A IMPLEMENTATION CONSIDERATIONS

We will introduce some implementation details for our model, specifically how the parameters for each distribution are learned.

Data Collection Model: In the first phase of the actor critic process we are in a *generation* phase of the data. Here, we have a prior over the latent variables $p_{\upsilon^Z}(z_t)$. This prior distribution can be defined as a Gaussian distribution with parameters as $p_{\upsilon^Z}(z_t) = \mathcal{N}(z_t; \mu_t^Z, \log \sigma_t^Z)$. To learn the parameters of this Gaussian prior we utilize an MLP (f) defined by $[\mu_t^Z, \log \sigma_t^Z] = f^Z(s_t)$ which takes the current state as input.

Now that we have the learned parameters, we can sample $z_t \sim p_{\upsilon^Z}(z_t)$ using the reparameterization trick given the learned mean and standard deviation. We denote this as $z_t \sim \text{reparameterize}(\mu_t^Z, \log \sigma_t^Z)$. At each time-step during the generation phase, we encounter a state from our environment. We find the distribution parameters and then sample z_t as denoted previously. The latent variable policy is then executed: $a_t \sim \pi_\theta(a_t|s_t, z_t)$. By applying action a_t on the environment we then obtain s_{t+1}, r_t, and repeat until termination. This leaves us with a sequence of states for use in inference. We denote the full trajectories generated in this phase with, τ, and $\tau_{(s)} = \{s_0, \ldots, s_T\}$ being the trajectories only containing states. Let us define the set of state only trajectories as, $\{\tau_{(s)}^0, \ldots, \tau_{(s)}^n\} \in \mathcal{D}_{(s)}$.

Training Model: The 2nd part of our actor critic algorithm where we learn the policy using the data in the inference phase. This is where we approximate the true posterior distribution over our latent variables. As in Z-forcing, we will use a *backwards* RNN, which takes the reversed sequence of states, $\tau_{(s)}^{\leftarrow}$ as input. We define this network as,

$$b_t = f^{\leftarrow}(s_{t+1}, b_{t+1}). \tag{10}$$

Each state b_t therefore contains information about future states and can be used to shape the approximate posterior distribution over latent variables to contain this information. We define the inference network with a normal distribution,

$$q_{\phi^Z}(z_t|b_t) = \mathcal{N}(z_t; \mu_t^q, \log \sigma_t^q), \tag{11}$$

where, as before, the parameters are learned with an MLP defined by $[\mu_t^q, \log \sigma_t^q] = f^q(b_t)$.

We derive a conditional generative model $p_\zeta(b|z)$ over the backwards states given the inferred latent variables $z_t \sim q_{\phi^Z}(z_t|b_t)$. Similar to before, we define the parameters of this Gaussian as $p_\zeta(b_t|z_t) = \mathcal{N}(z_t; \mu_t^{Ax}, \log \sigma_t^{Ax})$, with a MLP that produces $[\mu_t^{Ax}, \log \sigma_t^{Ax}] = f^{Ax}(z_t)$.

PGIF-DQN Implementation: The latent variable is utilized as an additional input to the DQN, in line with the other implementations of PGIF. In this model, we have a backwards loss used to train the backwards network parameters. Recall that the backwards RNN network hidden states are denoted as h_t. The final outputs of the backwards network are denoted as b_t. In our inference network, we condition on the backwards network hidden states, denoted by $q_\phi(z_t|h_t)$. The backwards loss is trained using the TD error with the backwards output network predicting the Q-values. The backwards loss is formulated as,

$$J_{\text{bwd}} = \mathbb{E}_{(s,a)\in\mathcal{B}}[\frac{1}{2}(r_t + \gamma_t \max_{a_{t+1}} Q_{\text{target}}(s_{t+1}, a_{t+1}) - b_{s_t,a_t})^2], \tag{12}$$

with sample batch $\mathcal{B}$ from an episodic replay buffer.

B ADDITIONAL RELATED WORK

Learning stochastic latent variable RNNs Deriving an approximate posterior over stochastic latent variables (Bayer & Osendorfer, 2015) conditioned on a backwards RNN hidden state has been practically useful in RNNs (Goyal et al., 2021) and has been able to overcome being trapped in local minima (Karl et al., 2017). Z-forcing (Goyal et al., 2021) is able to learn useful latent variables that capture higher level representations even with a strong auto-regressive decoder, by reconstructing hidden states in a backwards RNN. We use the main methods from Z-forcing to force our variables to learn a useful representation of the trajectory for the agent. Our method employs an auxiliary signal similar to (Karl et al., 2017) to design a cost with good convergence properties. Our work is a novel application of these stochastic latent variable RNNs in online and offline policy gradient methods.

