Supplementary Materials

S1. Details on the Training of a Regression model for Learning $\widehat{r^{\widehat{f}}}(\gamma;\mathbf{x})$

The success of `Cal-PIT` depends entirely on learning an accurate representation of $\widehat{r^{\widehat{f}}}(\gamma;\mathbf{x})$. One can in principle choose any regression algorithm and pair it with Algorithm 1. We use monotonic neural networks from Wehenkel & Louppe (2019) as our regression method as we find this architecture gives reasonably good results for all of our experiments. The network is constrained to be monotonic with respect to γ and uses identical sets of fully connected sub-networks to learn the monotonic dependence and the unconstrained dependence separately, with the two results merged in the final layer of the network. It is known that neural networks struggle with categorical inputs and in that case, tree-based regression methods or an additional embedding step might produce better results.

For Examples 1, 3 and the photometric redshift demonstration, we use a network architecture with four hidden layers with 1024, 512, 256, 128 neurons respectively (see supplementary material S3 for the details on Example 2). We use the ReLU activation function (Glorot et al., 2011) for all the hidden layers and the AdamW optimizer (Loshchilov & Hutter, 2019) with an initial learning rate of 0.001 and weight decay parameter set to 0.01. We follow a multiplicative weight decay schedule given by the rule: learning rate (epoch) = initial learning rate $\times$ 0.95^{epoch}. We minimize the Binary Cross Entropy (BCE; Good 1952) loss to train the regression models. The calibration data used to train the model is split into 90:10 partitions where 90% of the data is used to optimize the loss function and 10% of the data is used to calculate a validation loss every epoch on a fixed grid of α. To prevent our model from over-fitting we stop training once the validation loss does not decrease for 50 epochs and save the model with the best validation loss. We use a batch size of 2048 throughout and use an oversampling factor (K) of 2. We used PyTorch (Paszke et al., 2019) to create and train our neural network models and trained them on a single Nvidia A100 GPU. If a value of any hyperparameter is not explicitly mentioned here in the text, it implies that we used the default values set by PyTorch. Training times for all our experiments range from a few minutes to about an hour at maximum.

S2. Remarks on Example 3 (Prediction Sets)

The data for Example 3 in Appendix A consist of two groups with different spreads. The data generating process is given as follows:

$$\begin{aligned} X_{i,1-2} &\sim \text{Uniform}[-5,5]^2 \\ X_{i,0} &\sim \text{Bern}(0.2) \end{aligned}$$

$$\text{if } X_{i,1} < 0: \quad Y_i = \begin{cases} 0.3\beta^T X_i \epsilon_{1,i}, & \text{for } X_{i,0} = 0 \\ 0.3\gamma^T X_i \epsilon_{1,i} + \lambda\epsilon_{2,i}, & \text{for } X_{i,0} = 1 \end{cases}$$

$$\text{if } X_{i,1} > 0: \quad Y_i = \begin{cases} X_{i,1} + 0.3\beta^T X_i \epsilon_{1,i}, & \text{for } X_{i,0} = 0 \\ -X_{i,1} + 0.3\gamma^T X_i \epsilon_{1,i} + \lambda\epsilon_{2,i}, & \text{for } X_{i,0} = 1 \end{cases}$$

where, β and γ as random unit vectors of length 3.

Figure S1 shows that both `Cal-PIT (INT)` and `Cal-PIT (HPD)` have set sizes that are as small as their optimal counterparts ("Oracle Band" and "Oracle HPD", respectively), and that `Cal-PIT (HPD)` sets are indeed more informative (that is, the regions are smaller) than `Cal-PIT (INT)`.

