Counterfactual Explanations for Multivariate Time Series

Emre Ates*, Burak Aksar*, Vitus J. Leung†, and Ayse K. Coskun*
*Dept. of Electrical and Computer Eng.
Boston University
Boston, MA, USA
Email: {ates,baksar,acoskun}@bu.edu
†Sandia National Laboratories
Albuquerque, NM, USA
Email: vjleung@sandia.gov

***Abstract*—Multivariate time series are used in many science and engineering domains, including health-care, astronomy, and high-performance computing. A recent trend is to use machine learning (ML) to process this complex data and these ML-based frameworks are starting to play a critical role for a variety of applications. However, barriers such as user distrust or difficulty of debugging need to be overcome to enable widespread adoption of such frameworks in production systems. To address this challenge, we propose a novel explainability technique, *CoMTE*, that provides *counterfactual explanations for supervised machine learning frameworks on multivariate time series data*. Using various machine learning frameworks and data sets, we compare CoMTE with several state-of-the-art explainability methods and show that we outperform existing methods in comprehensibility and robustness. We also show how CoMTE can be used to debug machine learning frameworks and gain a better understanding of the underlying multivariate time series data.**

I. Introduction

Multivariate time series data analytics have been gaining popularity, especially due to the recent advancements in internet of things technologies and the omnipresence of real-time sensors [1]. Health care, astronomy, sustainable energy, and geoscience are some domains where researchers utilize multivariate time series along with machine learning (ML) based analytics to solve problems such as seismic activity forecasting, hospitalization rate prediction, and many others [2]. Large-scale computing system management has also been increasingly leveraging time series analytics for improving performance, efficiency, or security. For example, high performance computing (HPC) systems produce terabytes of instrumentation data per day in the form of logs, metrics, and traces [3], and monitoring frameworks organize system-wide resource utilization metrics as multivariate time series. Thousands of variables can be collected per compute node and each variable—representing different resource statistics such as network packet counts, CPU utilization, or memory statistics—is sampled on intervals of seconds to minutes [3]–[5]. Analyzing this data is invaluable for management and debugging [3], [6], but extensive manual analysis of these big data sets is not feasible. Researchers have recently started using ML to help analyze large-scale computing system telemetry data and gain valuable insights about system operation and user applications. Frameworks using ML methods can process large amounts of data and, in addition, benefit from the flexibility of the models that generalize to different systems and potentially previously unseen cases. ML frameworks using multivariate time series data have been shown to diagnose performance variations [7]–[10], improve scheduling [11] or improve system security by detecting unwanted or illegal applications on HPC systems [12].

While many advantages of ML are well-studied, there are also common drawbacks that need to be addressed before ML can be widely used in production systems. ML frameworks commonly have a taciturn nature, e.g., reporting only the final diagnosis when analyzing performance problems in HPC systems such as "`network contention on router-123`," without providing reasoning relating to the underlying data. Furthermore, the ML models within these frameworks are black boxes, which may perform multiple data transformations before arriving at a classification, and thus are often challenging to understand. The black-box nature of ML frameworks increases the difficulty of debugging mispredictions, degrades user trust, thus reduces the overall usefulness of the frameworks.

To address the broad ML *explainability* problem, a number of methods that explain black-box classifiers have been introduced. These methods can be divided into *local* and *global* explanations, based on whether they explain a single prediction or the complete classifier. Local explanations can also be divided into *sample-based* explanations that provide different samples i.e., time-windows, as explanations and *feature-based* explanations that indicate the features that impact the decision the most. However, most of the existing explainability methods are not designed for multivariate time series data, Thus, existing methods often fail to generate sufficiently simple explanations when working with complex multivariate time series data, e.g., HPC systems data.

Why do existing explainability methods fail to provide satisfactory explanations for high-dimensional multivariate time series data? One differentiating factor is the complexity of the data. Existing sample-based methods provide samples from the training set or synthetically generate samples [13], [14]. These methods are designed with the assumption that a single sample

is self-explanatory and users can easily distinguish between two samples; however, providing a time series sample with hundreds of variables is most often not an adequate explanation. On the other hand, existing feature-based methods [15], [16] provide a set of features and expect the users to know the meaning of each feature, as well as normal and abnormal values for them, which is not possible in many domains.

In this paper, we introduce *CoMTE*, a novel *counterfactual multivariate time series explainability method* that provides explanations for individual predictions. The counterfactual explanations consist of hypothetical samples that are as similar as possible to the sample that is explained, while having a different classification label; i.e., "if the values of these particular time series were different in the given sample, the classification label would have been different." Counterfactual explanations are generated by selecting time series from the training set and substituting them in the sample under investigation to obtain different classification results. In this way, end users can understand the classification decision by examining a limited number of variables.

