Counterfactual Explanations via Latent Structure for Time Series Classification

Akihiro Yamaguchi
Toshiba Corporation

Shizuo Kaji†
Kyushu University

Kaname Matsue
Kyushu University

Ryusei Shingaki
Toshiba Corporation

Abstract

There is a growing need for explainability in time series classification. Counterfactual (CF) generation creates in-distribution synthetic instances that flip the prediction to a desired class. We propose CELT, a model-agnostic CF generation method for time-series classifiers, including non-differentiable and one-class models. In the development phase, CELT learns a structured latent space in which desired-class latent instances form clusters and other latent instances are pushed away. In addition, the design enables segment-wise, time-local edits. In the deployment phase, CELT efficiently generates CFs by editing a minimal number of time-local segments, guided by the learned structure. We formulate both phases as mathematically sound optimization problems that uniformly handle supervised and one-class classification, and we demonstrate effectiveness on UCR datasets.

1 INTRODUCTION

With the popularization of the Internet of Things, *time series classification* (TSC), in which class labels of time-series instances are predicted by machine learning models, has attracted substantial attention from researchers and engineers in both academia and industry. On the other hand, in industrial applications that require high reliability and safety, practitioners and experts need reasons behind classifier decisions. However, many accurate TSC methods adopt black-box models that lack human interpretability.

In explainable AI (XAI), counterfactual explanations specify which features of an instance should change—and by how much—to obtain a desired prediction (Guidotti, 2022; Wachter et al., 2018). Unlike adversarial attacks, which often rely on imperceptible perturbations and move off the data manifold to flip a class label, *counterfactuals* (CFs)—*plausible*, on-manifold instances generated via human-perceptible modifications—aim to flip the prediction to a desired class (Yang et al., 2021). They should also be *sparse*, changing only a few features significantly. Above all, they must be *valid*: the modified instance actually changes the model's prediction to the desired class.

In time-series CF research, most existing methods have inherent limitations. Many methods advertised as "model-agnostic" (Wang et al., 2021; Filali Boubrahimi and Hamdi, 2022; Wang et al., 2024; Yamaguchi et al., 2024) rely on gradients and therefore assume differentiable classifiers, which excludes many state-of-the-art time-series classifiers, most of which are "non-differentiable" (Middlehurst et al., 2024b). Deng et al. (2024) address this gradient dependence, but their method is specialized to block-maxima forecasting and is not applicable to TSC. Yan and Wang (2023) target model-specific CFs. Ji et al. (2024) study anomaly repair but require differentiable anomaly-score functions. In addition, some methods (Bahri et al., 2024; Ates et al., 2021; Delaney et al., 2021) require access to training data at deployment, which can conflict with strict data-governance policies in industrial settings.

Although some model-agnostic methods (Todo et al., 2023; Spinnato et al., 2023; Redelmeier et al., 2024) without the above limitations have been proposed, important issues remain. CVAE (Todo et al., 2023) learns a latent structure by adding a supervised contrastive loss to a variational autoencoder (VAE) and therefore requires labeled abnormal data. For this reason, it cannot handle "one-class classification" (e.g., anomaly detection), where abnormal data are hard to collect in practice (Yamaguchi et al., 2022; Roy et al., 2024). LASTS (Spinnato et al., 2023) performs random search for CFs in a standard VAE latent space, which is inefficient in the absence of structural guidance. MCCE (Redelmeier et al., 2024) uses an autoregressive generative

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. † Shizuo Kaji is now at Kyoto University.

model to guide on-manifold counterfactuals, but is designed for tabular data rather than time-series data. These issues highlight the need for a design—inspired by deep one-class classification (Ruff et al., 2018; Hojjati and Armanfard, 2024)—that builds hyperspherical latent clusters for the normal (desired) class, thereby guiding efficient and effective CF generation.

Moreover, temporal locality is important for time-series CFs: edits should be restricted to a few short, contiguous segments. We refer to this property as segment-wise locality (time-local): after decoding, the modified regions remain concentrated in a small number of adjacent segments rather than spread across the whole sequence. However, existing model-agnostic methods (Todo et al., 2023; Spinnato et al., 2023; Wang et al., 2021) do not impose this constraint during latent-space editing, so changes that are nominally local can diffuse across the entire time series. This issue motivates an editing scheme that explicitly preserves segment-wise locality.

In this study, we propose *Counterfactual Explanations via Latent structure for Time series classification* (CELT), a model-agnostic CF generation method designed to address the aforementioned issues and inherent limitations. CELT first learns a per-classifier latent structure and then generates CFs without accessing training data. To efficiently generate high-quality CFs that are valid, sparse, and plausible, we formulate two optimization problems: (1) learning a structured latent space in development; and (2) generating CFs guided by this structure at deployment. The same formulation covers both supervised and one-class settings, including non-differentiable classifiers.

To guide efficient generation of high-quality CFs, the development phase (1) learns a structured latent space where latent instances of the desired class form clusters (inspired by deep one-class design in one-class settings), other latent instances are pushed away, and temporal order is preserved. This structure provides segment-wise edit controllability. The deployment phase (2) is formulated as a constrained discrete optimization that efficiently generates valid, sparse, and plausible CFs by decoding latent instances with minimal segment edits toward the target cluster of the desired class.

Our main contributions are summarized as follows:

- We propose CELT, a model-agnostic CF generation method that supports non-differentiable and one-class classifiers for TSC, while avoiding access to training instances at deployment.
- We formulate a mathematically grounded two-phase optimization: (1) learning a latent structure with desired-class clusters while preserving temporal order; and (2) generating CFs via minimal segment edits guided by the learned structure.
- We provide theoretical motivation for the core idea of the 2nd-phase optimization algorithm and empirically show efficient high-quality CF generation.

We present the problem setup in §2, describe our method in §§3–4, and then evaluate it in §§5–6, before providing related work and conclusion. The details are provided in the Appendix.

2 PRELIMINARIES

We use bold lowercase and bold uppercase letters (e.g., $\mathbf{v}$ and $\mathbf{V}$) to denote a vector and a matrix. Scalars are denoted by non-bold letters (e.g., v). For a positive integer $N \in \mathbb{N}$, we write $[N] \coloneqq \{1,2,...,N\}$. For a set $\mathcal{A}$, $|\mathcal{A}|$ denotes its cardinality. We represent a vector $\mathbf{v} \in \mathbb{R}^N$ as $[v_n]_{n=1}^N$ and a matrix $\mathbf{V} \in \mathbb{R}^{N \times M}$ as $[\mathbf{v}_n]_{n=1}^N$ with $\mathbf{v}_n \in \mathbb{R}^M$. We write $v_{n,m}$ for the (n,m)-th element of $\mathbf{V}$.

We consider a univariate time-series instance $\mathbf{x}^{\mathrm{or}} \in \mathbb{R}^Q$ and its counterfactual $\mathbf{x}^{\mathrm{cf}} \in \mathbb{R}^Q$, each of length Q. Because some time-series classifiers (e.g., ridge regression classifiers) do not expose class probabilities (Middlehurst et al., 2024b), we use only predicted class labels for broad compatibility. Let $f : \mathbb{R}^Q \to \mathcal{Y}$ denote the classifier, where $\mathcal{Y}$ is the set of class labels. Let $y^{\mathrm{or}} \in \mathcal{Y}$ be the original class of $\mathbf{x}^{\mathrm{or}}$ and $y^{\mathrm{cf}} \in \mathcal{Y}$ the desired alternative class, and assume $y^{\mathrm{or}} = f(\mathbf{x}^{\mathrm{or}}) \neq y^{\mathrm{cf}}$. Hereafter, "classifier" includes both supervised and one-class classifiers.

To evaluate CFs, we use the following metrics. First, *validity* is the fraction of CFs for which the classifier f correctly modifies the predicted class from the original y^{or} to the desired y^{cf}. A higher value is better. Because validity checks whether a CF meets its core definition, it is the most important metric. Second, *sparsity* is the proportion of timestamps whose values do not change significantly to the original values, defined as

$$\text{sparsity} \coloneqq \frac{1}{Q} \sum_{q=1}^{Q} \mathbb{I}\left(\left|x_q^{\mathrm{cf}} - x_q^{\mathrm{or}}\right| < \epsilon\right). \tag{1}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function, which returns 1 if its argument is true and 0 otherwise, following (Todo et al., 2023). Higher values indicate greater sparsity. We set $\epsilon > 0$ to 10% of the standard deviation on the training data in all experiments. Third, *implausibility* measures a CF's deviation from the data manifold, defined as the ℓ_1 distance to the nearest time-series instance in the dataset (following (Guidotti, 2022; Guo et al., 2023)). Lower values indicate CFs closer to the data manifold (i.e., more plausible and realistic). Other metrics are provided in the Appendix.

This study targets CF generation for time series in both supervised and one-class classification settings. Let $\mathcal{D}$

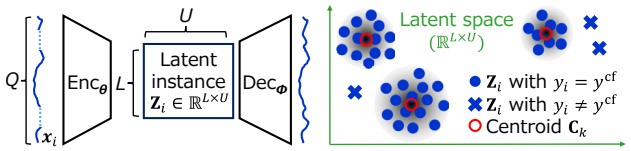

Figure 1: Structured latent space via an AE. (left) The latent space has shape $\mathbb{R}^{L\times U}$; along the temporal axis L, temporal order is preserved. (right) The latent space contains CF clusters where desired-class instances gather, while other instances are separated.

denote the training dataset used during development. For supervised classification, define $\mathcal{D} \coloneqq \{(\mathbf{x}_i, y_i)\}_{i=1}^I$, where $\mathbf{x}_i \in \mathbb{R}^Q$ is the i-th time-series instance and $y_i \in \mathcal{Y}$ its class label; in particular, $\{y^{\mathrm{or}}, y^{\mathrm{cf}}\} \subseteq \{y_i\}_{i=1}^I$. For one-class classification (e.g., anomaly detection), let $\mathcal{D} \coloneqq \{\mathbf{x}_i\}_{i=1}^I$ with $\mathbf{x}_i \in \mathbb{R}^Q$, where (almost) all instances in $\mathcal{D}$ belong to the normal class $y^{\mathrm{cf}} \in \mathcal{Y}$. In both settings, $\mathcal{D}$ is used only during development; at deployment, CFs are generated without access to $\mathcal{D}$. We now present a unified problem statement covering both settings.

Problem statement. Given a pre-trained classifier f, a training dataset $\mathcal{D}$, and a desired class $y^{\mathrm{cf}} \in \mathcal{Y}$ (the normal class in anomaly detection), we learn during development a structured latent space with respect to f and y^{cf}; this learned space constitutes the XAI model. At deployment, given the learned XAI model, f, y^{cf}, and an original instance $\mathbf{x}^{\mathrm{or}} \in \mathbb{R}^Q$ with $y^{\mathrm{or}} = f(\mathbf{x}^{\mathrm{or}}) \neq y^{\mathrm{cf}}$, we efficiently generate a counterfactual $\mathbf{x}^{\mathrm{cf}} \in \mathbb{R}^Q$ that is valid, sparse, and plausible, without accessing $\mathcal{D}$.

3 LEARNING A STRUCTURED LATENT SPACE

This section describes how we learn a structured latent space during development before generating CFs. We first give an overview and introduce distributional latent representations. We then define the loss functions and summarize the continuous optimization problem.

3.1 Overview

We learn a structured latent space with an extended autoencoder (AE), as shown in Fig. 1 (left). Let Enc$_{\boldsymbol{\theta}}$ and Dec$_{\boldsymbol{\phi}}$ denote the encoder and decoder with parameters $\boldsymbol{\theta} \in \mathbb{R}^{P_1}$ and $\boldsymbol{\phi} \in \mathbb{R}^{P_2}$, respectively. The latent space is $\mathbb{R}^{L\times U}$, where L is the temporal dimension and U is the channel dimension. Each row in the latent space corresponds to a time-series segment. As a result, editing a small number of rows in the latent space also locally suppresses the deformation of the decoded time-series instance. The architecture uses a convolutional neural network (NN) with residual blocks. Importantly, it preserves temporal order along axis L by omitting global pooling, dilation, and padding, unlike (Franceschi et al., 2019; Wang and Palpanas, 2021; Todo et al., 2023). This structure provides segment-wise edit controllability for CF generation at deployment. The architecture details are in the Appendix.

Unlike a standard AE, we learn a cluster-structured latent space where each cluster follows a spherical Gaussian, as shown in Fig. 1 (right). Given the number of clusters $K \in \mathbb{N}$, for $k \in [K]$ let the centroid and variance of the k-th cluster be $\mathbf{C}_k \in \mathbb{R}^{L\times U}$ and $\sigma_k^2 \in \mathbb{R}_+$, respectively. That is, $\mathcal{C} \coloneqq [\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_K] \in \mathbb{R}^{K\times L\times U}$ and $\boldsymbol{\sigma}^2 \coloneqq [\sigma_k^2]_{k=1}^K \in \mathbb{R}_+^K$, and both are learnable. To support plausible CFs at deployment, we learn during development a distributional representation for each time-series instance $\mathbf{x}_i \in \mathbb{R}^Q$ (for $i \in [I]$) rather than a single-point representation. Unlike standard VAEs with a single Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$, we regularize the encoder toward a learned set of cluster templates. This encourages latent representations to follow a richer cluster structure.

We first introduce distributional representations so that decoding any latent instance drawn from within a cluster yields a plausible time series (§3.2). We then learn latent representations by minimizing two loss functions. The first loss constructs *CF clusters*, parameterized by $\mathcal{C}$ and $\boldsymbol{\sigma}^2$, encouraging each cluster to contain only latent instances of $y^{\mathrm{cf}} \in \mathcal{Y}$ (§3.3). The second loss reduces reconstruction error with random short-segment masking (§3.4), inspired by (Zerveas et al., 2021; Dong et al., 2023). Finally, we minimize a weighted sum of these losses via a continuous optimization (§3.5); following (Kendall et al., 2018), the weighting parameters are jointly optimized with the other learnable parameters.

3.2 Distributional Representations

To ensure that decoding a latent instance drawn from within a cluster yields a plausible time series, we model each latent instance as a spherical Gaussian distribution rather than a deterministic point. The encoder outputs a latent mean Enc$_{\boldsymbol{\theta}}(\mathbf{x}_i)$ for $i \in [I]$, similar to VAEs. However, unlike a standard VAE, CELT uses learnable per-cluster variances $\boldsymbol{\sigma}^2 \in \mathbb{R}_+^K$ to induce a non-uniform metric in the latent space. Accordingly, the per-instance latent variance $\tilde{\sigma}_i^2 \in \mathbb{R}_+$ is obtained via a soft assignment over the cluster variances $\boldsymbol{\sigma}^2$; these variances $\boldsymbol{\sigma}^2$ are used during CF generation. We use vec$(\cdot)$ to flatten an $L \times U$ matrix into a length-LU vector; the stacking order (row/column) is immaterial. In addition, unvec$(\cdot)$ denotes the inverse of vec$(\cdot)$, i.e., unvec(vec$(\mathbf{V})) = \mathbf{V}$ for any $\mathbf{V} \in \mathbb{R}^{L\times U}$. Thus, we sample a latent instance $\mathbf{Z}_i \coloneqq \mathrm{unvec}(\tilde{\mathbf{z}}_i) \in \mathbb{R}^{L\times U}$ as

$$\mathcal{P}_i \coloneqq \mathcal{N}\big(\mathrm{vec}(\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)), \tilde{\sigma}_i^2 \mathbf{I}\big), \tilde{\mathbf{z}}_i \sim \mathcal{P}_i, \qquad (2)$$

where we omit the dependence of $\mathcal{P}_i$ on $\boldsymbol{\theta}$, $\mathbf{x}_i$, and $\tilde{\sigma}_i^2$. To obtain the latent variance $\tilde{\sigma}_i^2$, we first compute the distance between a latent mean and a cluster centroid:

$$d_{i,k}(\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i),\mathbf{C}_k) \coloneqq \|\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)-\mathbf{C}_k\|_F^2,\ k\in[K], \quad (3)$$

where $\|\cdot\|_F$ is Frobenius norm, and $\mathbf{d}_i \coloneqq [d_{i,k}]_{k=1}^K \in \mathbb{R}_+^K$. Next, we define the soft-minimum selector $\psi_{\mathbf{d}_i}:\mathbb{R}^K\to\mathbb{R}$ to differentiably select the minimum element of $\mathbf{v}\in\mathbb{R}^K$:

$$\psi_{\mathbf{d}_i}(\mathbf{v}) \coloneqq \frac{\sum_{k=1}^K v_k \exp(-\alpha d_{i,k})}{\sum_{k=1}^K \exp(-\alpha d_{i,k})}, \quad (4)$$

where $\alpha \geq 0$ controls the sharpness of the selection. As $\alpha\to\infty$, $\psi_{\mathbf{d}_i}(\mathbf{v})\to v_{k_i^*}$ with $k_i^* \coloneqq \arg\min_{k\in[K]} d_{i,k}$. We set $\alpha = 100$ in all experiments, following (Grabocka et al., 2014; Zhang et al., 2018). Finally, we define the latent variance $\tilde{\sigma}_i^2\in\mathbb{R}_+$ in Eq. (2) by applying the soft-minimum selector to the cluster variances $\boldsymbol{\sigma}^2\in\mathbb{R}_+^K$:

$$\tilde{\sigma}_i^2 \coloneqq \psi_{\mathbf{d}_i}(\boldsymbol{\sigma}^2). \quad (5)$$

For clarity, we define $\tilde{\sigma}_i \coloneqq \sqrt{\tilde{\sigma}_i^2}$ and $\sigma_k \coloneqq \sqrt{\sigma_k^2}$.

