Appendix

A: Details for COT-GAN

The family of cost functions $\mathcal{C}^{\mathcal{K}}(\mu, c)$ is given by

$$\mathcal{C}^{\mathcal{K}}(\mu, c) := \Bigg\{ c(x,y) + \sum_{j=1}^{J}\sum_{t=1}^{T-1} h_t^j(y)\Delta_{t+1}M^j(x) : J \in \mathbb{N}, (h^j, M^j) \in \mathcal{H}(\mu) \Bigg\},$$

where $\Delta_{t+1}M(x) := M_{t+1}(x_{1:t+1}) - M_t(x_{1:t})$ and $\mathcal{H}(\mu)$ is a set of functions depicting causality:

$$\mathcal{H}(\mu) := \{(h, M) : h = (h_t)_{t=1}^{T-1},\ h_t \in \mathcal{C}_b(\mathbb{R}^{n\times t}), M = (M_t)_{t=1}^{T} \in \mathcal{M}(\mu), M_t \in \mathcal{C}_b(\mathbb{R}^{n\times t})\},$$

with $\mathcal{M}(\mu)$ being the set of martingales on $\mathbb{R}^{n\times T}$ w.r.t. the canonical filtration and the measure μ, and $\mathcal{C}_b(\mathbb{R}^{n\times t})$ the space of continuous, bounded functions on $\mathbb{R}^{n\times t}$.

Moreover, in the implementation of COT-GAN, the dimensionality of the sets of $\mathbf{h} := (h^j)_{j=1}^J$ and $\mathbf{M} := (M^j)_{j=1}^J$ is bounded by a fixed $J \in \mathbb{N}$. The discriminator in COT-GAN is formulated by parameterizing $\mathbf{h}_{\varphi_1}$ and $\mathbf{M}_{\varphi_2}$ in the cost function $c^{\mathcal{K}}$ as two separate neural networks that respect causality,

$$c_{\varphi}^{\mathcal{K}}(x,y) = c(x,y) + \sum_{j=1}^{J}\sum_{t=1}^{T-1} h_{\varphi_1,t}^j(y)\Delta_{t+1}M_{\varphi_2}^j(x), \quad (1)$$

where $\varphi := (\varphi_1, \varphi_2)$ and J corresponds to the output dimensionality of the two networks. Thus, we update the parameters based upon the loss given by (**??**) between the empirical distributions of two mini-batches,

Given a mini-batch of size m from training data $\{x_{1:T}^d\}_{i=1}^m$ we define the empirical measure for the mini-batch as

$$\hat{\mu} := \frac{1}{m}\sum_{d=1}^{m} \delta_{x_{1:T}^d}.$$

As the last piece of the puzzle, **?** enforced $\mathbf{M}$ to be close to a martingale by a regularization term to penalize deviations from being a martingale on the level of mini-batches.

$$p_{\mathbf{M}}(\widehat{\mu}) := \frac{1}{mT}\sum_{j=1}^{J}\sum_{t=1}^{T-1}\left|\sum_{d=1}^{m}\frac{M_{t+1}^j(x_{1:t+1}^d) - M_t^j(x_{1:t}^d)}{\sqrt{\mathrm{Var}[M^j]} + \eta}\right|,$$

where $\mathrm{Var}[M]$ is the empirical variance of M over time and batch, and $\eta > 0$ is a small constant.

B: Training details

We used a smaller size of model with the same network architectures as COT-GAN to train all three datasets. The architectures for generator and discriminator are given in Tables 1 and 2.

Hyperparameter settings are as follows: the Sinkhorn regularizer $\epsilon = 0.8$, Sinkhorn iteration $L = 100$, the length-scale $l = 20$ and martingale penalty $\lambda = 1.5$. We used Adam optimizer with learning rate 0.0001, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. All models are trained for $60,000$ iterations.

Table 1: Generator architecture.

