A More Analysis

A.1 Objective for the Encoder, Model, and Policy

This section describes how the objective for the encoder, model, and policy (Eq. 6) is derived from our overall objective (Eq. 1). Our aim is to maximizing the sum of (information-augmented) rewards (Eq. 3), starting at state $\mathbf{s_t}$:

$$\begin{aligned}
\mathcal{L}(\theta; \mathbf{s_t}) &= E\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} \tilde{r}(\mathbf{s_{t'}}, \mathbf{a_{t'}}) \mid \mathbf{s_t}\right] \\
&= E\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} (r(\mathbf{s_{t'}}, \mathbf{a_{t'}}) + \lambda(\log m_\theta(\mathbf{z_{t'+1}} \mid \mathbf{z_{t'}}, \mathbf{a_{t'}}) - \log \phi_\theta(\mathbf{z_{t'}} \mid \mathbf{s_{t'}}))) \mid \mathbf{s_t}\right] \\
&= E\Bigg[r(\mathbf{s_t}, \mathbf{a_t}) + \lambda(\log m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi_\theta(\mathbf{z_t} \mid \mathbf{s_t})) \\
&\qquad + \sum_{t'=t+1}^{\infty} \gamma^{t'-t} (r(\mathbf{s_{t'}}, \mathbf{a_{t'}}) + \lambda(\log m_\theta(\mathbf{z_{t'+1}} \mid \mathbf{z_{t'}}, \mathbf{a_{t'}}) - \log \phi_\theta(\mathbf{z_{t'}} \mid \mathbf{s_{t'}}))) \mid \mathbf{s_t}\Bigg] \\
&= E\left[r(\mathbf{s_t}, \mathbf{a_t}) + \lambda(\log m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi_\theta(\mathbf{z_t} \mid \mathbf{s_t})) + \gamma Q_\psi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right].
\end{aligned}$$

Note that the reward at the current time step, $r(\mathbf{s_t}, \mathbf{a_t})$, is not influenced by the parameters θ, so we can drop this term:

$$\mathcal{L}(\theta; \mathbf{s_t}) \overset{\text{const.}}{=} E\left[\lambda(\log m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi_\theta(\mathbf{z_t} \mid \mathbf{s_t})) + \gamma Q_\psi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right].$$

The remaining difference between this objective and Eq. 5 is that the Q value term is scaled by γ. However, this difference has no effect on the optimization problem because the parameter λ is automatically tuned. If we scale the second term, Q, by some value (say, γ), then tuning λ to satisfy the bitrate constraint will result in a different value for λ (one which is γ times smaller). For optimizing this objective, we sample states $\mathbf{s_t}$ from the replay buffer.

A.2 Prior on the Initial Representation

For simplicity, we have omitted the prior $m(\mathbf{z_1})$ on the representation of the first observation. This prior cannot be predicted from prior observations. Instead, we fix $m(\mathbf{z_1})$ to be a zero-mean, unit-variance Gaussian. The information-augmented reward at the first time step is therefore

$$\tilde{r}_\lambda(\mathbf{s_0}, \mathbf{a_0}, \mathbf{s_1}) \triangleq \cancel{r(\mathbf{s_0}, \mathbf{a_0})} + \lambda\left(\log m_\theta(\mathbf{z_1}) - \log \phi_\theta(\mathbf{z_1} \mid \mathbf{s_1})\right).$$

A.3 MaxEnt RL and the Optimal Encoder

Maximum entropy (MaxEnt) RL is a special case of our compression objective. If the high-level policy is the identity function ($\pi^z(\mathbf{a_t} \mid \mathbf{z_t}) = \delta(\mathbf{a_t} = \mathbf{z_t})$) and the prior is the uniform distribution over $\mathcal{Z}$ (i.e., $m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) = \text{Unif}(\mathbf{z_{t+1}})$), then we recover standard MaxEnt RL. Another way of describing this connection is that MaxEnt RL is equivalent to imposing an information bottleneck on the final (i.e., action) output of the policy. This connection suggests that some of the empirically-observed benefits of MaxEnt RL might be derived from the fact that it implicitly is performing model compression. For example, the objective from MaxEnt RL can be interpreted as optimizing for behavior under which random actions (sampled from the prior) do not significantly decrease the expected reward. Nonetheless, directly optimizing for the model compression objective yields a prior that depends on time. As shown in our experiments, such a prior yields a policy that not only is more compressed, but is also more robust to sensor failure (i.e., the open-loop setting).