3.3 Cluster Loss

To efficiently generate valid CFs (§4), we define a cluster loss $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ that forms CF clusters, each parameterized by $\mathbf{C}_k$ and σ_k^2 for $k\in[K]$. A CF cluster contains only latent instances of the desired class $y^{\mathrm{cf}}\in\mathcal{Y}$ and is modeled as a Gaussian $\mathcal{N}(\mathrm{vec}(\mathbf{C}_k),\sigma_k^2\mathbf{I})$, while latent instances of other classes are pushed away. We use the pre-trained classifier f to determine class membership, i.e., whether $f(\mathbf{x}_i)=y^{\mathrm{cf}}$ for $i\in[I]$. For $\mathbf{x}_i$ with $f(\mathbf{x}_i)=y^{\mathrm{cf}}$, $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ pulls the latent mean $\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)\in\mathbb{R}^{L\times U}$ toward its nearest centroid $\mathbf{C}_k$ over $k\in[K]$ and regularizes the latent variance $\tilde{\sigma}_i^2$ toward 1, yielding compact, distributional clusters. This extends standard VAEs—which assume a single Gaussian prior $\mathcal{N}(\mathbf{0},\mathbf{I})$—by adopting multiple learnable cluster templates for richer structural guidance. Conversely, for $\mathbf{x}_i$ with $f(\mathbf{x}_i)\neq y^{\mathrm{cf}}$, the loss pushes $\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)$ away from the CF clusters with margin $M\in\mathbb{R}_+$. In summary, given M, we define $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ as follows:

$$\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}(\mathbf{x}_i) \coloneqq \begin{cases} \psi_{\mathbf{d}_i}\Big(\big[\mathrm{KL}\big(\mathcal{P}_i\big\|\mathcal{N}(\mathrm{vec}(\mathbf{C}_k),\mathbf{I})\big)\big]_{k=1}^K\Big) & \text{if } f(\mathbf{x}_i)=y^{\mathrm{cf}}, \\ \max\{0, M-\mathrm{dist}_{\boldsymbol{\theta}}(\mathbf{x}_i,\mathcal{C})\} & \text{if } f(\mathbf{x}_i)\neq y^{\mathrm{cf}}, \end{cases} \quad (6)$$

where $\mathcal{P}_i$ and $\psi_{\mathbf{d}_i}$ are in Eqs. (2) and (4), KL is Kullback-Leibler (KL) divergence, and the distance $\mathrm{dist}_{\boldsymbol{\theta}}$ is

$$\mathrm{dist}_{\boldsymbol{\theta}}(\mathbf{x}_i,\mathcal{C}) \coloneqq \psi_{\mathrm{sg}(\mathbf{d}_i)}\left(\left[\|\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)-\mathrm{sg}(\mathbf{C}_k)\|_F^2\right]_{k=1}^K\right),$$

where $\mathrm{sg}(\cdot)$ is a stop-gradient operator.

If all predicted classes in the training data are $y^{\mathrm{cf}}\in\mathcal{Y}$, the lower part of Eq. (6) is not used, and the desired class's latent instances are just gathered into hyperspherical clusters. This aligns with the idea of deep one-class classification trained solely on normal data (Ruff et al., 2018; Hojjati and Armanfard, 2024). Some readers might perceive an inconsistency: Eq. (6) uses KL divergence between distributions when $f(\mathbf{x}_i)=y^{\mathrm{cf}}$, whereas $\mathrm{dist}_{\boldsymbol{\theta}}$ uses the Frobenius norm between points when $f(\mathbf{x}_i)\neq y^{\mathrm{cf}}$. However, the following equation holds in our settings:

$$2\mathrm{KL}\big(\mathcal{P}_i\big\|\mathcal{N}(\mathrm{vec}(\mathbf{C}_k),\mathbf{I})\big) = \|\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}_i)-\mathbf{C}_k\|_F^2 + LU\big(\tilde{\sigma}_i^2-\ln\big(\tilde{\sigma}_i^2\big)-1\big).$$

3.4 Reconstruction Loss

We define a reconstruction loss $\mathcal{L}^{(\mathrm{Rec})}_{\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{C},\boldsymbol{\sigma}}$ that is robust to local, contiguous corruptions by masking short contiguous segments rather than independent pointwise masking (Zerveas et al., 2021; Dong et al., 2023). For each training instance $\mathbf{x}_i\in\mathbb{R}^Q$ for $i\in[I]$, we generate a set $\mathcal{M}$ consisting of binary masks $\mathbf{m}\in\{0,1\}^Q$ independently at each epoch. We sample the latent instance of a masked $\mathbf{x}_i$ as $\tilde{\mathbf{z}}_{i,\mathbf{m}}\sim\mathcal{N}(\mathrm{vec}(\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{m}\odot\mathbf{x}_i)),\tilde{\sigma}_i^2\mathbf{I})$ where $\odot$ indicates element-wise multiplication, and $\tilde{\sigma}_i^2$ is defined in Eq. (5). We then formulate $\mathcal{L}^{(\mathrm{Rec})}_{\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{C},\boldsymbol{\sigma}}$ as follows:

$$\mathcal{L}^{(\mathrm{Rec})}_{\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{C},\boldsymbol{\sigma}}(\mathbf{x}_i) \coloneqq \sum_{\mathbf{m}\in\mathcal{M}\cup\{\mathbf{1}\}} \|\mathrm{Dec}_{\boldsymbol{\phi}}(\mathrm{unvec}(\tilde{\mathbf{z}}_{i,\mathbf{m}}))-\mathbf{x}_i\|_2^2, \quad (7)$$

where $\mathbf{1}\in\{0,1\}^Q$ denotes no masking. Following (Dong et al., 2023), the masked segments follow a geometric distribution with mean 3, each feature (i.e., each timestamp) is masked on average by 50%, and $|\mathcal{M}|=3$. We can effectively approximate the sampling process of $\tilde{\mathbf{z}}_{i,\mathbf{m}}\in\mathbb{R}^{LU}$ by the reparametrization trick, that is, $\tilde{\mathbf{z}}_{i,\mathbf{m}} = \mathrm{vec}(\mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{m}\odot\mathbf{x}_i)) + \tilde{\sigma}_i\mathbf{e}$, where we sample $\mathbf{e}\in\mathbb{R}^{LU}$ independently from $\mathcal{N}(\mathbf{0},\mathbf{I})$.

3.5 Optimization Formulation

Finally, given desired class y^{cf}, pre-trained classifier f, and training dataset $\mathcal{D}$, we can jointly optimize the AE model parameters $(\boldsymbol{\theta},\boldsymbol{\phi})$ and the centroids $\mathcal{C}$ and variances $\boldsymbol{\sigma}^2$ of CF clusters as in

$$\underset{\substack{\mathcal{C}\in\mathbb{R}^{K\times L\times U},\boldsymbol{\sigma}^2\in\mathbb{R}_+^K \\ \boldsymbol{\theta}\in\mathbb{R}^{P_1},\boldsymbol{\phi}\in\mathbb{R}^{P_2}}}{\mathrm{minimize}} \sum_{i=1}^I \lambda_1\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}(\mathbf{x}_i)+\lambda_2\mathcal{L}^{(\mathrm{Rec})}_{\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{C},\boldsymbol{\sigma}}(\mathbf{x}_i), \quad (8)$$

where weighting parameters $\lambda_1\in\mathbb{R}_+$ and $\lambda_2\in\mathbb{R}_+$ balance the impact of each loss. Since different loss may be of different scales, it is difficult to properly pre-define

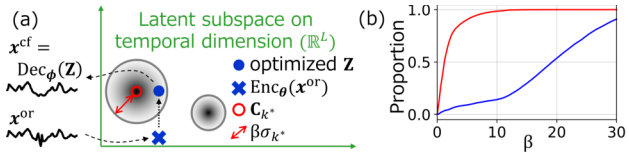

Figure 2: (a) CF generation, (b) Proportions inside radius $\beta\sigma_k$ for y^{cf} (red) vs others (blue) as β varies.

the values of λ_1 and λ_2. We follow the tuning strategy presented by (Kendall et al., 2018) to dynamically learn λ_1 and λ_2 during the training process. Specifically, we add a new regularization term $\ln(\rho_1\rho_2)$ to the objective function, and set λ_n to $\frac{1}{2\rho_n^2}$ for $n\in\{1,2\}$.

To improve numerical stability, we use $\tilde{\rho}_n := \ln(\rho_n^2)$ and $s_k := \ln(\sigma_k^2)$ as the optimization parameters, instead of directly using ρ_n and σ_k. Since all optimization parameters $\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{C}, \mathbf{s}$, and $\tilde{\boldsymbol{\rho}}$ are unconstrained and continuous, the final optimization problem also becomes unconstrained and continuous. Thus, we can efficiently learn it using standard stochastic gradient descent (SGD) algorithms. In addition, we automatically and heuristically determine the number of CF clusters K and the margin M in Eq. (6), as detailed in the Appendix.

The proposed method is designed so that a separate CELT model is learned for each desired class, and this entails additional training cost in general multi-class settings. The primary application scenarios we have in mind are industrial domains such as manufacturing, infrastructure, and medicine, where high reliability and safety are required, and in such settings it is often possible to spend a relatively long time in the development phase before deploying the model. Furthermore, in many real-world deployment scenarios, the number of desired classes for which users actually require CF explanations is itself not very large. Even if the classifier handles many class labels, the main interest in practice is typically in explanations for a limited number of classes, such as "how should this time series be modified to be classified as normal (or a small number of safe/desired states)?" The need to generate CFs for all classes is relatively rare. For these reasons, within the scope of applications we are targeting, we believe that learning CELT separately for each desired class does not cause a serious scalability issue in practice.

4 CF GENERATION

This section describes how to generate CFs using the latent structure learned in §3. We first give an overview, then formulate a constrained discrete optimization problem and present a greedy algorithm with theoretical motivation for its core edit strategy.

4.1 Overview

When we observe time series $\mathbf{x}^{\mathrm{or}}\in\mathbb{R}^Q$ with $y^{\mathrm{or}}\neq y^{\mathrm{cf}}$, the deployment-time procedure in Fig. 2 (a) is as follows: (i) initialize a latent instance as $\mathbf{Z}\leftarrow\mathbf{Z}^{\mathrm{or}} := \mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}^{\mathrm{or}})\in\mathbb{R}^{L\times U}$ by encoding $\mathbf{x}^{\mathrm{or}}$; (ii) select the target CF cluster; (iii) optimize $\mathbf{Z}$ to lie near the selected cluster's centroid; (iv) decode it to obtain the CF as $\mathbf{x}^{\mathrm{cf}} := \mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})\in\mathbb{R}^Q$.

4.2 Optimization Formulation

We adopt a metric induced by the per-cluster variances $\boldsymbol{\sigma}^2\in\mathbb{R}_+^K$ in the latent space: each CF cluster has a centroid $\mathbf{C}_k\in\mathbb{R}^{L\times U}$ and a variance $\sigma_k^2\in\mathbb{R}_+$ for $k\in[K]$. We encode $\mathbf{x}^{\mathrm{or}}$ into the latent space and select the target CF cluster index as the nearest under this metric:

$$k^* := \arg\min_{k\in[K]} \frac{1}{\sigma_k^2}\left\|\mathbf{C}_k-\mathbf{Z}^{\mathrm{or}}\right\|_F^2, \quad \mathbf{Z}^{\mathrm{or}} := \mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}^{\mathrm{or}}), \quad (9)$$

that is, we measure the closeness of $\mathbf{Z}^{\mathrm{or}}\in\mathbb{R}^{L\times U}$ to a CF cluster by the squared Mahalanobis distance.

We optimize a latent instance $\mathbf{Z} := [\mathbf{z}_l]_{l=1}^L \in \mathbb{R}^{L\times U}$ to generate a CF as $\mathbf{x}^{\mathrm{cf}} := \mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})$. In our AE, the L rows of $\mathbf{Z}$ preserve temporal order and locality; accordingly, we edit only a few time-local rows and keep the others equal to the original encoding, i.e., $\mathbf{z}_l = \mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}^{\mathrm{or}})_l$ for many $l\in[L]$. Moreover, CF clusters are modeled as Gaussians $\{\mathcal{N}(\mathrm{vec}(\mathbf{C}_k),\sigma_k^2\mathbf{I})\}_{k=1}^K$ composed solely of latent instances of the desired class $y^{\mathrm{cf}}\in\mathcal{Y}$; hence sampling $\mathbf{Z}$ near the selected centroid $\mathbf{C}_{k^*}$ increases the chance that decoding yields a valid and plausible CF. Accordingly, we explicitly constrain $\mathbf{Z}$ to lie near $\mathbf{C}_{k^*}$.

To summarize, given $\mathbf{x}^{\mathrm{or}}$, a desired class y^{cf}, and the learned encoder $\mathrm{Enc}_{\boldsymbol{\theta}}$ and CF clusters (parametrized by $\mathcal{C}$ and $\boldsymbol{\sigma}^2$), we minimize the number of edited rows of $\mathbf{Z}$ subject to a vicinity constraint around $\mathbf{C}_{k^*}$, as in

$$\underset{\mathbf{Z}\in\mathbb{R}^{L\times U}}{\text{minimize}} \sum_{l=1}^{L}\mathbb{I}(\mathbf{z}_l\neq\mathbf{z}_l^{\mathrm{or}}) \text{ s.t. } \|\mathbf{Z}-\mathbf{C}_{k^*}\|_F\le\beta\sigma_{k^*}, \quad (10)$$

where the target cluster index k^* is selected as in Eq. (9), $\beta>0$ is a scale parameter, and $\mathbf{Z}^{\mathrm{or}}=[\mathbf{z}_1^{\mathrm{or}},\mathbf{z}_2^{\mathrm{or}},\ldots,\mathbf{z}_L^{\mathrm{or}}]$.

4.3 Solution Method and Theoretical Analysis

To solve the constrained discrete optimization in Eq. (10), Algorithm 1 gives an efficient greedy procedure under a fixed target CF cluster. It initializes $\mathbf{Z}\leftarrow\mathbf{Z}^{\mathrm{or}}$ and iteratively updates $\mathbf{Z}$ by setting $\mathbf{z}_{l_n}\leftarrow\mathbf{c}_{k^*,l_n}$ for $n\in[L]$ in descending order of $\|\mathbf{c}_{k^*,l}-\mathbf{z}_l\|_2$, until the condition $\|\mathbf{Z}-\mathbf{C}_{k^*}\|_F\le\beta\sigma_{k^*}$ (and the optional classifier check $f(\mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z}))=y^{\mathrm{cf}}$) is met. While Algorithm 1 is greedy, it enjoys the following property under a fixed target CF cluster and without the optional classifier check: it returns a solution that edits the fewest number of rows to satisfy $\|\mathbf{Z}-\mathbf{C}_{k^*}\|_F\le\beta\sigma_{k^*}$.

Algorithm 1 CF generation by solving Eq. (10).

Input: Original time series $\mathbf{x}^{\mathrm{or}} \in \mathbb{R}^Q$; Desired class $y^{\mathrm{cf}} \in \mathcal{Y}$; Learned encoder $\mathrm{Enc}_{\boldsymbol{\theta}}$; Learned CF clusters $(\mathcal{C}, \boldsymbol{\sigma})$; Target CF cluster index $k^* \in [K]$; Scale parameter $\beta \in \mathbb{R}_+$
Output: Optimized latent instance $\mathbf{Z} := [\mathbf{z}_l]_{l=1}^L \in \mathbb{R}^{L \times U}$

1: $\mathbf{Z} \leftarrow \mathrm{Enc}_{\boldsymbol{\theta}}(\mathbf{x}^{\mathrm{or}})$.
2: Sort indices $l \in [L]$ in descending order of $\|\mathbf{c}_{k^*,l} - \mathbf{z}_l\|_2$.
3: Let the sorted indices be $[l_1, l_2, ..., l_L]$.
4: **for** $n = 1$ to L **do**
5: $\quad \mathbf{z}_{l_n} \leftarrow \mathbf{c}_{k^*,l_n}$.
6: $\quad$ **if** $\|\mathbf{Z} - \mathbf{C}_{k^*}\|_F \le \beta\sigma_{k^*}$ and (*optional classifier check* $f(\mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})) = y^{\mathrm{cf}}$) **then**
7: $\quad\quad$ **return** $\mathbf{Z}$
8: $\quad$ **end if**
9: **end for**

Proposition 1. *The output of Algorithm 1 without the optional classifier check at line 6 is optimal in Eq. (10).*

Proof. Note that Algorithm 1 does not perform the line-6 check $f(\mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})) = y^{\mathrm{cf}}$ and fixes the target cluster index k^*. If we change only one row, the biggest one-step decrease of $\|\mathbf{Z} - \mathbf{C}_{k^*}\|_F$ is obtained by replacing the row $\mathbf{z}_l$ with the largest current distance $\|\mathbf{z}_l - \mathbf{c}_{k^*,l}\|_2$ by $\mathbf{c}_{k^*,l}$ (this makes that row's distance zero). We repeat this choice, always taking the row with the largest distance, until the constraint $\|\mathbf{Z} - \mathbf{C}_{k^*}\|_F \le \beta\sigma_{k^*}$ holds. This uses the fewest possible row edits. Because the objective counts exactly the number of edited rows, this procedure is optimal for Eq. (10). □

4.4 Practical Considerations

When the latent structure is learned to minimize the cluster loss in §3.3, a properly chosen β in Eq. (10) leads the decoder $\mathrm{Dec}_{\boldsymbol{\phi}}$ to generate valid CFs (predicted as $y^{\mathrm{cf}} \in \mathcal{Y}$). However, it is difficult to identify a β that guarantees this property. In practice, we start from a conservatively large β and iteratively move $\mathbf{Z}$ toward the target centroid $\mathbf{C}_{k^*}$ until satisfying $f(\mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})) = y^{\mathrm{cf}}$.

