Appendix for Improved Contrastive Predictive Coding for Time Series Out-of-Distribution Detection Applied to Human Activity Data

Amirhossein Ahmadian and Fredrik Lindsten

A Sanity test for the advantage of RBF score in uncertainty estimation

To provide additional support regarding the advantage of using RBF kernel over log-bilinear function in CPC based OOD detection (without negative samples), we design a simple experiment that demonstrates the effect of noise on estimated uncertainty (OOD score) in each case. The purpose is to show that the behavior of RBF score is more reasonable on the data points highly corrupted by noise, which should be a positive characteristic for OOD detection in general. Specifically, in this experiment, we simply add random noise to the training in-distribution data (i.e., where the model is expected to be fully confident) at different magnitudes and measure the difference between OOD scores on the original and noisy data. Our hypothesis here is that the random noise should make the model with log-bilinear function over-confident in more cases than the one with RBF.

In practice, we use the best model of our proposed method trained for the KU-HAR dataset. To focus on comparison between the RBF and log-linear modes, we use the proposed OOD detection algorithm without quantization, once with the RBF kernel and the other time with the vanilla log-bilinear function (both in training and as the OOD score; similar to section 5.3). Each data point used here is a window (of size 80) from a time series in the training set of KU-HAR. Given a certain data point x, a noisy version of it ($\bar{x}$) is obtained by adding uniform random noise with scale α (more precisely, the noise added to each channel of x is drawn from $U(-0.5\alpha, 0.5\alpha)$). Then, the "relative confidence" is defined as $C(x) = S(x) - S(\bar{x})$ where $S(.)$ is the OOD score defined in the paper for either the vanilla or RBF without quantization modes (section 5.3). For each data point, we repeat this procedure with different amplitudes and drawing several noise samples. This gives us the estimated distribution of C versus α. Figure 1 compares the results between the RBF and log-linear modes.

Intuitively, we expect the model to be less confident on the inputs which are more contaminated by noise. It is seen in the graph that both the RBF and log-bilinear functions can lead to "overconfidence" in the sense that there are positive relative confidence values even for large noise amplitudes in both cases. However, the RBF mode is significantly better overall since the mean of C is always negative and also its distribution (considering the s.d.) leans towards the negative values in that mode.