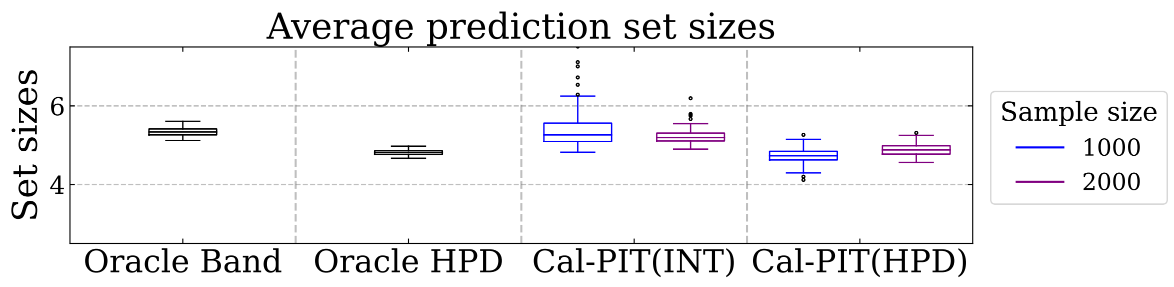


Figure S1: Average prediction set sizes for test points for different methods along with the ideal "Oracle Band" and "Oracle HPD". Box plots show the size distribution for multiple trials of the experiment. `Cal-PIT` achieves prediction sets that are at least as tight as those by other methods, while simultaneously providing more accurate coverage.

We saw that this example is difficult for both quantile regression (`QR`) and orthogonal quantile regression (`OQR`) to learn (see Figure A2). `OQR` augments the standard pinball loss of `QR` with a penalty on the correlation between prediction set size and coverage, which can improve conditional coverage in certain settings (Feldman et al., 2021), but is not very helpful in this example. Figure S2 shows that the initial prediction sets learned by `QR` have bad conditional coverage, but also do not have much correlation between size and coverage. Thus, the penalty applied by `OQR` is unable to substantially improve upon the `QR` results.

We emphasize that methods like `OQR` target *proxies* for conditional coverage, while our `Cal-PIT` method *directly* targets conditional coverage. Therefore, our method succeeds in more general settings. Example 1 is a case where penalizing the correlation between prediction set size and coverage is not a good proxy for achieving conditional coverage, so `OQR` is not as successful as `Cal-PIT` at achieving conditional coverage.