Figure 2 (b) shows, across the 26 datasets in §5.1 (supervised setting), the within-class proportions of latent instances that fall inside the radius $\beta\sigma_k$ over $k \in [K]$ as β varies (red: y^{cf}, blue: y^{or}). As β increases, y^{cf} instances fall within the radius at consistently higher rates than others; in particular, at β=10, 98.4% of y^{cf} lie inside the radius versus 14.1% for other classes. This suggests that the CF clusters behave as intended—concentrating y^{cf} while repelling others—though some leakage of other classes remains. Empirically, β=10 balances validity and leakage (Fig. 2 (b)); we therefore adopt β=10 as a practical default and keep the optional classifier check in Algorithm 1 (line 6), enforcing $f(\mathrm{Dec}_{\boldsymbol{\phi}}(\mathbf{Z})) = y^{\mathrm{cf}}$.

5 EXPERIMENTS

We describe the experimental setup and quantitative results. Further details are in the Appendix.

5.1 Experimental Setup

We train supervised time-series classifiers using QUANT (Dempster et al., 2024), which is recommended for its high accuracy and speed (Middlehurst et al., 2024b). QUANT first extracts features from training data in an unsupervised step and then trains a supervised classifier on these features. Building on this, we implement one-class time-series classifiers by training LOF (Breunig et al., 2000), which has good results in (Han et al., 2022), on the same QUANT features.

We compare the proposed method, **CELT**, with six baselines. Four existing methods—**CVAE** (Todo et al., 2023), **LASTS** (Spinnato et al., 2023), **MCCE** (Redelmeier et al., 2024), and **MCCE++**—are model-agnostic, support non-differentiable classifiers, and do not require deployment-time access to training data, as in CELT. MCCE++ is a variant of MCCE by choosing the best sparsity in Eq. (1) among the valid CFs. The remaining two methods are ablation variants of CELT: the "**w/o Clu**" does not learn CF clusters (i.e., with $\tilde{\sigma}_i = 1$, $\lambda_1 = 0$, and $\lambda_2 = 1$) in Eqs. (5) and (8); and the "**w/o Mask**" does not use the masked AE (i.e., with $\mathcal{M} = \emptyset$) in Eq. (7). Since CVAE uses supervised contrastive loss, its application is limited to supervised classification.

We use UCR Archive (Dau et al., 2018), setting the desired class y^{cf} and the original class y^{or} to 1 and 2, respectively. We use the same train/test split for both the classifiers and the XAI models. Since the default training data size yields insufficient accuracy for the one-class classification, we adjust the split: for each dataset, we use the last 30 instances of classes y^{cf} and y^{or} for testing, and the original training set plus all remaining test instances for training. For the 26 datasets in which both supervised and one-class classifiers achieve at least 50% test accuracy, Fig. 3 (a) reports their test accuracy. The results confirm that one-class classification is harder than supervised classification and that the number of correctly classified instances is highly skewed across datasets. For each dataset, we thus run the following experiments on up to 10 test instances that the classifiers correctly classifies, selecting instances in the original test-set order, unless stated otherwise.

5.2 Evaluation of CF Cluster Learning

We validate CF-cluster learning by comparing the latent spaces learned by CELT and "w/o Clu". Our goal is that latent instances of the desired class $y^{\mathrm{cf}} \in \mathcal{Y}$ concentrate within their CF clusters (modeled as spherical Gaussians), while latent instances of the original class $y^{\mathrm{or}} \in \mathcal{Y}$ fall outside these clusters. To test this, we fit a Gaussian mixture model (GMM) to the y^{cf} latent instances and then compute the log-likelihood of latent instances from both y^{cf} and y^{or} under the fitted GMM.

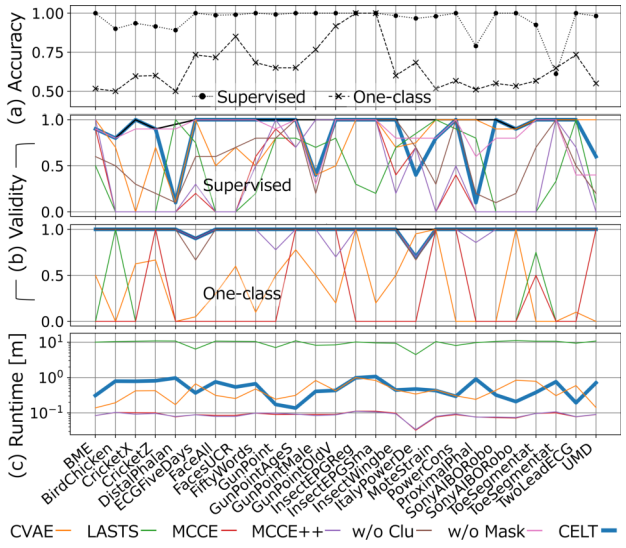

Figure 3: (a) Classifier accuracy, (b) validity, and (c) runtime of each method for each of the 26 datasets.

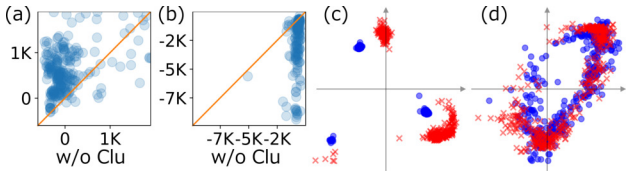

Figure 4: CF cluster effect: CELT vs. w/o Clu comparison. (a, b) Latent instance log-likelihoods for (a) y^{cf} and (b) y^{or} (x-axis: w/o Clu, y-axis: CELT). (c, d) 2D latent space visualizations by (c) CELT and (d) w/o Clu.

The number of components is chosen using the same procedure as for selecting the number of CF clusters K. Each GMM component uses spherical covariance. For clarity on the effect, in this section only, we evaluate supervised classification on the training data.

Figures 4 (a) and (b) scatter the log-likelihoods of latent instances for the desired class y^{cf} and the original class y^{or}, respectively. The x-axis uses "w/o Clu" and the y-axis uses CELT. In Fig. 4 (a), most points of CELT lie above the diagonal, indicating higher log-likelihoods for y^{cf}. In Fig. 4 (b), most points of CELT lie below the diagonal, indicating lower log-likelihoods for y^{or}. These results support that CELT learns CF clusters as intended.

Figures 4 (c) and (d) visualize the latent spaces learned by CELT and "w/o Clu" on ECGFiveDays dataset. For visualization, we learn a 2-D latent space ($\mathbb{R}^{2\times 1}$) by each method. Blue circles and red crosses represent latent instances for y^{cf} and y^{or}, respectively. Figure 4 (c) shows more clearly than Fig. 4 (d) that the latent instances of y^{cf} form compact clusters and are well-separated from the latent instances of y^{or}, indicating that CELT learns the intended latent structure via CF-cluster learning.

5.3 Evaluation of Validity

Figure 3 (b) shows the validity of each method on each test dataset. At a glance, validity of CELT tends to be better than that of the existing methods in both supervised and one-class classification. We then evaluate the average rank of each method in terms of validity across all 26 datasets. Figures 5 (a) and (b) show the critical difference (CD) diagrams for supervised and one-class classification, respectively. The values in parentheses indicate the average rank of the comparison methods, where a smaller value means better performance. Methods connected by a bold bar are not significantly different from each other at a 95% confidence level (Demšar, 2006). The CD values in (a) and (b) are 1.77 and 1.48.

From Fig. 5 (left), CELT is the best in both supervised and one-class classification and is significantly better than LASTS (standard VAE), MCCE (not tailored to time series), and MCCE++ (MCCE variant adapted to our experiments) at the 95% level. In supervised classification, CELT is better than CVAE (VAE with supervised contrastive loss), but the difference is not statistically significant. Figure 5 (left) further shows that adding CF-cluster learning and the masked AE (CELT) enhances validity compared with the ablations ("w/o Clu", "w/o Mask"). Under supervised classification, the gap between "w/o Clu" and CELT is large: CELT outperforms LASTS, MCCE, and MCCE++, whereas "w/o Clu" does not. These results indicate that the proposed method—especially the CF-cluster learning—improves validity.

5.4 Evaluation of Sparsity and Implausibility

We evaluate two metrics: sparsity and implausibility. If the counterfactual $\mathbf{x}^{\mathrm{cf}} \in \mathbb{R}^Q$ is identical to the original time-series instance $\mathbf{x}^{\mathrm{or}} \in \mathbb{R}^Q$, the metrics take their best values. However, $\mathbf{x}^{\mathrm{cf}} = \mathbf{x}^{\mathrm{or}}$ is not meaningful. Following (Yamaguchi et al., 2024), we thus compute these metrics only on valid CFs, i.e., test instances where the classifier f satisfies $f(\mathbf{x}^{\mathrm{cf}}) = y^{\mathrm{cf}} \neq y^{\mathrm{or}}$. For a fair comparison, we further restrict evaluation to the intersection of original instances on which both CELT and each existing method successfully generate valid CFs. This ensures that the set of original instances analyzed here is identical across all methods being compared.

Figure 5 (right) shows pairwise comparisons between CELT (y-axis) and each existing method (x-axis). Panels (c) and (d) report per-instance sparsity and implausibility, respectively. Labels (S) and (O) denote supervised and one-class classification. In the figure header, v1/v2/v3 indicate the number of instances where

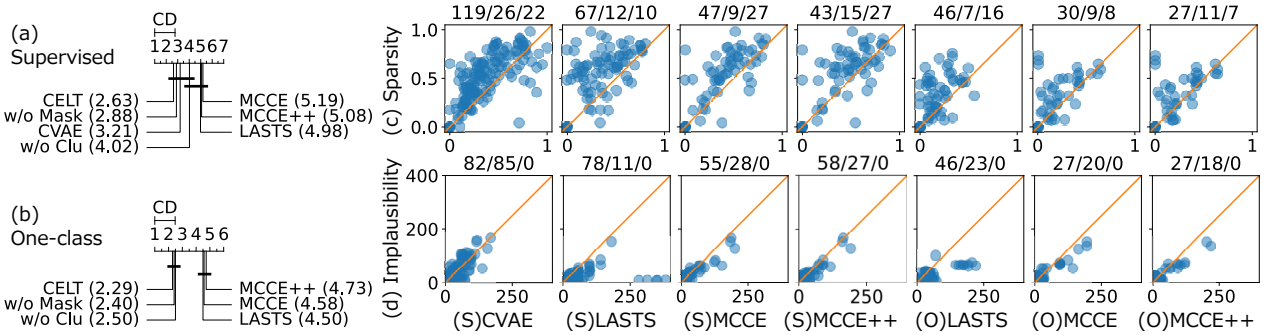

Figure 5: Left: CD diagrams for validity across the 26 datasets, comparing average ranks (lower is better). Right: Pairwise comparison of CELT (y-axis) vs. existing methods (x-axis) on (a) sparsity and (b) implausibility; (S) and (O) indicate supervised and one-class classification, and v1/v2/v3 are win/loss/draw counts.

CELT wins/loses/draws against the existing method.

In panel (c), most points lie above the diagonal and CELT's win count exceeds its loss count, indicating that CELT produces sparser CFs. Wilcoxon signed-rank tests confirm significance against every existing method ($p \leq 0.0068$). In panel (d), except for CVAE, many points lie below the diagonal with wins exceeding losses, showing lower implausibility for CELT; Wilcoxon tests are significant ($p \leq 0.018$). For CVAE, CFs sampled from the densest y^{cf} region in the latent space can be slightly more plausible than CELT's; however, the difference is not statistically significant ($p = 0.70$). In summary, CELT generates the sparsest CFs while maintaining competitive plausibility.

5.5 Evaluation of Runtime

Figure 3 (c) shows that CELT generates CFs at a practical speed during deployment, where speed matters, comparable to fast CVAE, with a worst-case of 1.08 minutes per instance. We report the supervised setting to enable a direct comparison with CVAE; the trend is similar in the one-class setting. MCCE and MCCE++ are fastest by pre-generating synthetic candidates during development; however, they can reduce CF quality on some time-series datasets (see §§ 5.3 and 5.4).

6 CASE STUDIES

We evaluate the CFs generated by CELT on three UCR datasets with known interpretable patterns. Because prior work mainly targets supervised classification and one-class studies on UCR datasets are limited, we focus on the one-class setting. We follow §5.1 for the rest of the setup. Figure 6 shows original time-series instances (blue) and their valid CFs (red).

ECGFiveDays is a dataset for classifying patient conditions from electrocardiogram (ECG) signals. Cardiologists report that the T-wave is the primary clinically salient difference between the classes (Keogh and Rakthanmanon, 2013). Prior interpretable TSC methods discover T-wave as shapelets in both supervised and one-class classification (Ye and Keogh, 2009; Yamaguchi and Nishikawa, 2018). Figure 6 (a) shows that CELT locates and locally edits the T-wave segment in the context of CF generation, as expected.

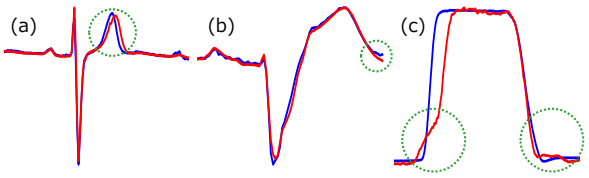

Figure 6: Original time series (blue) and their CFs (red) in (a) ECGFiveDays, (b) TwoLeadECG, and (c) GunPoint datasets. Circles mark discriminative patterns.

TwoLeadECG dataset contains ECG signals where the clinically salient difference between normal and abnormal cases is the QT interval (Nachimuthu et al., 2012). Prior work—a CF method limited to differentiable supervised classifiers (Wang et al., 2024) and a NN-based anomaly detector (Roy et al., 2024)—has identified this characteristic. Figure 6 (b) shows an abnormal ECG (blue) and its synthetic normal CF (red). The CF modifies the QT interval properly, consistent with prior findings (Roy et al., 2024; Wang et al., 2024).

GunPoint is a time-series dataset for classifying gun-drawing and finger-pointing motions. In the gun class, a bump appears when drawing from a holster, whereas in the pointing class an overshoot dip appears when lowering the arm (Ye and Keogh, 2009; Lines et al., 2012). Prior work on CF explanations for supervised classification—either model-specific (Karlsson et al., 2018, 2020) or restricted to differentiable classifiers (Ya-

maguchi et al., 2024)—has discovered these discriminative patterns. Figure 6 (c) shows an original pointing sequence (blue) and its CF gun sequence (red). The CF clearly captures both patterns by adding the bump and suppressing the overshoot dip.

7 RELATED WORK

This section and Table 1 summarize related work. The details are provided in the Appendix.

7.1 CFs for TSC

Existing methods fall into model-specific and model-agnostic groups. Model-specific methods (Karlsson et al., 2018, 2020; Yan and Wang, 2023) target particular classifiers and thus are difficult to apply to many TSC methods (Middlehurst et al., 2024b). Among model-agnostic methods, some construct CFs by using (a part of) training data $\mathcal{D}$ at deployment (Bahri et al., 2024; Ates et al., 2021; Delaney et al., 2021), making integration difficult in industries with strict data-access policies. The remaining model-agnostic methods can be grouped by whether they rely on the classifier input gradients. Gradient-based methods (Wang et al., 2021; Filali Boubrahimi and Hamdi, 2022; Wang et al., 2024; Yamaguchi et al., 2024) assume differentiability w.r.t the input and thus are difficult to apply to many TSC methods (Middlehurst et al., 2024b). Deng et al. (2024) removes this gradient dependence but targets block-maxima forecasting and is not applicable to TSC tasks.

In this study, we focus on model-agnostic CF generation without requiring classifier gradients or training data access at deployment. Redelmeier et al. (2024) can in principle be applied to time-series data, but it is designed for tabular data. Spinnato et al. (2023) lacks structural guidance and provides no explicit mechanism to control segment-wise, time-local edits. CVAE (Todo et al., 2023) introduces structure in the VAE latent space via a supervised contrastive loss and reports faster, higher-quality CF generation than prior methods (Wang et al., 2021; Ates et al., 2021). However, CVAE does not provide an explicit mechanism to control segment-wise, time-local edits.

Compared with Todo et al. (2023); Spinnato et al. (2023); Redelmeier et al. (2024), CELT formulates mathematically sound optimization and achieves the best validity and sparsity while keeping plausibility (see §§5.3 and §§5.4).

7.2 CFs for Anomaly Detection

Compared with supervised settings, only a few studies address CF explanations for unsupervised anomaly detection, and they have limitations. Ji et al. (2024) requires differentiable anomaly-score functions and assumes that the overall anomaly score aggregates feature-wise scores. Datta et al. (2022) targets tabular data with categorical variables and requires likelihood-based anomaly detectors. Haldar et al. (2021) restricts the classifier to AEs. Todo et al. (2023) was evaluated on (supervised) time-series anomaly detection for ECG signals, but it relies on supervised contrastive loss and therefore cannot be applied to one-class classification or unsupervised anomaly detection. In principle, Redelmeier et al. (2024); Spinnato et al. (2023) can be applied to one-class classification including unsupervised anomaly detection, but their CF quality is not sufficient (see §§5.3 and §§5.4).

CELT handles both supervised and one-class settings in a unified manner and generates higher-quality CFs.