Combining model-based and model-free RL Our proposed PGIF is a way of enabling a model-free RL algorithm to "look into the future". Although we do not learn explicit models of the MDP, our work may nevertheless be considered as combining elements of both model-based and model-free RL. Previous works have also claimed to do this, although in distinct ways. For example, previous work has proposed learning a low dimensional encoding of the environment and then performing planning in this abstract representation along with model-free RL (Fra, 2019). VPNs (Oh et al., 2017) improve representations in model-based RL, and use a single network along with supervised learning to learn transition and reward dynamics in an abstract representation of the state space. Attention augment agents utilize dynamics model roll-outs as input during policy optimization (Racanière et al., 2017). Our work takes a very different approach and incorporates a representation of future information using the hidden states of a backwards RNN, while leveraging VPN-style losses only to enforce information in the RNN to be propagated from inputs to outputs.

Auxiliary objectives in RL Incorporating auxiliary objectives in RL generally takes the form of intrinsic rewards. State-space density models have been used to derive intrinsic rewards that incentivize exploration, challenging the agent to find states that generate "surprise" (Ostrovski et al., 2017). Other works use the prediction error in an inverse model as a metric for curiosity (Pathak et al., 2017). The auxiliary objectives we utilize in this work serve the purpose of improving the incorporation of future dynamics information into policy optimization.

Value Prediction Networks In more detail, the VPN loss introduces the following additional learnable functions:

a) Encoding network : $f^{(\text{enc})} : s \to x$

b) Outcome encoder: $f^{(\text{out})} : x, a \to \hat{\gamma}, \hat{r}$

c) Value predictor: $f^{(\text{val})} : x \to \hat{V}(x)$

d) Transition predictor: $f^{(\text{trans})} : x, a \to \hat{x}'$

Given a backwards state b_t, we first embed this to $x_t^0 = f^{\text{emb}}(b_t)$ with (a). We then make predictions on future rewards $\hat{r}_t^l$, transition dynamics x_t^l, discounts $\hat{\gamma}_t^l$, and values $\hat{V}(x_t^l)$ using (b, c, d) for $l = 0, \ldots, k$. Finally, we compute the VPN loss as,

$$J_{\text{VPN}}^t = \sum_{l=0}^{k-1} (R_{t+l} - \hat{V}(x_t^l))^2 + (r_{t+l} - \hat{r}_t^l)^2 + (\log_\gamma \gamma_{t+l} - \log_\gamma \hat{\gamma}_t^l)^2, \tag{13}$$

where R_{t+l} is the empirically observed future discounted reward in the trajectory. The full auxiliary loss $J_{\text{Ax}}(\zeta)$ is then given by summing up the loss in (13) over all timesteps in all trajectories.

C Full Algorithm

We show the full algorithms for PGIF-PPO (Algorithm 2) with State based forcing and PGIF-SAC (Algorithm 4) with state based forcing in this section. Using VPN forcing simply changes the auxiliary loss calculations. The shared Backwards RNN Pass algorithm is in Algorithm 3.

We implement SAC with discrete actions for BSUITE umbrella-length environment using methods described in (Christodoulou, 2019).

D Experimental Parameters

In this section, we give the hyperparameters used for each of our experiments in Tables 3, 4, and 5. For MiniGrid, the parameters are in Table 6.

Algorithm 2 Algorithm with Advantage Policy Gradient and State Based Forcing

Require: Initial policy parameters: θ, Prior parameters: $\upsilon^{\text{Z}}, \phi^{\text{Z}}$, KL weight: β, Auxiliary loss Parameters: θ^{Ax}, Auxiliary loss weight: α Backwards net input parameters: θ^{in}, Backwards RNN parameters: θ^{bkw}, Backwards net output parameters: $\theta^{\text{bkw-out}}$, Initial hidden states $\mathbf{h_i}$.