These explanations can then be used to debug misclassifications, understand how the classifier makes decisions, provide adaptive dashboards that highlight important metrics from ongoing application runs, and extract knowledge on the nature of normal or anomalous behavior of a system. Our specific contributions are as follows:

- Design of a formal problem statement for multivariate time series explainability.
- Design and implementation of heuristic algorithms that generate counterfactual explanations for time series[1].
- Experimental evaluation of the proposed explainability method to explain several ML frameworks that work with multivariate time series data, using 3 different HPC telemetry data sets and a motion classification data set, and comparisons of CoMTE with state-of-the-art explainability methods using both novel and standard measures of explanation success. CoMTE generates the most comprehensible explanations and performs better than the baselines in terms of *faithfulness* and *robustness*.

II. Related Work

Counterfactual explanations have recently been introduced to explain ML models. Wachter et al. are among the first to use the term "counterfactual" for ML explanations [17]. Counterfactual explanations have also been used in image [18], tabular data [19], document [20] and univariate time series [21], [22] classification. Diverse Counterfactual Explanations (DiCE) is an open-source counterfactual explanation method for black-box classifiers [23]. Contrastive explanations method [14] generates synthetic counterfactuals, while using autoencoders to ensure the generated sample is realistic. Recent work also addresses the problem of counterfactuals for data that contains actionable or immutable features [24], [25].

[1]Our implementation is available at https://github.com/peaclab/CoMTE

Local feature-based explanations highlight certain features that impact the classification of a sample of interest. These methods include the local interpretable model-agnostic explanation (LIME) method, which fits a linear model to classifier decisions for samples in the neighborhood of the sample to be explained [15], and Shapley additive explanations (SHAP) method, which derives additive Shapley values for each feature to represent the impact of that feature [16]. These feature-based models do not support using time series directly; however, they can be applied to sets of features extracted from the time series.

To the best of our knowledge, CoMTE is the first method to generate counterfactual explanations for ML frameworks that work with high-dimensional multivariate time series. Furthermore, it is common for multivariate time series data to have various physical constraints, such as in HPC time series data. Existing explainability methods [15], [17], [22] often require synthetic or random data generation that does not consider these constraints.

III. The Counterfactual Time Series Explanation Problem

Our goal is to provide counterfactual explanations for ML methods that operate on time series data. Given a black-box ML framework that takes multivariate time series as input and returns class probabilities, the explanations show which time series need to be modified, and how, to change the classification result of a sample in the desired way. For example, "if `MemFree` (a variable in an HPC performance analytics framework) was not decreasing over time, this time window would not be classified as a memory leak (an anomaly affecting the performance of an HPC system)." Given a sample and a class of interest, the counterfactual can be used to learn why the sample is *not* classified as the class of interest, because it is almost identical to the given sample but is classified as the class of interest.

We define the *counterfactual time series explanation problem* as follows. Given a sample to be explained and a class of interest, find (1) the *distractor*, which is a sample from the training set that is classified as the class of interest, and (2) which small set of time series to substitute from the distractor to the given sample such that the classification label changes to the class of interest. In the previous example of explaining a memory leak, the distractor would be a run from the training set without a memory leak where `MemFree` is not decreasing, and the set of variables would include `MemFree`. We assume a black box model for the ML classifier, thus having no access to the internal weights or gradients.

We use a single distractor instead of combining various distractors in one explanation, in order to (1) reduce the search space of possible substitutions, (2) guarantee a possible solution as long as the distractor is classified as the class of interest, and (3) improve the usability of CoMTE in cases where it is necessary to investigate other related data, e.g., logs in addition to time series data.

A. Problem Statement

In this paper, we represent multivariate time series classification models using $f(x) = y : \mathbb{R}^{m \times t} \rightarrow [0,1]^k$ where the model takes a time series over m variables of length t and returns the probability for k classes. We use the shorthand $f_c(x)$ as the probability for class $c \in [1,k]$. Our goal is to find the *optimum counterfactual explanation* for a given test sample x_{test} and a class of interest c. We define an optimum counterfactual explanation as a modified sample x' that is constructed using x_{test} such that (1) $f_c(x')$ is maximized and (2) the difference between x' and x_{test} is minimized. The class of modifications that we consider to construct x' are substitutions of entire variables from a distractor sample x_{dist}, chosen from the training set, to x_{test}. We define the difference between x' and x_{test} as the number of variables that were substituted to x_{test} in order to obtain x'.

The optimum counterfactual explanation can be constructed by finding x_{dist} among the training set and A, which minimizes

$$L\left(f, c, A, x'\right) = \left(1 - f_c(x')\right)^2 + \lambda ||A||_1, \tag{1}$$

where

$$x' = (I_m - A)x_{test} + Ax_{dist}, \tag{2}$$

λ is a tuning parameter, I_m is the $m \times m$ identity matrix, and A is a binary diagonal matrix where $A_{j,j} = 1$ if variable j of x_{test} is going to be swapped with that of x_{dist}, 0 otherwise. The problem of finding a counterfactual explanation, x_{dist} and A, that maximize $f_c(x')$ is analogous to the hitting set problem and thus NP-hard [26], so we focus on designing approximation algorithms and heuristics that generate acceptable explanations efficiently.