In MaxEnt RL, the optimal policy can be expressed in terms of the *soft* value function: $\pi(\mathbf{a_t} \mid \mathbf{s_t}) \propto e^{\tilde{Q}(\mathbf{s_t}, \mathbf{a_t})}$, where $\tilde{Q}(\mathbf{s_t}, \mathbf{a_t})$ is the expected, entropy-regularized return. We can express the optimal encoder learned by RPC in similar terms. First, we reparametrize the Q function in terms of $\mathbf{z_t}$ instead of $\mathbf{a_t}$: $Q^\pi(\mathbf{s_t}, \mathbf{z_t}) = \mathbb{E}_{\pi^z(\mathbf{a_t}|\mathbf{z_t})}[Q(\mathbf{s_t}, \mathbf{a_t})]$. The optimization problem for the encoder is

$$\begin{aligned}
&\max_{\phi(\mathbf{z_t}|\mathbf{s_t})} \mathbb{E}_{\phi(\mathbf{z_t}|\mathbf{s_t})}\left[\sum_t \gamma^t \tilde{r}(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})\right] \\
&\quad = \mathbb{E}_{\phi(\mathbf{z_t}|\mathbf{s_t})}\left[r(\mathbf{s_t}, \mathbf{a_t}) + \log m(\mathbf{z_t} \mid z_{t-1}, a_{t-1}) - \log \phi(\mathbf{z_t} \mid \mathbf{s_t}) + \gamma Q(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right].
\end{aligned}$$

Algorithm 1 Robust Predictable Control. The updates below are written using a learning rate of η. In practice we perform gradient steps using the Adam [24] optimizer.

```
1:  Initialize prior $m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$, encoder $\phi_\theta(\mathbf{z_{t+1}} \mid \mathbf{s_t})$, Q function $Q_\psi(\mathbf{s_t}, \mathbf{a_t})$, and high-level
    policy $\pi_\theta^z(\mathbf{a_t} \mid \mathbf{z_t})$.
2:  Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
3:  Initialize dual variable $\log \lambda \leftarrow \log(1e-6)$
4:  while not converged do
5:      Sample a batch of transitions: $\{(\mathbf{s_t}, \mathbf{a_t}, \mathbf{r_t}, \mathbf{s_{t+1}}) \sim \mathcal{D}\}$
6:      Compute information cost: $\mathbf{c_t} \leftarrow \mathbb{E}[\log \phi_\theta(\mathbf{z_t} \mid \mathbf{s_t}) - \log m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})]$    ▷ Eq. 4
7:      Compute information-regularized reward $\tilde{\mathbf{r}} \leftarrow \mathbf{r_t} - \lambda \mathbf{c_t}$
8:      Update Q function: $\psi \leftarrow \psi - \eta \nabla_\psi \mathcal{L}(\psi)$    ▷ Eq. 5
9:      Update prior, encoder, and high-level policy: $\theta \leftarrow \theta + \eta \nabla_{theta} \mathcal{L}(\mathbf{s_t})$    ▷ Eq. 6
10:     Update dual variable: $\log \lambda \leftarrow \log \lambda - \eta(C - \mathbb{E}[c_t])$
11: return $\pi_\theta^z(\mathbf{a_t} \mid \mathbf{z_t}), \phi_\theta(\mathbf{z_t} \mid \mathbf{s_t}), m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$
```

Using calculus of variations, we determine that the optimal encoder is given by

$$\phi(\mathbf{z_t} \mid \mathbf{s_t}) = \frac{m(\mathbf{z_t} \mid \mathbf{z_{t-1}}, \mathbf{a_{t-1}}) e^{Q^\pi(\mathbf{s_t}, \mathbf{z_t})}}{\int m(\mathbf{z_t}' \mid \mathbf{z_{t-1}}, \mathbf{a_{t-1}}) e^{Q^\pi(\mathbf{s_t}, \mathbf{z_t}')} d\mathbf{z_t}'}.$$

Thus, the optimal encoder is trained to *tilt* the predictions from the prior by the Q function.

A.4 Value of information.

An optimal agent must balance these information costs against the *value of information* gained from these observations. Precisely, the value of information is how much more reward an optimal agent could receive if it observes the representation $\mathbf{z_t}$ instead of predicting $\mathbf{z_t}$ from the previous representation and action. We expect that the optimal policy will only look at representations where the value of information is greater than the cost of information. We confirm this prediction experimentally in Fig. 4. In practice, the policy learned by RPC looks at every observation, but may only look at a few bits from that observation.

B Experimental Details

B.1 Implementation Details

We implemented RPC on top of the SAC [17] implementation in TF-Agents [15]. Unless otherwise noted, we used the default hyperparameters from that implementation and trained all agents for 3e6 steps. We provide pseudocode in Alg. 1.