Generator	Configuration
Input	$z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
0	LSTM(state size = 64), BN
1	LSTM(state size = 128), BN
2	Dense(8*8*256), BN, LeakyReLU
3	reshape to 4D array of shape (m, 8, 8, 256)
4	DCONV(N256, K5, S1, P=SAME), BN, LeakyReLU
5	DCONV(N128, K5, S2, P=SAME), BN, LeakyReLU
6	DCONV(N64, K5, S2, P=SAME), BN, LeakyReLU
7	DCONV(N1, K5, S2, P=SAME)

Table 2: Discriminator architecture.

Discriminator	Configuration
Input	
0	CONV(N64, K5, S2, P=SAME), BN, LeakyReLU
1	CONV(N128, K5, S2, P=SAME), BN, LeakyReLU
2	CONV(N256, K5, S2, P=SAME), BN, LeakyReLU
3	reshape to 3D array of shape (m, T, -1)
4	LSTM(state size = 256), BN
5	LSTM(state size = 64)

C: Evaluation metrics

To compute our three metrics, let us first assume that we have a set of real data samples ($\mathcal{P}$) and synthetic data samples ($\mathcal{S}$). EMD is defined as:

$$EMD(\mathcal{P}, \mathcal{S}) = \min_{\phi:\mathcal{P}\to\mathcal{S}} \sum_{p\in\mathcal{P}} \|p - \phi(p)\| \quad (2)$$

where $\phi : \mathcal{P} \to \mathcal{S}$ is a bijection. MMD is defined as:

$$\widehat{MMD}^2(\mathcal{P}, \mathcal{S}) = \frac{1}{n(n-1)}\sum k(p,p) + \frac{1}{n(n-1)}\sum k(s,s) - \frac{2}{n^2}\sum k(p,s) \quad (3)$$

where k denotes a positive-definite kernel (e.g. RBF kernel) and n is the number of (real or synthetic) samples.

Lastly, to compute the KNN score, we first split our real and synthetic samples $\mathcal{P}$ and $\mathcal{S}$ into training and test datasets $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$ so that $\mathcal{D} = \mathcal{D}_{tr} \cup \mathcal{D}_{te}$. We train the KNN classifier $f : \mathcal{X}_{tr} \to [0, 1]$ using training data. The accuracy of the trained classifier is then obtained using test samples $\mathcal{D}_{te}$ and given as:

$$\hat{t} = \frac{1}{n_{te}} \sum_{(z_i, l_i)\in\mathcal{D}_{te}} \mathbb{I}\left[\left(f(z_i) > \frac{1}{2}\right) = l_i\right] \quad (4)$$

where $f(z_i)$ estimates the conditional probability distribution $p(l = 1|z_i)$. A classifier accuracy approaching random chance (50%) indicates better synthetic data. As suggested by **?**, we use a 1-NN classifier to obtain the score.

D: More figures

In this section, we provide more results in larger figures for visual comparisons.

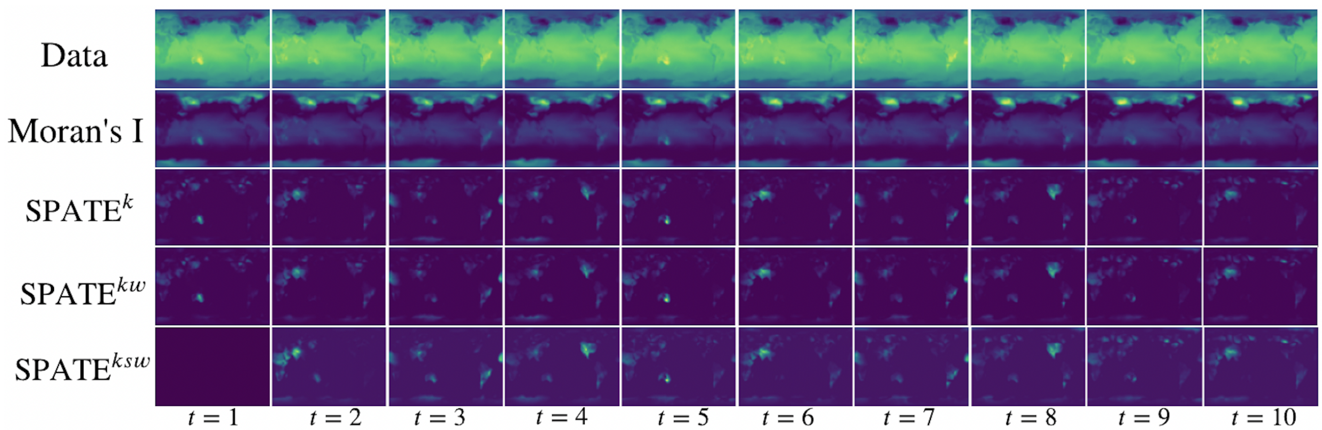

Figure 1: Larger version of Figure 2 for the purpose of visual comparison.