B. The Rationale for the Chosen Explanation

Explainability techniques targeting multivariate time series frameworks need to consider several properties that general-purpose explainability techniques do not. In most domains that work on time series analytics, time series data is more complex than traditional machine learning data sets by several aspects. A single sample is most often not explainable because of the volume of data it contains; therefore, sample-based methods often fail to provide comprehensible explanations. Furthermore, meanings and values of variables may not be straightforward to understand without additional information and comparison points. In order to address these challenges, our explainability approach is *simultaneously* sample- and feature-based. We provide a counterfactual sample from the training set and indicate which of the many time series in the sample need to be modified to have a different classification result. This results in an explanation that is easy to understand since it requires interpreting only a minimal number of variables. CoMTE makes interpreting easier by including examples of the same variable for both the distractor and the given test sample x_{test} in the explanation.

CoMTE chooses x_{dist} from the training set as part of the explanation instead of providing synthetic samples as in previous work [16], [17], as synthetically generated data might not result in meaningful explanations. For example, in HPC performance analytics, many variables represent resource utilization values that have constraints; e.g., the rate of change and the maximum/minimum values of certain counters are bounded by physical constraints of the CPUs, servers, and other components. Choosing a distractor x_{dist} from the training set guarantees that the time series in the explanation are feasible and realistic, and also enables the users to inspect the logs and other information besides time series that belong to the distractor sample.

IV. CoMTE: Counterfactual Explanations

We present a greedy search algorithm that generates counterfactual explanations for a black-box classifier and a faster optimization of this algorithm. Our goal is to find counterfactual explanations for a given test sample x_{test} and class of interest c. Recall that a counterfactual explanation is a minimal modification to x_{test} such that the probability of being part of c is maximized. CoMTE aims to find a minimal number of time series substitutions from the chosen distractor instance x_{dist} that will flip the prediction.

We relax the loss function L in Eqn. (1), using

$$L\left(f, c, A, x'\right) = \left((\tau - f_c(x'))^+\right)^2 + \lambda(||A||_1 - \delta)^+, \tag{3}$$

where x' is defined in Eqn. (2), τ is the target probability for the classifier, i.e., the threshold after which increasing $f_c(x)$ does not subjectively improve the explanations, δ is the threshold below which reducing the number of variables does not improve the explanations, and $x^+ = \max(0, x)$, which is the rectified linear unit (ReLU). ReLU is used to avoid penalizing explanations shorter than δ. We empirically set $\tau = 0.95$ and we set $\delta = 3$, as it is shown to be a suitable number of variables in an explanation [27].

Our explainability method operates by choosing multiple distractor candidates and, finding the best A matrix for each distractor. Among the different A matrices, we choose the matrix with the smallest loss value. We next present how CoMTE chooses distractors, and two different algorithms for choosing matrix A for a given distractor.

A. Choosing Distractors

As we seek to find the minimum number of substitutions, it is intuitive to start with distractors that are as similar to the test sample as possible. Hence, we choose distractor candidates by picking the n nearest neighbors of x_{test} in the training set that are correctly classified as the class of interest.

To quickly query for nearest neighbors, we keep all correctly classified training set instances in per-class *KD-Tree*s. The number of distractors to try out is a tuning parameter of our algorithm, chosen based on the acceptable running time. If the number of training instances is large, methods like random sampling or k-means can be used to reduce the number of instances before constructing the KD-tree. We use the KD-tree implementation in scikit-learn [28], and use Euclidean distance to select n-nearest neighbors, but other distance measures, such as dynamic time warping, can also be used.

Algorithm 1 Random Restart Hill Climbing

Require: Instance to be explained x_{test}, class of interest c, model f, distractor x_{dist}, loss function $L(f, c, A, x')$, max attempts, max iters, num restarts

for $i \in$ [0,num restarts] **do**
  Randomly initialize A; attempts $\leftarrow 0$; iters $\leftarrow 0$
  $x' \leftarrow (I_m - A)x_{test} + Ax_{dist}$
  $l \leftarrow L(f, c, A, x')$
  while attempts $\leq$ max attempts **and** iters $\leq$ max iters **do**
    iters++
    $A_{tmp} \leftarrow$ RandomNeighbor(A)
    $x' \leftarrow (I_m - A_{tmp})x_{test} + A_{tmp}x_{dist}$
    if $L(f, c, A_{tmp}, x') \leq l$ **then**
      attempts $\leftarrow 0$; $A \leftarrow A_{tmp}$; $l \leftarrow L(f, c, A, x')$
    else
      attempts++
    end if
  end while
end for

B. Sequential Greedy Algorithm

The greedy algorithm for solving the hitting set problem is shown to have an approximation factor of $log_2|U|$, where U is the union of all the sets [29]. Thus, one algorithm we use is the *Sequential Greedy Algorithm*. In each iteration, we first replace each variable in x_{test} by the corresponding variable from x_{dist} and choose the variable that leads to the highest increase in $f_c(x')$. After we replace a variable in x_{test}, we continue the greedy search with the remaining variable set until the prediction probability exceeds τ in Eqn. (3).