State-based RPC. We parameterized the model $m_\theta(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$ by predicting the *difference* between the $\mathbf{z_t}$ and $\mathbf{z_{t+1}}$. That is, we trained a 2-layer neural network (both layers have 256 units with ReLU activations) to model $p(\mathbf{z_{t+1}} - \mathbf{z_t} \mid \mathbf{z_t}, \mathbf{a_t})$. We parameterized the encoder $\phi_\theta(\mathbf{z_t} \mid \mathbf{s_t})$ as a 2-layer neural network (both layers have 256 units with ReLU activations). Both the model and encoder output the mean and (diagonal) standard deviation of a multivariate Normal distribution. We squashed the mean to be within $[-30, 30]$ and squashed the standard deviation to be within $[0.1, 10.0]$. We used the following function for squashing:

```
def squash_to_range(t, low=-np.inf, high=np.inf):
  if low == -np.inf:
    t_low = t
  else:
    t_low = -low * tf.nn.tanh(t / (-low))
  if high == np.inf:
    t_high = t
  else:
    t_high = high * tf.nn.tanh(t / high)
  return tf.where(t < 0, t_low, t_high)
```

Unless otherwise noted, we set the dimension of $\mathbf{z_t}$ to 50, though found this parameter has little effect.

We parameterized the high-level policy $\pi_\theta^z(\mathbf{a_t} \mid \mathbf{s_t})$ as a 2-layer neural network (both layers have 256 units with ReLU activations). Following the TF-Agents implementation of SAC, this network first predicts a Normal distribution, which is then squashed by `tf.tanh` to only output actions within the action space.

Image-based RPC. Our image-based version of RPC was based on DrQ [46]. Unless otherwise specified, we used the same hyperparameters and architecture as that paper. For example, we used the same action repeat (4 actions) and frame stack (3 frames) as that paper. The only architectural difference was the policy network. We parameterized the encoder $\phi_\theta(\mathbf{z_t} \mid \mathbf{s_t})$ as follows:

```
tf.keras.Sequential([
  tf.keras.layers.Lambda(lambda t: tf.cast(t, tf.float32) / 255.0),  # Normalize image
  tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(2, 2), activation='relu'),
  tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
  tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
  tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(tfp.layers.IndependentNormal.params_size(latent_dim), activation=_activation),
  tfp.layers.IndependentNormal(latent_dim),
])
```

We used the same model and high-level policy as in the state-based experiments.

Dual parameter λ. We updated the dual parameter λ via dual gradient ascent to satisfy the bitrate constraint. We parametrized this variable as $\log \lambda$ to ensure that it remains positive. We initialized $\lambda = 10^{-6} \approx 0$, and performed updates using `tf.keras.optimizers.Adam(learning_rate=3e-4)`. We confirmed via visual inspection that the bitrate constraint was satisfied.

RNN baseline. We based the RNN baseline experiments on SAC-RNN implementation in TF-Agents [15].[4] We trained the agent on sequences of length 20. For fair comparison, we used the same number of *transitions* per batch as other agents that trained on individual transitions. The default batch size in TF-Agent's implementation of SAC is 256, so we trained the RNN on batches with 256 // 20 = 13 sequences.

Model-based baselines. The state-space model and latent-space models used the same model architecture as RPC. This model was trained using standard maximum likelihood with the same optimizer that RPC used for optimizing $m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$.

B.2 Learning Compressed Policies (Fig. 2)

We ran this experiment with 5 random seeds, with error bars depicting the [25%, 75%] quantiles across random seeds. To compute the return of each policy, we took the average of the last 50 evaluations, corresponding to the last 5e5 training steps (out of a total of 3e6 training steps). For each task, we normalized the return by the median return of the best performing baseline. Fig. 10 plots returns without normalization.

B.3 Behavior of Compressed Policies (Fig. 3a (bottom))

We used the `two-way` task from Leurent [29]. We initially observed that the agent often crashed into other cars, so we modified the reward function by adding a penalty of -50 when the agent crashes. The original environment has discrete actions, so we added a wrapper around the environment that allows RPC to command continuous actions, which would then be discretized. We trained RPC for 1e5 steps on this environment.

B.4 Behavior of Compressed Policies (Fig. 3b)

For this experiment, we used a modified version of the `two-way` task from Leurent [29]. We modified the environment to disable passing, so the car was forced to remain in the same lane. To encourage

[4] `https://github.com/tensorflow/agents/blob/master/tf_agents/agents/sac/examples/v2/train_eval_rnn.py`

aggressive driving, we modified the reward function to be $(x/2000)^{10}$, where x is the distance traveled. We trained RPC for 2e5 steps. The "Compressed Policy" refers to RPC using a bitrate constraint of 0.1.

