APPENDIX

A DOMAINS FOR EMPIRICAL EVALUATION

We consider six domains of Metaworld Yu et al. (2020) and three domains of DM Control Tunyasuvunakool et al. (2020) for our empirical evaluation.

For all our experiments we use the internal state representation as the agent state. This includes reward learning, policy learning and world model training. The state features are as described in Yu et al. (2020). Further we follow Park et al. (2022) to utilize the packaged task rewards in MetaWorld for our synthetic oracles.

B REWARD ARCHITECTURE FOR PRIOR

We follow Park et al. (2022) reward architecture for Metaworld and DMControl domains.

C WORLD MODEL LEARNING

We generally follow the training paradigm suggested in TWM Robine et al. (2023) to learn our forward dynamics model. Furter, we follow the same architecture with minor changes to work with continuous control domains : :

1. TWM is proposed for discrete actions image based atari Brockman et al. (2016) domains, therefore we modify the observation encode layer to take in a 1-D state vector (instead of 3 channel image vector). Additionally, the original TWM makes use of frame-stacking which we do not.
2. We change the state predictor to take continuous valued vector based action space (as in our case) instead of discrete 1-D actions (as in Atari).

Finally, we modify the world model sequence length and memory length to be same as the query length such that we can feed the whole trajectory as an input to the TWM model.

To obtain an attention map from TWM we auto-repressively feed in a trajectory, i.e. first we feed in the first state,action tuple as trajectory of size 1, then the next state, action tuple and so on. At the final step we take the attention vector from the attention layers in the architecture.

C.1 OBSERVATION MODEL

The observation model in our world model encodes the input state,action tuple. The observation model has two components, an encode and a decoder both of which are multi-layer perceptrons.

The encoder has two hidden layers of size 512 and an output layer of size (32,). The decoder layer has an input size of (32,) followed by two hidden layers of size 512 and the output shape is same as the number of state features. As recommended by Robine et al. (2023) we use SiLU activation in the MLP.

C.2 LATENT STATE PREDICTOR

We follow Robine et al. (2023) to contruct the world model architecture with minor changes required to adapt to continuous control action space and 1-D state vector. That is, vanilla TWM requires categorical action embeddings (because of the discrete action space) but we do not. Finally, TWM Robine et al. (2023) can operate in different modes with respect to the prediction heads from the latent state predictor such as, next state prediction, reward prediction, and discount prediction. For Hindsight PRIOR we only require the next-state prediction output head.

Similarly TWM supports different input modes i.e. it can take (state,action) tuple and (state, action, reward) tuple. We compute state conditioned hindsight priors and therefore only need the (state, action) as input. Robine et al. (2023) is inconclusive on the need for rewards in the inputs. Moreover,

since PRIOR is learning the reward model to begin with we only use (state, action) as input. Robine et al. (2023) uses a Transformer XL architecture which we borrow as is.

C.3 Training World Model

We share the replay buffer of the agent with the world model. Similar to PbRL paradigm, the world model first gets access to a bank of state transitions during PbRL's pre-training step. Once the pretraining step is complete the world model is trained on this data (to get the observation model). After this pretraining step only the dynamics model is trained on incoming data every jth step of PbRL loop. The world model is trained on its bank of transitions as suggested by Robine et al. (2023).

D Policy Evaluation Metrics

Algorithms are evaluated according to their learning curves over the course of policy and reward training along with their normalized returns for Deep Mind Control Suite and normalized success rates for MetaWorld Lee et al. (2021b). An algorithm's normalized return or success rate measures how well the policies trained jointly with a preference-learned reward function recovers the performance of a policy trained on the target reward function. For each episode of policy training, the PbRL or SAC policy's return or success rate is computed and then the PbRL policy is evaluated based on how well it is able to recover "optimal" performance approximated with SAC trained on the ground truth reward. The normalized returns are computed as:

$$\text{normalized returns} = \frac{1}{T}\sum_{t}\frac{r_\psi(s_t, \pi_\phi^{\hat{r}_\psi}(a_t))}{r_\psi(s_t, \pi_\phi^{r_\psi}(a_t))}, \tag{6}$$

where T is the number of policy training training steps, $\bar{r}_\psi$ is the target reward function, $\pi_\phi^{\hat{r}_\psi}$ is the policy trained in conjunction the learned reward function, and $\pi_\phi^{\bar{r}_\psi}$ is the policy trained on the target reward function. The normalized success rates are computed as:

$$\text{normalized success rates} = \frac{1}{T}\sum_{t}\frac{\text{success}(\pi_\phi^{\hat{r}_\psi}(a_t))}{\text{success}(\pi_\phi^{r_\psi}(a_t))}, \tag{7}$$

where $\text{success}(\cdot)$ indicates whether action a_t resulted in the policy reaching the goal state.