C. Random-Restart Hill Climbing

Although the greedy method is able to find a minimal set of explanations, searching for the best explanation by substituting the variables one by one can become slow for data sets with many variables. For a faster algorithm, we apply a derivative-free optimization algorithm hill-climbing to minimize the loss L in Eqn. (3).

Hill-climbing attempts to iteratively improve the current state by choosing the best successor state. The successor state is defined as adding or removing one variable to the set of substituted variables, i.e., A. This method only looks at the current state and possible states in the near future [30]. Since it is easy for hill-climbing to settle in local minima, we use *random restarting*, as shown in the *Random Restart Hill-Climbing* in Algorithm 1. This algorithm starts with a random initialization point for A and evaluates L for random neighbors of A until it finds a better neighbor. If a better neighbor is found, the search continues from the new A. We use the randomized optimization algorithms in `mlrose` package.

In some cases, hill climbing may contain variables in A that do not affect the target probability, $f_c(x')$. We check for this possible scenario by pruning the output, i.e., removing variables that have no effect. In rare cases, all variables may be pruned by this step. In this scenario, we use greedy search to find a viable solution.

D. How to Measure Good Explanations

There is no consensus yet on measures for comparing explainability methods in academia [31], [32]. In this work, we aim to provide several tenets of good explanations with our explainability method.

Faithfulness to the original model: An explanation is faithful to the classifier if it reflects the actual reasoning process of the model. It is a first-order requirement of any explainability method to accurately reflect the classifier's decision process and not mislead users [15]. To test the faithfulness of CoMTE, we explain a simple model with a known reasoning process and report the precision and recall of our explanations.

Comprehensibility by humans: Understanding an explanation should not require specialized knowledge about ML. A recent survey states that humans prefer only 1 or 2 causes instead of an explanation that covers the actual and full list of causes [27]. This is especially important for multivariate data sets such as HPC telemetry data, since each variable represents a different counter/gauge and understanding each counter typically requires expertise. To evaluate comprehensibility, we compare the *number of variables* that are returned in explanations, i.e., lower is better.

Robustness to changes in the sample: It is important for explanations to be robust [33], which ensures that users can trust the ML models and explanations. A good explanation would not only explain the given sample, but provide similar explanations for similar samples. A measure that have been used to evaluate robustness is the *local Lipschitz constant* $\mathcal{L}$ [34], [35], which is defined as follows for a given x_{test} instance:

$$\mathcal{L}(x_{test}) = \max_{x_j \in \mathcal{N}_k(x_{test})} \frac{\|\xi(x_{test}) - \xi(x_j)\|_2}{\|x_{test} - x_j\|_2}, \qquad (4)$$

where $\xi(x)$ is the explanation for instance x, and $\mathcal{N}_k(x)$ is the k-nearest neighbors of x_{test} in the training set. We modify the method of calculating this measure by using nearest neighbors from the training set instead of randomly generated samples because it is challenging to generate realistic random time series. The maximum constant is chosen because the explanations should be robust against the worst-case. Intuitively, the Lipschitz constant measures the ratio of change of explanations to changes in the samples. We use A as $\xi(x)$.

Generalizability of explanations: Each explanation should be generalizable to similar samples, i.e., the lessons learned from one explanation should apply to other predictions of the classifier; otherwise, humans using the explanations would not be able to gain an intuitive understanding of the model. Furthermore, for misclassifications, it is more useful for the explanations to uncover types of misclassifications instead of a single mishap. We measure generalizability by applying an explanation's substitutions to other samples. If the same time series substitutions from the same distractor can flip the prediction of other samples, the explanation is generalizable.

V. Experimental Setup

This section describes the time series data sets and ML frameworks we use to evaluate CoMTE as well as the baseline explainability methods we implement for comparisons.

A. Data Sets

We use four high-dimensional multivariate time series data sets: three HPC system telemetry data sets and a motion classification data set. All three HPC system telemetry data sets contain data collected using Lightweight Distributed Metric Service (LDMS) [3]. LDMS collects data from the `/proc` filesystem as well as PAPI [36] and Cray performance counters. One sample always corresponds to the data collected from a single compute node of an application run.

HPAS: The goal of collecting this data set is to diagnose performance variations on HPC applications using system telemetry data collected via LDMS. We run 8 applications, Cloverleaf, CoMD, miniAMR, miniGhost, miniMD, Kripke, SW4lite, and MILC [37]–[40] on Voltrino[2], each on 4 nodes, with and without performance anomalies. We use the cpuoccupy, memorybandwidth, cachecopy, memleak, memeater, and netoccupy anomalies from the HPC performance anomaly suite (HPAS) [5]. We collect a total of 617 samples (350 training, 267 test) with 839 time series each. We extract 45-second time windows with 30-second overlaps from each sample.

Taxonomist: The goal of this data set is to classify the different applications running on HPC systems using LDMS data. A previously released data set [41] contains 11 different applications. We use all of the data, which has 4728 samples (3776 training, 952 test) with 563 time series each.

