Supplementary Material

A Data Modeling

In this section, we provide further details for our data modeling. Our diffusion model generates full environment transitions i.e., a concatenation of states, actions, rewards, next states, and terminals where they are present. For the purposes of modeling, we normalize each continuous dimension (non-terminal) to have 0 mean and 1 std. We visualize the marginal distributions over the state, action, and reward dimensions on the standard halfcheetah medium-replay dataset in Figure 8 and observe that the synthetic samples accurately match the high-level statistics of the original dataset.

We note the difficulties of appropriately modeling the terminal variable which is a binary variable compared to the rest of the dimensions which are continuous for the environments we investigate. This is particularly challenging for "expert" datasets where early termination is rare. For example, walker2d-expert only has $\approx 0.0001\%$ terminals. In practice, we find it sufficient to leave the terminals un-normalized and round them to 0 or 1 by thresholding the continuous diffusion samples in the middle at 0.5. A cleaner treatment of this variable could be achieved by leveraging work on diffusion with categorical variables [31].

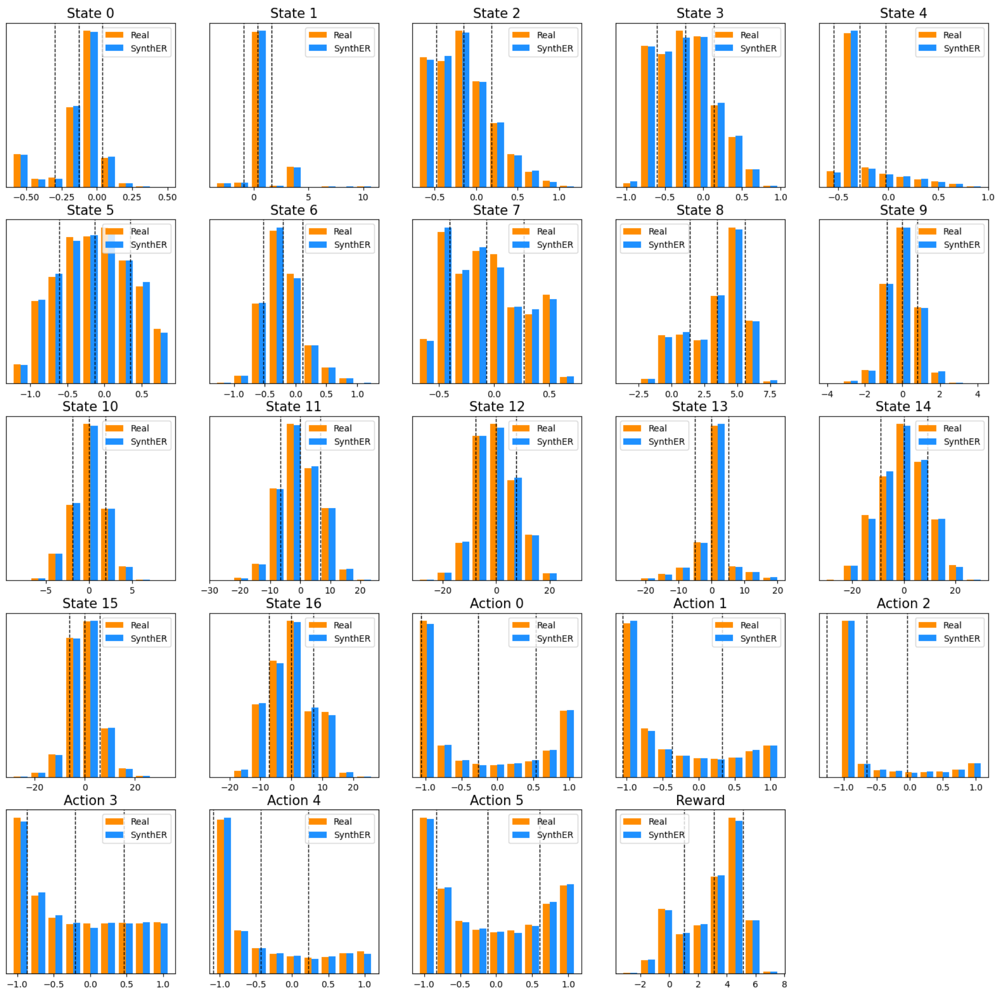

Figure 8: Histograms of the empirical marginal distribution of samples from SYNTHER in **blue** on the halfcheetah medium-replay dataset against the original data in **orange**. Dashed lines indicate the mean $\pm$ one standard deviation in the original dataset. SYNTHER faithfully reproduces the high-level statistics of the dataset.

A.1 Data Compression

An immediate advantage of sampling data from a generative model is compression. In Table 5, we compare the memory requirements of SYNTHER and the original data by the number of 32-bit floating point numbers used by each for some sample D4RL [21] datasets. For the original data, this simply scales linearly with the size of the dataset. On other hand, SYNTHER amortizes this in the number of parameters in the denoising network, resulting in a high level of dataset compression, at the cost of sampling speed. This property was also noted in the continual learning literature with generative models summarizing previous tasks [64]. As we discuss in Appendix B.3, sampling is fast with 100K transitions taking around 90 seconds to generate.