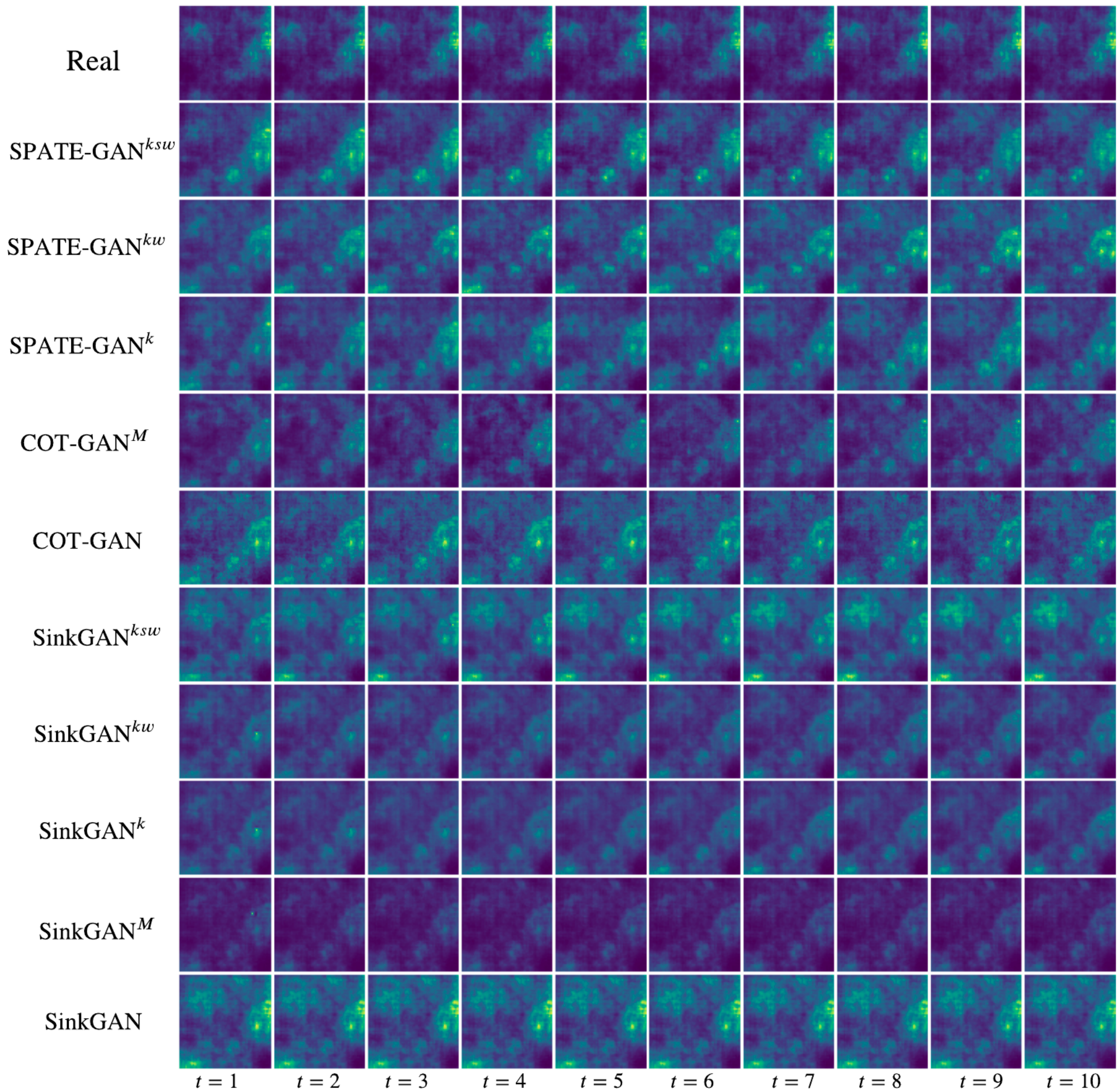

Figure 2: More selected samples for log-Gaussian Cox process (LGCP) dataset.