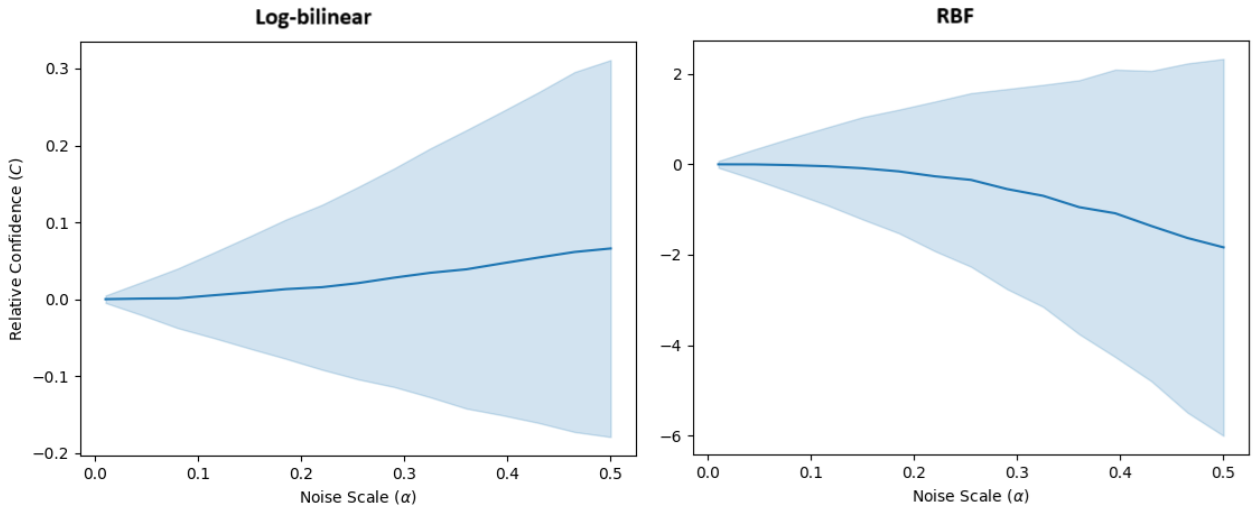

Figure 1: The relative confidence of CPC trained models versus the scale of uniform noise added to in-distribution (KU-HAR) data; when using the log-bilinear and RBF functions/scores. The graphs show the mean and standard deviation of relative confidence. They are obtained by using 100 training data points (time series windows) and 50 samples of noise per each data point.

This outcome can be further explained in theory by noting that the log-bilinear score function $s_L(z_w, c) = \exp(z_w^T W c)$ is equivalent to the exponential of logits of a binary linear classifier in the representation space with a decision hyperplane determined by $z_w^T W$. This score is not "distance aware", in the sense that c can be placed very far from z_w while leading to a very large score depending on which side of the hyperplane it lies at. In contrast, the RBF kernel $s_R(z_w, c) = \exp(-||z_w - c||_v^v)$ leads to a function that necessarily approaches zero when the Euclidean distance between c and z_w vectors approaches infinity. Although in practice we add the random noise to the input data, and not the representation space, the results suggest that the noise is several times more likely to move c and z_w far from each other than moving them closer.

B Experiments with Constant OOD Signals

In this section, we compare the proposed OOD detection approach with the Skip-Step CPC method by using a particular type of synthetic OOD data, which is "constant" (blank) signal. The main reason for this experiment is to verify the claim about the impact of autocorrelation of input time series on the behavior of the Skip-CPC, or any method that uses negative samples from the test data (section 3), and show that our negative-sample free OOD score is more robust in this respect. We pick the constant time series for this purpose (i.e., a series where all the values are the same for all time steps and dimensions) as an extreme case of auto-correlated data.

We use the best models trained on KU-HAR in this part for Skip-Step CPC and our proposed QR-CPC. The assumed OOD data are time series of length 80 with constant value b, where for each OOD data point b is chosen randomly from a uniform distribution [-0.5,0.5]. The Skip-Step CPC method includes a reconstruction part in addition to the CPC classifier-like component. Since our discussion here is about the behavior of the latter on the assumed OOD data, the performance of Skip-Step CPC is also reported while excluding the reconstruction term in its loss and OOD score. Table 1 reports the performance of the Skip-Step CPC and QR-CPC methods, in addition to QR-CPC without context quantization.

The QR-CPC shows the best AUROC performance in this problem. A remarkable observation is the severe negative impact of the reconstruction term in Skip-CPC. This aligns with the known limitations of autoencoders in OOD/anomaly detection, specifically their tendency to reconstruct blank images. The Skip-Step CPC without reconstruction is outperformed by our proposed method on the constant OOD data, though its performance is still significantly higher than random. To further investigate whether the autocorrelation in the input data can directly affect the Skip-Step CPC behavior, we obtain the histogram of softmax values in this method (i.e., normalized values of zWc for positive and negative pairs) on the generated constant series and compare it to the case of KU-HAR OOD data. The result (figure 2) shows a clearly different pattern between the two cases; for constant OODs, the majority of values are concentrated around $1/N$, where N is the number of samples (feature vectors) used in the softmax ($N = 80$ since the series length are 80 here). This matches our guess that highly auto-correlated data can cause the feature vectors to almost collapse around a single value and lead to a type of bias in OOD score of the methods like Skip-Step CPC.

Table 1: AUROC performance of OOD detection with KU-HAR data as in-distribution and constant signals (with random magnitude) as OOD. The results are obtained with 500 OOD data points.