Table 5: SYNTHER provides high levels of dataset compression *without sacrificing downstream performance* in offline reinforcement learning. Statistics shown are for the standard D4RL MuJoCo walker2d datasets which has a transition dimension of 42, and the residual denoiser used for evaluation on these environments in Section 4.1. Figures are given to 1 decimal place.

Dataset	**# FP32s in Original Dataset**	**# Diffusion Parameters**	**Compression**
mixed	12.6M		1.9×
medium	42M	6.5M	6.5×
medium-expert	84M		12.9×

B Hyperparameters

B.1 TVAE and CTGAN

In Section 3.1, we compared SYNTHER to the VAE and GAN baselines, TVAE and CTGAN. As these algorithms have not been used for reinforcement learning data before, we performed a hyperparameter search [42] across the following spaces:

Table 6: Hyperparameter search space for TVAE. We highlight the default choice in **bold**.

Parameter	**Search Space**
no. layers	{ 1, **2**, 3, 4 }
width	{ 64, **128**, 256, 512 }
batch size	{ 250, **500**, 1000 }
embedding dim	{ 32, 64, **128**, 256 }
loss factor	{ 0.02, 0.2, **2**, 20}

Table 7: Hyperparameter search space for CTGAN. We highlight the default choice in **bold**.

Parameter	**Search Space**
no. layers	{ 1, **2**, 3, 4 }
width	{ 64, 128, **256**, 512 }
batch size	{ 250, **500**, 1000 }
embedding dim	{ 32, 64, **128**, 256 }
discriminator steps	{ **1**, 2}

These ranges are similar to those listed in Tables 10 and 11 of Kotelnikov et al. [42]. We used 30 trials along with the default.

B.2 Denoising Network

The formulation of diffusion we use in our paper is the Elucidated Diffusion Model (EDM, Karras et al. [38]). We parametrize the denoising network D_θ as an MLP with skip connections from the previous layer as in Tolstikhin et al. [69]. Thus each layer has the form given in Equation (3).

$$x_{L+1} = \text{linear}(\text{activation}(x_L)) + x_L \tag{3}$$

The hyperparameters are listed in Table 8. The noise level of the diffusion process is encoded by a Random Fourier Feature [57] embedding. The base size of the network uses a width of 1024 and depth of 6 and thus has $\approx$ 6M parameters. We adjust the batch size for training based on dataset size. For online training and offline datasets with fewer than 1 million samples (medium-replay datasets) we use a batch size of 256, and 1024 otherwise.

