A Implementation Details

A.1 Practical implementation

Normalization As a primitive representation learning method, we normalized the states and rewards before training to reduce model prediction errors. Specifically, we estimated the mean and variance of the states based on the dataset and normalized the states accordingly. We also estimated the average reward for a single step and reduced the reward to the ratio compared to the average reward, which ensures the advantage term not to be too large.

Model training We used single three-layer neural network for the transition model $T_\phi(s,a) = N(\mu_\phi(s,a), \Sigma_\phi(s,a))$ to predict the reward r and next state s' and single two-layer neural network for the guidance model $g_\omega(s) = N(\mu_\omega(s), \Sigma_\omega(s))$ to suggest the reference state s''. For both V-value function and Q-value function, twins two-layer networks are adopted to reduce estimation errors. Note that if advantage weighting factors are not required, the V-value and Q-value networks can also be removed accordingly.

Policy Optimization After model training, SDV performs policy optimization similar to the MBPO [**?**] framework. Specifically, SDV first rolls out a batch of transitions with a step size of k from the transition model and shapes the predicted immediate rewards according to Eq. **??**. Then, the rewards-shaped trajectories are placed in the replay buffer D_{model}. Finally, SDV samples from both the offline dataset and the synthetic data in a ratio of $f : 1-f$ and optimizes the policy though the SAC algorithm [**?**] with the samples.

A.2 Model training

In our experiments, both the guidance model and the transition model used single neural networks to output Gaussian distributions on the next state (and reward for transition model), given the current state (and action for transition model):

$$g_\omega(s' \mid s) = N(\mu_\omega(s), \Sigma_\omega(s)) \tag{1}$$

$$T_\phi(s', r \mid s, a) = N(\mu_\phi(s,a), \Sigma_\phi(s,a)) \tag{2}$$

And because the model advantage weighting factor was added, dual V-value and Q-value networks were also trained (if advantage weighting is not used, the value function networks can be omitted). The network architecture are shown in Table 1

Table 1: Model architecture.

Network	Hidden Layers $\times$ Units	Activation Function
Guidance model	2×256	ReLU
Transition model	3×256	Swish
Dual V-value	2×256	ReLU
Dual Q-value	2×256	ReLU

Before policy optimization, we conducted 500k steps gradient updates for model training. We sampled a batch of 256 transitions from the offline dataset in each step and used the Adam optimizer [**?**] for parameter optimization. The hyperparameters for model training are shown in Table 2.

Table 2: Hyperparameters for model training.

Hyperparameter	Value
$\lambda_T,\ \lambda_g,\ \lambda_V,\ \lambda_Q$	1e-4
α	10
γ	0.99
β	5e-3
τ	0.7
Batch size	256
Optimiser	Adam

A.3 Policy optimization

After model training, we adopted an MBPO-like framework for policy training. First we sampled a batch of 256 transitions from the augmented dataset $D \cup D_{model}$, and then used Soft Actor-Critic (SAC) [**?**] to train the policy. During the policy optimization, we conducted 1M step gradient updates on each task. For SAC [**?**], we used automatic entropy adjustment, where the entropy target was set to a standard heuristic of -dim(A). The hyperparameters we mainly adjusted were the ratio of the real data f, the rollout step length k and the reward penalty coefficient λ. Other basic hyperparameters shared by all task are shown in Table 3 for reference.

Table 3: Basic hyperparameters for policy training.

Hyperparameter	Value
Actor learning rate	3e-4
Critic learning rate	3e-4
SAC discount factor	0.99
Soft update parameter	5e-3
Target entropy	-dim(A)
Batch size	256
Replay buffer size	1e6
Rollout batch size (b)	5e4

The hyperparameters we mainly adjusted were the rollout step length $k \in \{1,\ 2,\ 5\}$, the real data ratio $f \in \{0.1,\ 0.5,\ 0.8,\ 0.9\}$ and the reward penalty coefficient $\lambda \in \{0.5,\ 1,\ 1.5,\ 5\}$ and the specific setting for each task is shown in Table 4. We found that setting f to 0.8 or 0.9 worked better on most tasks and as for k, 1 or 2 was more appropriate. This might be because reward shaping had a significant impact on the model for OOD states, so the length of steps per rollout could not be too long, nor should it add too much augmented data. Otherwise, the agent might fall into suboptimal. For the value of λ, we mainly referred to the proportional relationship between the average single-step reward and the average double-validation distance in the offline dataset and found 0.5 or 1 was more appropriate for most tasks. In summary, the values of these hyperparameters were conservative, which might be related to the larger fixed value of the model advantage weighting factor α.

A.4 Evaluation procedure

We evaluated the current policy every 1k steps during policy optimization and report the average normalized score over 3 seeds for each task. When evaluating the SAC policy, we deterministically

Table 4: Adjusted hyperparameters for policy training.