Skip-Step CPC without reconstruction	71.0
Skip-Step CPC	43.5
QR-CPC	81.0
QR-CPC without quantization	80.4

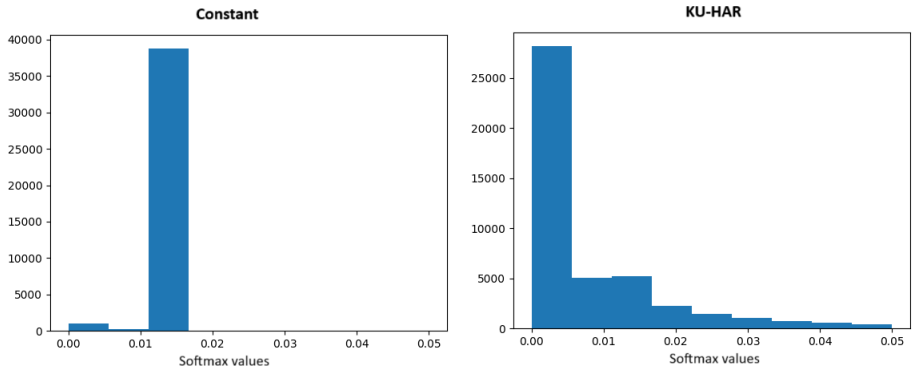

Figure 2: The histogram of softmax values obtained in the classifier-like OOD detection of the Skip-Step CPC method (excluding reconstruction) on the synthetic constant signals (with random magnitude) versus KU-HAR data. The used Skip-Step CPC model is the best one trained on KU-HAR.

C Additional results with vanilla/normalized modes and non-scaled KU-HAR

The effect of replacing the RBF kernel (Gaussian/Laplace) score function in QR-CPC by the vanilla log-linear function and vMF kernel (normalized representations) was studied in section 5.3 of the paper. One might wonder whether the poor performance of the vanilla method can be caused by over-training the model (i.e., a type of over-confidence or overfitting in logits score, when CPC is viewed as a classification task). Thus, we show the performance of the vanilla method (log-bilinear score) across a wide range of training epochs in figure 3 and compare it to the RBF score, without context quantization. As reported in the

paper, it is verified that this method is still largely outperformed by RBF even after this optimization, although approximately there exists a sweet spot for the number of training epochs.

When experimenting with the vMF mode (i.e., normalized representation vectors with $v = 2$, without quantization), we observed that the squared distance $||z_w^+ - c||_2^2$, used in our proposed loss function, tends to be several times smaller during the training than the case of original QR-CPC. This is not strange since both z_w and c are normalized by their L2 norms in this mode. Specifically, at the end of first epoch, the average ratio between this term in the QR-CPC and vMF modes was close to 17 and 16 on HAR and KU-HAR respectively. Therefore, to compensate for this difference and present a fair comparison, we divide the best temperature found for the QR-CPC by the mentioned ratios, and look at the performance of the vMF mode at a few different temperatures around that (by dividing $\{1.0, 2.0, ..., 6.0\}$ by the ratio) in addition to the original optimal value (τ=3.0). The results are given in table 2. Although downscaling the temperature increases the performance considerably, the results still suggest that the advantage of Gaussian kernel (unnormalized representations) should not be simply a coincident with choosing a good temperature since the best AUROC obtained in the Gaussian case is 75.6 on HAR and 76.4 on KU-HAR .

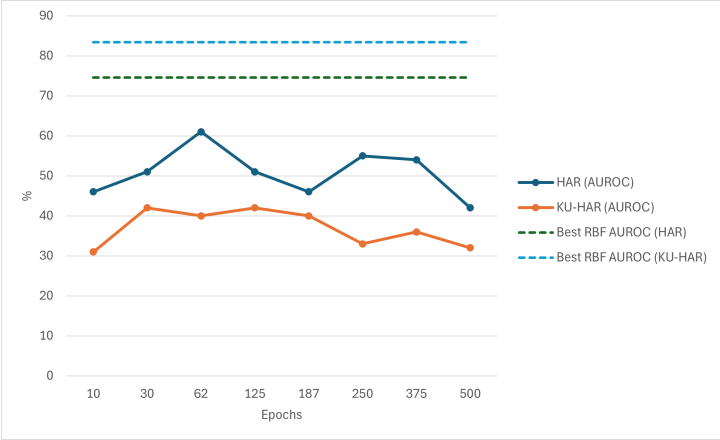

Figure 3: Validation performance of the vanilla CPC OOD detection (i.e., when the proposed method is modified to use log-bilinear score) versus the number of training epochs compared to the best performance with RBF (Gaussian or Laplace) score. Context quantization is not used here.