For the following offline datasets, we observe more performant samples by increasing the width up to 2048: halfcheetah medium-expert, hopper medium, and hopper medium-expert. This raises the network parameters to $\approx$ 25M, which remains fewer parameters than the original data as in Table 5. We provide ablations on the depth and type of network used in Table 10.

Table 8: Default Residual MLP Denoiser Hyperparameters.

Parameter	Value(s)
no. layers	6
width	1024
batch size	{ 256 for online and medium-replay, 1024 otherwise }
RFF dimension	16
activation	relu
optimizer	Adam
learning rate	3×10^{-4}
learning rate schedule	cosine annealing
model training steps	100K

B.3 Elucidated Diffusion Model

For the diffusion sampling process, we use the stochastic SDE sampler of Karras et al. [38] with the default hyperparameters used for the ImageNet, given in Table 9. We use a higher number of diffusion timesteps at 128 for improved sample fidelity. We use the implementation at `https://github.com/lucidrains/denoising-diffusion-pytorch` which is released under an Apache license.

Table 9: Default ImageNet-64 EDM Hyperparameters.

Parameter	Value
no. diffusion steps	128
σ_{min}	0.002
σ_{max}	80
S_{churn}	80
S_{tmin}	0.05
S_{tmax}	50
S_{noise}	1.003

The diffusion model is fast to train, taking approximately 17 minutes for 100K training steps on a standard V100 GPU. It takes approximately 90 seconds to generate 100K samples with 128 diffusion timesteps.

C SynthER Ablations

We consider ablations on the number of generated samples and type of denoiser used for our offline evaluation in Section 4.1.

C.1 Size of Upsampled Dataset

In our main offline evaluation in Section 4.1, we upsample each dataset (which has an original size of between 100K to 2M) to 5M. We investigate this choice for the walker medium-replay dataset in Figure 9 and choose 10 levels log-uniformly from the range [50K, 5M]. Similarly to He et al. [26], we find that performance gains with synthetic data eventually saturate and that 5M is a reasonable heuristic for all our offline datasets.

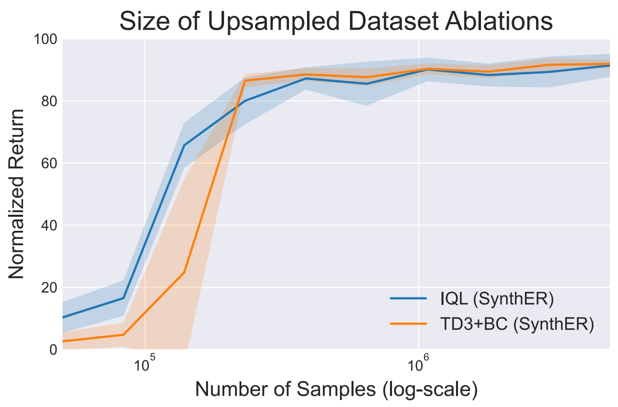

Figure 9: Ablations on the number of samples generated by SYNTHER for the offline walker medium-replay dataset. We choose 10 levels log-uniformly from the range [50K, 5M]. We find that performance eventually saturates at around 5M samples.

C.2 Network Ablations

We ablate the hyperparameters of the denoising network, comparing 3 settings of depth from $\{2, 4, 6\}$ and analyze the importance of skip connections. The remaining hyperparameters follow Appendix B.2. We choose the hopper medium-expert dataset as it is a large dataset of 2M. As we can see in Table 10, we see a positive benefit from the increased depth and skip connections which leads to our final choice in Table 8.

Table 10: Ablations on the denoiser network used for SYNTHER on the hopper medium-expert dataset. We observe that greater depth and residual connections are beneficial for downstream offline RL performance. We show the mean and standard deviation of the final performance averaged over 4 seeds.