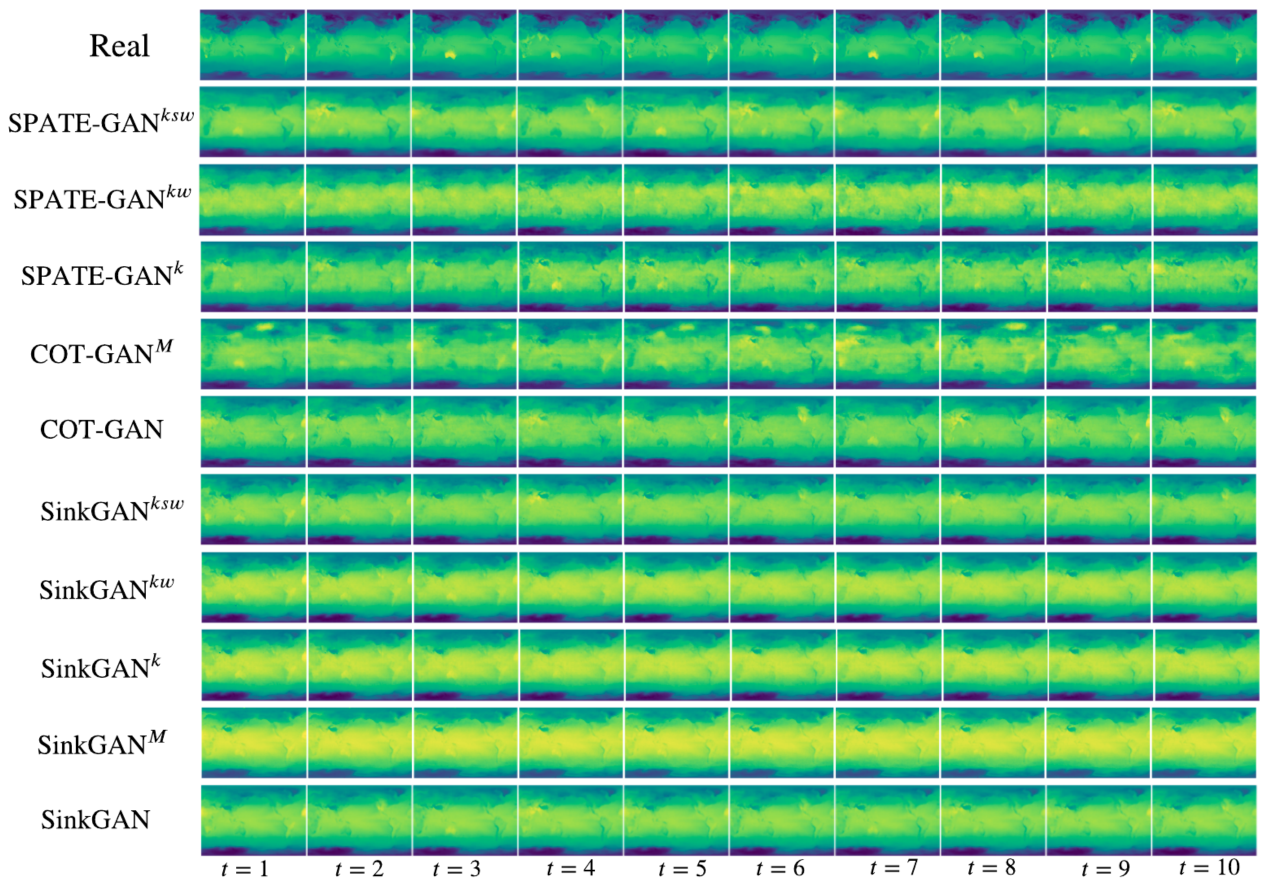

Figure 3: More selected samples for extreme weather (EW) dataset.