Table 2: Validation performance of OOD detection in the vMF (l2 normalized) mode for different values of temperature (τ) used in the loss function, without context quantization. The number of training epochs is 250, as it is the best found with Gaussian kernel (unnormalized representations).

	τ	AUROC (%)
HAR		
	3.0	55
	0.05	52.9
	0.11	57.5
	0.17	48.5
	0.23	38.3
	0.29	45.8
	0.35	**61.6**
KU-HAR		
	3.0	35.9
	0.06	49.5
	0.12	53.6
	0.18	65.0
	0.25	68.0
	0.31	**68.2**
	0.37	67.1

As explained in the paper, on KU-HAR, we always rescale the OOD time series by a constant (0.3) in order to avoid a trivial problem in which signal semantics/dynamics do not matter. Figure 4 shows the histograms of the norms of input time series,

compared between the in-distribution and OOD validation data, which reveals that the overlap between the norms is small if rescaling is not applied. To justify our preprocessing method further, in table 3 we report the performance of QR-CPC and compare it to the OC-SVM baseline on the original and rescaled KU-HAR datasets (where, as before, the in-distribution and OOD classes are indoor and outdoor activities). It seems that OC-SVM works better on this task when QR-CPC is trained for 250 epochs (which is the optimal number for the rescaled version of KU-HAR). However, if we simply reduce the number of training epochs to 1, then QR-CPC reaches a performance close to OC-SVM. This experiment suggests that without rescaling, the defined OOD detection on KU-HAR can be possibly solved by a trivial rule (mapping the average magnitude of test time series to an OOD score) and training a neural model quickly leads to a type of "overfitting" in this situation.

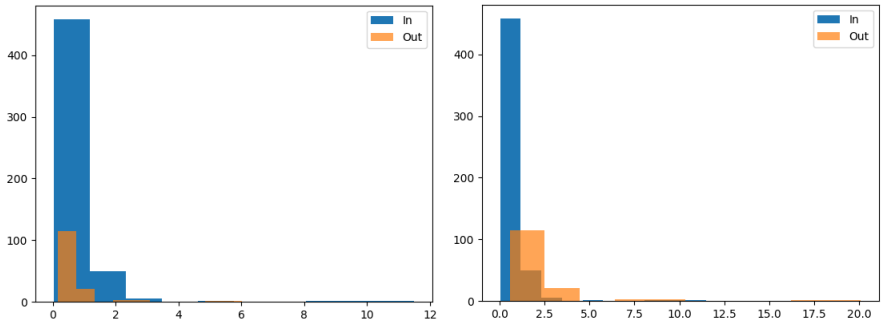

Figure 4: Histograms of L1 norm of the test time series (average of $|x|$ taken over time and dimensions) with rescaled KU-HAR on the left and original KU-HAR (i.e., without OOD rescaling) data on the right. In all of our experiments, "KU-HAR" refers to the rescaled version of data (that is, the harder one).

Table 3: AUROC performance (validation) obtained with and without rescaling the OOD data in KU-HAR. QR-CPC uses Gaussian kernel here, and the other hyperparameters of the two methods are tuned on the rescaled KU-HAR.