for policy-step $k = 0, 1, 2, \ldots, N$ **do**
  Collect set of Trajectories $\mathcal{D} = \{\tau_i, \ldots\}$:
  repeat
    $[\mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}}] = f_{\upsilon^{\text{Z}}}(s_t)$
    $z_t \sim$ reparameterization$(\mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}})$
    Execute: $\pi_\theta(a_t|s_t, z_t) = f_\theta(s_t, z_t)$
    Observe s_{t+1}, r_t from environment.
    $\tau_i = \tau_i \cup \{s_t, a_t, s_{t+1}, r_t\}$
  until episode termination
  Compute advantage estimates $\hat{A}_t^k$ using GAE.
  for Trajectory: $\tau_i \in \mathcal{D}$ **do**
    $\mathbf{b_i}, \mathbf{h_i} =$ BackwardsPass$(\tau_i, \mathbf{h}_i, \theta^{(in)}, \theta^{\text{bkw}}, \theta^{\text{bkw-out}})$
    $\tau_i = \tau_i \cup \{\mathbf{b_i}\}$
  Reset: $J_{\text{PG}} = 0$
  for timestep $t \in \{0, \ldots, T\}$ **do**
    $\forall \{s_t, b_t, a_t\} \in \mathcal{D}$: $[\mu_t^{\text{Z}}, \log \sigma_t^{\text{Z-PG}}] = f_{\phi^{\text{Z}}}(b_t, s_t)$
    $z_t^{\text{PG}} \sim$ reparameterization$(\mu_t^{\text{Z-PG}}, \log \sigma_t^{\text{Z-PG}})$
    $\pi_\theta(a_t|s_t, z_t^{\text{PG}}) = f_\theta(s_t, z_t^{\text{PG}})$
    $[\mu_t^{\text{Ax}}, \log \sigma_t^{\text{Ax}}] = f_{\theta^{\text{Ax}}}(z_t^{\text{PG}})$
    $J_{\text{KL}} =$ KLDivergence$(\mu_t^{\text{Z-PG}}, \log \sigma_t^{\text{Z-PG}}, \mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}})$
    $J_{\text{Ax}} =$ LogProbGaussian$(b_t, \mu_t^{\text{Ax}}, \log \sigma_t^{\text{Ax}})$
    $J_{\text{PG}} \overset{+}{=} \mathbb{E}_{\tau_i \in \mathcal{D}}\Big[\log \pi_\theta(a_t|s_t, z_t^{\text{PG}}) \hat{A}_t^k\Big] - \alpha J_{\text{Ax}} - \beta J_{\text{KL}}$
  Update all parameters w.r.t: J_{PG}
  Update value function estimates with any method.

Algorithm 3 BackwardsPass

Require: Trajectory: τ, Hidden states h, Input MLP Parameters: θ^{in}, Backwards RNN Parameters: θ^{bkw}, Output net parameters: $\theta^{\text{bkw-out}}$.

$\tau^{-1} =$ Reverse-Order(τ)
$x_{\text{bkw}} = f_{\theta^{\text{in}}}(\tau^{-1})$
$h_{\text{bkw}} = f_{\theta^{\text{bkw}}}(x_{\text{bkw}}, h)$ (Operates over the entire sequence)
$x_{\text{bkw-out}} = f_{\theta^{\text{bkw-out}}}(h_{\text{bkw}})$,
$x_{\text{bkw-out}} =$ Reverse-Order$(x_{\text{bkw-out}})$
$h_{\text{bkw}} =$ Reverse-Order(h_{bkw})
return $x_{\text{bkw-out}}, h_{\text{bkw}}$

Algorithm 4 SAC Algorithm with State Based Forcing

Require: Initial policy parameters: θ, Z forcing parameters: $\upsilon^{\text{Z}}, \upsilon^{\text{U}}, \phi^{\text{Z}}, \phi^{\text{U}}$, KL weight: β, Auxiliary loss Parameters: $\theta^{\text{Ax-PG}}, \theta^{\text{Ax-TD}}$, Auxiliary loss weight: α Backwards net input parameters: θ^{in}, Backwards RNN parameters: $\theta^{\text{bkw-Z}}, \theta^{\text{bkw-U}}$, Backwards net output parameters: $\theta^{\text{bkw-out-Z}}, \theta^{\text{bkw-out-U}}$, Initial hidden states $\mathbf{h_i}^{\text{Z}}, \mathbf{h_i}^{\text{U}}$.