Network	Depth	Eval. Return
MLP	2	86.8±18.7
	4	89.9±17.9
	6	**100.4± 6.9**
Residual MLP	2	78.5±11.3
	4	**99.3±14.7**
	6	**101.1±10.5**

D RL Implementation

For the algorithms in the offline RL evaluation in Section 4.1, we use the 'Clean Offline Reinforcement Learning' (CORL, Tarasov et al. [67]) codebase. We take the final performance they report for the baseline offline evaluation. Their code can be found at `https://github.com/tinkoff-ai/CORL` and is released under an Apache license.

For the online evaluation, we consider Soft Actor-Critic (SAC, Haarnoja et al. [24]) and use the implementation from the REDQ [12] codebase. This may be found at `https://github.com/watchernyu/REDQ` and is released under an MIT license. We use the 'dmcgym' wrapper for the DeepMind Control Suite [70]. This may be found at `https://github.com/ikostrikov/dmcgym` and is released under an MIT license.

D.1 Data Augmentation Hyperparameters

For the data augmentation schemes we visualize in Figure 1a, we define:

1. Additive Noise [45]: adding $\epsilon \sim \mathcal{N}(0, 0.1)$ to s_t and s_{t+1}.
2. Multiplicative Noise [45]: multiplying s_t and s_{t+1} by single number $\epsilon \sim \text{Unif}([0.8, 1.2])$.

3. Dynamics Noise [8]: multiplying the next state delta $s_{t+1} - s_t$ by $\epsilon \sim \text{Unif}([0.5, 1.5])$ so that $s_{t+1} = s_t + \epsilon \cdot (s_{t+1} - s_t)$.

D.2 Online Running Times

Our online implementation in Section 4.2 uses the default training hyperparameters in Appendix B.2 to train the diffusion model every 10K online steps, and generates 1M transitions each time. On the 200K DMC experiments, 'SAC (SynthER)' takes ≈ 21.1 hours compared to ≈ 22.7 hours with REDQ on a V100 GPU. We can further break down the running times of 'SAC (SynthER)' as follows:

- Diffusion training: 4.3 hours
- Diffusion sampling: 5 hours
- RL training: 11.8 hours

Therefore, the majority of training time is from reinforcement learning with an update-to-data ratio (UTD) of 20. We expect the diffusion training may be heavily sped-up with early stopping, and leave this to future work. The default SAC algorithm with UTD=1 takes ≈ 2 hours.

E Further Offline Results

In this section, we include additional supplementary offline experiments to those presented in Section 4.1.

E.1 AntMaze Data Generation

We further verify that SynthER can generate synthetic data for more complex environments such as AntMaze [21]. This environment replaces the 2D ball from Maze2D with the more complex 8-DoF "Ant" quadruped robot, and features: non-Markovian policies, sparse rewards, and multitask data. In Table 11, we see that SynthER improves the TD3+BC algorithm where it trains (on the 'umaze' dataset) and achieves parity otherwise.

Table 11: We show synthetic data from SynthER achieves at least parity for more complex offline environments like AntMaze-v2, evaluated with the TD3+BC algorithm. We show the mean and standard deviation of the final performance averaged over 6 seeds.

Environment		**TD3+BC [22]**	
		Original	**SynthER**
AntMaze	umaze	70.8±39.2	**88.9±4.4**
	medium-play	0.3±0.4	0.5±0.7
	large-play	0.0±0.0	0.0±0.0

E.2 Offline Data Mixtures

In Section 4.1, we considered exclusively training on synthetic data. We present results in Table 12 with a 50-50 mix of real and synthetic data to confirm that the two are compatible with each other, similar to Section 4.2. We do so by including as many synthetic samples as there are real data. As we stated before, we do not expect an increase in performance here due to the fact that most D4RL datasets are at least 1M in size and are already sufficiently large.

Table 12: We verify that the synthetic data from SynthER can be mixed with the real data for offline evaluation. The 50-50 mix achieves parity with the original data, same as the synthetic data. We show the mean and standard deviation of the final performance averaged over 8 seeds.