Cori: We collect this data set from Cori[3] to test our explainability method with data from large-scale systems and applications. The goal of this data set is to again use LDMS telemetry data to classify applications. Cori is a Cray XC40 supercomputer with 12,076 nodes. We run 6 applications, LAMMPS, QMCPACK, HACC, NEKBone, miniAMR, and HPCG [42]–[45] on 64 nodes for 15-30 minutes. We collect a total of 9216 samples (7373 training, 1843 test) with 819 time series each.

NATOPS: This previously released data set is from the motion classification domain [46][4]. We chose this data set because of the relatively high number of time series per sample, compared to other time series data sets commonly used in the ML domain. The NATOPS data contains a total of 24 time series representing the X, Y and Z coordinates of the left and right hand, wrist, thumb, and elbows, as captured by a Kinect 2 sensor. We keep the original training and test set of 180 samples each, with 50 second time windows.

B. Machine Learning Techniques

We evaluate our explainability techniques by explaining 3 different ML pipelines.

Statistical Feature Extraction + Logistic Regression: The logistic regression classifier is inherently interpretable, so we use this pipeline for simpler tests of our explanations in experiments where we need a ground truth for explanations. For input feature vector x the logistic regression model we use calculates the output $y = S(w \cdot x)$, where $S(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. Thus, the classifier only learns the weight vector w during training[5]. Any feature x_i for which the corresponding weight w_i is zero has no effect on the classification. Similarly, features can be sorted based on their impact on the classifier decision using $|w_i|$. We extract 11 statistical features: the minimum, maximum, mean, standard deviation, skew, kurtosis, 5^{th}, 25^{th}, 50^{th}, 75^{th} and 95^{th} percentiles from each of the time series, which are commonly used in previous work [7], [8], [12].

Statistical Feature Extraction + Random Forest: This technique represents a commonly used pipeline to classify time series data for failure prediction, diagnose performance variability, or classify applications [7], [8], [10], [47]. For example, Tuncer et al. [7] diagnose performance anomalies by extracting statistical features from system telemetry data and using random forests to classify the type or absence of anomalies. We extract the same 11 statistical features as the logistic regression method. Then, we train scikit-learn's random forest classifier using these features.

Autoencoder: Borghesi et al. have proposed an autoencoder architecture for anomaly detection using HPC time series data [9]. We implement this architecture, which learns a compressed representation of "healthy" data, and samples with high mean reconstruction error are classified as anomalous. In order to convert the mean error, which is a positive real number, to class probabilities between 1 and 0, we subtract the chosen threshold from the error and use the sigmoid function.

C. Baseline Methods

We compare our explainability method with popular explainability methods, LIME [15], SHAP [16], as well as Random, which picks a random subset of the variables as the explanation.

LIME operates by fitting an interpretable linear model to the classifier's predictions of random data samples, weighted based on their distance to the test sample. Random data is generated based on the range observed in the training set. In our evaluation, we use the open-source LIME implementation[6]. Another challenge with LIME is that it requires the number of features in the explanation as a user input. In our experiments, we leave the value as the default (10), and we first use CoMTE to find how many features are sufficient, then trim LIME's output to match that number.

SHAP operates by calculating feature importance values using model parameters; however, since we do not have access to model parameters, we use the open source KernelSHAP

[2] www.sandia.gov/asc/computational_systems/HAAPS.html

[3] docs.nersc.gov/systems/cori/

[4] We use the version found in UCR time series classification repository at www.timeseriesclassification.com.

[5] Other formulations of logistic regression include a b term such that $y = S(w \cdot x + b)$, but we omit this for better interpretability.

[6] www.github.com/marcoctr/lime

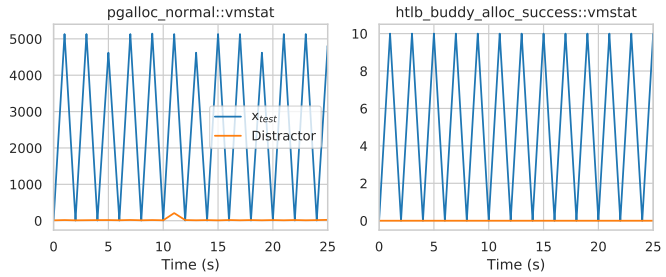

Fig. 1. The explanation of CoMTE for a correctly classified time window with the "memory leak" label. CoMTE provides two variables as an explanation to change the classification label from "memory leak" to "healthy." The variables are shown in the y-axes and the variable names are above the plots.