B.5 Value of Information (Fig. 4 (top))

The value of observing $\mathbf{z_t}$ is given by

$$\text{ValueOfInfo}(\mathbf{z_{t+1}}) = \mathbb{E}\left[\sum_t \gamma^t r(\mathbf{s_t}, \mathbf{a_t}) \mid \mathbf{z_{t+1}}\right] - \mathbb{E}\left[\sum_t \gamma^t r(\mathbf{s_t}, \mathbf{a_t}) \mid \mathbf{z_t}\right],$$

while the cost of observing $\mathbf{z_t}$ is given by

$$\text{CostOfInfo}(\mathbf{z_{t+1}}) = \mathbb{E}[\log \phi(\mathbf{z_{t+1}} \mid \mathbf{s_{t+1}}) - \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})].$$

Intuitively, the optimal policy will only choose to look at representations $\mathbf{z_t}$ when $\text{ValueOfInfo}(\mathbf{z_{t+1}}) > \text{CostOfInfo}(\mathbf{z_{t+1}})$. We compare these two quantities in the experiments (Fig. 4 (top)). Note that both the cost of information and value of information depend on the encoder, $\phi(\mathbf{s_t})$.

For this experiment, we applied RPC to the `HalfCheetah-v2` environment using a bitrate constraint of 10.0.

B.6 Sparse Representations (Fig. 4 (bottom))

For this experiment, we applied RPC to the `HalfCheetah-v2` environment. Since both the encoder $\phi(\mathbf{z_t} \mid \mathbf{s_t})$ and model $m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$ predict *diagonal* normal distributions, we can compute the KL divergence between each coordinate. For the plot in Fig. 4 (bottom), we sorted these KLs before plotting.

B.7 Hierarchical RL (Fig. 8)

Pre-training. We learned the representation of actions by applying RPC to the `push-v2` task in Metaworld [48]. To make the subsequent optimization easier, we set the dimension of $\mathbf{z_t}$ to 3.

Hierarchical RL. We optimize over the list of $\mathbf{z_t}$s using CMA-ES [18]. We provide pseudocode in Fig. 9. The aim of this approach is to test whether compression results in behaviors that can transfer to new tasks. We use CMA-ES as a black-box optimizer because of its simplicity, and emphasize that more sophisticated optimization routines (e.g., actor-critic RL with actions corresponding to $\mathbf{z_t}$) would likely perform better.

For the `pusher` task (which was the same as the training task), we used one behavior $\mathbf{z_t}$ for a horizon of 100 steps. For the `pusher_wall` task, we used two behaviors $\mathbf{z_t}$, each with horizon 100 steps. For the `button` task, we used one behavior $\mathbf{z_t}$ for a horizon of 100 steps. For the `drawer_open` task, we used two behaviors $\mathbf{z_t}$, each with a horizon of 75 steps.

```
def objective_fn(z_list, horizon):
  env.reset()
  total_r = 0.0
  for z in z_list:
    for _ in range(horizon):
      a = policy(z)
      _, r, done, _ = env.step(a)
      total_r += r
      if done:
        break
      z = prior(z, a)
  loss = -1 * total_r
cma.fmin(objective_fn)
```

Figure 9: Pseudocode for hierarchical RL

Action Repeat. In some tasks, actions correspond to a desired pose. Thus, action repeat would correspond to moving to a desired pose, which seemed like a reasonable baseline in these manipulation tasks. Our experiments highlight that the behaviors learned by RPC do more than move to a particular pose. In addition, the behaviors learned by RPC do obstacle avoidance and objective manipulation.

B.8 Robustness to missing observations (Fig. 5)

For this experiment, we trained RPC using a bitrate of 0.1 of `HalfCheetah-v2` and 3.0 for `Walker2d-v2`. We evaluated all methods by dropping observations independently. Note that none of the methods were trained using observation dropout, so this experiment explicitly tests robustness to new disturbances introduced at test time.

B.9 Adversarial Robustness to Dynamics (Fig. 6 (left))

The adversary aims to apply a small perturbation to that state to make the policy perform as poorly as possible. The adversary's objective is

$$\min_{\mathbf{s}_{\text{adv}} \in \mathcal{B}(\mathbf{s})} V^\pi(\mathbf{s}_{\text{adv}}) = Q^\pi(\mathbf{s}_{\text{adv}}, \mathbf{a} \sim \pi(\mathbf{a} \mid \mathbf{s}_{\text{adv}})).$$

We instantiate the adversary using projected gradient descent [32] with step size 0.1. For fair comparison, we evaluate each policy on states collected by rolling out that policy. This experiment used the `Ant-v2` environment, applying RPC using a bitrate of 0.3. We repeated this attack on 20 states sampled from each policy's state distribution. The dark line shows the average across these 20 attacks. As expected, the agent is more vulnerable to attacks in some states than in other states.