for policy-step $k = 0, 1, 2, \ldots, N$ **do**
  Collect set of Trajectories $\mathcal{D} = \{\tau_i, \ldots\}$:
  repeat
    $[\mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}}] = f_{\upsilon^{\text{Z}}}(s_t)$
    $z_t \sim \text{reparameterization}(\mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}})$
    Execute: $\pi_\theta(a_t|s_t, z_t) = f_\theta(s_t, z_t)$
    Observe s_{t+1}, r_t from environment.
    $\tau_i = \tau_i \cup \{s_t, a_t, s_{t+1}, r_t\}$
  until episode termination
  for Trajectory: $\tau_i \in \mathcal{D}$ **do**
    $\mathbf{b_i}^{\text{Z}}, \mathbf{h_i}^{\text{Z}} = \text{BackwardsPass}^{\text{Z}}(\tau_i, \mathbf{h}_i^{\text{Z}}, \theta^{\text{in-Z}}, \theta^{\text{bkw-Z}}, \theta^{\text{bkw-out-Z}})$
    $\mathbf{b_i}^{\text{U}}, \mathbf{h_i}^{\text{U}} = \text{BackwardsPass}^{\text{U}}(\tau_i, \mathbf{h}_i^{\text{U}}, \theta^{\text{in-U}}, \theta^{\text{bkw-U}}, \theta^{\text{bkw-out-U}})$
    $\tau_i = \tau_i \cup \{\mathbf{b_i}^{\text{U}}, \mathbf{b_i}^{\text{Z}}\}$
  Reset: $J_{\text{PG}}, J_{\text{Q}} = 0$
  for timestep $t \in \{0, \ldots, T\}$ **do**
    $\forall \{s_t, b_t, a_t, s_{t+1}, b_t^{\text{Z}}, b_t^{\text{U}}\} \in \mathcal{D}$
    Compute a_{t+1}
    $[\mu_{t+1}^{\text{Z}}, \log \sigma_{t+1}^{\text{Z}}] = f_{\upsilon^{\text{Z}}}(s_{t+1})$
    $z_{t+1} \sim \text{reparameterization}(\mu_{t+1}^{\text{Z}}, \log \sigma_{t+1}^{\text{Z}})$
    $a_{t+1} \sim \pi_\theta(a_{t+1}|s_{t+1}, z_{t+1}) = f_\theta(s_{t+1}, z_{t+1})$
    Compute target Q (U-Tar to denote distribution parameters)
    $[\mu_{t+1}^{\text{U-Tar}}, \log \sigma_{t+1}^{\text{U-Tar}}] = f_{\upsilon^{\text{U}}}(s_{t+1})$
    $u_{t+1}^{\text{Tar}} \sim \text{reparameterization}(\mu_{t+1}^{\text{U-Tar}}, \log \sigma_{t+1}^{\text{U-Tar}})$
    $y(r_t, s_{t+1}, d) = r_t + \gamma(1-d)\Big(Q_{\psi\text{-target}}(s_{t+1}, a_{t+1}, u_{t+1}^{\text{Tar}}) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1}, z_{t+1}))\Big)$
    Update Q functions
    $[\mu_t^{\text{U}}, \log \sigma_t^{\text{U}}] = f_{\phi^{\text{U}}}(b_t^{\text{Q}}, s_t)$
    $u_t \sim \text{reparameterization}(\mu_{t+1}^{\text{U}}, \log \sigma_{t+1}^{\text{U}})$
    $J_{\text{KL-TD}} = \text{KLDivergence}(\mu_t^{\text{U}}, \log \sigma_t^{\text{U}}, \mu_t^{\text{U-Tar}}, \log \sigma_t^{\text{U-Tar}})$
    $[\mu_t^{\text{Ax-TD}}, \log \sigma_t^{\text{Ax-TD}}] = f_{\theta^{\text{Ax-TD}}}(u_t)$
    $J_{\text{Ax-TD}} = \text{LogProbGaussian}(b_t^{\text{Q}}, \mu_t^{\text{Ax-TD}}, \log \sigma_t^{\text{Ax-TD}})$
    $J_Q \stackrel{+}{=} (Q_\psi(s_t, a_t, u_t) - y(r_t, s_{t+1}, d)) + \alpha J_{\text{Ax-TD}} + \beta J_{\text{KL-TD}}$
    Update Target Networks
    $(\psi - \text{target}) \leftarrow \rho(\psi - \text{target}) + (1-\rho)\psi$
    Update Policy
    $[\mu_t^{\text{Z-PG}}, \log \sigma_t^{\text{Z-PG}}] = f_{\phi^{\text{Z}}}(b_t^{\text{Z}}, s_t)$
    $z_t^{\text{PG}} \sim \text{reparameterization}(\mu_t^{\text{Z-PG}}, \log \sigma_t^{\text{Z-PG}})$
    $a_t \sim \pi_\theta(a_t|s_t, z_t^{\text{PG}}) = f_\theta(s_t, z_t^{\text{PG}})$
    $[\mu_t^{\text{Ax-PG}}, \log \sigma_t^{\text{Ax-PG}}] = f_{\theta^{\text{Ax-PG}}}(z_t^{\text{PG}})$
    $J_{\text{KL-PG}} = \text{KLDivergence}(\mu_t^{\text{Z-PG}}, \log \sigma_t^{\text{Z-PG}}, \mu_t^{\text{Z}}, \log \sigma_t^{\text{Z}})$
    $J_{\text{Ax-PG}} = \text{LogProbGaussian}(b_t^{\text{Z}}, \mu_t^{\text{Ax-PG}}, \log \sigma_t^{\text{Ax-PG}})$
    $J_{\text{PG}} \stackrel{+}{=} \mathbb{E}_{\tau_i \in \mathcal{D}}\Big[Q_\psi(s_t, a_t, u_t) - \alpha \log \pi_\theta(a_t|s_t, z_t^{\text{PG}})\Big] + \alpha J_{\text{Ax-PG}} + \beta J_{\text{KL-PG}}$
  Update all parameters w.r.t: J_{PG} or J_{Q}
  Update value function estimates with any method.