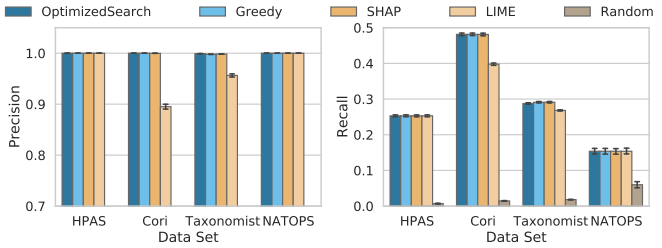

Fig. 2. Precision and recall of the explanations for a classifier with known feature importances. CoMTE (OptimizedSearch and Greedy), and SHAP have perfect precision. LIME has lower precision for Cori and Taxonomist data sets, which indicates that although some features have no impact on the classifier decision, they are included in the LIME explanations. The low recall indicates that not every feature is used in every local decision.

implementation[7], which estimates SHAP values without using model weights.

Neither SHAP nor LIME can be directly applied to time series models, so we apply them to interpret the classifier used in feature-based frameworks.

VI. Evaluation

In this section, we evaluate CoMTE and compare it with other explainability methods based on qualitative comparisons and the metrics described in Sec. IV-D. We aim to answer several questions: (1) Are the explanations minimal? (2) Are the explanations faithful to the original classifier? (3) Are the explanations robust, or do we get different explanations based on small perturbations of the input? (4) Are the explanations generalizable to different samples? (5) Are the explanations useful in understanding the classifier?

Qualitative Evaluation Our first-order evaluation is to use CoMTE and the baselines to explain a realistic classifier. Similar to the framework proposed by Tuncer et al. [7], we use the random forest classifier with feature extraction and the HPAS dataset, which includes different types of performance anomalies. We choose the "memory leak" anomaly from the HPAS data set. Our goal is to understand better the classifier's understanding of the "memory leak" anomaly. After training the random forest pipeline, we choose a correctly classified time window with the memory leak label as x_{test}, and the "healthy" class as the class of interest. We run CoMTE, LIME, and SHAP with the same x_{test} and compare the results.

Our explanation contains two time series, and is shown in Fig. 1. The first variable to be substituted is `pgalloc_normal` from `/proc/vmstat`, which is a counter that represents the number of page allocations per second. It is immediately apparent that the nodes with memory leaks perform many memory allocations and act periodically.

The second variable in Fig. 1 is `htlb_buddy_alloc_successes`, which shows the number of successful huge page allocations. Memory leaks do not need to cause huge page allocations in a system with fragmented memory, where huge page allocations would fail. This indicates that our training set is biased towards systems with less fragmented memory, most probably because our benchmarks are all short-lived.

The baseline methods, LIME and SHAP, can only explain the random forest classifier, and they can not provide explanations that relate to the time series. The SHAP explanation contains 187 features with very similar SHAP values, and it is difficult to decide how many features are sufficient for a good explanation. Similarly, LIME requires the number of features in the explanation as a user input. Finally, it is left to the user to interpret the values of different features, e.g., the 75th percentile of `pgalloc_normal` was 5,127; however, this does not inform the user of normal values for this counter, or whether it was too high or too low.

It is important to note that both LIME and SHAP use randomly generated data for the explanations. In doing so, these methods assume that all of the features are independent variables; however, many features are in fact dependent, e.g., features generated from the same variable. Without knowledge of this, these random data generation methods may test the classifier with infeasible synthetic runs, e.g., runs where the 75th percentile of the one variable is lower than the 50th percentile of the same variable. CoMTE does not generate synthetic data and uses the whole time series instead of just the statistical features, so it is not affected by this.

Comprehensibility We measure comprehensibility using the number of variables in the explanation. CoMTE returns 2 time series for the qualitative evaluation example in Fig. 1, and in most cases the number of time series in our explanations is below 3; however, for some challenging cases, it can reach up to 10. SHAP explanations typically have hundreds of features for HPC time series data. LIME requires the number of features as an input; however, it does not provide any guidelines for deciding this value.

Faithfulness We test whether the explainability methods actually reflect the decision process of the models, i.e., whether they are faithful to the model. For every data set, we train a logistic regression model with $L1$ regularization. We change the $L1$ regularization parameter until less than 10 variables are used by the classifier. The resulting classifier uses features derived from 5 variables for HPAS and Cori data, 9 for NATOPS, and 8 for Taxonomist. Because we know the used variables, we

[7]www.github.com/slundberg/shap

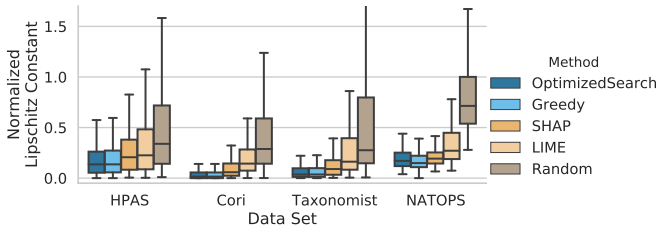

Fig. 3. Robustness of explanations to changes in the test sample. CoMTE (OptimizedSearch and Greedy) is the most robust to small changes in the input, resulting in more predictable explanations and a better user experience. The Lipschitz constant is normalized to be comparable between different data sets. A Lower value indicates better robustness.