Table 1: Comparison to related work. MA: model-agnostic, NG: no classifier-gradient requirement, ND: no training-data access at deployment, AT: applicability to TSC, SW: segment-wise CF edits.

CF method	MA	NG	ND	AT	SW
Karlsson et al. (2018)	×	–	×	○	○
Karlsson et al. (2020)	×	–	×	○	○
Yan and Wang (2023)	×	–	○	○	–
Bahri et al. (2024)	○	○	×	○	○
Ates et al. (2021)	○	○	×	○	×
Delaney et al. (2021) w/o CAM	○	○	×	○	×
Delaney et al. (2021) w/ CAM	△	○	×	○	○
Wang et al. (2021)	○	×	○	○	×
Filali Boubrahimi and Hamdi (2022)	○	×	○	○	–
Wang et al. (2024)	○	×	○	○	○
Yamaguchi et al. (2024)	○	×	○	○	○
Deng et al. (2024)	○	○	○	×	○
Ji et al. (2024)	△	×	○	○	○
Datta et al. (2022)	△	○	○	×	–
Haldar et al. (2021)	△	×	○	○	–
Todo et al. (2023) (supervised only)	○	○	○	○	×
Spinnato et al. (2023)	○	○	○	○	×
Redelmeier et al. (2024)	○	○	○	○	–
CELT (ours)	○	○	○	○	○

8 CONCLUSION

We proposed CELT, a model-agnostic counterfactual (CF) method that supports non-differentiable and one-class classifiers and does not require access to training data at deployment, for time series classification (TSC). CELT learns CF-clusters in the latent space during development and then generates high-quality CFs efficiently via minimal time-local edits at deployment. We formulated both phases as optimization problems, justified the core idea of the deployment phase algorithm, and demonstrated effectiveness on UCR datasets in terms of CF quality and interpretability, CF-cluster learning, and deployment runtime.

Acknowledgements

The research was the collaborative work of Toshiba Corporation and Kyushu University, based on funding from Toshiba Corporation.

References

Ates, E., Aksar, B., Leung, V. J., and Coskun, A. K. (2021). Counterfactual explanations for multivariate time series. In *ICAPAI*, pages 1–8. IEEE Computer Society.

Bahri, O., Li, P., Filali Boubrahimi, S., and Hamdi, S. M. (2024). Discord-based counterfactual explanations for time series classification. *Data Min. Knowl. Discov.*, 38(6):3347–3371.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: identifying density-based local outliers. In *SIGMOD*, pages 93–104. ACM.

Datta, D., Chen, F., and Ramakrishnan, N. (2022). Framing algorithmic recourse for anomaly detection. In *KDD*, pages 283–293. ACM.

Dau, H. A., Keogh, E., Kamgar, K., Yeh, Michael, C.-C., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., and Hexagon-ML (2018). The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

Delaney, E., Greene, D., and Keane, M. T. (2021). Instance-based counterfactual explanations for time series classification. In *ICCBR*, pages 32–47. Springer-Verlag.

Dempster, A., Schmidt, D. F., and Webb, G. I. (2024). Quant: a minimalist interval method for time series classification. *Data Min. Knowl. Discov.*, 38(4):2377–2402.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

Deng, Y., Galib, A. H., Tan, P.-N., and Luo, L. (2024). Unraveling block maxima forecasting models with counterfactual explanation. In *KDD*, pages 562–573. ACM.

Dong, J., Wu, H., Zhang, H., Zhang, L., Wang, J., and Long, M. (2023). Simmtm: a simple pre-training framework for masked time-series modeling. In *NeurIPS*, pages 29996–30025. Curran Associates Inc.

European Parliament and the Council of the European Union (2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). Official Journal of the European Union, L 119. https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng.

Filali Boubrahimi, S. and Hamdi, S. M. (2022). On the mining of time series data counterfactual explanations using barycenters. In *CIKM*, pages 3943–3947. ACM.

Franceschi, J.-Y., Dieuleveut, A., and Jaggi, M. (2019). Unsupervised scalable representation learning for multivariate time series. In *NeurIPS*, pages 4650–4661. Curran Associates Inc.

Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2014). Learning Time-series Shapelets. In *KDD*, pages 392–401. ACM.

Guidotti, R. (2022). Counterfactual explanations and how to find them: literature review and benchmarking. *Data Min. Knowl. Discov.*, 36(6):1–55.

Guo, H., Nguyen, T. H., and Yadav, A. (2023). Counternet: End-to-end training of prediction aware counterfactual explanations. In *KDD*, pages 577–589. ACM.

Haldar, S., John, P. G., and Saha, D. (2021). Reliable counterfactual explanations for autoencoder based anomalies. In *IKDD CODS & COMAD*, pages 83–91. ACM.

Hamerly, G. and Elkan, C. (2003). Learning the k in k-means. In *NIPS*, pages 281–288. MIT Press.

Han, S., Hu, X., Huang, H., Jiang, M., and Zhao, Y. (2022). Adbench: anomaly detection benchmark. In *NeurIPS*, pages 32142–32159. Curran Associates Inc.

Hojjati, H. and Armanfard, N. (2024). Dasvdd: Deep autoencoding support vector data descriptor for anomaly detection. *IEEE Trans. on Knowl. and Data Eng.*, 36(8):3739–3750.

Ji, X., Xue, A., Wong, E., Sokolsky, O., and Lee, I. (2024). Ar-pro: Counterfactual explanations for anomaly repair with formal properties. In *NeurIPS*, pages 16133–16159. Curran Associates Inc.

Kan, Z. et al. (2024). Benchmarking counterfactual interpretability in deep learning models for time series classification. *arXiv preprint arXiv:2408.12666*.

Karlsson, I., Rebane, J., Papapetrou, P., and Gionis, A. (2018). Explainable time series tweaking via irreversible and reversible temporal transformations. In *ICDM*, pages 207–216. IEEE Computer Society.

Karlsson, I., Rebane, J., Papapetrou, P., and Gionis, A. (2020). Locally and globally explainable time series tweaking. *Knowl. Inf. Syst.*, 62(5):1671–1700.

Kendall, A., Gal, Y., and Cipolla, R. (2018). Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, pages 7482–7491. IEEE Computer Society.

Keogh, E. and Rakthanmanon, T. (2013). Fast shapelets: A scalable algorithm for discovering time series shapelets. In *SDM*, pages 668–676. SIAM.

Lines, J., Davis, L. M., Hills, J., and Bagnall, A. (2012). A shapelet transform for time series classification. In *KDD*, pages 289–297. ACM.

Middlehurst, M., Ismail-Fawaz, A., Guillaume, A., Holder, C., Guijo-Rubio, D., Bulatova, G., Tsaprounis, L., Mentel, L., Walter, M., Schäfer, P., and Bagnall, A. (2024a). aeon: a python toolkit for learning from time series. *J. Mach. Learn. Res.*, 25(1).

Middlehurst, M., Schäfer, P., and Bagnall, A. (2024b). Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Min. Knowl. Discov.*, 38(4):1958–2031.

Nachimuthu, S., Assar, M. D., and Schussler, J. M. (2012). Drug-induced qt interval prolongation: mechanisms and clinical management. *Therapeutic Advances in Drug Safety*, 3(5):241–253.

Office of the Federal Register, National Archives and Records Administration (2025). Other requirements relating to uses and disclosures of protected health information – minimum necessary. Electronic Code of Federal Regulations (eCFR), Title 45, Part 164, Subpart E. https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164/subpart-E/section-164.514.

Pawelczyk, M., Broelemann, K., and Kasneci, G. (2020). Learning model-agnostic counterfactual explanations for tabular data. In *WWW*, pages 3126–3132. ACM.

Redelmeier, A., Jullum, M., Aas, K., and Løland, A. (2024). Mcce: Monte carlo sampling of valid and realistic counterfactual explanations for tabular data. *Data Min. Knowl. Discov.*, 38(4):1830–1861.

Roy, M., Halder, A., Majumder, S., and Biswas, U. (2024). Attentivecgru: Gru based autoencoder with attention mechanism and automated fuzzy thresholding for ecg arrhythmia detection. *Applied Soft Computing*, 167:33.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. (2018). Deep one-class classification. In *ICML*, volume 80, pages 4393–4402. PMLR.

Spinnato, F., Guidotti, R., Monreale, A., Nanni, M., Pedreschi, D., and Giannotti, F. (2023). Understanding any time series classifier with a subsequence-based explainer. *ACM Trans. Knowl. Discov. Data*, 18(2).

Todo, W., Selmani, M., Laurent, B., and Loubes, J. (2023). Counterfactual explanation for multivariate times series using a contrastive variational autoencoder. In *ICASSP*, pages 1–5. IEEE Computer Society.

U.S. Department of Health & Human Services (2013). Minimum necessary requirement. https://www.hhs.gov/hipaa/for-professionals/privacy/guidance/minimum-necessary-requirement/index.html.

Wachter, S., Mittelstadt, B., and Russell, C. (2018). Counterfactual explanations without opening the black box: automated decisions and the gdpr. *Harvard Journal of Law and Technology*, 31(2):841–887.

Wang, Q. and Palpanas, T. (2021). Deep learning embeddings for data series similarity search. In *KDD*, pages 1708–1716. ACM.

Wang, Z., Samsten, I., Miliou, I., Mochaourab, R., and Papapetrou, P. (2024). Glacier: guided locally constrained counterfactual explanations for time series classification. *Machine Learning*, 113(3):1–31.

Wang, Z., Samsten, I., Mochaourab, R., and Papapetrou, P. (2021). Learning time series counterfactuals via latent space representations. In *Discovery Science*, pages 369–384. Springer-Verlag.

Yamaguchi, A. and Nishikawa, T. (2018). One-class learning time-series shapelets. In *Big Data*, pages 2365–2372. IEEE Computer Society.

Yamaguchi, A., Ueno, K., Shingaki, R., and Kashima, H. (2024). Learning counterfactual explanations with intervals for time-series classification. In *CIKM*, pages 4158–4162. ACM.

Yamaguchi, A., Ueno, K., Uchida, K., Matsumoto, E., and Saida, T. (2022). Development of advanced ai technologies for condition diagnosis of high voltage switchgear in substations. *CIGRE Science and Engineering*, CSE 026:1–11.

Yan, J. and Wang, H. (2023). Self-interpretable time series prediction with counterfactual explanations. In *ICML*, volume 202, pages 39110–39125. PMLR.

Yang, F., Liu, N., Du, M., and Hu, X. (2021). Generative counterfactuals for neural networks via attribute-informed perturbation. *SIGKDD Explor. Newsl.*, 23(1):59–68.

Ye, L. and Keogh, E. (2009). Time series shapelets: A new primitive for data mining. In *KDD*, pages 947–956. ACM.

Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., and Eickhoff, C. (2021). A transformer-based framework for multivariate time series representation learning. In *KDD*, pages 2114–2124. ACM.

Zhang, Q., Wu, J., Zhang, P., Long, G., and Zhang, C. (2018). Salient subsequence learning for time series

clustering. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 2193–2207.

Zhao, Y., Nasrullah, Z., and Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. **Yes**
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. **Yes. Specifically, for each method we report the average runtime at deployment where speed is required.**
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. **Yes**

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. **Yes**
   (b) Complete proofs of all theoretical results. **Yes**
   (c) Clear explanations of any assumptions. **Yes**

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). **No**
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). **Yes**
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). **Yes**
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). **Yes**

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. **Yes**
   (b) The license information of the assets, if applicable. **Yes**
   (c) New assets either in the supplemental material or as a URL, if applicable. **Yes**
   (d) Information about consent from data providers/curators. **Not Applicable**
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. **Not Applicable**

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. **Not Applicable**
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. **Not Applicable**
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. **Not Applicable**

Supplementary Material for "Counterfactual Explanations via Latent Structure for Time Series Classification"

A RELATED WORK DETAILS

This section reviews related work and contrasts it with the proposed method (CELT) in two research areas: counterfactual (CF) explanations for time series classification (TSC) and anomaly detection.

A.1 Counterfactual Explanations for Time Series Classification

CFs for time series can assist domain experts and practitioners, but the literature remains limited. Existing methods fall into two groups: model-specific and model-agnostic. Model-specific approaches (Yan and Wang, 2023; Karlsson et al., 2018, 2020) target particular classifier families and therefore fail to cover many state-of-the-art (SOTA) TSC classifiers (Middlehurst et al., 2024b). Among model-agnostic methods, some require access to training data at deployment (e.g., nearest-unlike retrieval and segment substitution) (Bahri et al., 2024; Ates et al., 2021; Delaney et al., 2021). This requirement may conflict with data-minimization policies in real deployments (European Parliament and the Council of the European Union, 2016; Office of the Federal Register, National Archives and Records Administration, 2025; U.S. Department of Health & Human Services, 2013).[1] The remaining model-agnostic methods can be grouped by whether they rely on classifier input gradients $\partial f/\partial \mathbf{x}$. Many methods rely on classifier gradients ($\partial f/\partial \mathbf{x}$) (Yamaguchi et al., 2024; Filali Boubrahimi and Hamdi, 2022; Wang et al., 2021, 2024), thereby presupposing differentiability with respect to the input and thus excluding many high-performing non-neural or hybrid TSC classifiers (Middlehurst et al., 2024b). Deng et al. (2024) remove this dependence, but their method is tailored to block-maxima forecasting and is not applicable to TSC tasks.

This study focuses on the remaining category of model-agnostic CF generation methods that require neither differentiability nor access to training data at deployment. This category covers many widely used, accurate, and fast TSC classifiers (Middlehurst et al., 2024b). MCCE (Redelmeier et al., 2024) targets tabular data. Its modeling is sufficiently general to be applied to TSC, but its design is not tailored to time series. LASTS (Spinnato et al., 2023) builds on prior work for tabular data (Pawelczyk et al., 2020) and employs a CNN-based VAE tailored to TSC. However, it lacks structural guidance in the latent space for efficiently generating high-quality CFs and provides no explicit mechanism to control segment-wise, time-local edits. CVAE (Todo et al., 2023) is, to our knowledge, the most similar to CELT. It introduces structure in the VAE latent space via a supervised contrastive loss to facilitate CF generation and reports faster, higher-quality CF generation than the prior methods (Wang et al., 2021; Ates et al., 2021). Although CVAE employs kernel density estimation to identify CFs in the latent space, this approach entails an exhaustive grid search, resulting in a trade-off between grid accuracy and efficiency. Moreover, CVAE does not provide an explicit mechanism to control segment-wise, time-local edits in the original time series, despite their importance for time-series CFs.

Table 2 summarizes the comparison. Unlike the above methods, CELT generates high-quality CFs without requiring differentiability or deployment-time access to training data, through mathematically sound formulations. Like CELT, the existing methods (CVAE, LASTS, and MCCE) are model-agnostic, work with non-differentiable time-series classifiers, and do not access training data at deployment. The key difference is that CELT achieves the best sparsity (with statistically significant gains), the numerically highest validity (though not statistically different from CVAE), and competitive plausibility, as demonstrated in Sections 5.3 and 5.4.

[1] CELT generates CFs as synthetic time-series instances without accessing training data at deployment, which can alleviate such concerns, although it does not guarantee legal compliance by itself.

Method	model-agnostic	no classifier-gradient requirement	no training-data access at deployment	supports TSC	segment-wise CF edits
Yan and Wang (2023)	No	N/A	Yes	Yes	N/A
Karlsson et al. (2018)	No	N/A	No	Yes	Yes
Karlsson et al. (2020)	No	N/A	No	Yes	Yes
Bahri et al. (2024)	Yes	Yes	No	Yes	Yes
Ates et al. (2021)	Yes	Yes	No	Yes	No
Delaney et al. (2021) w/o CAM	Yes	Yes	No	Yes	No
Delaney et al. (2021) w/ CAM	Limited	Yes	No	Yes	Yes
Yamaguchi et al. (2024)	Yes	No	Yes	Yes	Yes
Filali Boubrahimi and Hamdi (2022)	Yes	No	Yes	Yes	N/A
Wang et al. (2021)	Yes	No	Yes	Yes	No
Wang et al. (2024)	Yes	No	Yes	Yes	Yes
Deng et al. (2024)	Yes	Yes	Yes	No	Yes
Todo et al. (2023)	Yes	Yes	Yes	Yes	No
Redelmeier et al. (2024)	Yes	Yes	Yes	Yes	N/A
Spinnato et al. (2023)	Yes	Yes	Yes	Yes	No
CELT (ours)	Yes	Yes	Yes	Yes	Yes

Table 2: Comparison of CF explanation methods for TSC across five aspects. Only methods that are model-agnostic, require no classifier gradients, require no access to training data at deployment, and support TSC can utilize a wide range of SOTA time-series classifiers (Middlehurst et al., 2024b). This design may be compatible with data-minimization practices in some deployments; we do not assess legal compliance in this paper.

A.2 Counterfactual Explanations for Anomaly Detection

Most work on CF explanations focuses on supervised settings; only a few studies address unsupervised anomaly detection. Existing methods have narrow applicability. Ji et al. (2024) require differentiable anomaly-score functions (i.e., the classifier in our setting) and assume that the overall anomaly score aggregates feature-wise scores. Datta et al. (2022) target tabular data with categorical features and require a likelihood-based detector. Haldar et al. (2021) restrict classifiers to AE-based models. Although CVAE (Todo et al., 2023) considers time-series anomaly detection as an application, it uses a supervised contrastive loss and is therefore inapplicable to one-class classification or unsupervised anomaly detection. Although MCCE (Redelmeier et al., 2024) and LASTS (Spinnato et al., 2023) are, in principle, applicable to unsupervised anomaly detection, the resulting CF quality is insufficient for time-series one-class classification.