Method	KU-HAR	KU-HAR - No Rescaling
QR-CPC (250 epochs)	76.4	80.7
QR-CPC (1 epoch)	31.8	94.4
OC-SVM	56.7	95.3

D Discussion on Gaussian and Laplace kernels

We observe that choosing the distance norm as either $v = 1$ or $v = 2$ in the loss function of QR-CPC, i.e., using a Gaussian or Laplace kernel, is of high importance for OOD detection performance. Table 4 compares the best performance that is achieved with each kernel on each of the two datasets. One observation that might need explanation is that the Gaussian kernel looks to be better on HAR while it is worse than the Laplace kernel on KU-HAR. Although part of this phenomenon might be just because of dataset-specific characteristics of these two particular datasets, we speculate that there is a systematic relation here as well. Specifically, the important structural difference between the HAR and KU-HAR dataset is the segmentation (with respect to activities) in these two datasets. Let us consider how positive pairs, i.e., a context and its following window, are fetched in our training algorithm (section 4.3). Since in HAR, all the activities of a subject are concatenated into a single series, the model is exposed to some noisy (false positive) samples during training as the context and its corresponding window can belong to different activities (near the boundaries between activities). Considering this, unlike KU-HAR, we expect the distribution of positive representation distances ($||z_w^+ - c||_v^v$) in HAR to have an obvious tail on the right side representing the false (outlier) positive pairs. As illustrated in figure 5, the histogram of training distances we obtain with Gaussian kernel on HAR is more aligned with this requirement compared to the one with Laplace kernel.

Furthermore, to provide more evidence on the relation between the segmentation and kernel choice, we create a third dataset based on HAR to compare the performance of the kernels on. We segment the activities according to the HAR labels for this dataset, thus the training set and contrastive algorithm follow the "segmented mode" (similar to KU-HAR). After hyperparameter tuning, the AUROC performance achieved on this "segmented HAR" dataset with Gaussian and Laplace kernels are 66.3 and 58.3 respectively. Thus, a takeaway here can be that the Laplace kernel is more likely to be the appropriate choice in QR-CPC when the training data are segmented with respect to activities.

Table 4: Performance of QR-CPC (% best validation AUROC) with Gaussian ($v = 2$) and Laplace ($v = 1$) kernels

Data	Gaussian	Laplace
HAR	75.6	71.7
KU-HAR	76.4	83.5
Segmented HAR	58.3	66.3

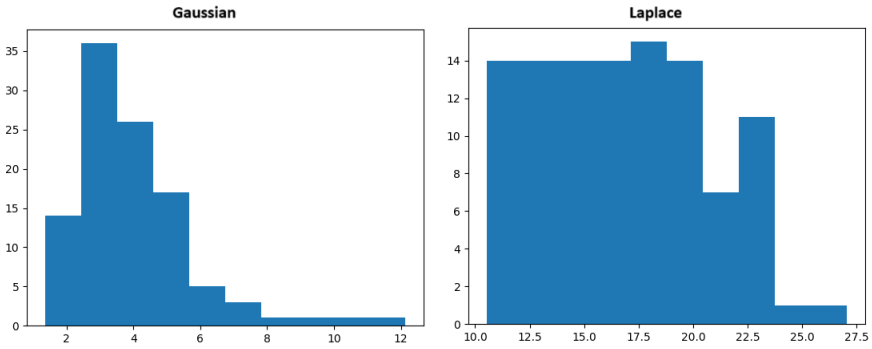

Figure 5: The histogram of positive representations distances, that is $||z_w^+ - c||_v^v/\tau$, obtained during training of the proposed method on HAR (epochs 245 to 250), with Gaussian ($v = 2$) and Laplace ($v = 1$) kernels used in the loss function.

E More details on data and implementations

All the software implementation and experiments are carried out using Python (3.11), PyTorch (2.2), Scikit-learn (1.4), and Scipy (1.13). Each OOD detection method has a score function $S(x)$ at test time which is compared to a threshold to detect the OOD inputs (i.e., x is OOD when $S(x) > \tau$). We use threshold-independent metrics for the final evaluation (AUROC and FPR@95TPR), thus there is no need to specify a certain threshold value. Detailed information regarding the experimented data and methods are provided in the following.