Parameter	Value
Optimizer	Adam
Learning rate	$5e^{-4}$
Batch size	250
Actor and Critic network dimensions	$(256, 256)$
RNN Dim	15
RNN-Embedding Dim	20
Z or Q Dim	5
Z-force Neural net dim	20
Initial random exploration steps	10000
Replay Buffer Size	1,000,000 steps
Discount	0.99
Evaluation Episodes	5

Table 3: Parameters used for PGIF and SAC in the online experiments with MuJoCo.

Parameter	Value
Optimizer	Adam
Learning rate	$1e^{-3}$
Value penalty	True
Batch size	250
Actor and Critic network dimensions	$(200, 300)$
Q value ensemble	2
RNN Dim	15
RNN-Embedding Dim	20
Z or Q Dim	10
Z-force Neural net dim	20
Initial random exploration steps	10000
Replay Buffer Size	$1,000,000$ steps
Discount	0.99
Evaluation Episodes	5
BRAC Value Penalty α	0.1

Table 4: Parameters used for offline RL experiments

Parameter	Value
Hidden Dim	128
Activation	ReLu
Heads	1
Heads	1
Layers	3
Attention Dropout	0.1

Table 5: Parameters used for the transformers experiments.

Parameter	Value
Adam optimizer learning rate	$7 \cdot 10^{-4}$
$\beta_1; \beta_2$	$0.9; 0.999$
ϵ	10^{-5}
entropy coefficient	10^{-2}
RNN Dim	75
RNN-Embedding Dim	90
Z or Q Dim	5
Z-force Neural net dim	80
value loss coefficient	0.5
discount	0.99
maximum norm of gradient in PPO	0.5
number of PPO epochs	4
batch size for PPO	256
entropy coefficient	10^{-2}
clip parameter	0.2

Table 6: Parameters for benchmark tasks in MiniGrid

Training the behavior policy for BRAC offline RL experiments uses behavior cloning with 300,000 time steps.

For the purpose of the backwards RNN, since we have variable episode length we pad the episode sequence to the maximum length with zeros. The maximum number of timesteps is generally 1000 in MuJoCo. The training of the RNN with this long sequence and the pre-processing steps involved are expensive computationally. For offline experiments, our replay buffer is episodic, so we train with an entire episode at each update. For online we train with a batch of randomly sampled steps from the replay buffer (and obtain the respective episodes that those steps were in for the training of the backwards RNN).

For the weight parameters α and β, we recommend that the weight on the KL be increased at a small fixed constant rate during training so that the final returned policy uses minimal information about the future. We use an initial weight of $1e^{-4}$ for the KL and auxiliary losses and increase it by $1e^{-6}$ at each training iteration for experiments to a max weight of 1. Thus, at the end of training the policy and value functions should be minimally reliant on this privileged information.

For an empirical comparison to Value driven hindsight modelling (HiMo), the only codebase is a notebook and is quite rudimentary (See: `https://github.com/deepmind/deepmind-research/blob/master/himo/himo_example.ipynb`); We attempt a good faith implementation on the MiniGrid set of environments for HiMo to make at least some comparisons, but do not extend this method into other environments relying on different codebases.

E Online Experiment Learning Curves

In this section, we show the training curves for the online experiments with full observability in Figure 4. We compare PGIF against SAC only, since we see that PPO performs much worse from initial experiments. Note that in practicality, it is best to reduce the maximum number of steps in the MuJoCo environments to 500 for better computational efficiency in the backwards RNN steps (without much change in final reward). We note that our method appears to have a small reduction in standard error compared to vanilla SAC.