B.10 Adversarial Robustness to Observations (Fig. 6 (right))

We optimize the adversary to corrupt the state using the following objective:

$$\min_{\mathbf{s}_{\text{adv}} \in \mathcal{B}(\mathbf{s})} Q^\pi(\mathbf{s}, \mathbf{a} \sim \pi(\mathbf{a} \mid \mathbf{s}_{\text{adv}})).$$

The adversary is optimized using PGD, using three steps of size 0.1 for each state. Note that we only change the observation, not the true state of the environment. This difference from the previous experiment is important, as it allows us to unroll the policy and adversary for an entire trajectory. We used the `Ant-v2` environment for this experiment.

B.11 Robust RL (Fig. 7)

We followed prior work in setting up the robust RL experiments [41]. To modify the mass, we scaled the `env.model.body_mass` attribute of the environment by a fixed constant. To modify friction, we scaled the `env.model.geom_friction` attribute of the environment by a fixed constant. We used perturbations much larger than prior work [41] because we found that standard RL was already robust to smaller ranges of perturbations.

C Proofs

C.1 Model Compression is a Lower Bound on Open-Loop Control (Lemma 5.2)

In this section we formally define the assumptions for Lemma 5.2 and then provide the proof.

First, we define an open loop policy with hidden state $\mathbf{z_t}$. This policy updates its hidden state as $m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})$ and produces actions by sampling $p(\mathbf{a_t} \mid \mathbf{z_t})$. Note that the open-loop sequence of actions generated by the policy is evaluated under the *true* system dynamics, $p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t})$. The open loop policy does not observe the transitions from the true system dynamics.

The main idea of the proof will be to apply an evidence lower bound on the expected reward objective. This idea alone almost completes the proof. The main technical challenge is accounting for discount factors: a naïve application of an ELBO will result in discounted rewards but an undiscounted information term.

Proof. The **first step** is to recognize that the expected discounted return objective can be written as the expected *terminal* reward of a mixture of finite-length episodes. Define $p_H(\tau)$ as a distribution over length-H episodes. We can then write the expected discounted return objective as follows:

$$\mathbb{E}_{p(\tau)}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s_t}, \mathbf{a_t})\right] = \sum_{H=1}^{\infty} \gamma^H \mathbb{E}_{p_H(\tau)}\left[r(\mathbf{s_H}, \mathbf{a_H})\right].$$

We can now obtain a lower bound on the *log* of the expected return objective:

$$
\begin{aligned}
&\log \mathbb{E}_{p(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s_t}, \mathbf{a_t})\right] \\
&\quad = \log\left(\sum_{H=1}^{\infty} \gamma^H \mathbb{E}_{p(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})}\left[r(\mathbf{s_H}, \mathbf{a_H})\right]\right) \\
&\quad = \log\left(\frac{1-\gamma}{\gamma} \sum_{H=1}^{\infty} \gamma^H \mathbb{E}_{p(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})}\left[r(\mathbf{s_H}, \mathbf{a_H})\right]\right) + \log\frac{\gamma}{1-\gamma} \\
&\quad \geq \frac{(1-\gamma)}{\gamma} \sum_{H=1}^{\infty} \gamma^H \log\left(\mathbb{E}_{\mathbf{p}(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})}\left[r(\mathbf{s_H}, \mathbf{a_H})\right]\right) + \log\frac{\gamma}{1-\gamma}. \qquad (7)
\end{aligned}
$$

The inequality is Jensen's inequality. The constant $\frac{1-\gamma}{\gamma}$ was introduced because Jensen's inequality must be applied to a proper probability distribution.

The **second step** is to define a variational distribution $q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})$ as

$$
q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}}) = \prod_t p(\mathbf{a_t} \mid \mathbf{z_t}) q(\mathbf{z_t} \mid \mathbf{s_t}) p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}).
$$

We can then obtain a lower bound on Eq. 7

$$
\begin{aligned}
&\log \mathbb{E}_{p(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s_t}, \mathbf{a_t})\right] \\
&\geq \frac{1-\gamma}{\gamma} \sum_{H=1}^{\infty} \gamma^H \mathbb{E}_{q(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})}\left[\log r(\mathbf{s_H}, \mathbf{a_H}) + \log p(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}}) - \log q(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})\right] + \log\frac{\gamma}{1-\gamma} \\
&= \frac{1-\gamma}{\gamma} \sum_{H=1}^{\infty} \gamma^H \mathbb{E}_{q(\mathbf{s_{1:H}}, \mathbf{a_{1:H}}, \mathbf{z_{1:H}})}\left[\log r(\mathbf{s_H}, \mathbf{a_H}) + \sum_{t=1}^{H} \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi(\mathbf{z_t} \mid \mathbf{s_t})\right] + \log\frac{\gamma}{1-\gamma}. \qquad (8)
\end{aligned}
$$