E.1 Data Preprocessing

Some of the preprocessing steps and parameters we use for the HAR dataset are inspired by [26]. For each subject, all the measurements are concatenated into a single time series. All the time series in HAR have 561 dimensions (channels). We use the default training-test partitioning of this dataset which means there is no overlap between the subjects in the training and test partitions. To create the test (validation) set for OOD detection, every time series in the test partition is split into non-overlapping windows of length $L_T = 8$. Then, only the windows which are labeled with a single activity label (i.e., have the same activity label for all the L_T time steps) are used as the test data points. This results in the following number of test data points in each of the classes 0 to 5 respectively: 39-27-22-47-53-52 .

For KU-HAR, we use the "trimmed interpolated" version of the dataset. All the series in KU-HAR experiments have 6 dimensions (channels), which correspond to acceleration and rate of rotation along the axes . As said before, we down-sample all the time series of this dataset by a factor of 10 (using scipy "decimate" function). We also exclude all the time series with length smaller than $L_T = 80$ from our data. Moreover, data normalization is applied to all the data in KU-HAR. Specifically, the per-channel mean and standard deviation of all time series in the training set are computed after clamping all the values to the range of -10 to 10. All the training and test time series are then clamped to the same range and standardized using the per-channel mean and standard deviation. The data of subjects with IDs 1024 to 1086 are used in training and the rest of subjects are held out for test (validation). The data points in the test set are obtained by splitting each time series into non-overlapping windows of length $L_T = 80$. Consequently, following the in-distribution vs. OOD data split defined in the paper (indoor vs. outdoor activities), the number of in-distribution and OOD data points in this set are 2110 and 589 respectively.

E.2 Proposed Method (QR-CPC)

We always use 15 negative samples for the contrastive training, both in our method and CPC-Anomaly/Skip-Step CPC methods (thus, $N = 16$). In the training loop, the context and positive window come from $\hat{x}$, where $\hat{x} = D_p^r$, i.e., the rth time series in D_p. If $|D_p| > 1$ (KU-HAR dataset), r is drawn randomly from $[1, |D_p|]$ with uniform probability. When $|D_p| = 1$ (HAR dataset), r is always 1. For the HAR dataset, the indices of negative windows are obtained by drawing 15 numbers with uniform probability and without replacement on the range $[1, i_{pos} - 2] \cup [i_{pos} + 1, N_W]$ where i_{pos} is the index of the positive window and N_W is the number windows in $\hat{x}$. For KU-HAR, the indices of negative time series (I_N) are firstly found by drawing 15

numbers with uniform probability (with replacement) from $[1, |D_p|] - \{r\}$. Then, for each $i \in I_N$, D_p^i is split into windows of length L_W and one of the windows is chosen randomly (with uniform probability) to be added to the set of negative windows.

In the experiments on HAR, the feature extractor (f) is a neural network with four 1D convolution layers (convolution over time) which have channels of number 400-300-200-100, kernel sizes 1-2-2-2 respectively, and a 'valid' padding for every layer except for the first one which has a 'same' padding. Additionally, all the layers except the last one use ReLU activation function and 1D Batch Normalization (with momentum 0.03). The autoregressive (g) model is a GRU network with 2 layers and the hidden dimension 100, which is followed by a linear layer mapping the hidden state of the GRU (at the last time step) to the output of dimension 100. In such a GRU network (the PyTorch GRU class), the hidden state of each layer at time t is a function of the input at the same time and hidden state at time $t-1$, where in each layer (except for the first one), the input is equal to the hidden state of its preceding layer. When feeding an input to the GRU (which comes from f), we append a zero vector to the end of the sequence as an end of sequence token.

In the KU-HAR experiments, f has an structure similar to the one for HAR but the number of channels and kernel sizes are 10-20-30-100 and 1-2-2-38 respectively. It is reminded that the kernel size in the last layer should be chosen such that an input sequence of length $L_W = 40$ is aggregated into a single vector. The g network is also similar to the case of HAR with the difference that its hidden dimension is 40, which is mapped to the output of dimension 100.