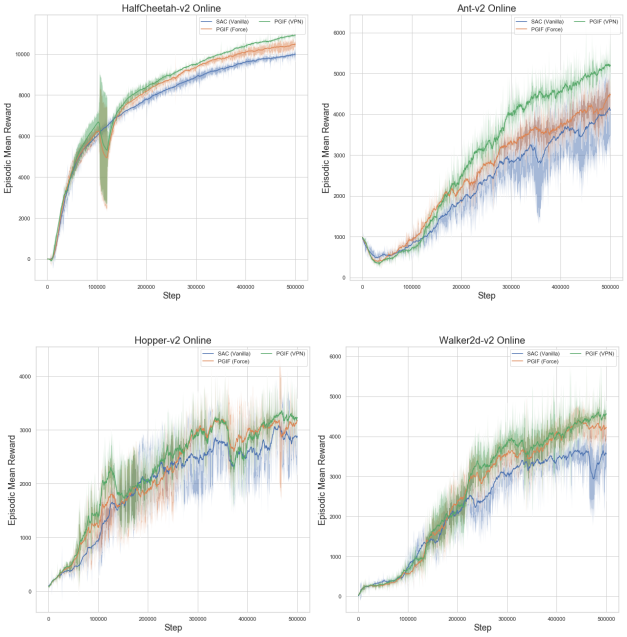

Figure 4: The online episodic mean reward evaluated over 5 episodes every 500 steps for MuJoCo continuous control RL tasks with full observability. We show the average over 5 random seeds. $500,000$ environment step interactions are used. The shaded area shows the standard error.

We aim to investigate the effect PGIF learning with a lower batch size. We notice that PGIF has better performance at lower batch sizes. It may be an interesting direction to further investigate this anomaly. In Figure 5, we show the performance of PGIF with a batch size of 25.

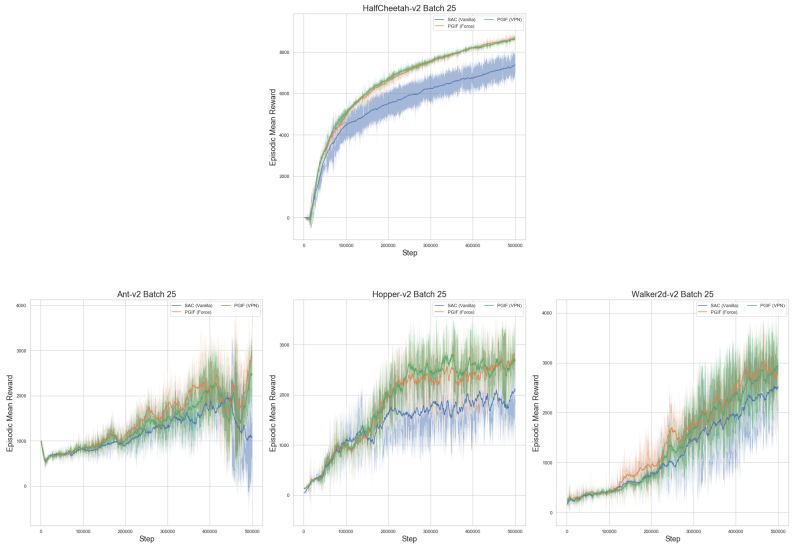

Figure 5: The online episodic mean reward evaluated over 5 episodes every 500 steps for MuJoCo tasks with full observability. We show the average over 5 random seeds. $500,000$ environment step interactions are used. The model is updated with a batch size of 25. The shaded area shows the standard error.

F ABLATION ANALYSIS

In the following experiment, we compare the contributions of both PGIF implemented for the policy only vs. PGIF implemented for the value function only. It appears that the main contribution of our method comes from the implementation using the value function. "Best" refers to the best of either VPN or State forcing auxiliary losses. Results are in Table 7 and 8.

Environment	Value Only - Best	Policy Only - Best	PGIF-Best
HalfCheetah	12453	10713	12780
Ant	4573	4230	5173
Hopper	3188	2805	3211
Walker2d	4689	3786	4519

Table 7: PGIF actor vs critic ablation analysis. Performance on the online RL tasks showing the average episodic return. The final average return is shown after training the algorithm for $500,000$ episodes and then evaluating the policy over 5 episodes. Results show an average of 5 random seeds.

Environment	Value Only - Best	Policy Only - Best	PGIF-Best
ant-medium	3279	2877	3250
ant-medium-expert	2659	3001	3048
hopper-medium	2155	2402	2327
walker2d-medium	3888	3874	3989
halfcheetah-medium	5762	6029	6037
halfcheetah-medium-expert	5116	5389	5418
antmaze-umaze	0.5	0.95	0.95

Table 8: Actor vs critic PGIF ablation analyse. Performance on the offline RL tasks showing the average episodic return. The final average return is shown after training the algorithm for $500,000$ episodes and then evaluating the policy over 5 episodes. Results show an average of 5 random seeds. The value after $\pm$ shows the standard error.