Task	f	λ	k
halfcheetah-r	0.1	0.5	2
hopper-r	0.8	0.5	1
walker2d-r	0.5	1	1
halfcheetah-mr	0.9	0.5	2
hopper-mr	0.9	1.5	5
walker2d-mr	0.9	1	2
halfcheetah-m	0.8	0.5	1
hopper-m	0.1	5	1
walker2d-m	0.8	1	1

took the mean action. The normalization procedure for the scores is that proposed by [**?**], where 100 represents the score of an expert policy trainned by SAC in real environment and 0 represents a random policy.

A.5 Experimental environment

We used the following hardware and software for all experiments:

- CPU: Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz
- GPU: NVIDIA GeForce RTX 2080 Ti
- OS: Ubuntu 22.04.1 LTS
- Python: 3.7.16
- PyTorch: 1.13.1
- D4RL: 1.1
- Gym: 0.23.1
- MuJoCo: 2.3.1

B Toy Environments Demo

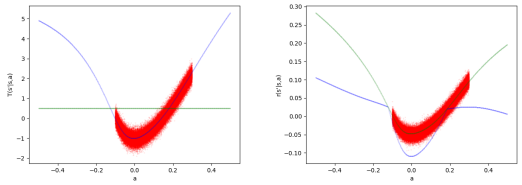
In order to intuitively demonstrate the effectiveness of double validation and model advantage weighting, the core techniques of SDV, we have constructed two simple toy environment to show the impact on model training and policy optimization respectively. Both states and actions are one-dimensional, and the figures show single transition from the same initial state.

B.1 Demo for model training

In our first toy environment, which is used to demonstrate model training, the rewards and normalized state values are consistent (i.e. $R(s,a) = \frac{s'-\mu_{s'}}{\sigma_{s'}}$). The offline dataset extracts transitions (s,a,r,s') in the current state s using a Gaussian distribution of $N(\mu(s,a),\sigma(s,a))$, where the actions $a \in [-0.1, 0.3]$. For better demonstration, we trained the guidance model with advantage weighting, but trained the transition model with MLE method. The results are shown in Figure 1, where the red points represent the dataset in all subfigures.

Figure 1a shows the mean states predicted by the guidance model (green) and the transition model (blue), with the horizontal axis representing the action space and the vertical axis representing the next state s'. Note that since the values of rewards and normalized states are consistent, a larger state value indicates a better reward. It can be seen that the guidance model provides a proper guidance, as the corresponding state is in the dataset and has a relatively high reward.

And Figure 1b shows the comparison between estimated reward (green) and the shaped reward (blue). It indicates that after reward shaping based on double validation, the states in the dataset with high reward are concentrated near the guidance state while the reward curve outside the dataset is significantly suppressed, making it more conservative compared to the pure MLE model.



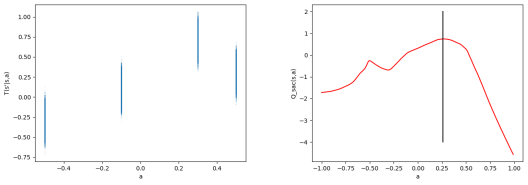
(a) Next state demo (b) Reward shape demo

Figure 1: The results for model training. In 1a, the green curve is predictions of the guidance model and the blue curve is transition model. In 1b, the green curve is the original predictions of reward and the blue curve is the penalized reward.

B.2 Demo for policy optimization

As is shown in Figure 2a, we have constructed a second toy environment to show the impact of the two techniques on policy optimization. In this environment, the initial state is the same as that in B.1 while the rewards are equal to the value of unnormalized states and the actions in the dataset are only selected from $\{-0.5, -0.1, 0.3, 0.5\}$. The Gaussian distribution used to generate data are as follows:

$$T(s,a) = \begin{cases} N(-0.3,\ 0.2), & a = -0.5 \\ N(0.1,\ 0.2), & a = -0.1 \\ N(0.7,\ 0.2), & a = 0.3 \\ N(0.3,\ 0.2), & a = 0.5 \end{cases} \tag{3}$$



(a) Dataset for training. (b) Q-value of SDV policy.

Figure 2: Toy environment results for policy training. In 2a, the blue points are the offline data generated by Eq. 3. In 2b, the red curve is the Q-value of the SDV policy for initial state and the actions in $[-1, 1]$. The black vertical bar is the action with highest Q-value.

We conducted 10k step model training on this dataset, followed by 10k step policy optimization. The hyperparameters for model training and the basic hyperparameters for policy training were consistent with the evaluation experiments, and the adjusted hyperparameters setting were: $f = 0.5$, $\lambda = 1$ and $k = 5$. After policy training, we plotted the curve of Q-value function estimated by the SAC algorithm according to the actions in the initial state. The result is shown in Figure 2b, where the black vertical bar represents the action with the highest Q-value. It shows that even if only a few single-step transitions on a few actions are provided, the action with the highest Q-value estimated by SDV is still near the optimal action in the dataset. For actions far from the optimal action or the actions outside the dataset, SDV pessimistically estimates their Q-values, ensuring the policy training to be sufficiently conservative.