Table 3 summarizes the comparison. CELT provides a unified formulation for both supervised and one-class TSC. Compared to MCCE and LASTS, CELT generates high-quality CFs for non-differentiable one-class time-series classifiers, as demonstrated in Sections 5.3 and 5.4.

Method	model-agnostic	no classifier-gradient requirement	supports one-class (OC)	supports time-series	segment-wise CF edits
Ji et al. (2024)	Limited (linearly decomposable)	No	Yes	Yes	Yes
Datta et al. (2022)	Limited (likelihood-based)	Yes	Yes	No	N/A
Haldar et al. (2021)	Limited (autoencoder-based)	No	Yes	Yes	N/A
Todo et al. (2023)	Yes	Yes	No	Yes	No
Redelmeier et al. (2024)	Yes	Yes	Yes	Yes	N/A
Spinnato et al. (2023)	Yes	Yes	Yes	Yes	No
CELT (ours)	Yes	Yes	Yes	Yes	Yes

Table 3: Comparison of CF explanation methods for anomaly detection across five aspects. Only methods that are model-agnostic, do not require classifier gradients, support one-class classification, and support time-series data can be applied to one-class classifiers based on SOTA time-series classification methods (Middlehurst et al., 2024b).

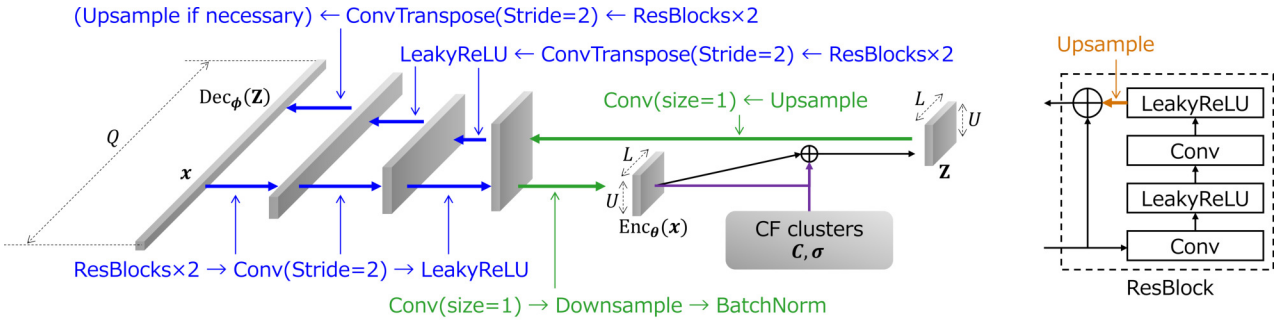

Figure 7: Convolutional autoencoder architecture preserving temporal order along the length (temporal) axis L. To focus on the autoencoder architecture, this figure does not include the derivation of the cluster loss (Section 3.3) or the masking procedure used in the reconstruction loss (Section 3.4).

B IMPLEMENTATION DETAILS

This section complements Section 3 with implementation details for learning the latent structure.

We use an $L\times U$-dimensional latent space with L=16 and U=20 for all experiments. We optimize Eq. (8) with the Adam optimizer using a batch size of 64 in all experiments. Unless otherwise noted, we use PyTorch defaults. In the following subsections, we describe the autoencoder design and the hyperparameter settings for the cluster loss.

B.1 Autoencoder Architecture

Figure 7 shows the autoencoder architecture introduced in Section 3. It consists of a convolutional encoder and a symmetric decoder. All convolutions are 1D with no padding or dilation (hereafter denoted Conv). Unless noted, the kernel size is 3 and the stride is 1. Importantly, unlike previous architectures (Franceschi et al., 2019; Wang and Palpanas, 2021; Todo et al., 2023), we omit global pooling, dilation, and padding. This design preserves the sequential order and locality of the time series in every layer's output.

The encoder and decoder use the ResBlocks in Fig. 7 (right). Unlike previous architectures (Franceschi et al., 2019; Wang and Palpanas, 2021), which learn time-series latent representations with ResNet-style blocks that rely on padding and dilation, each of our ResBlocks applies Conv → LeakyReLU → Conv → LeakyReLU → Upsampling, and thereby the block's input and output lengths are aligned without dilation or padding. The number of input and output channels for the Conv is the same in each ResBlock.

Encoder. Following the diagram (blue/green right arrows), the encoder stages are: [ResBlocks ×2] → Conv (stride =2) → LeakyReLU. This progressively halves the sequence length and increases channels from $1\to50\to100\to200$, performing local aggregation. The final-stage output then passes through 1×1 Conv → Downsampling → BatchNorm to produce Enc$_{\boldsymbol{\theta}}(\cdot)$ with U channels and length L. We apply BatchNorm to reduce scale imbalance across LU dimensions.

Latent sampling (reparametrization). Let $\mathbf{x}\in\mathbb{R}^Q$ be an input time-series instance. At the module indicated by the purple arrows in Fig. 7, given the latent mean Enc$_{\boldsymbol{\theta}}(\mathbf{x})\in\mathbb{R}^{L\times U}$ from the encoder and the per-instance latent variance $\tilde{\sigma}^2$ from Eq. (5), we apply the reparametrization trick to sample a latent instance $\mathbf{Z}\in\mathbb{R}^{L\times U}$.

Decoder. Following the diagram (green/blue left arrows), the decoder applies Upsampling → 1×1 Conv to $\mathbf{Z}$ to match the size of the final encoder convolution map. The decoder then repeats [ResBlocks ×2] → transposed convolution (stride = 2) → LeakyReLU three times, which progressively upsamples the sequence and changes channels 200 → 100 → 50 → 1. The final layer uses no activation (no LeakyReLU). Depending on Q, the reconstruction may be shorter than Q because of stride-2 operations; therefore, we apply a final upsampling step so that the output exactly matches Q, producing Dec$_{\boldsymbol{\phi}}(\mathbf{Z})\in\mathbb{R}^Q$.

Although repeated convolutions may slightly blur locality, Section 5.4 shows quantitatively that, compared with methods that do not preserve locality, our architecture achieves higher sparsity in CF generation.

B.2 Hyperparameter Setting for Cluster Loss

This section explains how we set the hyperparameters of the cluster loss $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ introduced in Section 3: the number of CF clusters $K \in \mathbb{N}$ and the margin $M \in \mathbb{R}_+$ in Eq. (6). Because exhaustive search per dataset is costly, we choose K and M automatically using a simple heuristic.

We first pre-train the autoencoder with only the reconstruction loss $\mathcal{L}^{(\mathrm{Rec})}_{\boldsymbol{\theta},\boldsymbol{\phi},\mathcal{C},\boldsymbol{\sigma}}$, temporarily omitting $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$. After the latent space stabilizes, we determine K and M from the pre-trained latent instances. We then add $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ and fine-tune the latent structure by optimizing the full objective in Eq. (8).

For pre-training, we optimize Eq. (8) for the first 1,000 epochs with $\tilde{\sigma}_i = 1$, $\lambda_1 = 0$, and $\lambda_2 = 1$ in Eqs. (5) and (8) for all $i \in [I]$. In fine-tuning, we learn all optimization parameters—including the weighting parameters λ_1 and λ_2 in Eq. (8)—for the remaining 1,000 epochs, using the hyperparameters K and M identified as follows.

We estimate the hyperparameters K and M from the latent instances immediately after pre-training.

Estimating the number of CF clusters K. We use the G-means algorithm (Hamerly and Elkan, 2003) to infer the number of CF clusters K. G-means tests whether the points assigned to each tentative center are normally distributed at the 0.01 significance level. If the test rejects normality, the cluster is split. We also use the centroids of the resulting clusters to initialize the CF cluster centroids.

Estimating the margin M. Given the G-means centroid $\mathbf{C}_k \in \mathbb{R}^{L \times U}$ for $k \in [K]$, we set the margin $M \in \mathbb{R}_+$ in Eq. (6) as

$$M = \tfrac{1}{2} \min_{k_1 \neq k_2} \left\| \mathbf{C}_{k_1} - \mathbf{C}_{k_2} \right\|_F, \qquad k_1, k_2 \in [K],$$

i.e., half of the minimum inter-centroid distance. This choice provides a conservative, safe upper bound on the per-cluster "radius" in the latent space where pre-training has just finished: balls of radius M around different centroids do not overlap.

Despite being heuristic, these settings yield high-quality CFs without additional tuning (Section 5).

C ADDITIONAL THEORETICAL CHARACTERIZATION

This section provides additional details on Proposition 1 in Section 4.3 and on the separability of the learned CF clusters in Section 3.3. The former clarifies why the greedy row-selection strategy in Algorithm 1 is well motivated for minimizing the number of edited rows in Eq. (10). The latter discusses how the cluster loss $\mathcal{L}^{(\mathrm{Clu})}_{\boldsymbol{\theta},\mathcal{C},\boldsymbol{\sigma}}$ in Section 3.3 promotes separation between the desired class $y^{\mathrm{cf}} \in \mathcal{Y}$ and the other classes, at least on the training data. Although these statements do not provide a complete theoretical guarantee for the entire method, they serve as supplementary justification for the main design choices of the proposed approach.

C.1 Another Proof of Proposition 1

While we provided the proof of Proposition 1 in Section 4.3, we can also prove Proposition 1 through two standard properties of greedy methods: the greedy choice property and optimal substructure. For each row $l \in [L]$, define

$$d_l \coloneqq \|\mathbf{z}^{\mathrm{or}}_l - \mathbf{c}_{k^*,l}\|_2^2,$$

which is the contribution of the l-th row of the original latent instance $\mathbf{Z}^{\mathrm{or}}$ to the distance from the target centroid $\mathbf{C}_{k^*}$. Algorithm 1 edits rows in descending order of d_l. If only one row can be edited, then the largest one-step decrease of $\|\mathbf{Z} - \mathbf{C}_{k^*}\|_F^2$ is achieved by replacing the row $\mathbf{z}^{\mathrm{or}}_l$ with the largest d_l by $\mathbf{c}_{k^*,l}$. Therefore, among all optimal solutions that satisfy Eq. (10) with the minimum number of edited rows, there always exists one that includes the row with the largest d_l. This is the greedy choice property. Next, once the row with the largest d_l is edited first, the remaining task is again to choose the minimum number of rows among the remaining ones so that the residual distance becomes at most $\beta\sigma_{k^*}$. That is, after fixing the first edited row, the objective is still the number of edited rows, and the constraint is still of the same form, only with a reduced residual. Therefore, if we solve this remaining subproblem optimally and combine it with the first greedy edit, we obtain an optimal solution to the original problem. This is the optimal substructure property. Consequently, Algorithm 1 without the optional classifier check returns an optimal solution for Eq. (10).

C.2 CF Cluster Separability

Based on the margin M defined in Section B.2, if the loss in the lower term of Eq. (6) is reduced to zero and $\alpha \to \infty$ in Eq. (4), we can partially characterize the separability on the training data. Let $\mathbf{C}_k \in \mathbb{R}^{L \times U}$ be the cluster centroid obtained by G-means for $k \in [K]$. Then, in the latent space immediately after pre-training, each centroid $\mathbf{C}_k$ forms a ball $B(\mathbf{C}_k, M)$ of radius M, and these balls do not intersect. Furthermore, when the lower term in the cluster loss in Eq. (6) for $f(\mathbf{x}_i) \neq y^{\mathrm{cf}}$ is minimized to zero, all training instances of the non-desired class are excluded from any $B(\mathbf{C}_k, M)$. That is, the CF clusters of radius M completely separate the desired class y^{cf} from all others in the training data. Moreover, in the upper term of Eq. (6) for $f(\mathbf{x}_i) = y^{\mathrm{cf}}$, the KL divergence is minimized, and its gradients w.r.t the latent mean $\boldsymbol{\mu}_i$ are $(\boldsymbol{\mu}_i - \mathbf{C}_k)$, and thus during training, it pulls the latent mean $\boldsymbol{\mu}_i$ of the training instance $\mathbf{x}_i$ where $f(\mathbf{x}_i) = y^{\mathrm{cf}}$ towards the corresponding cluster centroid $\mathbf{C}_k$, thereby improving separability.

D EXPERIMENTAL DETAILS

This section complements Section 5 with additional experimental details. We execute all experiments on an Amazon AWS g5.12xlarge instance running Ubuntu OS with four GPUs and 192 GiB of RAM. For context, please read Section 5 first.

D.1 Details about Experimental Setup

This section adds setup details for Section 5.1: time-series classifiers, existing methods to compare, and UCR datasets.

Pre-trained classifiers. In supervised classification, we use `QUANTClassifier` from the aeon library (Middlehurst et al., 2024a). In one-class classification, we use `QUANTTransformer` to produce the same features as `QUANTClassifier`, and then apply `LOF` from the PyOD library (Zhao et al., 2019). We use default parameters with aeon v0.9.0 and PyOD v1.1.2. Please see Section D.4 for additional classifiers.

Existing methods. We use the source code for CVAE (Todo et al., 2023), LASTS (Spinnato et al., 2023), and MCCE (Redelmeier et al., 2024) from their official repositories. We use default parameters for all methods unless noted. Since the growing-sphere search in LASTS can run indefinitely, we set a 10-minute timeout for each generation. We use the UCR archive datasets, where time-series instances are pre-split, whereas Todo et al. (2023) targeted long time-series data from PTB-XL. Consequently, we do not perform random cropping—the preprocessing step used by CVAE. In addition, CVAE may fail to operate correctly for certain time-series lengths (e.g., odd Q). Therefore, following the CELT architecture, after the final transposed-convolution step we apply upsampling, if needed, so that the output matches the input sequence length Q. In each method, all features (all timestamps) of the time series are assumed actionable for CF generation.

Datasets. We select 26 datasets from the UCR archive and use only two classes: class 1 (y^{cf}) and class 2 (y^{or}). We include a dataset only if (i) each of the y^{cf} and y^{or} classes has at least 30 training instances, (ii) there are no missing values, (iii) all time series have the same length, (iv) training a CVAE model on the dataset does not trigger GPU out-of-memory errors, and (v) the test accuracy is at least 50% in both the one-class and the supervised classification settings. The selected datasets are listed in Tables 14–23. Further details for each dataset are available on the UCR archive web page (Dau et al., 2018).

D.2 Details about Experimental Results

This section adds details that support the evaluations of validity, sparsity, and implausibility in Section 5.3 and Section 5.4.

For validity, we first conducted the Friedman test before constructing the CD diagrams in Section 5.3. The p-values are 2.3×10^{-6} (supervised setting) and 8.9×10^{-9} (one-class setting), indicating statistically significant differences among methods. As a post-hoc step, we used the Nemenyi test: the critical differences are 1.77 (supervised setting) and 1.48 (one-class setting), matching the CD values used in Fig. 5 (left).

As noted in Section 5.4, we evaluated sparsity and implausibility on the intersection of instances for which both CELT and each existing method generated valid CFs, and conducted Wilcoxon signed-rank tests. Tables 4 and 5 report the p-values for sparsity and implausibility against each method, respectively. For sparsity, CELT

Table 4: Sparsity: Wilcoxon signed-rank tests between CELT and each existing method. CELT consistently outperforms all existing methods.

Existing method	Supervised		One-class	
	p-value	Significant ($\alpha=0.05$)	p-value	Significant ($\alpha=0.05$)
CVAE	2.9×10^{-14}	Yes	—	—
LASTS	2.2×10^{-10}	Yes	4.3×10^{-7}	Yes
MCCE	1.0×10^{-6}	Yes	2.9×10^{-4}	Yes
MCCE++	6.4×10^{-5}	Yes	0.0068	Yes

Table 5: Implausibility: Wilcoxon signed-rank tests between CELT and each existing method. CELT is a little worse than CVAE (no significant difference) in the supervised setting, but outperforms the remaining existing methods.

Existing method	Supervised		One-class	
	p-value	Significant ($\alpha=0.05$)	p-value	Significant ($\alpha=0.05$)
CVAE	0.70	No	—	—
LASTS	7.2×10^{-14}	Yes	1.7×10^{-5}	Yes
MCCE	1.5×10^{-3}	Yes	0.014	Yes
MCCE++	2.1×10^{-4}	Yes	0.018	Yes

consistently outperforms all existing methods in both the supervised and one-class settings. For implausibility, CELT is worse than CVAE in the supervised setting, but the difference is not significant; against the remaining methods, CELT consistently performs better.

Tables 14 and 15 report per-dataset validity. Tables 16 and 17 report per-dataset average sparsity. Tables 18 and 19 report per-dataset average implausibility. Each table pair corresponds to the supervised and one-class settings, respectively. The "±" values denote standard deviations. As noted in Section 5.4, sparsity and implausibility are computed only over valid CFs. If a method yields no valid CFs on a dataset, we mark "—" and assign the worst rank. In all tables, the number in parentheses is the method's rank (1 is best). We also report the mean rank and the number of wins at the bottom of each table.

Tables 14 and 15 directly support Fig. 5 (left) (Section 5.3). By contrast, Tables 16–19 use a protocol that differs from Fig. 5 (right) (Section 5.4): the tables evaluate each method on its own instance set, whereas Fig. 5 (right) evaluates on the intersection of instances on which both CELT and each existing method produce valid CFs, ensuring a like-for-like comparison. For a fairer comparison, please refer to Fig. 5 (right) (Section 5.4). Nevertheless, the mean rank and the number of wins at the bottom of each table remain consistent with Section 5.3 and Section 5.4—not only for validity but also for sparsity and implausibility.