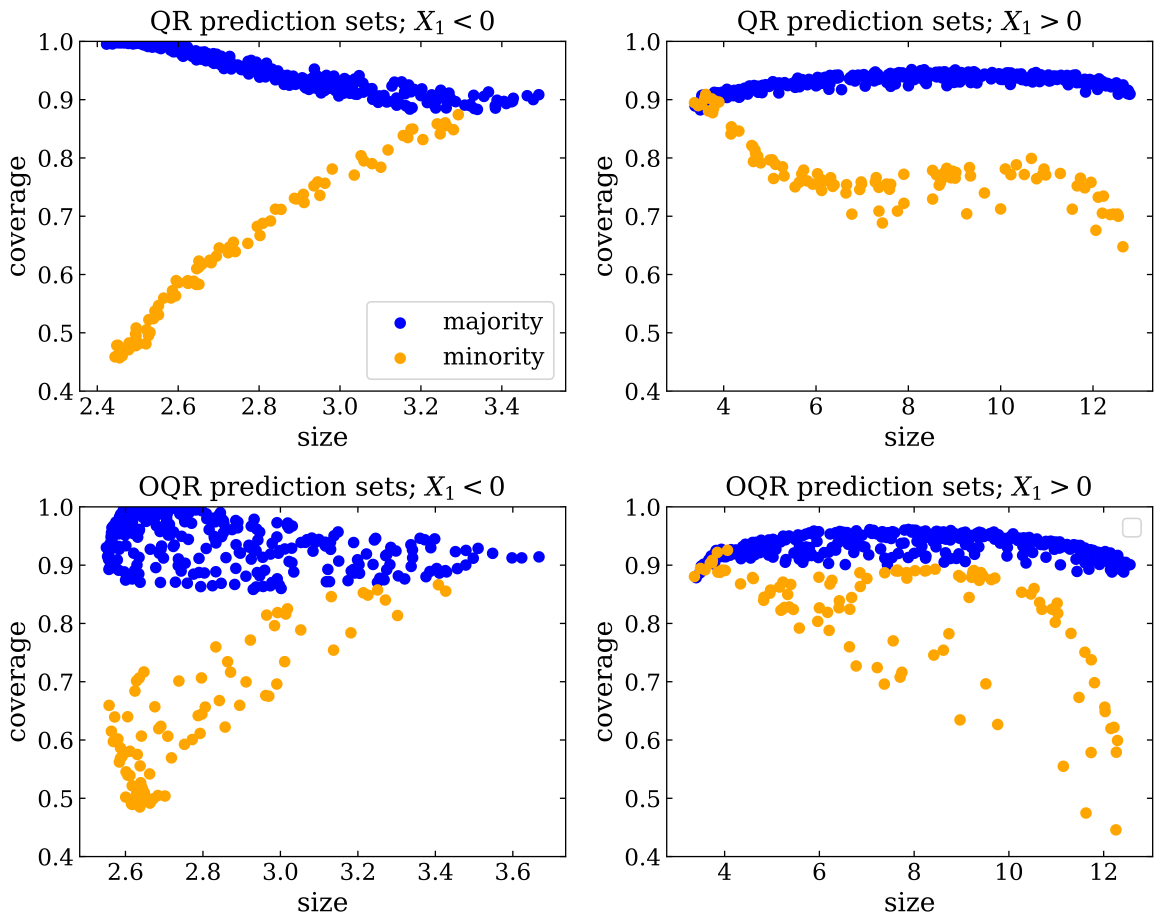


Figure S2: *Top:* Prediction sets from quantile regression (`QR`). We see clear correlations between size and coverage, but note that X_0 is not actually available as a predictor, i.e. we cannot "see" the blue and orange colors. The overall correlations, without the colors, are weak. *Bottom:* Prediction sets from orthogonalized quantile regression (`OQR`). Because the overall correlation between size and coverage is weak, penalizing it does not change the results very much. In particular, we still see high correlations (and bad conditional coverage) in the minority group.

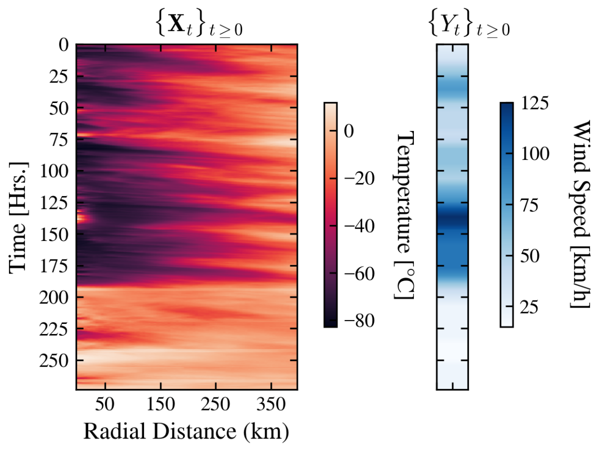
S3. Generating Synthetic Example 3 (High-Dimensional Sequence Data)

S3.1. Synthetic Model for Radial Profiles (Input Variable).

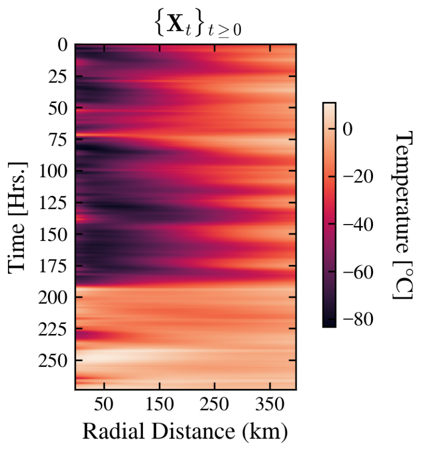
We fit our TC example to TC intensity and location data from National Hurricane Center's (NHC) HURDAT2 best track database (Landsea & Franklin, 2013), and GOES long-wave infrared imagery from National Oceanic and Atmospheric Administration's (NOAA) MERGIR database (Janowiak et al., 2020). HURDAT2 best tracks are provided at 6-hour time resolution, while the GOES IR imagery is available at a 30-minute×4-km resolution over both the North Atlantic (NAL) and Eastern North Pacific (ENP) basins from 2000–2020. Every thirty minutes during the lifetime of a storm, we record a ∼800 km×800 km “stamp” of IR imagery surrounding the TC location, showing cloud-top temperatures for the storm. Figure 5 (left) shows two such stamps.