E Hyper-parameters

A results in the paper are reported over five random seeds: $[12345, 23456, 34567, 45678, 56789]$.

E.1 Train Hyper-parameters

This section specifies the hyper-parameters (e.g. learning rate, batch size, etc) used for the experiments and results. The SAC, and PEBBLE experiments all match those used in Haarnoja et al. (2018) and Lee et al. (2021a) respectively. The SAC hyper-parameters are specified in Table 2, the PEBBLE hyper-parameters are given in Table 3, the hyper-parameters used to train on with Hindsight PRIOR are in Table 4, and finally the hyperparameters used for training our world model in Table 5. Table 5 only mentions the hyper-parameters that we change in the prescribed Robine et al. (2023) configuration.

Table 2: Training hyper-parameters for SAC (Haarnoja et al., 2018).

HYPER-PARAMETER	VALUE
Learning rate	1e-3 (cheetah), 5e-4 (walker), 1e-4 (quadruped), 3e-4 (Meta-World)
Batch size	512 (DMC), 1024 (Meta-World)
Total timesteps	500k, 1M (quadruped, sweep into)
Optimizer	Adam (Kingma & Ba, 2015)
Critic EMA τ	5e-3
Critic target update freq.	2
$(\mathcal{B}_1, \mathcal{B}_2)$	$(0.9, 0.999)$
Initial Temperature	0.1
Discount γ	0.99

Table 3: PEBBLE hyper-parameters (Lee et al., 2021a).

HYPER-PARAMETER	VALUE
Learning rate PEBBLE	3e-4 (Metaworld), 5e-4 (Walker, Cheetah), 1e-4 (Quadruped)
Optimizer	Adam (Kingma & Ba, 2015)
Segment length l	50
Feedback amount / number queries (M)	1000/100, 100/10 (DMC) 10000/50, 4000/20, 2000/25, 400/10 (Meta-World)
Steps between queries (K)	20000 (walker, cheetah), 30000 (quadruped), 500 (Meta-World)

Table 4: Hindsight PRIOR hyper-parameters.

HYPER-PARAMETER	VALUE
λ	1000 (MetaWorld), 5 (DMC)
update frequency j	2000 steps

Table 5: World Model hyper-parameters in Hindsight PRIOR

HYPER-PARAMETER	VALUE
world model sequence length	50
world model memory length	50
world model batch size	100

F NORMALIZED RETURNS AND SUCCESS RATES BY TASK

The mean normalized scores for each algorithm on each task are given for MetaWorld in Table 6 and DMC in Table 7.

A two-tailed paired t-test with dependent means (significant at p ¡ .05) was performed over the normalized returns and success rates to determine that Hindsight PRIOR's performance gains are statistically significant over:

1. **MetaWorld**: PEBBLE ($t = -3.92$, $p = 0.006$), SURF ($t = -2.85$, $p = 0.025$), RUNE ($t = -5.39$, $p = 0.001$), and MRN ($t = -4.91$, $p = 0.002$)
2. **DMC**: PEBBLE ($t = -3.47$, $p = 0.00843$), SURF ($t = -2.52$, $p = .03541$), RUNE ($t = -7.745967$, $p = 0.00006$), and MRN ($t = -2.392232$, $p = 0.0437$)

Table 6: Normalized Success Rate for MetaWorld domains

Env/Algorithm	**PRIOR**	**PEBBLE**	**SURF**	**RUNE**	**MRN**
Window Open	**0.55**	0.40	0.45	0.35	0.43
Door Open	**0.65**	0.62	0.55	0.53	0.53
Drawer Open	**0.68**	0.59	0.59	0.59	0.53
Sweep Into	**0.69**	0.51	0.54	0.52	0.62
Button Press	**0.69**	0.65	0.65	0.67	0.65
Door Unlock	**0.68**	0.62	0.64	0.53	0.54