The final line follows from simplifying the definitions of $p(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})$ and $q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})$. Note that we have used the assumption that $r(\mathbf{s_t}, \mathbf{a_t}) > 0$ to ensure that $\log r(\mathbf{s_t}, \mathbf{a_t})$ is well defined.

Our **third step** is to do the opposite of the first step: recognize that expectations over a mixture of finite-horizon episodes are equivalent to expectations over a discounted infinite-length episode. With this, we can rewrite Eq. 8 as follows:

$$
\begin{aligned}
&\log \mathbb{E}_{p(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s_t}, \mathbf{a_t})\right] \\
&\geq \frac{1-\gamma}{\gamma} \mathbb{E}_{q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})}\left[\sum_{H=1}^{\infty} \gamma^H \left(\log r(\mathbf{s_H}, \mathbf{a_H}) + \sum_{t=1}^{H} \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi(\mathbf{z_t} \mid \mathbf{s_t})\right)\right] + \log\frac{\gamma}{1-\gamma}. \qquad (9)
\end{aligned}
$$

We can simplify the double summation using the following identity, where $\mathbf{x_t}$ is shorthand for $\log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi(\mathbf{z_t} \mid \mathbf{s_t})$:

$$
\begin{aligned}
\sum_{H=1}^{\infty} \gamma^H \sum_{t=1}^{H} \mathbf{x_t} &= \gamma(\mathbf{x_1}) + \gamma^2(\mathbf{x_1} + \mathbf{x_2}) + \gamma^3(\mathbf{x_1} + \mathbf{x_2} + \mathbf{x_3}) + \cdots \\
&= \mathbf{x_1}(\gamma + \gamma^2 + \gamma^3 + \cdots) + \mathbf{x_2}(\gamma^2 + \gamma^3 + \cdots) + \cdots \\
&= \mathbf{x_1}\frac{\gamma}{1-\gamma} + \mathbf{x_2}\frac{\gamma^2}{1-\gamma} + \mathbf{x_3}\frac{\gamma^3}{1-\gamma} + \cdots \\
&= \frac{1}{1-\gamma} \sum_{t=1}^{\infty} \gamma^t \mathbf{x_t}.
\end{aligned}
$$

Applying this identity to Eq. 9, we get

$$
\begin{aligned}
&\log \mathbb{E}_{p(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})} \left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s_t}, \mathbf{a_t}) \right] \\
&\geq \frac{1-\gamma}{\gamma} \mathbb{E}_{q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})} \left[\sum_{t=1}^{\infty} \gamma^t \left(\log r(\mathbf{s_t}, \mathbf{a_t}) + \frac{1}{1-\gamma} \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \frac{1}{1-\gamma} \log \phi(\mathbf{z_t} \mid \mathbf{s_t}) \right) \right] + \log \frac{\gamma}{1-\gamma} \\
&\geq \frac{1}{\gamma} \mathbb{E}_{q(\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})} \left[\sum_{t=1}^{\infty} \gamma^t \left((1-\gamma) \log r(\mathbf{s_t}, \mathbf{a_t}) + \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) - \log \phi(\mathbf{z_t} \mid \mathbf{s_t}) \right) \right] + \log \frac{\gamma}{1-\gamma}.
\end{aligned}
$$

Exponentiating both sides, we obtain the desired result.

□

This derivation is useful because it relates model compression to open-loop control. Precisely, it says that a compressed model is one that will perform well under open loop rollouts.

C.2 Proof of Lemma 5.3

Proof. To start, we define the distribution over trajectories $\tau = (\mathbf{s_{1:\infty}}, \mathbf{a_{1:\infty}}, \mathbf{z_{1:\infty}})$ produced by π^{open} and π^{reactive}:

$$
\begin{aligned}
p^{\text{open}}(\tau) &= p_1(\mathbf{s_1}) \prod_t p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}) \pi^z(\mathbf{a_t} \mid \mathbf{z_t}) m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) \\
p^{\text{reactive}}(\tau) &= p_1(\mathbf{s_1}) \prod_t p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}) \pi^z(\mathbf{a_t} \mid \mathbf{z_t}) \phi(\mathbf{z_t} \mid \mathbf{s_t}).
\end{aligned}
$$