The radial profile, defined as $T(r) = \frac{1}{2\pi}\int_0^{2\pi} T_b(r,\theta)d\theta$, captures the structure of cloud-top temperatures T_b as a function of radius r from the TC center and serves as an easily interpretable description of the depth and location of convection near the TC core (McNeely et al., 2020; Sanabia et al., 2014). The radial profiles are computed at 5-km resolution from 0-400km (d = 80) (Figure 5, center). Finally, at each time t we stack the preceding 24 hours (48 profiles) into a structural trajectory, $\mathbf{S}_{<t}$, consisting of an image of the most recent 48 rows of the data. We visualize these summaries over time with Hovmöller diagrams (Hovmöller (1949); see Figure 5, right).

Figure S1 (left) shows an example sequence of observed radial profiles every 30 minutes for a real TC, along with observed wind speed Y. We interpolate Y, which is available every six hours, to a 30-minute resolution. Our goal is to create a synthetic example that has a similar dependency structure as actual TCs.



(a) Observed



(b) Reconstructed

Figure S1: Observed (*left*) and reconstructed (*right*) radial profiles $\mathbf{X}_t$ over time for Hurricane GUSTAV (2008). The radial profiles are recorded every 30 mins. We obtain a decent reconstruction by using the three EOFs. Observed wind speed values Y_t, are recorded every six hours but interpolated on the same 30 min grid.

Using the radial profiles from all TC data, we perform a principal component analysis (PCA). Figure S2 (right) shows the first three principal components, or empirical orthogonal functions (EOFs). Figure S1 shows the observation and reconstruction of the TC using just these three EOFs. To create the synthetic data in Example 2, we use a similar reconstruction scheme:

Let $\Delta PC_t := PC_t - PC_{t-30m}$ be the 30-minute change in a PC coefficient at time t for observed data. We fit a vector autoregression (VAR) model to $(\Delta PC1_t, \Delta PC2_t, \Delta PC3_t)$ to capture the dependence of each component on its own lags as well as the lags of the other components. The model chosen by the BIC criterion has order 3, for a lag of 90 minutes (Hyndman, 2018).

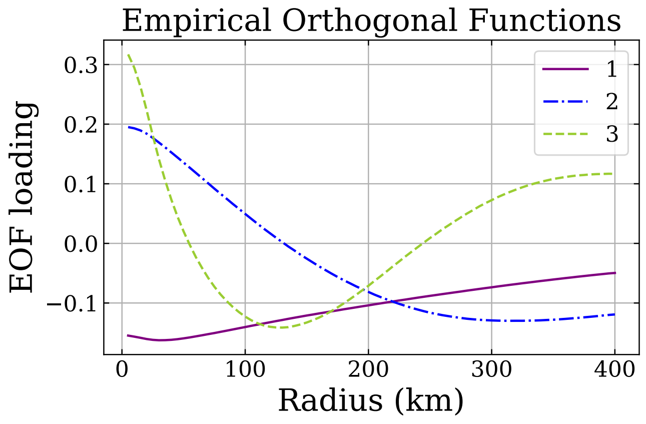


Figure S2: Top 3 PCA components, or empirical orthogonal functions (EOFs), for TC radial profiles.

With the fitted VAR model, we can jointly simulate synthetic time series data for $PC1, PC2, PC3$. A TC structural trajectory is constructed by multiplying simulated time series of PCA coefficients with their corresponding eigenvectors (Figure S2).

S3.2. Synthetic Model for Intensities (Output Variable)

To model the time evolution of intensities Y, we fit a time series regression of intensity change on its past values together with PC coefficients for present and past TC structure.