In the following experiment, we compare the contribution of the auxiliary losses without the backwards RNN. Results are in Table 9.

Environment	VPN-Aux	StateForce-Aux	Both	PGIF-Best
HalfCheetah	9950	9813	9917	12781
Ant	4165	4111	4197	5173
Walker	3324	2997	3120	4519
Hopper	2951	2680	2742	3211
AntMaze	0.1	0	0	0.5

Table 9: PGIF without backwards RNN auxiliary loss ablation analysis. Performance on the online RL tasks showing the average episodic return. The final average return is shown after training the algorithm for $500,000$ episodes and then evaluating the policy over 5 episodes. Results show an average of 5 random seeds.

G EXPERIMENTS USING TRANSFORMERS

Using transformers for processing the future trajectory into latent variables offers a few key benefits over traditional RNN architectures, namely better computational efficiency with long sequences and improved ability to model long timescale interactions. These provide a few interesting benefits for use with our architecture, namely due to the fact that some of our environments generate sequences

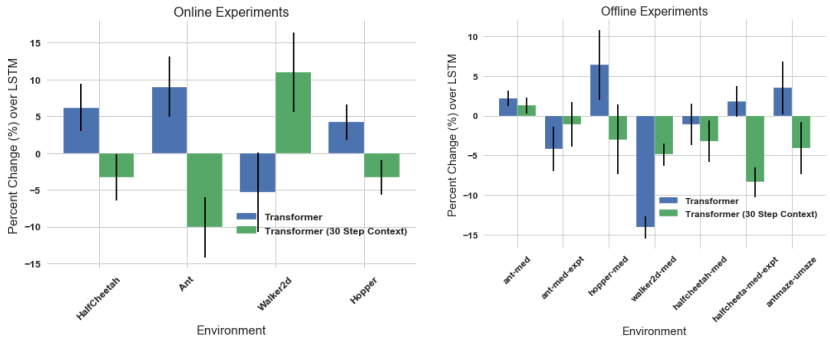

Figure 6: The percent improvement evaluated over 5 episodes by replacing the LSTM backwards network with a transformer. The 30 timestep transformer has a context size of a 30 timesteps look-ahead into the future of the trajectory. We show the average over 5 random seeds taking the final policy evaluation mean episodic reward as the value. $500,000$ environment step interactions are used. The error bars shows the standard error.

up to 1000 timesteps, which can be onerous for an RNN to process. In the following experiments, we simply replace our RNN with a transformer. This is inspired by work that models trajectories using a transformer (Janner et al., 2021) for offline RL. Additionally, we show the results of experiments where we use a K-fixed length context of future states as the input to our transformer. A short length context is much faster to process than the entire sequence of upstream states, but there is a decrease in performance, suggesting that the entire trajectory contains useful information. We show these results in Figure 6. The run-time of training was decreased by approximately 40%.

H ANTMAZE EXPERIMENT

We additionally make comparisons to SAC and PPO-LSTM is the AntMaze environment (Florensa et al., 2017) where we have a simple U-shaped maze with a goal at the end having sparse reward. The agent receives a reward of $+1$ if it reaches the goal (within an L2 distance of 5) and 0 elsewhere. The Ant starts at one end of a U-shaped corridor and must navigate to the goal location at the other end. This task is particularly challenging since the reward is extremely sparse. We show the results of this experiment here, where we see that the baselines are unable to make any progress on this challenging task, while PGIF is able to solve the task to a point where it successfully navigates to the goal location in the maze 50% of the time. We hypothesize our algorithm has benefits in these environments since as soon as it obtains a reward signal it can adapt quickly and make use of the signal by incorporating it in both policy and value optimization, therefore accelerating learning.

Method	Mean Episodic Reward
PPO-LSTM	0.00 ± 0.00
SAC	0.00 ± 0.00
PGIF-PPO (VPN)	0.20 ± 0.13
PGIF-PPO (Force)	0.30 ± 0.15
PGIF-SAC (VPN)	$\mathbf{0.40} \pm 0.16$
PGIF-SAC (Force)	$\mathbf{0.50} \pm 0.16$

Table 10: Mean episodic reward on the AntMaze environment over 10 random seeds trained with 3 million environment steps. A sparse reward of $+1$ is obtained at the end of the episode if the agent successfully reaches the goal state within an L2 distance of 5. The number after $\pm$ is the standard error.

