Hyperparameter	Value
n (line 3)	1
M	200
Optimizer	Adam
actor/critic/entropy learning rate	$3 \cdot 10^{-4}$
model learning rate	10^{-4}
model L2 penalty	10^{-5}
initial entropy coeff	1.0
target entropy	$-dim(\mathcal{A})$
γ	0.99
polyak factor	0.995
replay size	1e5
replay batch size	256
actor/critic hidden layers	[256, 256]
actor/critic nonlinearity	ReLU
actor max logvar	2
actor min logvar	-20
model hidden layers	[128, 128]
model nonlinearity	Swish
model max logvar	2
model min logvar	-20

Table 1: Algorithm 1 hyperparameters

B Algorithm hyperparameters

Table 1 lists the hyperparameters used in algorithm 1, our model-based augmentation of SAC with value-aware models.

We use Glorot initialization [18] for all networks with defaults parameters via the PyTorch machine learning framework [30].

Due to limited availability of computational resources, we use a single model network instead of an ensemble, which is a trend in recent model-based RL methods.

As mentioned in section 4.3, algorithm 1 uses only one gradient step on each call to lines 8 and 9 to update critic and actor respectively.

We use clipped double Q-learning [17] with target networks, updated after every critic update (line 8) with polyak averaging of the parameters. In eq. (7), we sample 10 future states from the model and average the clipped Q-value to approximate the inner expectation.

We use automatic temperature tuning for the entropy coefficient as described in the original paper [20]. We take one dual gradient step on the entropy coefficient's loss function right after the actor update in line 9 of algorithm 1.

We collect 10000 initial transitions by choosing from the action space uniformly at random. Only after this initial phase do we start acting according to the current policy.

We evaluate the current policy every 2000 timesteps by turning off exploration (sampling from the mean of the learned Gaussian policy) and computing the average return over 10 episodes.

In terminal transitions due to timeouts, we ignore the termination flag and instead bootstrap from the final state [29].

We use the Pytorch Lightning [12] framework for training the model. On every call to line 7 of algorithm 1, we subsample 20% of the current replay buffer as a holdout set, with a maximum of 5000 transitions. After the initial 10000 random uniform transitions, we train the model for a maximum of 1000 epochs. Otherwise, over the normal course of the algorithm, we train the model for a maximum of 20 epochs or 200 gradient steps. We calculate the validation loss on the holdout set at the end of every epoch. We early stop training if the validation loss hasn't improved over 5 consecutive epochs, where improvement is characterized as any decrease in loss.

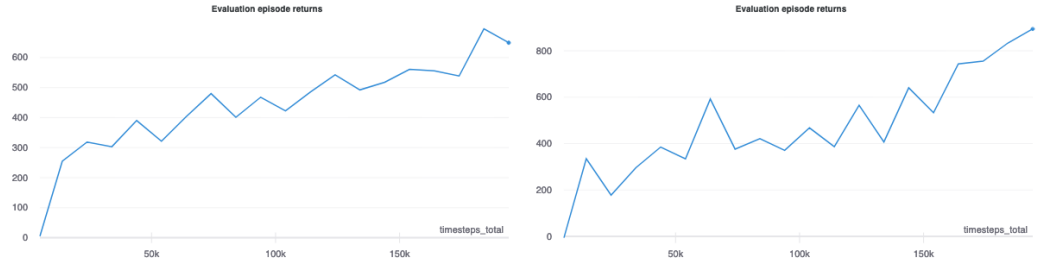

Figure 4: SAC learning curves on Hopper-v3 (left) and Walker2d-v3 (right) of the runs used in the second experiment.

C Supervised learning experiments details

Experiments in the artifical domain. We use the dynamics architecture described in section 5, a state space $\mathcal{S} \subseteq \mathbb{R}^{11}$, an action space $\mathcal{A} \subseteq [-1, 1]^3$, and a horizon of 200 timesteps. The state and action spaces were chosen to mimic the spaces of the Hopper-v3 task from OpenAI Gym.

We collect 50000 transitions for training (with 20% of them held out for validation) and 1000 transitions for testing. We use a batch size of 128 for training over a maximum of 1000 epochs. We early stop training if the validation loss (calculated at the end of every epoch) has not decreased for 3 consecutive epochs.

The value function architecture consists of a single-layer MLP with 64 hidden units and ReLU nonlinearities. We use orthogonal initialization [34] with default parameters via PyTorch.

Experiments on SAC data. Figure 4 shows the episode returns attained by the SAC agent from which we gathered data for the second experiment.

We take actor, critic, and replay data from different checkpoints of each SAC run to train the model with. The model architecture is as described in section 5, with each hidden layer having 32 units. We split the replay data into 70% for training, 20% for validation, and 10% for testing. For each model training run, we use the Adam with learning rate 10^{-3} to optimize the model for a maximum of 1000 epochs, early stopping training if the validation loss, calculated once every epoch, does not decay for 10 consecutive epochs. We report in fig. 2 the test loss at the end of each training run.