D.3 Additional Metrics

In addition to validity, sparsity, and implausibility, which were mentioned in Section 2, this section adds two more metrics. The first one is the number of modified segments (*NumSeg*) metric, which is introduced in the Segment sparsity section of Kan et al. (2024). NumSeg counts the number of time-series segments that are modified from the original time series, and together with sparsity, it can be used to evaluate the segment-wise locality of CFs. The second metric is the Mean Absolute Difference (*MAD*) between the original time series and the CFs. Unlike sparsity, MAD is threshold-free.

Figure 8 summarizes pairwise comparisons between CELT (y-axis) and each existing method (x-axis). The top and bottom rows report per-instance NumSeg and MAD, respectively. Labels (S) and (O) denote supervised and one-class classification settings, respectively. In the figure header, v1/v2/v3 indicate the number of instances where CELT wins/loses/draws against the existing method. Other experimental settings are the same as in Sections 5.1 and 5.4.

As in Section D.2, we conducted Wilcoxon signed-rank tests. Tables 6 and 7 report the p-values for NumSeg and MAD against each method, respectively. For NumSeg, CELT is worse than LASTS in the one-class setting, but the difference is not significant; against the remaining methods, CELT consistently performs better although the

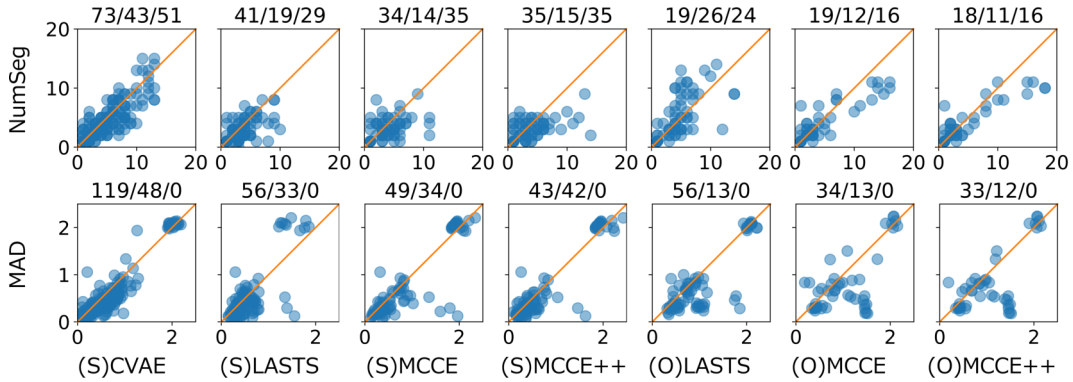

Figure 8: Pairwise comparison of CELT (y-axis) vs. existing methods (x-axis) on (top) NumSeg and (bottom) MAD; (S) and (O) indicate supervised and one-class classification, and v1/v2/v3 are win/loss/draw counts.

Table 6: NumSeg: Wilcoxon signed-rank tests between CELT and each existing method. CELT consistently outperforms all existing methods in the supervised setting although the difference is not significant in the one-class setting.

Existing method	Supervised		One-class	
	p-value	Significant ($\alpha=0.05$)	p-value	Significant ($\alpha=0.05$)
CVAE	0.00083	Yes	—	—
LASTS	0.00079	Yes	0.093	No
MCCE	0.0048	Yes	0.070	No
MCCE++	0.0029	Yes	0.057	No

difference is not significant in the one-class setting. For MAD, CELT is better than all existing methods although there is no significant difference compared with MCCE and MCCE++ in the supervised setting.

Tables 20 and 21 report per-dataset NumSeg. Tables 22 and 23 report per-dataset MAD. Each table pair corresponds to the supervised and one-class settings, respectively. The "±" values denote standard deviations. As in Section 5.4, NumSeg and MAD are computed only over valid CFs. If a method yields no valid CFs on a dataset, we mark "—" and assign the worst rank. In all tables, the number in parentheses is the method's rank (1 is best). We also report the mean rank and the number of wins at the bottom of each table.

Tables 20–23 use a protocol that differs from Fig. 8. The tables evaluate each method on its own instance set, whereas Fig. 8 evaluates on the intersection of instances on which both CELT and each existing method produce valid CFs, ensuring a like-for-like comparison. For a fairer comparison, please refer to Fig. 8, as in sparsity and implausibility.

D.4 Experiments on Additional Classifiers

To further support the wide applicability of the proposed method, we conducted additional experiments on the three representative datasets: ECGFiveDays, GunPoint, and TwoLeadECG. In these experiments, we add the following two gradient-free black-box classifiers using default parameters with aeon v0.9.0 and PyOD v1.1.2:

- **Supervised setting:** `Catch22Classifier`, which is fast and reasonably accurate.
- **One-class setting:** `catch22+IForest`, which applies Isolation Forest (IForest) on the catch22 features.

Tables 8–10 compare the Rank Mean of Validity, Sparsity, and Implausibility. We observe similar trends between `QUANTClassifier` and `Catch22Classifier`, and between `QUANT+LOF` and `catch22+IForest`. Furthermore, Tables 11–13 report the dataset-wise values of Validity, Sparsity, and Implausibility in the supervised setting for `Catch22Classifier` (we observed similar trends also in the one-class setting). We can see that even when we replace the classifiers, the actual metric values exhibit similar behavior in CELT and CVAE, by comparing with Tables 14, 16, and 18. Overall, these additional experiments suggest that the proposed method remains effective across different classifier choices.

Table 7: MAD: Wilcoxon signed-rank tests between CELT and each existing method. CELT is better than all existing methods although there is no significant difference compared with some of them.

Existing method	Supervised		One-class	
	p-value	Significant ($\alpha=0.05$)	p-value	Significant ($\alpha=0.05$)
CVAE	2.8×10^{-10}	Yes	—	—
LASTS	0.017	Yes	2.3×10^{-7}	Yes
MCCE	0.090	No	0.0015	Yes
MCCE++	0.31	No	0.0039	Yes

Table 8: Rank Mean for Validity.

Method	CELT	CVAE	LASTS	MCCE
`QUANTClassifier`	1.50	2.33	2.83	3.33
`Catch22Classifier`	1.17	2.00	2.83	4.00
`QUANT+LOF`	1.00	N/A	2.00	3.00
`catch22+IForest`	1.00	N/A	2.17	2.83

Table 9: Rank Mean for Sparsity.

Method	CELT	CVAE	LASTS	MCCE
`QUANTClassifier`	1.33	2.00	3.67	3.00
`Catch22Classifier`	1.67	2.00	3.33	3.00
`QUANT+LOF`	1.00	N/A	2.00	3.00
`catch22+IForest`	1.00	N/A	2.17	2.83

Table 10: Rank Mean for Implausibility.

Method	CELT	CVAE	LASTS	MCCE
`QUANTClassifier`	2.00	1.00	4.00	3.00
`Catch22Classifier`	2.33	1.00	3.33	3.33
`QUANT+LOF`	1.00	N/A	2.00	3.00
`catch22+IForest`	1.00	N/A	2.17	2.83

Table 11: Validity under `Catch22Classifier`.

Dataset	CELT	CVAE	LASTS	MCCE	Classifier Accuracy
ECGFiveDays	1.00 (1.0)	0.90 (2.5)	0.90 (2.5)	0.00 (4.0)	1.00
GunPoint	0.90 (1.0)	0.80 (2.0)	0.10 (3.0)	0.00 (4.0)	0.98
TwoLeadECG	1.00 (1.5)	1.00 (1.5)	0.40 (3.0)	0.10 (4.0)	1.00

Table 12: Sparsity under `Catch22Classifier`.

Dataset	CELT	CVAE	LASTS	MCCE
ECGFiveDays	0.73±0.15 (1.0)	0.64±0.11 (2.0)	0.53±0.20 (3.0)	– (4.0)
GunPoint	0.77±0.08 (1.0)	0.75±0.13 (2.0)	0.02±0.00 (3.0)	– (4.0)
TwoLeadECG	0.75±0.19 (3.0)	0.76±0.13 (2.0)	0.50±0.07 (4.0)	0.87±0.00 (1.0)

Table 13: Implausibility under `Catch22Classifier`.

Dataset	CELT	CVAE	LASTS	MCCE
ECGFiveDays	11.70±2.96 (2.0)	7.78±4.46 (1.0)	38.68±11.20 (3.0)	– (4.0)
GunPoint	18.65±11.49 (2.0)	17.92±10.48 (1.0)	121.20±0.00 (3.0)	– (4.0)
TwoLeadECG	5.54±1.89 (3.0)	3.94±1.64 (1.0)	15.75±4.74 (4.0)	5.05±0.00 (2.0)

Table 14: Validity for each dataset under supervised classification.

	CELT	CVAE	LASTS	MCCE	MCCE++	w/o Clu	w/o Mask
BME	0.90 (4.0)	1.00 (1.5)	0.50 (7.0)	0.90 (4.0)	1.00 (1.5)	0.60 (6.0)	0.90 (4.0)
BirdChicken	0.80 (1.5)	0.70 (3.0)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.50 (4.0)	0.80 (1.5)
CricketX	1.00 (1.0)	0.00 (5.5)	0.00 (5.5)	0.00 (5.5)	0.00 (5.5)	0.30 (3.0)	0.90 (2.0)
CricketZ	0.90 (1.5)	0.70 (3.0)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.20 (4.0)	0.90 (1.5)
DistalPhalanxOutlineAgeGroup	0.10 (4.0)	0.10 (4.0)	1.00 (1.0)	0.00 (6.5)	0.00 (6.5)	0.10 (4.0)	0.90 (2.0)
ECGFiveDays	1.00 (2.0)	1.00 (2.0)	0.75 (4.0)	0.20 (7.0)	0.30 (6.0)	0.60 (5.0)	1.00 (2.0)
FaceAll	1.00 (1.5)	0.50 (4.0)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.60 (3.0)	1.00 (1.5)
FacesUCR	1.00 (1.5)	0.70 (3.5)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.70 (3.5)	1.00 (1.5)
FiftyWords	1.00 (1.5)	0.50 (5.5)	0.20 (7.0)	0.60 (4.0)	0.50 (5.5)	0.80 (3.0)	1.00 (1.5)
GunPoint	1.00 (1.5)	0.80 (6.0)	0.80 (6.0)	0.90 (3.5)	1.00 (1.5)	0.80 (6.0)	0.90 (3.5)
GunPointAgeSpan	1.00 (2.5)	1.00 (2.5)	0.80 (5.0)	0.70 (6.5)	0.70 (6.5)	1.00 (2.5)	1.00 (2.5)
GunPointMaleVersusFemale	0.40 (4.5)	0.40 (4.5)	0.70 (3.0)	1.00 (1.5)	1.00 (1.5)	0.20 (7.0)	0.30 (6.0)
GunPointOldVersusYoung	1.00 (3.0)	0.50 (7.0)	0.80 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
InsectEPGRegularTrain	1.00 (3.5)	1.00 (3.5)	0.30 (7.0)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)
InsectEPGSmallTrain	1.00 (3.5)	1.00 (3.5)	0.20 (7.0)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)
InsectWingbeatSound	1.00 (1.5)	0.70 (4.5)	0.70 (4.5)	0.40 (6.0)	0.20 (7.0)	1.00 (1.5)	0.80 (3.0)
ItalyPowerDemand	0.40 (7.0)	0.75 (3.0)	0.85 (1.0)	0.70 (5.0)	0.70 (5.0)	0.70 (5.0)	0.80 (2.0)
MoteStrain	0.80 (3.5)	1.00 (1.5)	1.00 (1.5)	0.00 (6.5)	0.00 (6.5)	0.30 (5.0)	0.80 (3.5)
PowerCons	1.00 (2.5)	1.00 (2.5)	0.90 (5.0)	0.40 (7.0)	0.50 (6.0)	1.00 (2.5)	1.00 (2.5)
ProximalPhalanxOutlineAgeGroup	0.10 (5.0)	1.00 (1.0)	0.80 (2.0)	0.00 (6.5)	0.00 (6.5)	0.20 (4.0)	0.60 (3.0)
SonyAIBORobotSurface1	1.00 (1.0)	0.90 (2.0)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.10 (4.0)	0.80 (3.0)
SonyAIBORobotSurface2	0.90 (1.5)	0.90 (1.5)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.20 (4.0)	0.80 (3.0)
ToeSegmentation1	1.00 (2.0)	1.00 (2.0)	0.00 (6.0)	0.00 (6.0)	0.00 (6.0)	0.70 (4.0)	1.00 (2.0)
ToeSegmentation2	1.00 (3.5)	1.00 (3.5)	0.33 (7.0)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)
TwoLeadECG	1.00 (2.0)	1.00 (2.0)	1.00 (2.0)	0.50 (5.5)	0.70 (4.0)	0.50 (5.5)	0.40 (7.0)
UMD	0.60 (2.0)	1.00 (1.0)	0.10 (6.0)	0.20 (4.5)	0.00 (7.0)	0.20 (4.5)	0.40 (3.0)
Rank Mean	2.63	3.21	4.98	5.19	5.08	4.02	2.88
Number of Wins	19	13	4	5	7	7	13

Table 15: Validity for each dataset under one-class classification.

	CELT	LASTS	MCCE	MCCE++	w/o Clu	w/o Mask
BME	1.00 (2.0)	0.50 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
BirdChicken	1.00 (2.5)	0.00 (5.5)	1.00 (2.5)	0.00 (5.5)	1.00 (2.5)	1.00 (2.5)
CricketX	1.00 (2.0)	0.62 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
CricketZ	1.00 (3.0)	0.67 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
DistalPhalanxOutlineAgeGroup	1.00 (2.0)	0.00 (5.0)	0.00 (5.0)	0.00 (5.0)	1.00 (2.0)	1.00 (2.0)
ECGFiveDays	0.90 (1.5)	0.05 (4.0)	0.00 (5.5)	0.00 (5.5)	0.90 (1.5)	0.67 (3.0)
FaceAll	1.00 (2.0)	0.30 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
FacesUCR	1.00 (2.0)	0.60 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
FiftyWords	1.00 (2.0)	0.10 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
GunPoint	1.00 (1.5)	0.50 (4.0)	0.00 (5.5)	0.00 (5.5)	0.78 (3.0)	1.00 (1.5)
GunPointAgeSpan	1.00 (3.0)	0.78 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
GunPointMaleVersusFemale	1.00 (3.0)	0.50 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
GunPointOldVersusYoung	1.00 (2.5)	0.20 (6.0)	1.00 (2.5)	1.00 (2.5)	0.70 (5.0)	1.00 (2.5)
InsectEPGRegularTrain	1.00 (2.5)	1.00 (2.5)	0.00 (5.5)	0.00 (5.5)	1.00 (2.5)	1.00 (2.5)
InsectEPGSmallTrain	1.00 (2.0)	0.20 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
InsectWingbeatSound	1.00 (2.0)	0.50 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
ItalyPowerDemand	0.70 (2.5)	0.95 (1.0)	0.00 (5.5)	0.00 (5.5)	0.70 (2.5)	0.67 (4.0)
MoteStrain	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)	1.00 (3.5)
PowerCons	1.00 (3.0)	0.00 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
ProximalPhalanxOutlineAgeGroup	1.00 (1.5)	0.00 (5.0)	0.00 (5.0)	0.00 (5.0)	0.86 (3.0)	1.00 (1.5)
SonyAIBORobotSurface1	1.00 (2.0)	0.00 (5.0)	0.00 (5.0)	0.00 (5.0)	1.00 (2.0)	1.00 (2.0)
SonyAIBORobotSurface2	1.00 (2.5)	1.00 (2.5)	0.00 (5.5)	0.00 (5.5)	1.00 (2.5)	1.00 (2.5)
ToeSegmentation1	1.00 (2.0)	0.00 (6.0)	0.75 (4.0)	0.50 (5.0)	1.00 (2.0)	1.00 (2.0)
ToeSegmentation2	1.00 (2.0)	0.00 (5.0)	0.00 (5.0)	0.00 (5.0)	1.00 (2.0)	1.00 (2.0)
TwoLeadECG	1.00 (2.0)	0.10 (4.0)	0.00 (5.5)	0.00 (5.5)	1.00 (2.0)	1.00 (2.0)
UMD	1.00 (3.0)	0.00 (6.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)	1.00 (3.0)
Rank Mean	2.29	4.50	4.58	4.73	2.50	2.40
Number of Wins	25	4	8	7	22	24

Table 16: Sparsity for each dataset under supervised classification.