I ABLATIONS DEMONSTRATING THE IMPACT OF Z-FORCING

I.1 INCREASING NETWORK CAPACITY

In order to show that the performance improvements from PGIF are not due to increased network capacity that the PGIF networks provide, we show learning curves using SAC with increased network capacities past 256, with the number of hidden units $\in \{(256, 256), (300, 300), (500, 500), (600, 600))\}$. We show the results on two fully observable online MuJoCo continuous control tasks in Figure 7.

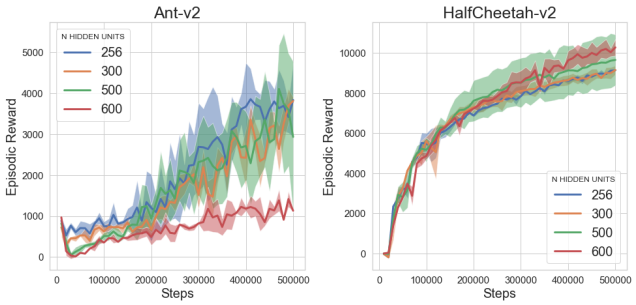

Figure 7: The online episodic mean reward evaluated over 3 seeds every 1000 steps for MuJoCo online RL tasks. The number of hidden states in the policy and value network is varied. We show the average over 3 random seeds. The shaded area shows the standard deviation.

I.2 n-STEP RETURNS

We now show that SAC does not benefit from n-step returns in Figure 8. This is to show that n-step methods are not competitive with SAC. We show SAC on two fully observable online MuJoCo continuous control tasks with $n \in \{1, 3, 5, 10\}$.

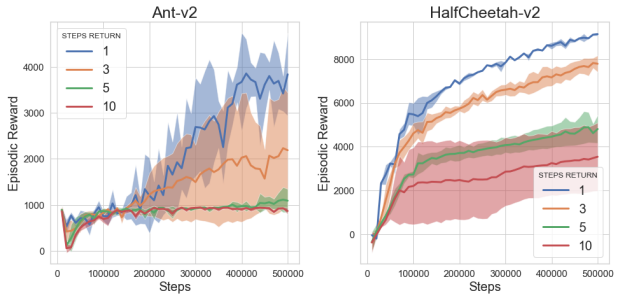

Figure 8: The online episodic mean reward evaluated over 3 seeds every 1000 steps for MuJoCo RL tasks. The number of steps (n) using an n-step return is varied. We show the average over 3 random seeds. The shaded area shows the standard deviation.

I.3 MULTIPLE UPDATES PER STEP

We now show that SAC does not benefit from multiple updates per step in Figure 9. This is to show that more Bellman updates per step do not lead to significantly better performance with SAC. We show SAC on two fully observable online MuJoCo continuous control tasks with $n \in \{1, 3\}$.

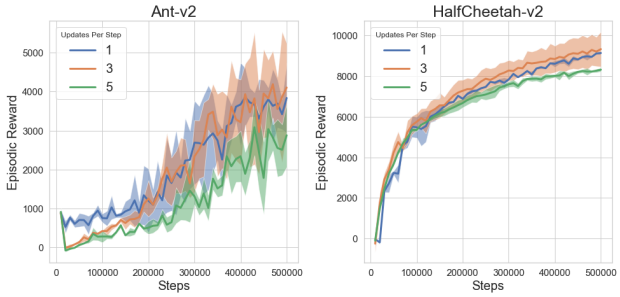

Figure 9: The online episodic mean reward evaluated over 3 seeds every 1000 steps for MuJoCo RL tasks. The number of updates per step is varied. We show the average over 3 random seeds. The shaded area shows the standard deviation.

I.4 NO Z-FORCING

We now demonstrate how our method performs when the auxiliary loss is removed, to show that it the loss required for increased performance. In this experiment to KL loss is kept. We show this ablation on a partially observable online Hopper-v2 MuJoCo environment in Figure 10.

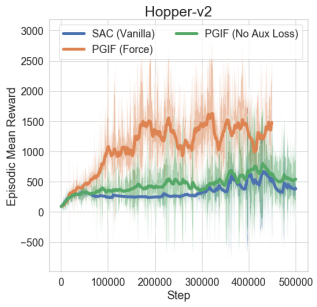

Figure 10: The online episodic mean reward evaluated over 3 seeds every 1000 steps for a MuJoCo RL task. The auxiliary loss is removed in. We show the average over 3 random seeds. The shaded area shows the standard deviation.

J COMPUTING INFRASTRUCTURE

The cluster used to run these experiments has 688 NVIDIA V100-SXM2 GPUs.