Environment	**TD3+BC [22]**			**IQL [41]**		
	Original	**SynthER**	**50-50**	**Original**	**SynthER**	**50-50**
locomotion average	59.0±4.9	60.0±5.1	59.2±3.7	62.1±3.5	63.7±3.5	62.7±5.1

F Latent Data Generation with V-D4RL

We provide full details for the experiments in Section 4.3 that scale SYNTHER to pixel-based environments by generating data in latent space for the DrQ+BC [50] and BC algorithms. Concretely, for the DrQ+BC algorithm, we consider parametric networks for the shared CNN encoder, policy, and Q-functions, f_ξ, π_ϕ, and Q_θ respectively. We also use a random shifts image augmentation, aug. Therefore, the Q-value for a state s and action a is given by $Q_\theta(f_\xi(\text{aug}(s)), a)$. The policy is similarly conditioned on an encoding of an augmented image observation.

The policy and Q-functions both consist of an initial 'trunk' which further reduces the dimensionality of the CNN encoding to $d_{\text{feature}} = 50$, followed by fully connected layers. We represent this as $\pi_\phi = \pi_\phi^{\text{fc}} \circ \pi_\phi^{\text{trunk}}$ and $Q_\theta = Q_\theta^{\text{fc}} \circ Q_\theta^{\text{trunk}}$. This allows us to reduce a pixel-based transition to a low-dimensional latent version. Consider a pixel-based transition (s, a, r, s') where $s, s' \in \mathbb{R}^{84\times84\times3}$. Let $h = f_\xi(\text{aug}(s))$ and $h' = f_\xi(\text{aug}(s'))$. The latent transition we generate is:

$$(\pi_\phi^{\text{trunk}}(h), Q_\theta^{\text{trunk}}(h), a, r, \pi_\phi^{\text{trunk}}(h'), Q_\theta^{\text{trunk}}(h'))$$

This has dimension $4 \cdot d_{\text{feature}} + |a| + 1$ and includes specific supervised features for both the actor and the critic; we analyze this choice in Appendix F.1. For example, on the 'cheetah-run' environment considered in V-D4RL, since $|a| = 6$, the overall dimension is 207 which is suitable for our residual MLP denoising networks using the same hyperparameters in Table 8. This allows us to retain the fast training and sampling speed from the proprioceptive setting but now in pixel space.

To obtain a frozen encoder f_ξ and trunks π_ϕ^{trunk}, Q_θ^{trunk}, we simply train in two stages. The first stage trains the original algorithm on the original data. The second stage then retrains only the fully-connected portions of the actor and critic, π_ϕ^{fc} and Q_θ^{fc}, with synthetic data. Thus, our approach could also be viewed as fine-tuning the heads of the networks. The procedure for the BC algorithm works the same but without the critic.

We use the official V-D4RL [50] codebase for the data and algorithms in this evaluation. Their code can be found at `https://github.com/conglu1997/v-d4rl` and is released under an MIT license.

F.1 Ablations On Representation

We analyze the choice of low-dimensional latent representation we use in the previous section, in particular, using specific supervised features for both the actor and critic. We compare this against using actor-only or critic-only features for both the actor and critic, which corresponds to a choice of $\pi_\phi^{\text{trunk}} = Q_\theta^{\text{trunk}}$, in Table 13. We note that both perform worse with an especially large drop-off for the critic-only features. This may suggest that non-specific options for compressing the image into low-dimensional latents, for example, using auto-encoders [40], could be even less suitable for this task.

Table 13: Ablations on the latent representation used for SYNTHER on the V-D4RL cheetah expert dataset. We observe that separate specific supervised features are essential for downstream performance with a particularly large decrease if we only used critic features for the actor and critic. We show the mean and standard deviation of the final performance averaged over 4 seeds.

Latent Representation	**Eval. Return**
Actor and Critic (Ours)	**52.3**$\pm$**7.0**
Actor Only	43.5$\pm$7.3
Critic Only	16.0$\pm$2.8