Table 7: Normalized Returns for DM Control domains

Env/Algorithm	**PRIOR**	**PEBBLE**	**SURF**	**RUNE**	**MRN**
Walker Walk	**0.60**	0.46	0.66	0.47	0.59
Cheetah Run	**0.51**	0.33	0.36	0.39	0.46
Quadruped Walk	**0.65**	0.66	0.64	0.60	0.54

G ADAPTED BASELINES FOR PBRL

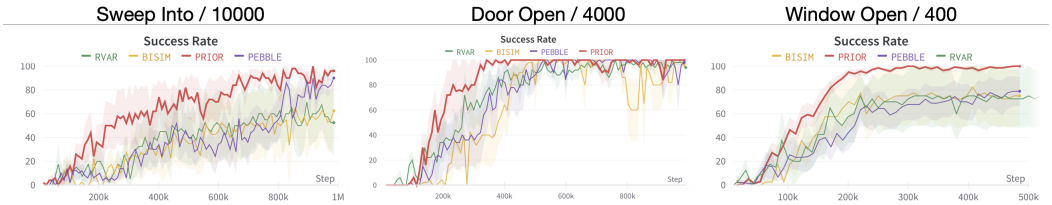

Figure 5: Learning curves of PRIOR, BISIM, RVAR and baseline PEBBLE

G.1 UNINFORMED RETURN REDISTRIBUTION (RVAR)

Inspired from Ren et al. (2021) we consider the Uninformed return redistribution baseline that essentially has reward targets as the mean reward in the trjaectory, i.e. $r_{target} = \frac{G}{|\tau|}$. This essentially reduces the variance of the trajectory rewards (hence the name RVAR). As discussed previously, RVAR is marginally better than PEBBLE and PRIOR is superior to such an uninformed redistribution technique. (See Fig. 5).

G.2 BISIMULATION METRIC : BISIM

To incorporate the bisimulation metrics into PbRL, we use the following bisimulation metric loss:

$$\mathcal{J}_{bisim}(\psi) = (||z_i - z_j||_1 - |r_i - r_j| - \gamma||z_i^{'(z_i,a_i)} - z_j^{'(z_j,a_j)})^2, \tag{8}$$

adapted from Equation 4 in Kemertas & Aumentado-Armstrong (2021). We use the reward model to provide the predicted rewards r_i, r_j required by bisim loss. Further we use the penultimate layer as the embedding layer to obtain z_i, z_j. Next, we add an additional head from the penultimate layer that predicts the next state embedding to obtain z_i', z_j' as next state predictors. Finally, we reuse the trajectory buffer (as used by Hindsight PRIOR) to obtain the states on which we compute the bisim-target distance and finally optimize the loss as above. We verify that the BISIM adapted baseline offers only marginal improvements over baseline PEBBLE as given in Fig. 5.

G.3 Normalized Redistribution PRIOR (NRP)

Readers may question whether the raw attention values extracted from the forward dynamics prediction task are the best choice or certain post processing may further improve PRIOR's performance. We construct a variant of PRIOR referred to as PRIOR-NRP where we perform a min-max normalization of the attention vector α. That is :

$$\hat{\alpha}_i = softmax(\frac{\alpha_i - \alpha_{min}}{\alpha_{max} - \alpha_{min}}) \tag{9}$$

The above equation amplifies the attention values thereby preventing the reward targets to become uniform. Fig. 6 shows that default PRIOR (without any postprocessing of attention) performs well and NRP like post-processing is not required.