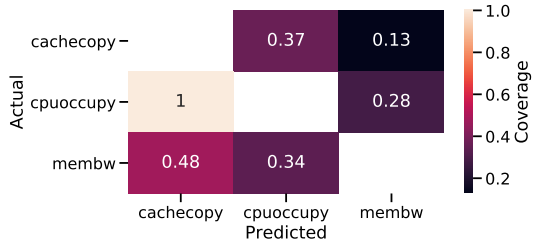

Fig. 4. We report coverage for each (true class, predicted class) pair. For the cpuoccupy runs misclassified as cachecopy, every explanation is applicable to every other sample with the same misclassification.

can rank the explanations based on precision and recall.

We acquire explanations for each sample in the test set and show the average precision and recall. To ensure that the other explainability methods are not at a disadvantage, we first run the greedy search method and get the number of variables in the explanation. Then, we get the same number of variables from each method.

The results in Fig. 2 show that both our method and SHAP have perfect precision. Recall values of the explanations are lower than one because not every feature in the classifier is effective for every decision. Notably, LIME has low precision for the Cori and Taxonomist data sets, which indicates that there may be features in LIME explanations that are actually not used by the classifier at all. This outcome could be due to the randomness in the data sampling stage of LIME.

Robustness For robustness, we calculate the Lipschitz constant given in Eqn. (4) for each test sample and show average results in Fig. 3. According to the results, CoMTE is the most robust explainability technique. One reason is that CoMTE does not use random data generation for explanations, which reduces the randomness in the explanations. For the NATOPS data set, the greedy method has better robustness compared to the optimized search, because the greedy method inspects every variable before generating an explanation, thus finds the best variable, while the optimized search stops after finding a suitable explanation.

Generalizability We test whether our explanations for one x_{test} are generalizable to other samples. We use the HPAS data set and random forest classifier with feature extraction, where 3 classes are confused with each other. For each misclassified test instance, we get an explanation and apply the same variable substitutions using the same distractor to other test samples with the same (true class, predicted class) pair.

We report the *coverage*, which is the percentage of misclassifications that the explanation applies (i.e., successfully flips the prediction for) to the selected (true, predicted) class pair in Fig. 4. According to our results, on average, explanations for one mispredicted sample are applicable to over 40% of similarly mispredicted samples. This shows that users do not need to manually inspect the explanation for every misprediction, and instead, they can obtain a general idea of the classifiers error characteristics from a few explanations, which is one of the goals of explainability.

VII. Conclusion and Future Work

This paper, for the first time, investigated explainability for ML frameworks that use multivariate time series data sets. Multivariate time series data is widely used in many scientific and engineering domains, and ML-based HPC analysis and management methods that show a lot of promise to improve HPC system performance, efficiency, and resilience. Explainability is an important requirement for any ML framework that seeks widespread adoption in real-world production systems.

We defined the counterfactual time series explainability problem and presented CoMTE, a method to can generate feasible explanations. We also demonstrated the use of CoMTE to explain various frameworks, and showed that our method outperforms existing explainability methods in terms of comprehensibility and robustness. In the future, we plan to explore approximation algorithms for the optimization problem that may lead to better explanations in a shorter time. Additionally, we plan to develop support for minimizing the target probability for the class of interest, $f_c(x')$, as opposed to maximizing, which can provide novel explanations for multiclass classification tasks.

Acknowledgment

This work has been partially funded by Sandia National Laboratories. Sandia National Labs is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525.

References

[1] R. Assaf and A. Schumann, "Explainable deep neural networks for multivariate time series predictions," in *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 7 2019, pp. 6488–6490.

[2] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, p. 6085, Apr 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-24271-9

[3] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014, pp. 154–165.

[4] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, D. Gregori, S. Tinti, C. Gianfreda, and P. Altoè, "The D.A.V.I.D.E. big-data-powered fine-grain power and performance monitoring support," in *International Conference on Computing Frontiers*, 2018, p. 303–308.

[5] E. Ates, Y. Zhang, B. Aksar, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "HPAS: An HPC performance anomaly suite for reproducing performance variations," in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP 2019, no. 40, 2019.

[6] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proceedings of the 5th European Conference on Computer Systems*, 2010, pp. 111–124.

[7] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Diagnosing performance variations in HPC applications using machine learning," in *International Supercomputing Conference (ISC-HPC)*, 2017, pp. 355–373.

[8] ——, "Online diagnosis of performance variation in HPC systems using machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 883–896, April 2019.

[9] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, p. 634–644, Oct 2019.

[10] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data mining-based analysis of HPC center operations," in *2017 IEEE International Conference on Cluster Computing*, 2017, pp. 766–773.

[11] C.-T. Yang, K.-C. Lai, and H.-Y. Tung, "On construction of a well-balanced allocation strategy for heterogeneous multi-cluster computing environments," *The Journal of Supercomputing*, vol. 56, no. 3, pp. 270–299, Jun 2011.

[12] E. Ates, O. Tuncer, A. Turk, V. J. Leung, J. Brandt, M. Egele, and A. K. Coskun, "Taxonomist: Application detection through rich monitoring data," in *Euro-Par 2018: Parallel Processing*. Cham: Springer International Publishing, 2018, pp. 92–105.