Note that $R(\tau) \triangleq \sum_t \gamma^t r(\mathbf{s_t}, \mathbf{a_t})$ is a deterministic function of a random variable, τ. We can now write the difference in returns as $\mathbb{E}_{\pi^{\text{reactive}}}[R(\tau)] - \mathbb{E}_{\pi^{\text{open}}}[R(\tau)]$. The main idea of our proof will be to show that this difference is not too large.

$$
\begin{aligned}
\mathbb{E}_{\pi^{\text{reactive}}}[R(\tau)] - \mathbb{E}_{\pi^{\text{open}}}[R(\tau)] &= \int R(\tau)(p^{\text{reactive}}(\tau) - p^{\text{open}}(\tau)) d\tau \\
&\overset{(a)}{\leq} R_{\max} \int |(p^{\text{reactive}}(\tau) - p^{\text{open}}(\tau))| d\tau \\
&\overset{(b)}{\leq} R_{\max} \sqrt{\frac{1}{2} KL(p^{\text{reactive}}(\tau) \| p^{\text{open}}(\tau))}. \qquad (10)
\end{aligned}
$$

We used Hölder's inequality in *(a)* and Pinsker's inequality in *(b)*. Our second step is to relate the KL divergence to the compression objective.

$$
\begin{aligned}
&KL(p^{\text{reactive}}(\tau) \| p^{\text{open}}(\tau)) \\
&= \mathbb{E}_{p^{\text{reactive}}} \Bigg[\cancel{\log p_1(\mathbf{s_1})} + \sum_t \cancel{p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t})} + \cancel{\log \pi^z(\mathbf{a_t} \mid \mathbf{z_t})} + \log \phi(\mathbf{z_t} \mid \mathbf{s_t}) \\
&\qquad - \cancel{\log p_1(\mathbf{s_1})} - \sum_t \left(\cancel{p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t})} + \cancel{\log \pi^z(\mathbf{a_t} \mid \mathbf{z_t})} + \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) \right) \Bigg] \\
&= \mathbb{E}_{p^{\text{reactive}}} \left[\sum_t \log \phi(\mathbf{z_t} \mid \mathbf{s_t}) - \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t}) \right] \leq C.
\end{aligned}
$$

where C is the information constraint (Eq. 3). We then substitute this simplified expression for the KL into Eq. 10:

$$
\mathbb{E}_{\pi^{\text{reactive}}}[R(\tau)] - \mathbb{E}_{\pi^{\text{open}}}[R(\tau)] \leq R_{\max} \sqrt{\frac{1}{2} C}.
$$

Rearranging terms, we obtain the desired result:

$$
\mathbb{E}_{\pi^{\text{open}}}[R(\tau)] \geq \mathbb{E}_{\pi^{\text{reactive}}}[R(\tau)] - R_{\max} \sqrt{\frac{1}{2} C}.
$$

□

For this result to be non-vacuous, we need that $\sqrt{\frac{1}{2}C} < 1$, so the average per-transition c must satisfy $c = (1-\gamma)C \leq 2 \cdot (1-\gamma)$:

$$\mathbb{E}_\pi[\log \phi(\mathbf{z_t} \mid \mathbf{s_t}) - \log m(\mathbf{z_{t+1}} \mid \mathbf{z_t}, \mathbf{a_t})] \leq 2 \cdot (1-\gamma).$$

For example, when $\gamma = 0.99$, we would need that the average per-transition KL be less than 0.02.

D Additional Experiments

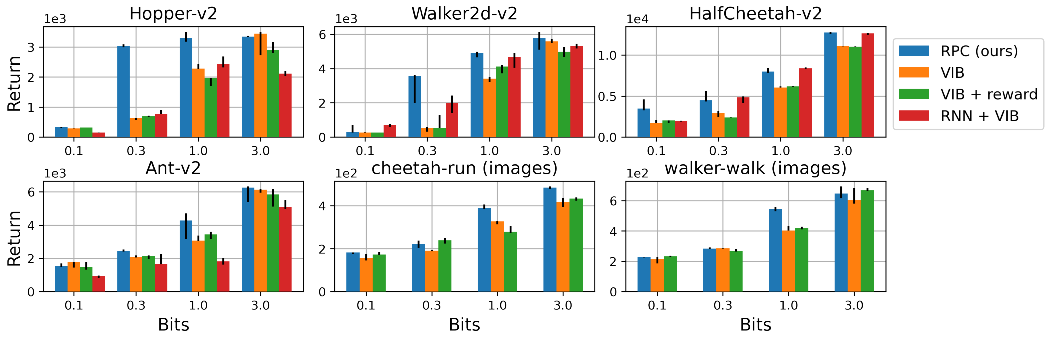

Figure 10: **Learning Compressed Policies.** This plot shows the same experiment as Fig. 2 on a wider range of bitrates, without using normalized returns.