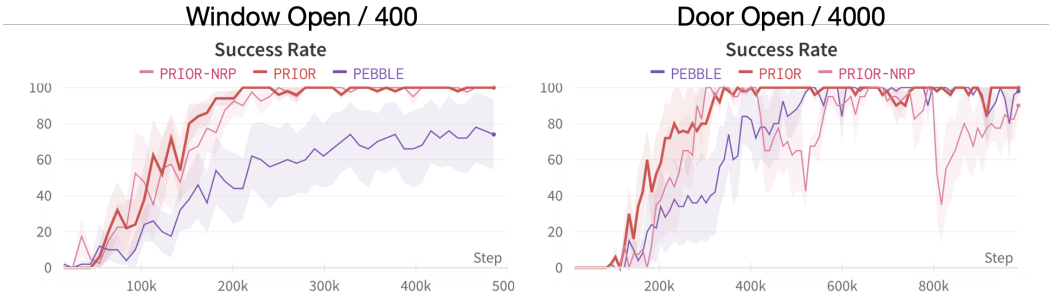

Figure 6: Learning curves of PRIOR, PRIOR-NRP and baseline PEBBLE

H Long Trajectories

Figure 7 shows the performance of PRIOR compared to PEBBLE when the trajectory length is 4x (200). While the Transformer XL architecture is already known for scaling to much longer trajectory lengths Robine et al. (2023) our experiments conclude that this is indeed the case for the challenging continuous control domains.

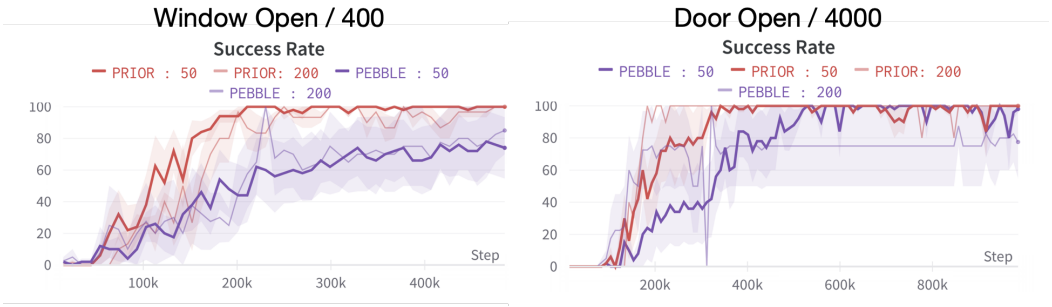

Figure 7: Learning curves of PRIOR and PEBBLE on query segment lengths of 50 and 200.

I QUALITATIVE STATE IMPORTANCE ASSESSMENT

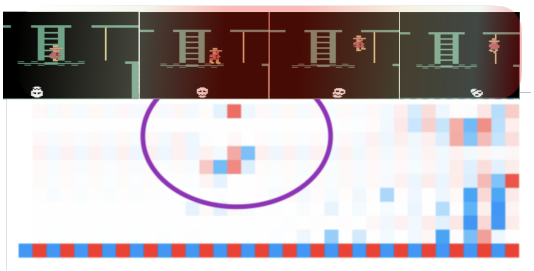

Figure 8: Attention analysis on Montezuma's Revenge. The plot shows a situation where the agent (red animated character) attempts to jump from the green platform towards the rope. Each row represents a layer in the Transformer Model and the columns represent time step where left most column is time t-T and right most column is the present state of the agent. In hindsight, we compute this attention map to obtain the attention over the past states (given as blue cells) and attention over past actions (given as red cells). Note that the map begins with a blue cell (as $s_0, a_0, a_1, a_1 \cdots$). We highlight that the agent attends to past states which corresponds to point of launch which can be considered as an important summary state for the complete trajectory of jumping from the platform to the rope.

We qualitatively evaluate the states identified by the attention weights $\boldsymbol{\alpha}$ as important for a given trajectory. We evaluate whether the attention map $\mathcal{A}$ salience over a trajectory can capture critical events. We conduct experiments on Atari (Montezuma's Revenge) and Metaworld (Window-Open, Door Open) domains, and seek to answer whether the world model attends to states in the complete history (even for Markovian transitions) and whether that correlates with "critical" events (similar to how **?** describes it). From figure 8 we can see that the forward model does attend to past states and actions. Moreover, upon closer inspection by executing known maneuvers, such as jumping across the platform to the rope in Montezuma's Revenge, we find that the agent is attending to past critical events like the point of launch. We do not expect attention over states in history to be aligned with human's reward function as human may have arbitrary preference unknown to the dynamics model. However, we do expect the attention to be aligned with "critical events" loosely defined as apriori states enough to summarize a trajectory. To further investigate this we force the reward model to predict rewards aligned with the PRIOR reward targets by very high value λ_{prior} in equation eqn and find that the reward model is unable to learn preference-relevant reward model at all. This shows that PRIOR reward targets are not task specific but contains salience information on states which can be considered "critical" in the sampled trajectory.