Let $Z := \text{logit}(Y/200)$ so that simulated values of intensities Y are reasonable, i.e. fall between 0 and 200. We then define $\Delta Z_t = Z_t - Z_{t-6h}$. Finally, we fit the following linear regression model for ΔZ:

$$\begin{aligned}\Delta Z_t = \beta_0 &+ \beta_1 Z_{t-6h} + \beta_2 \Delta Z_{t-6h} + \beta_3 PC1_t + \beta_4 PC2_t + \beta_5 PC3_t \\ &+ \beta_6 PC1_{t-6h} + \beta_7 PC2_{t-6h} + \beta_8 PC3_{t-6h} + \beta_9 PC1_{t-12h} \\ &+ \beta_{10} PC2_{t-12h} + \beta_{11} PC3_{t-18h} + \beta_{12} PC2_{t-24h} + \epsilon_t \end{aligned} \tag{C.1}$$

where ϵ_t is Gaussian noise with mean 0 and standard deviation set to the root mean squared error between the real and predicted radial profiles in the training set. Note that ΔZ_t has dependencies on its own lagged values as well as lagged values of PC_t. Figure 6 in Section 3.2 shows an example TC with simulated radial profiles that update every 30 minutes, with accompanying simulated wind speed Y every 30 minutes.
As a sanity check, we check that the marginal distributions of the simulated and real wind speed values (Y) look similar, as shown in Figure S3.

S3.3. Fitting an Initial CDE (Convolutional MDN)

With our trained VAR model, we generate a very long time series for $PC1, PC2, PC3$ with a value of the PC's randomly selected from the training set of storms as the initial point. The time series is then divided into 24-hour-long chunks and the structural trajectory and intensities are reconstructed. We create 8000 such instances for our training set, 8000 more for our calibration set, and 4000 instances for our test sets. We

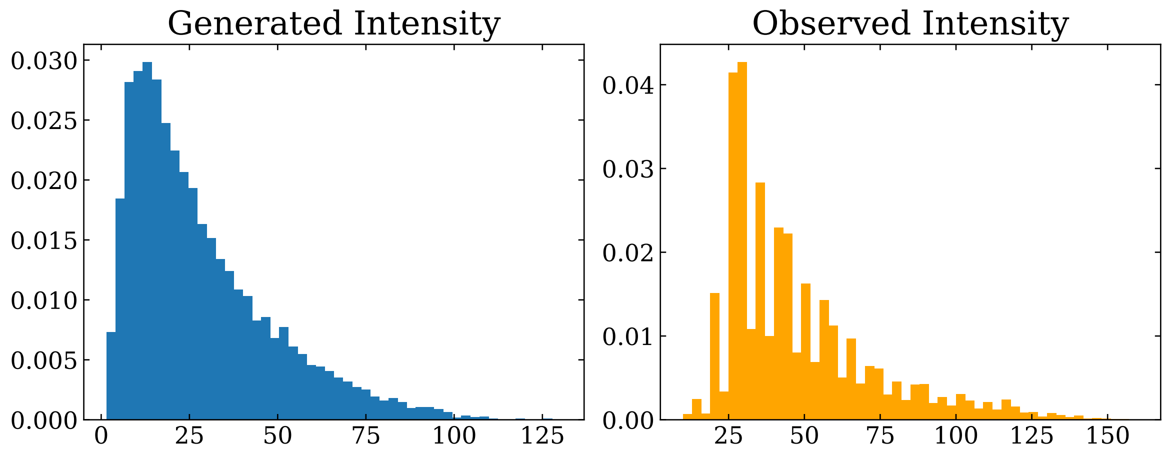


Figure S3: *Left:* Marginal distribution of generated wind speed values Y, based on the model in Equation C.1. *Right:* Marginal distribution of observed wind speed values.

rejected a 24-hour long window between each chunk of the time series to ensure that each instance has no memory of the previous ones.

We fit a unimodal Gaussian neural density model to estimate the conditional density $f(y|\mathbf{s})$ of TC intensities given past radial profiles. Specifically, we fit a convolutional mixture density network (ConvMDN, D'Isanto & Polsterer (2018)) with a single Gaussian component, two convolutional and two fully connected layers which gives an initial estimate of $f(y|\mathbf{s})$.

We then use a convolutional neural network (LeCun et al., 1989; Fukushima & Miyake, 1982) model with two convolutional layers followed by five fully connected layers which take the structural trajectory images and the coverage level (α) as inputs training. The network output is restricted to be monotonic with respect to α (Wehenkel & Louppe, 2019). The activation functions, optimizers and learning rater decay schemes are same as mentioned in supplementary material S1.