The model parameters are optimized using ADAM with learning rate 1e-4, weight decay 1e-5, and batch size 16 (more accurately, at each iteration for updating the model parameters, the stochastic loss function is the sum of over the losses over 16 samples, each from a different subject). The same optimization hyperparameters have been used for the other implemented methods that use neural networks unless it is stated otherwise.

In the context clustering stage, windows of length of $3L_W$ are obtained from the time series in training set and encoded into the set of context vectors D_C using $u(x) = g(f(x))$. The windows are obtained with a stride of 1 and 40 for the HAR and KU-HAR datasets respectively. K-means method with Kmeans++ initialization is used to cluster the vectors in D_C and find the prototypes (cluster means).

E.3 CPC-Anomaly

The encoder network used in our implementation of CPC-Anomaly, which encodes windows to feature vectors, is exactly same as f in QR-CPC. An additional linear layer (without bias) has the role of the weight matrix (W in the standard CPC loss) in this method. At test time, initially, a set of windows are randomly taken from the training time series and encoded using f to obtain the negative representations Z^-. Then, the same Z^- are used to compute the score on all test instances.

E.4 Skip-Step CPC

While windows of time series are encoded into feature vectors in our method, Skip-Step CPC uses a sequence-to-sequence feature extractor. The networks f and g used in our implementation of Skip-Step CPC are generally similar to QR-CPC. There are two differences in f. First, the kernel size in the last layer is 2 here (as windowing is not used). Secondly, the whole convolutional network is causal in accordance to the model used in that paper (the output for time t is only a function of input at t and earlier time steps). This is achieved by padding the sequences with zero on the left side. Moreover, there is an additional component in the Skip-Step CPC model employed for reconstruction. This reconstruction head is a feedforward network that maps the output of the convolutional network (f) from the dimension 100 to the input dimension (561 or 6), and has two hidden layers (ReLu activation, 100 units) in addition to a linear output layer.

Assuming the skip-step hyperparameter is t_s, the score function of Skip-Step CPC on the test input $\tilde{x}$ of length L_T in our problem setup is as the following:

$$S(\tilde{x}) = -\log \frac{\exp(z_{t_0+t_s}^T W c_{t_0})}{\sum_{t=1}^{L_T} \exp(z_t^T W c_{t_0})} (\tilde{x}_{t_0} - \hat{x}_{t_0})^2 \tag{1}$$

where $t_0 = L_T - 2 - t_s$, $\hat{x}$ is the reconstruction of $\tilde{x}$, and the squared error is computed as the average per dimension.

E.5 Deep SVDD

The encoder network used in our Deep SVDD implementation is a 1D convolution network which is similar to the f in QR-CPC, though with the necessary adjustments. Specifically, we set the bias parameter equal to zero and remove the affine transform from batch normalization in all the layers. We use the simple Deep SVDD loss function (without soft boundary). To initialize the center point, we use the encoder model which is initialized randomly as usual to obtain the representations on each training data point (windows of length L_T), and then fix the center to the mean of the representations.

E.6 COCA

For COCA, the default model architecture implemented by its authors (consisting of convolutional and LSTM networks) is used. However, we make a few changes to the model hyperparameters to make it closer to the model used in QR-CPC. The model in our experiments has three 1D convolution layers, with number of channels 400-200-200 (for HAR) or 10-30-30 (KU-HAR) and kernel sizes 2-8-8. The number of projection channels is 100, and the number of hidden units of LSTM is either

100 (HAR) or 40 (KU-HAR). The window size used in training is equal to L_T (8 for HAR, and 80 for KU-HAR) with 50% overlap (due to the pipeline used in the COCA code, prior to windowing, we concatenate all the time series of the training set into a single training time series). We do not use soft boundary in the training objective. At inference time, the score function proposed in the COCA paper ($S_i(.)$) is used as the OOD score, which is based on the distance of the pair of representations returned by the model with respect to the center point.