	CELT	CVAE	LASTS	MCCE	MCCE++
BME	0.78 ± 0.09 (1.0)	0.57 ± 0.15 (4.0)	0.51 ± 0.11 (5.0)	0.63 ± 0.14 (3.0)	0.71 ± 0.13 (2.0)
BirdChicken	0.42 ± 0.20 (1.0)	0.31 ± 0.12 (2.0)	— (4.0)	— (4.0)	— (4.0)
CricketX	0.29 ± 0.15 (1.0)	— (3.5)	— (3.5)	— (3.5)	— (3.5)
CricketZ	0.35 ± 0.16 (1.0)	0.23 ± 0.11 (2.0)	— (4.0)	— (4.0)	— (4.0)
DistalPhalanxOutlineAgeGroup	0.84 ± 0.00 (1.0)	0.79 ± 0.00 (2.0)	0.50 ± 0.10 (3.0)	— (4.5)	— (4.5)
ECGFiveDays	0.73 ± 0.14 (2.0)	0.64 ± 0.13 (4.0)	0.61 ± 0.20 (5.0)	0.66 ± 0.01 (3.0)	0.78 ± 0.10 (1.0)
FaceAll	0.24 ± 0.08 (1.0)	0.17 ± 0.03 (2.0)	— (4.0)	— (4.0)	— (4.0)
FacesUCR	0.25 ± 0.07 (1.0)	0.17 ± 0.02 (2.0)	— (4.0)	— (4.0)	— (4.0)
FiftyWords	0.55 ± 0.10 (1.0)	0.29 ± 0.13 (4.0)	0.15 ± 0.02 (5.0)	0.32 ± 0.09 (3.0)	0.34 ± 0.12 (2.0)
GunPoint	0.80 ± 0.12 (1.0)	0.72 ± 0.17 (2.0)	0.28 ± 0.15 (5.0)	0.47 ± 0.16 (3.0)	0.46 ± 0.11 (4.0)
GunPointAgeSpan	0.66 ± 0.10 (1.0)	0.51 ± 0.34 (4.0)	0.28 ± 0.18 (5.0)	0.59 ± 0.12 (3.0)	0.61 ± 0.11 (2.0)
GunPointMaleVersusFemale	0.40 ± 0.27 (2.0)	0.53 ± 0.28 (1.0)	0.01 ± 0.03 (5.0)	0.06 ± 0.08 (4.0)	0.06 ± 0.08 (3.0)
GunPointOldVersusYoung	0.13 ± 0.18 (2.0)	0.11 ± 0.08 (3.0)	0.17 ± 0.23 (1.0)	0.10 ± 0.13 (4.0)	0.09 ± 0.12 (5.0)
InsectEPGRegularTrain	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)
InsectEPGSmallTrain	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)	0.00 ± 0.00 (3.0)
InsectWingbeatSound	0.59 ± 0.10 (2.0)	0.38 ± 0.16 (5.0)	0.43 ± 0.15 (4.0)	0.55 ± 0.11 (3.0)	0.64 ± 0.09 (1.0)
ItalyPowerDemand	0.28 ± 0.19 (4.0)	0.38 ± 0.13 (1.0)	0.27 ± 0.13 (5.0)	0.34 ± 0.05 (3.0)	0.37 ± 0.08 (2.0)
MoteStrain	0.36 ± 0.10 (1.0)	0.18 ± 0.03 (3.0)	0.28 ± 0.07 (2.0)	— (4.5)	— (4.5)
PowerCons	0.54 ± 0.11 (1.0)	0.35 ± 0.08 (3.0)	0.21 ± 0.11 (5.0)	0.33 ± 0.15 (4.0)	0.54 ± 0.06 (2.0)
ProximalPhalanxOutlineAgeGroup	0.75 ± 0.00 (3.0)	0.95 ± 0.06 (1.0)	0.81 ± 0.13 (2.0)	— (4.5)	— (4.5)
SonyAIBORobotSurface1	0.40 ± 0.12 (1.0)	0.32 ± 0.06 (2.0)	— (4.0)	— (4.0)	— (4.0)
SonyAIBORobotSurface2	0.34 ± 0.08 (1.0)	0.24 ± 0.06 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation1	0.42 ± 0.16 (1.0)	0.20 ± 0.07 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation2	0.37 ± 0.18 (1.0)	0.19 ± 0.09 (4.0)	0.17 ± 0.00 (5.0)	0.21 ± 0.02 (3.0)	0.33 ± 0.11 (2.0)
TwoLeadECG	0.62 ± 0.15 (2.0)	0.77 ± 0.08 (1.0)	0.49 ± 0.16 (4.0)	0.41 ± 0.14 (5.0)	0.54 ± 0.16 (3.0)
UMD	0.71 ± 0.16 (1.0)	0.47 ± 0.04 (4.0)	0.53 ± 0.00 (3.0)	0.61 ± 0.12 (2.0)	— (5.0)
Rank Mean	1.54	2.67	3.90	3.62	3.27
Number of Wins	19	6	3	2	4

Table 17: Sparsity for each dataset under one-class classification.

	CELT	LASTS	MCCE	MCCE++
BME	0.48 ± 0.04 (1.0)	0.43 ± 0.00 (2.0)	— (3.5)	— (3.5)
BirdChicken	0.07 ± 0.00 (2.0)	— (3.5)	0.17 ± 0.00 (1.0)	— (3.5)
CricketX	0.45 ± 0.04 (1.0)	0.09 ± 0.03 (2.0)	— (3.5)	— (3.5)
CricketZ	0.49 ± 0.06 (1.0)	0.09 ± 0.02 (4.0)	0.39 ± 0.04 (3.0)	0.42 ± 0.02 (2.0)
DistalPhalanxOutlineAgeGroup	0.50 ± 0.12 (1.0)	— (3.0)	— (3.0)	— (3.0)
ECGFiveDays	0.70 ± 0.09 (1.0)	0.33 ± 0.00 (2.0)	— (3.5)	— (3.5)
FaceAll	0.21 ± 0.04 (1.0)	0.15 ± 0.04 (2.0)	— (3.5)	— (3.5)
FacesUCR	0.17 ± 0.04 (1.0)	0.13 ± 0.02 (2.0)	— (3.5)	— (3.5)
FiftyWords	0.26 ± 0.08 (1.0)	0.17 ± 0.00 (2.0)	— (3.5)	— (3.5)
GunPoint	0.29 ± 0.25 (1.0)	0.14 ± 0.08 (2.0)	— (3.5)	— (3.5)
GunPointAgeSpan	0.42 ± 0.13 (1.0)	0.18 ± 0.16 (4.0)	0.27 ± 0.17 (3.0)	0.28 ± 0.17 (2.0)
GunPointMaleVersusFemale	0.35 ± 0.24 (1.0)	0.32 ± 0.12 (2.0)	0.09 ± 0.09 (4.0)	0.10 ± 0.10 (3.0)
GunPointOldVersusYoung	0.05 ± 0.08 (3.0)	0.00 ± 0.00 (4.0)	0.07 ± 0.13 (1.5)	0.07 ± 0.12 (1.5)
InsectEPGRegularTrain	0.00 ± 0.00 (1.5)	0.00 ± 0.00 (1.5)	— (3.5)	— (3.5)
InsectEPGSmallTrain	0.00 ± 0.00 (1.5)	0.00 ± 0.00 (1.5)	— (3.5)	— (3.5)
InsectWingbeatSound	0.50 ± 0.06 (1.0)	0.44 ± 0.03 (2.0)	— (3.5)	— (3.5)
ItalyPowerDemand	0.30 ± 0.12 (1.0)	0.24 ± 0.13 (2.0)	— (3.5)	— (3.5)
MoteStrain	0.40 ± 0.00 (1.0)	0.20 ± 0.00 (2.0)	0.15 ± 0.00 (3.5)	0.15 ± 0.00 (3.5)
PowerCons	0.26 ± 0.17 (2.0)	— (4.0)	0.16 ± 0.09 (3.0)	0.30 ± 0.13 (1.0)
ProximalPhalanxOutlineAgeGroup	0.73 ± 0.09 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface1	0.31 ± 0.07 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface2	0.33 ± 0.01 (1.0)	0.16 ± 0.02 (2.0)	— (3.5)	— (3.5)
ToeSegmentation1	0.24 ± 0.09 (1.0)	— (4.0)	0.15 ± 0.03 (3.0)	0.21 ± 0.01 (2.0)
ToeSegmentation2	0.22 ± 0.10 (1.0)	— (3.0)	— (3.0)	— (3.0)
TwoLeadECG	0.63 ± 0.19 (1.0)	0.33 ± 0.00 (2.0)	— (3.5)	— (3.5)
UMD	0.56 ± 0.02 (3.0)	— (4.0)	0.60 ± 0.04 (2.0)	0.63 ± 0.01 (1.0)
Rank Mean	1.27	2.63	3.13	2.96
Number of Wins	22	2	2	3

Table 18: Implausibility for each dataset under supervised classification.

	CELT	CVAE	LASTS	MCCE	MCCE++
BME	18.77 ± 4.59 (3.0)	24.33 ± 12.81 (4.0)	30.76 ± 11.40 (5.0)	15.42 ± 7.33 (1.0)	15.47 ± 5.63 (2.0)
BirdChicken	116.05 ± 25.70 (2.0)	97.50 ± 40.20 (1.0)	— (4.0)	— (4.0)	— (4.0)
CricketX	77.64 ± 21.31 (1.0)	— (3.5)	— (3.5)	— (3.5)	— (3.5)
CricketZ	56.35 ± 17.70 (1.0)	83.17 ± 29.23 (2.0)	— (4.0)	— (4.0)	— (4.0)
DistalPhalanxOutlineAgeGroup	6.18 ± 0.00 (2.0)	5.05 ± 0.00 (1.0)	7.35 ± 1.22 (3.0)	— (4.5)	— (4.5)
ECGFiveDays	12.62 ± 4.99 (2.0)	7.84 ± 3.29 (1.0)	29.54 ± 2.16 (5.0)	14.93 ± 3.33 (3.0)	15.26 ± 2.19 (4.0)
FaceAll	42.40 ± 5.46 (2.0)	32.37 ± 4.48 (1.0)	— (4.0)	— (4.0)	— (4.0)
FacesUCR	44.46 ± 9.81 (2.0)	36.08 ± 9.99 (1.0)	— (4.0)	— (4.0)	— (4.0)
FiftyWords	75.23 ± 10.97 (2.0)	67.31 ± 20.70 (1.0)	139.18 ± 0.45 (5.0)	101.29 ± 33.88 (4.0)	82.74 ± 7.87 (3.0)
GunPoint	22.63 ± 12.02 (2.0)	18.54 ± 10.02 (1.0)	46.28 ± 25.58 (5.0)	26.06 ± 4.90 (3.0)	28.91 ± 3.34 (4.0)
GunPointAgeSpan	13.60 ± 6.20 (1.0)	25.82 ± 23.14 (4.0)	39.30 ± 12.92 (5.0)	15.97 ± 6.13 (3.0)	15.67 ± 6.01 (2.0)
GunPointMaleVersusFemale	14.51 ± 10.54 (2.0)	8.74 ± 2.71 (1.0)	48.20 ± 4.91 (5.0)	28.56 ± 14.78 (3.0)	35.25 ± 10.44 (4.0)
GunPointOldVersusYoung	16.11 ± 8.63 (3.0)	22.41 ± 25.44 (4.0)	36.52 ± 13.47 (5.0)	6.82 ± 1.07 (2.0)	6.55 ± 1.79 (1.0)
InsectEPGRegularTrain	6.83 ± 1.22 (1.0)	14.82 ± 5.40 (4.0)	359.51 ± 15.48 (5.0)	12.17 ± 0.00 (2.0)	12.17 ± 0.00 (3.0)
InsectEPGSmallTrain	7.90 ± 1.07 (1.0)	14.46 ± 3.30 (4.0)	297.52 ± 15.84 (5.0)	12.77 ± 0.00 (3.0)	12.77 ± 0.00 (2.0)
InsectWingbeatSound	49.39 ± 13.26 (2.0)	43.97 ± 21.30 (1.0)	97.50 ± 1.09 (5.0)	69.61 ± 13.32 (3.0)	95.45 ± 21.44 (4.0)
ItalyPowerDemand	3.54 ± 0.68 (4.0)	2.94 ± 1.67 (2.0)	4.71 ± 1.49 (5.0)	2.51 ± 1.36 (1.0)	3.47 ± 1.34 (3.0)
MoteStrain	23.26 ± 4.12 (3.0)	13.36 ± 5.46 (1.0)	19.77 ± 0.46 (2.0)	— (4.5)	— (4.5)
PowerCons	38.07 ± 9.18 (3.0)	37.66 ± 7.81 (2.0)	56.14 ± 0.54 (5.0)	35.10 ± 3.05 (1.0)	51.98 ± 13.06 (4.0)
ProximalPhalanxOutlineAgeGroup	2.89 ± 0.00 (2.0)	2.61 ± 0.53 (1.0)	5.38 ± 2.06 (3.0)	— (4.5)	— (4.5)
SonyAIBORobotSurface1	18.81 ± 2.92 (2.0)	18.03 ± 1.67 (1.0)	— (4.0)	— (4.0)	— (4.0)
SonyAIBORobotSurface2	20.08 ± 5.90 (1.0)	22.40 ± 6.85 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation1	88.82 ± 18.36 (2.0)	76.67 ± 12.87 (1.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation2	148.51 ± 16.40 (2.0)	146.24 ± 22.25 (1.0)	179.30 ± 0.00 (4.0)	189.48 ± 7.20 (5.0)	172.05 ± 13.87 (3.0)
TwoLeadECG	4.89 ± 0.86 (2.0)	3.97 ± 1.26 (1.0)	9.59 ± 1.39 (5.0)	7.82 ± 0.00 (4.0)	5.59 ± 2.00 (3.0)
UMD	17.71 ± 1.53 (1.0)	43.15 ± 15.43 (3.0)	49.06 ± 0.00 (4.0)	25.56 ± 2.60 (2.0)	— (5.0)
Rank Mean	1.96	1.90	4.33	3.27	3.54
Number of Wins	7	15	0	3	1

Table 19: Implausibility for each dataset under one-class classification.

	CELT	LASTS	MCCE	MCCE++
BME	23.29 ± 0.28 (1.0)	62.38 ± 0.00 (2.0)	— (3.5)	— (3.5)
BirdChicken	124.44 ± 0.00 (1.0)	— (3.5)	161.57 ± 0.00 (2.0)	— (3.5)
CricketX	61.01 ± 6.69 (1.0)	179.33 ± 24.22 (2.0)	— (3.5)	— (3.5)
CricketZ	67.53 ± 7.01 (1.0)	178.14 ± 19.34 (4.0)	101.10 ± 17.82 (2.0)	103.75 ± 11.95 (3.0)
DistalPhalanxOutlineAgeGroup	6.97 ± 0.78 (1.0)	— (3.0)	— (3.0)	— (3.0)
ECGFiveDays	10.29 ± 2.78 (1.0)	55.18 ± 0.00 (2.0)	— (3.5)	— (3.5)
FaceAll	39.53 ± 4.97 (2.0)	35.00 ± 0.76 (1.0)	— (3.5)	— (3.5)
FacesUCR	38.62 ± 7.07 (2.0)	35.91 ± 3.45 (1.0)	— (3.5)	— (3.5)
FiftyWords	71.77 ± 14.53 (2.0)	70.25 ± 0.00 (1.0)	— (3.5)	— (3.5)
GunPoint	14.64 ± 3.91 (1.0)	25.67 ± 1.58 (2.0)	— (3.5)	— (3.5)
GunPointAgeSpan	21.00 ± 6.57 (3.0)	33.40 ± 7.05 (4.0)	9.22 ± 2.22 (1.0)	9.44 ± 1.98 (2.0)
GunPointMaleVersusFemale	12.43 ± 7.67 (1.0)	40.85 ± 24.51 (4.0)	30.53 ± 1.59 (2.0)	31.77 ± 2.11 (3.0)
GunPointOldVersusYoung	7.89 ± 4.75 (3.0)	20.84 ± 10.19 (4.0)	3.96 ± 0.21 (1.0)	4.16 ± 0.77 (2.0)
InsectEPGRegularTrain	13.18 ± 8.50 (2.0)	7.65 ± 1.25 (1.0)	— (3.5)	— (3.5)
InsectEPGSmallTrain	8.55 ± 3.28 (1.0)	38.66 ± 32.74 (2.0)	— (3.5)	— (3.5)
InsectWingbeatSound	39.38 ± 4.73 (1.0)	49.20 ± 0.43 (2.0)	— (3.5)	— (3.5)
ItalyPowerDemand	3.02 ± 0.52 (2.0)	2.65 ± 0.18 (1.0)	— (3.5)	— (3.5)
MoteStrain	12.57 ± 0.00 (1.0)	15.22 ± 0.00 (2.0)	19.17 ± 0.00 (3.5)	19.17 ± 0.00 (3.5)
PowerCons	58.19 ± 7.82 (2.0)	— (4.0)	49.55 ± 26.10 (1.0)	59.12 ± 6.98 (3.0)
ProximalPhalanxOutlineAgeGroup	3.21 ± 0.78 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface1	19.04 ± 0.38 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface2	20.64 ± 0.14 (1.0)	37.72 ± 0.51 (2.0)	— (3.5)	— (3.5)
ToeSegmentation1	110.80 ± 33.15 (1.0)	— (4.0)	169.13 ± 32.65 (2.0)	209.24 ± 8.02 (3.0)
ToeSegmentation2	117.81 ± 20.92 (1.0)	— (3.0)	— (3.0)	— (3.0)
TwoLeadECG	3.54 ± 0.52 (1.0)	11.07 ± 0.00 (2.0)	— (3.5)	— (3.5)
UMD	14.34 ± 1.32 (1.0)	— (4.0)	23.84 ± 7.34 (3.0)	18.60 ± 0.04 (2.0)
Rank Mean	1.38	2.56	2.88	3.17
Number of Wins	18	5	3	0

Table 20: NumSeg for each dataset under supervised classification.