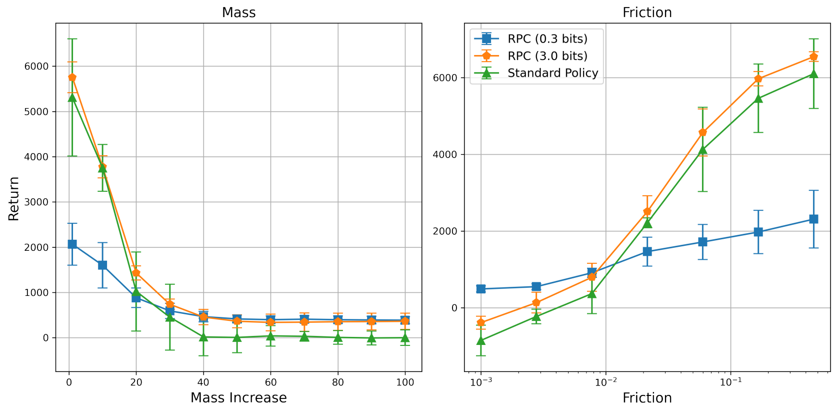

Figure 11: **Robust RL**: This plot shows the same experiment as Fig. 7 with error bars corresponding to the standard deviation across five *training* random seeds.

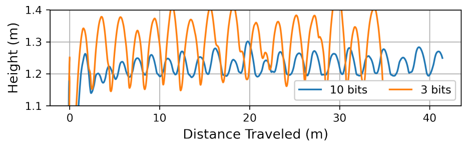

Figure 12: Behavior of compressed walking policies.

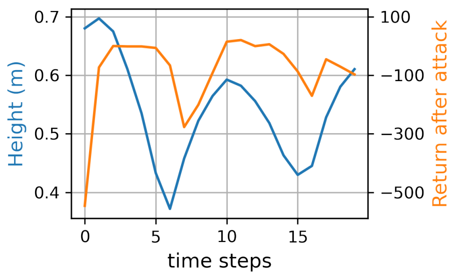

Figure 13: **Adversarial Robustness**: The agent is most vulnerable just before its feet leave the ground

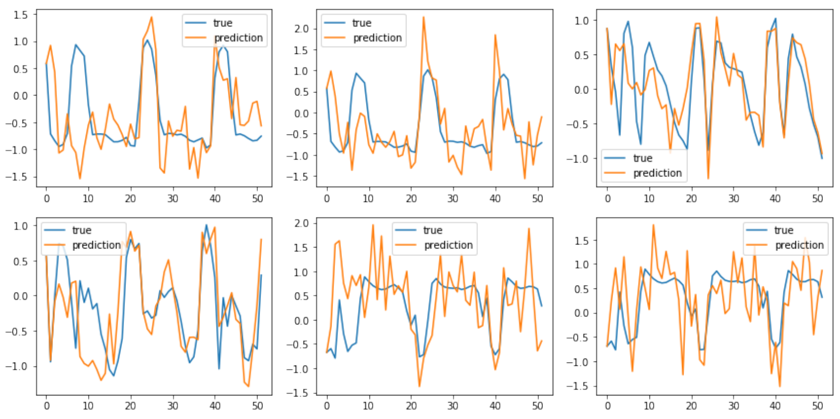

Figure 14: **Open-loop predictions of representations**

Visualizing more compressed policies. In Fig. 12, we demonstrate that compressed policies also learn different walking gaits on a locomotion task. The agent requires many more bits each time it touches the ground, so the compressed policy takes bigger steps to minimize the number of footsteps. See the project website for videos comparing the behaviors learned for varying levels of compression.

Behavior of attacked policies. We visualize the behavior of the adversarial attack on dynamics described in Sec. 6.3. Visualizing the uncompressed policy in Fig. 13, we observe that the policy is most vulnerable to attack the moment the robot launches off the ground for each step. This makes sense, as the actions right before takeoff dictate the path taken when the robot is in the air before the next step.

Open-loop predictions of representations. In Fig. 14, we plot the 6 coordinates of the representation $\mathbf{z_t}$ learned by RPC for the `HalfCheetah-v2` task. Recall that RPC learns a sparse representation, so we choose the 6 coordinates with the largest variance. We observe that open-loop (i.e., autoregressive) predictions from the latent-space model, m_θ, closely match the true representations.