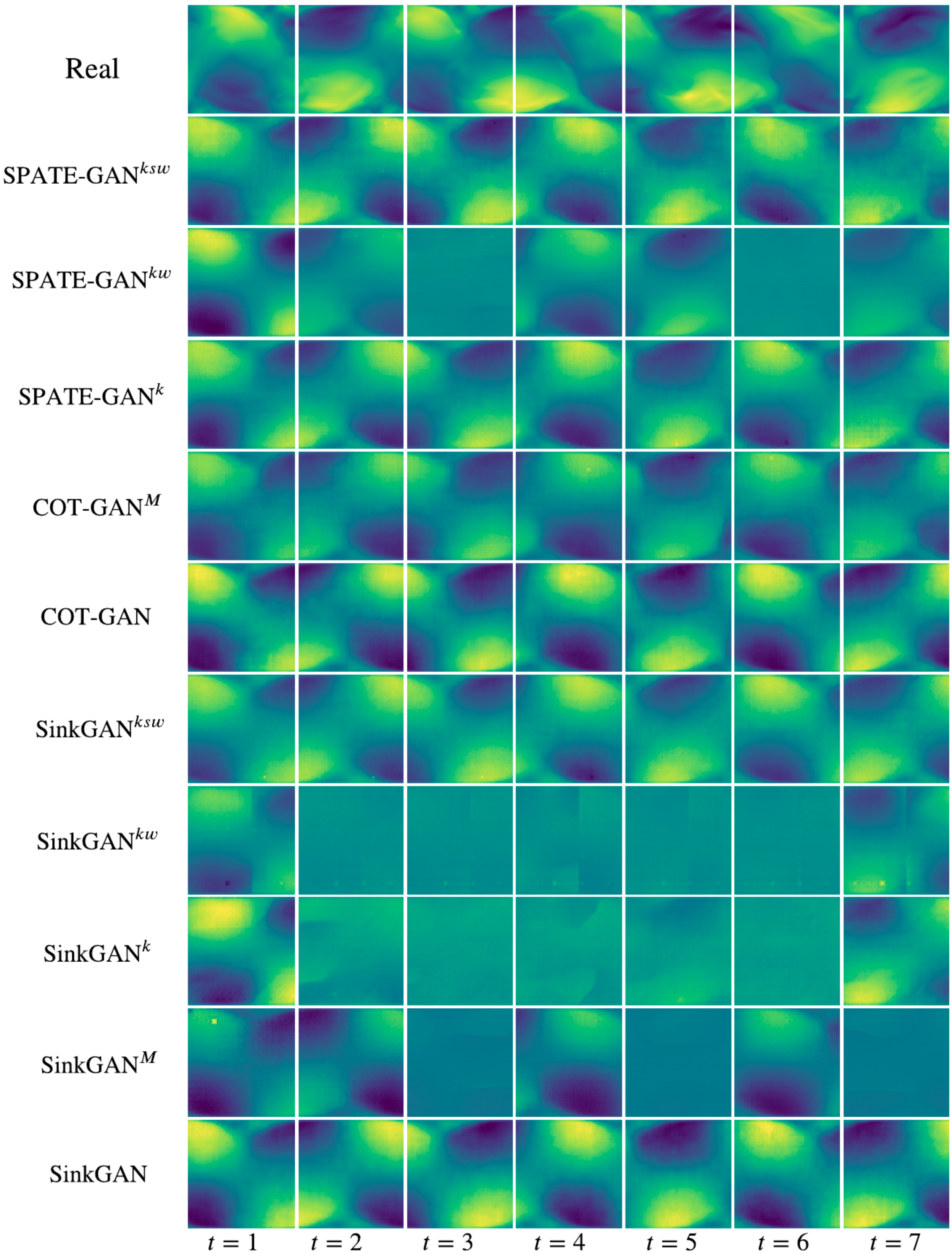

Figure 4: More selected samples for turbulent flow (TF) dataset.