	CELT	CVAE	LASTS	MCCE	MCCE++
BME	3.89 ± 0.99 (1.0)	4.40 ± 2.33 (2.0)	5.20 ± 2.23 (4.0)	5.11 ± 2.51 (3.0)	5.90 ± 2.30 (5.0)
BirdChicken	4.88 ± 2.03 (1.0)	6.29 ± 1.67 (2.0)	— (4.0)	— (4.0)	— (4.0)
CricketX	7.10 ± 2.88 (1.0)	— (3.5)	— (3.5)	— (3.5)	— (3.5)
CricketZ	9.33 ± 2.26 (2.0)	8.57 ± 2.38 (1.0)	— (4.0)	— (4.0)	— (4.0)
DistalPhalanxOutlineAgeGroup	2.00 ± 0.00 (1.5)	2.00 ± 0.00 (1.5)	6.60 ± 1.62 (3.0)	— (4.5)	— (4.5)
ECGFiveDays	1.90 ± 1.14 (1.0)	2.30 ± 0.71 (2.0)	2.60 ± 0.88 (5.0)	2.50 ± 0.50 (4.0)	2.33 ± 1.25 (3.0)
FaceAll	11.30 ± 2.41 (1.0)	11.40 ± 1.85 (2.0)	— (4.0)	— (4.0)	— (4.0)
FacesUCR	11.30 ± 2.61 (2.0)	10.86 ± 1.81 (1.0)	— (4.0)	— (4.0)	— (4.0)
FiftyWords	2.80 ± 1.17 (1.0)	4.60 ± 1.74 (3.0)	4.00 ± 0.00 (2.0)	5.67 ± 1.49 (5.0)	5.20 ± 1.17 (4.0)
GunPoint	1.90 ± 0.70 (1.0)	2.62 ± 1.36 (2.0)	4.06 ± 1.48 (5.0)	3.67 ± 1.15 (4.0)	3.50 ± 1.20 (3.0)
GunPointAgeSpan	2.70 ± 1.35 (1.0)	3.60 ± 1.91 (4.0)	2.75 ± 0.66 (2.0)	3.71 ± 1.39 (5.0)	3.29 ± 0.88 (3.0)
GunPointMaleVersusFemale	3.25 ± 1.30 (5.0)	2.75 ± 1.48 (4.0)	1.14 ± 0.35 (1.0)	2.00 ± 1.34 (2.5)	2.00 ± 1.41 (2.5)
GunPointOldVersusYoung	1.90 ± 1.22 (4.0)	2.00 ± 0.63 (5.0)	1.88 ± 1.36 (3.0)	1.60 ± 0.80 (1.5)	1.60 ± 1.02 (1.5)
InsectEPGRegularTrain	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)
InsectEPGSmallTrain	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)	1.00 ± 0.00 (3.0)
InsectWingbeatSound	3.70 ± 1.68 (1.0)	5.29 ± 0.88 (4.0)	4.43 ± 2.66 (2.0)	5.50 ± 0.87 (5.0)	5.00 ± 1.00 (3.0)
ItalyPowerDemand	4.00 ± 1.22 (1.0)	4.07 ± 0.77 (2.0)	4.24 ± 0.88 (4.0)	4.29 ± 0.70 (5.0)	4.14 ± 1.36 (3.0)
MoteStrain	4.62 ± 1.22 (1.0)	5.40 ± 1.28 (3.0)	4.90 ± 0.70 (2.0)	— (4.5)	— (4.5)
PowerCons	6.00 ± 1.84 (2.0)	7.20 ± 2.44 (5.0)	7.00 ± 1.89 (3.5)	5.25 ± 1.92 (1.0)	7.00 ± 2.61 (3.5)
ProximalPhalanxOutlineAgeGroup	2.00 ± 0.00 (3.0)	0.20 ± 0.40 (1.0)	1.25 ± 1.30 (2.0)	— (4.5)	— (4.5)
SonyAIBORobotSurface1	6.20 ± 1.17 (1.0)	7.00 ± 1.33 (2.0)	— (4.0)	— (4.0)	— (4.0)
SonyAIBORobotSurface2	5.89 ± 1.45 (1.0)	6.11 ± 1.52 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation1	6.90 ± 2.12 (1.0)	8.70 ± 2.49 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation2	5.33 ± 2.87 (2.0)	7.33 ± 1.70 (3.0)	3.00 ± 0.00 (1.0)	10.33 ± 0.94 (4.0)	13.00 ± 0.82 (5.0)
TwoLeadECG	2.30 ± 0.90 (2.0)	1.80 ± 0.98 (1.0)	3.10 ± 0.70 (4.0)	3.20 ± 0.98 (5.0)	2.57 ± 1.05 (3.0)
UMD	2.50 ± 0.50 (1.0)	3.00 ± 0.63 (2.5)	4.00 ± 0.00 (4.0)	3.00 ± 1.00 (2.5)	— (5.0)
Rank Mean	1.71	2.56	3.27	3.79	3.67
Number of Wins	18	7	4	4	3

Table 21: NumSeg for each dataset under one-class classification.

	CELT	LASTS	MCCE	MCCE++
BME	3.00 ± 0.00 (1.0)	12.00 ± 0.00 (2.0)	— (3.5)	— (3.5)
BirdChicken	7.00 ± 0.00 (1.0)	— (3.5)	11.00 ± 0.00 (2.0)	— (3.5)
CricketX	10.25 ± 2.11 (2.0)	5.20 ± 0.75 (1.0)	— (3.5)	— (3.5)
CricketZ	10.33 ± 0.75 (2.0)	5.00 ± 0.71 (1.0)	14.67 ± 1.11 (3.0)	15.33 ± 2.69 (4.0)
DistalPhalanxOutlineAgeGroup	6.50 ± 1.89 (1.0)	— (3.0)	— (3.0)	— (3.0)
ECGFiveDays	2.11 ± 1.10 (1.0)	5.00 ± 0.00 (2.0)	— (3.5)	— (3.5)
FaceAll	11.40 ± 2.58 (2.0)	9.33 ± 1.70 (1.0)	— (3.5)	— (3.5)
FacesUCR	10.90 ± 1.81 (2.0)	9.83 ± 3.02 (1.0)	— (3.5)	— (3.5)
FiftyWords	5.10 ± 1.97 (2.0)	5.00 ± 0.00 (1.0)	— (3.5)	— (3.5)
GunPoint	3.33 ± 1.56 (1.0)	4.11 ± 1.20 (2.0)	— (3.5)	— (3.5)
GunPointAgeSpan	3.11 ± 0.99 (4.0)	3.00 ± 2.20 (3.0)	2.56 ± 0.68 (1.5)	2.56 ± 0.68 (1.5)
GunPointMaleVersusFemale	2.90 ± 1.45 (3.0)	4.00 ± 1.10 (4.0)	2.40 ± 1.20 (1.5)	2.40 ± 1.28 (1.5)
GunPointOldVersusYoung	1.20 ± 0.40 (2.0)	1.00 ± 0.00 (1.0)	1.60 ± 1.02 (3.5)	1.60 ± 1.02 (3.5)
InsectEPGRegularTrain	1.00 ± 0.00 (1.5)	1.00 ± 0.00 (1.5)	— (3.5)	— (3.5)
InsectEPGSmallTrain	1.00 ± 0.00 (1.5)	1.00 ± 0.00 (1.5)	— (3.5)	— (3.5)
InsectWingbeatSound	5.25 ± 0.97 (2.0)	4.75 ± 0.83 (1.0)	— (3.5)	— (3.5)
ItalyPowerDemand	4.86 ± 1.46 (2.0)	4.05 ± 1.32 (1.0)	— (3.5)	— (3.5)
MoteStrain	5.00 ± 0.00 (1.5)	5.00 ± 0.00 (1.5)	6.00 ± 0.00 (3.5)	6.00 ± 0.00 (3.5)
PowerCons	6.25 ± 1.48 (1.5)	— (4.0)	6.25 ± 2.49 (1.5)	8.00 ± 1.41 (3.0)
ProximalPhalanxOutlineAgeGroup	3.57 ± 2.06 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface1	6.33 ± 2.05 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface2	5.50 ± 0.50 (1.0)	6.00 ± 0.00 (2.0)	— (3.5)	— (3.5)
ToeSegmentation1	9.25 ± 1.64 (2.0)	— (4.0)	9.00 ± 2.83 (1.0)	9.50 ± 1.50 (3.0)
ToeSegmentation2	9.80 ± 4.96 (1.0)	— (3.0)	— (3.0)	— (3.0)
TwoLeadECG	1.90 ± 0.83 (1.0)	3.00 ± 0.00 (2.0)	— (3.5)	— (3.5)
UMD	2.67 ± 0.94 (2.0)	— (4.0)	4.33 ± 1.70 (3.0)	2.33 ± 0.47 (1.0)
Rank Mean	1.65	2.19	3.00	3.15
Number of Wins	14	11	4	3

Table 22: MAD for each dataset under supervised classification.

	CELT	CVAE	LASTS	MCCE	MCCE++
BME	0.20 ± 0.08 (1.0)	0.32 ± 0.14 (3.0)	0.42 ± 0.15 (5.0)	0.40 ± 0.43 (4.0)	0.23 ± 0.11 (2.0)
BirdChicken	0.58 ± 0.34 (2.0)	0.48 ± 0.24 (1.0)	— (4.0)	— (4.0)	— (4.0)
CricketX	0.64 ± 0.21 (1.0)	— (3.5)	— (3.5)	— (3.5)	— (3.5)
CricketZ	0.59 ± 0.27 (1.0)	0.81 ± 0.23 (2.0)	— (4.0)	— (4.0)	— (4.0)
DistalPhalanxOutlineAgeGroup	0.12 ± 0.00 (1.5)	0.12 ± 0.00 (1.5)	0.24 ± 0.05 (3.0)	— (4.5)	— (4.5)
ECGFiveDays	0.22 ± 0.10 (2.0)	0.26 ± 0.08 (3.0)	0.34 ± 0.12 (5.0)	0.28 ± 0.04 (4.0)	0.20 ± 0.09 (1.0)
FaceAll	0.68 ± 0.14 (1.0)	0.81 ± 0.15 (2.0)	— (4.0)	— (4.0)	— (4.0)
FacesUCR	0.65 ± 0.22 (1.0)	0.81 ± 0.09 (2.0)	— (4.0)	— (4.0)	— (4.0)
FiftyWords	0.56 ± 0.20 (1.0)	0.72 ± 0.30 (3.0)	0.74 ± 0.05 (5.0)	0.74 ± 0.15 (4.0)	0.61 ± 0.25 (2.0)
GunPoint	0.18 ± 0.11 (1.0)	0.20 ± 0.09 (2.0)	0.52 ± 0.16 (5.0)	0.43 ± 0.14 (3.0)	0.46 ± 0.14 (4.0)
GunPointAgeSpan	0.24 ± 0.10 (1.0)	0.33 ± 0.27 (3.0)	0.49 ± 0.22 (5.0)	0.29 ± 0.08 (2.0)	0.34 ± 0.12 (4.0)
GunPointMaleVersusFemale	0.56 ± 0.36 (2.0)	0.33 ± 0.17 (1.0)	1.32 ± 0.28 (3.0)	1.41 ± 0.41 (5.0)	1.36 ± 0.42 (4.0)
GunPointOldVersusYoung	1.45 ± 0.73 (3.0)	0.78 ± 0.25 (1.0)	1.24 ± 0.62 (2.0)	1.54 ± 0.75 (4.0)	1.58 ± 0.74 (5.0)
InsectEPGRegularTrain	2.08 ± 0.04 (5.0)	2.00 ± 0.07 (4.0)	1.26 ± 0.03 (1.0)	1.91 ± 0.04 (2.0)	1.92 ± 0.04 (3.0)
InsectEPGSmallTrain	2.04 ± 0.04 (5.0)	2.04 ± 0.09 (4.0)	1.36 ± 0.01 (1.0)	1.89 ± 0.04 (2.0)	1.89 ± 0.04 (3.0)
InsectWingbeatSound	0.61 ± 0.08 (4.0)	0.79 ± 0.09 (5.0)	0.48 ± 0.10 (2.0)	0.59 ± 0.14 (3.0)	0.46 ± 0.07 (1.0)
ItalyPowerDemand	0.63 ± 0.30 (5.0)	0.36 ± 0.12 (1.0)	0.48 ± 0.17 (3.0)	0.44 ± 0.12 (2.0)	0.48 ± 0.17 (4.0)
MoteStrain	0.48 ± 0.12 (1.0)	0.67 ± 0.10 (3.0)	0.53 ± 0.10 (2.0)	— (4.5)	— (4.5)
PowerCons	0.51 ± 0.16 (2.0)	0.74 ± 0.22 (4.0)	0.73 ± 0.25 (3.0)	0.94 ± 0.30 (5.0)	0.48 ± 0.07 (1.0)
ProximalPhalanxOutlineAgeGroup	0.12 ± 0.00 (2.0)	0.08 ± 0.02 (1.0)	0.13 ± 0.04 (3.0)	— (4.5)	— (4.5)
SonyAIBORobotSurface1	0.39 ± 0.11 (1.0)	0.45 ± 0.08 (2.0)	— (4.0)	— (4.0)	— (4.0)
SonyAIBORobotSurface2	0.56 ± 0.10 (1.0)	0.63 ± 0.09 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation1	0.51 ± 0.21 (1.0)	0.81 ± 0.23 (2.0)	— (4.0)	— (4.0)	— (4.0)
ToeSegmentation2	0.55 ± 0.09 (1.0)	0.85 ± 0.02 (5.0)	0.73 ± 0.00 (3.0)	0.81 ± 0.08 (4.0)	0.59 ± 0.13 (2.0)
TwoLeadECG	0.22 ± 0.09 (2.0)	0.14 ± 0.03 (1.0)	0.30 ± 0.08 (4.0)	0.39 ± 0.13 (5.0)	0.30 ± 0.13 (3.0)
UMD	0.22 ± 0.10 (1.0)	0.52 ± 0.15 (3.0)	0.70 ± 0.00 (4.0)	0.40 ± 0.05 (2.0)	— (5.0)
Rank Mean	1.90	2.50	3.48	3.69	3.42
Number of Wins	15	7	2	0	3

Table 23: MAD for each dataset under one-class classification.

	CELT	LASTS	MCCE	MCCE++
BME	0.65 ± 0.35 (1.0)	0.66 ± 0.00 (2.0)	— (3.5)	— (3.5)
BirdChicken	1.32 ± 0.00 (2.0)	— (3.5)	0.73 ± 0.00 (1.0)	— (3.5)
CricketX	0.35 ± 0.03 (1.0)	1.02 ± 0.05 (2.0)	— (3.5)	— (3.5)
CricketZ	0.30 ± 0.04 (1.0)	1.02 ± 0.07 (4.0)	0.45 ± 0.09 (3.0)	0.41 ± 0.01 (2.0)
DistalPhalanxOutlineAgeGroup	0.25 ± 0.06 (1.0)	— (3.0)	— (3.0)	— (3.0)
ECGFiveDays	0.26 ± 0.08 (1.0)	0.57 ± 0.00 (2.0)	— (3.5)	— (3.5)
FaceAll	0.77 ± 0.09 (1.0)	0.86 ± 0.09 (2.0)	— (3.5)	— (3.5)
FacesUCR	0.83 ± 0.09 (1.0)	0.91 ± 0.10 (2.0)	— (3.5)	— (3.5)
FiftyWords	0.74 ± 0.19 (1.0)	1.06 ± 0.00 (2.0)	— (3.5)	— (3.5)
GunPoint	0.41 ± 0.15 (1.0)	0.60 ± 0.17 (2.0)	— (3.5)	— (3.5)
GunPointAgeSpan	0.35 ± 0.09 (1.0)	1.02 ± 0.67 (4.0)	1.01 ± 0.51 (2.0)	1.01 ± 0.51 (3.0)
GunPointMaleVersusFemale	0.53 ± 0.26 (2.0)	0.52 ± 0.13 (1.0)	1.13 ± 0.29 (3.5)	1.13 ± 0.31 (3.5)
GunPointOldVersusYoung	1.70 ± 0.65 (3.0)	2.08 ± 0.14 (4.0)	1.61 ± 0.68 (1.0)	1.62 ± 0.67 (2.0)
InsectEPGRegularTrain	2.05 ± 0.05 (1.0)	2.07 ± 0.04 (2.0)	— (3.5)	— (3.5)
InsectEPGSmallTrain	2.04 ± 0.05 (1.0)	2.10 ± 0.12 (2.0)	— (3.5)	— (3.5)
InsectWingbeatSound	0.66 ± 0.04 (1.0)	0.71 ± 0.01 (2.0)	— (3.5)	— (3.5)
ItalyPowerDemand	0.47 ± 0.14 (1.0)	0.57 ± 0.24 (2.0)	— (3.5)	— (3.5)
MoteStrain	0.34 ± 0.00 (1.0)	0.50 ± 0.00 (2.0)	0.56 ± 0.00 (3.5)	0.56 ± 0.00 (3.5)
PowerCons	1.15 ± 0.27 (2.0)	— (4.0)	1.34 ± 0.24 (3.0)	1.08 ± 0.12 (1.0)
ProximalPhalanxOutlineAgeGroup	0.15 ± 0.02 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface1	0.43 ± 0.06 (1.0)	— (3.0)	— (3.0)	— (3.0)
SonyAIBORobotSurface2	0.46 ± 0.06 (1.0)	0.85 ± 0.08 (2.0)	— (3.5)	— (3.5)
ToeSegmentation1	0.63 ± 0.13 (1.0)	— (4.0)	0.86 ± 0.09 (2.0)	0.96 ± 0.10 (3.0)
ToeSegmentation2	0.84 ± 0.18 (1.0)	— (3.0)	— (3.0)	— (3.0)
TwoLeadECG	0.22 ± 0.13 (1.0)	0.35 ± 0.00 (2.0)	— (3.5)	— (3.5)
UMD	0.78 ± 0.08 (3.0)	— (4.0)	0.44 ± 0.28 (1.0)	0.52 ± 0.23 (2.0)
Rank Mean	1.27	2.63	2.98	3.12
Number of Wins	21	1	3	1