[13] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the International Conference on Machine Learning*, vol. 70, Aug 2017, pp. 1885–1894.

[14] A. Dhurandhar, P.-Y. Chen, R. Luss, C.-C. Tu, P. Ting, K. Shanmugam, and P. Das, "Explanations based on the missing: Towards contrastive explanations with pertinent negatives," in *Advances in Neural Information Processing Systems (NeurIPS) 31*, 2018, pp. 592–603.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, 2016, p. 1135–1144.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *NeurIPS 30*, 2017, pp. 4765–4774.

[17] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the GDPR," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[18] A. Akula, S. Wang, and S.-C. Zhu, "Cocox: Generating conceptual and counterfactual explanations via fault-lines," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, p. 2594–2601, 2020.

[19] M. Pawelczyk, K. Broelemann, and G. Kasneci, "Learning model-agnostic counterfactual explanations for tabular data," in *Proceedings of The Web Conference 2020*, 2020, pp. 3126–3132.

[20] D. Martens and F. Provost, "Explaining data-driven document classifications," *MIS Q.*, vol. 38, no. 1, p. 73–100, Mar. 2014.

[21] E. Delaney, D. Greene, and M. T. Keane, "Instance-based counterfactual explanations for time series classification," arXiv:2009.13211 [cs.LG], 2020.

[22] I. Karlsson, J. Rebane, P. Papapetrou, and A. Gionis, "Explainable time series tweaking via irreversible and reversible temporal transformations," in *IEEE International Conference on Data Mining*, 2018, pp. 207–216.

[23] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining machine learning classifiers through diverse counterfactual explanations," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2020, p. 607–617.

[24] A.-H. Karimi, G. Barthe, B. Balle, and I. Valera, "Model-agnostic counterfactual explanations for consequential decisions," in *International Conference on Artificial Intelligence and Statistics*, 2020, pp. 895–905.

[25] R. Poyiadzi, K. Sokol, R. Santos-Rodriguez, T. De Bie, and P. Flach, "Face: Feasible and actionable counterfactual explanations," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '20, 2020, p. 344–350.

[26] E. Ates, B. Aksar, V. J. Leung, and A. K. Coskun, "Counterfactual explanations for machine learning on multivariate time series data," arXiv:2008.10781 [cs.LG], 2020.

[27] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, p. 1–38, Feb 2019.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct 2011.

[29] K. Chandrasekaran, R. Karp, E. Moreno-Centeno, and S. Vempala, "Algorithms for implicit hitting set problems," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011, p. 614–629.

[30] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[31] Z. C. Lipton, "The mythos of model interpretability," *Queue*, vol. 16, no. 3, p. 31–57, Jun. 2018.

[32] P. Schmidt and F. Biessmann, "Quantifying interpretability and trust in machine learning systems," arXiv:1901.08558 [cs.LG], Jan 2019.

[33] H. Lakkaraju, N. Arsov, and O. Bastani, "Robust and stable black box explanations," *Proceedings of the International Conference on Machine Learning*, 2020.

[34] D. Alvarez-Melis and T. S. Jaakkola, "On the robustness of interpretability methods," arXiv:1806.08049 [cs.LG], 2018.

[35] D. Alvarez-Melis and T. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *NeurIPS 31*, 2018, pp. 7775–7784.

[36] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing*, 2010, pp. 157–173.

[37] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[38] A. Kunen, T. Bailey, and P. Brown, "Kripke-a massively parallel transport mini-app," Lawrence Livermore National Laboratory, Tech. Rep., 2015.

[39] B. Sjogreen, "SW4 final report for iCOE," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2018.

[40] The MIMD Lattice Computation (MILC) Collaboration, "MILC benchmark application," http://www.physics.utah.edu/~detar/milc/, 2016.

[41] E. Ates, O. Tuncer, A. Turk, V. J. Leung, J. Brandt, M. Egele, and A. K. Coskun, "Artifact for Taxonomist: Application Detection through Rich Monitoring Data," https://doi.org/10.6084/m9.figshare.6384248.v1, Aug 2018.

[42] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, p. 1–19, 1995.

[43] J. Kim, A. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley, and et al., "Qmcpack: An open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules, and solids," *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, May 2018.

[44] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, "Hacc: Extreme scaling and performance across diverse architectures," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–10.

[45] J. Dongarra, M. A. Heroux, and P. Luszczek, "A new metric for ranking high-performance computing systems," *National Science Review*, vol. 3, no. 1, p. 30–35, Mar 2016.

[46] N. Ghouaiel, P.-F. Marteau, and M. Dupont, "Continuous pattern detection and recognition in stream - a benchmark for online gesture recognition," *International Journal of Applied Pattern Recognition*, vol. 4, no. 2, 2017.

[47] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for GPU error prediction in a large scale HPC system," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2018, pp. 95–106.