F Hyperparameters selection

This section provides the values of hyperparameters that are used when reporting the performance of the proposed method as well as the compared methods (table 1 in the paper), and also describes how they have been optimized. For each method, we only consider a few hyperparameters as the primary ones which should be tuned, and thus many hyperparameters including neural network configurations are only set by an initial guess. To tune the hyperparameters, we use grid search over a handful of candidate values (considering our intuitive guess and computational resources) and select the combination which results in the highest AUROC on the validation set. In the following, the details are given for each method.

QR-CPC. The four hyperparameters that we optimize for our proposed method are the number of training epochs, the temperature (τ), the kernel norm (v), and number of prototypes (N_p). The candidate values examined for these hyperparameters on both datasets are $\{125, 250, 500\}$, $\{1.0, 2.0, 3.0, 4.0, 5.0\}$, $\{1, 2\}$, and $\{5, 10, 20, 30, 40\}$ respectively. The optimal values we use are $(250 \text{ epochs}, \tau = 3.0, v = 2, N_p = 10)$ for HAR and $(125 \text{ epochs}, \tau = 3.0, v = 1, N_p = 20)$ for KU-HAR.

CPC-Anomaly. CPC-Anomaly results on HAR are reported with 250 training epochs and 200 test-time negative samples. On KU-HAR, the training epochs are 500, and 100 negative samples are used. The candidate values for the number of epochs and samples were $\{125, 250, 500\}$ and $\{100, 200, 400\}$.

Skip-Step CPC. The main hyperparameters in this method are the number of epochs and the number of skip time steps (t_s), i.e., the gap between the time until which the context (c_{t_0}) is computed and the time at which the feature $z_{t_0+t_s}$ is obtained. Considering the length of test instances (L_T) for each dataset, we search for optimal t_s over $\{1, 2, .., 5\}$ and $\{1, 10, 20, .., 50\}$ on HAR and KU-HAR respectively. The number of epochs considered were $\{125, 250, 500\}$. The final values used are $t_s = 5$ with 125 epochs (HAR) and $t_s = 1$ with 500 epochs (KU-HAR).

OC-SVM. Given that RBF kernel is a popular kernel in SVM-like methods and one of the best kernels for most problems, we choose the kernel in OC-SVM as RBF. Using Scikit-learn "OneClassSVM class", we tried the automatic kernel spread (γ) calculation as well as setting γ to two times of the automatic value and half of it. For the ν parameter, the default value 0.5 and the values $\{0.2, 0.8\}$ were considered. The optimal hyperparameters found are $\nu = 0.8$, $\gamma = 0.001$ on HAR and $\nu = 0.2$, $\gamma = 0.004$ on KU-HAR.

Deep SVDD. For this method, we tried to tune the number of training epochs and λ, which is the coefficient for the weight decay regularization on the model parameters. The values that we searched over are $\{125, 250, 500\}$ for the number of epochs as before, and $\{10^{-5}, 10^{-4}, 10^{-3}\}$ for λ. The reported performance is obtained with 125 epochs, $\lambda = 10^{-5}$ on HAR, and 250 epochs, $\lambda = 10^{-4}$ on KU-HAR.

COCA. In the early experiments with COCA, we observed signs of collapse when the training was run on our datasets, in the sense that the loss quickly converged to a small value (resulting in a poor OOD detection performance). For this reason, the number of training epochs we consider for this method is smaller than the others. Also, we try larger values for the coefficient of the variance term (ω_2) in the loss function than the default value 0.1, as this term can help to mitigate the collapse (the coefficient of the invariance term is always ω_1=1). Specifically, the used candidate hyperparameter values are $\{5, 10, 20, 50, 100\}$ and $\{0.1, 0.5, 1.0\}$ for the number of epochs and ω_2 respectively. The values we use finally on are 10 epochs, $\omega_2 = 0.5$ on HAR, and 10 epochs, $\omega_2 = 0.1$ on KU-HAR.