SAND2020-11064C

Explainable Machine Learning Frameworks for Managing HPC Systems

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

***Abstract*—Recent research on supercomputing proposes a variety of machine learning frameworks that are able to detect performance variations, find optimum application configurations, perform intelligent scheduling or node allocation, and improve system security. Although these goals align well with HPC systems' needs, barriers such as the lack of user trust or the difficulty of debugging need to be overcome to enable the widespread adoption of such frameworks in production systems. This paper evaluates a new *counterfactual time series explainability method* and compares it against state-of-the-art explainability methods for supervised machine learning frameworks that use multivariate HPC system telemetry data. The *counterfactual time series explainability method* outperforms existing methods in terms of comprehensibility and robustness. We also show how explainability techniques can be used to debug machine learning frameworks and gain a better understanding of HPC system telemetry data.**

***Index Terms*—high performance computing, monitoring, explainability, interpretability, machine learning, time series**

I. Introduction

High performance computing (HPC) systems produce terabytes of instrumentation data per day in the form of logs, metrics, and traces. HPC monitoring frameworks organize system-wide resource utilization metrics as multivariate time series, each time series representing a different resource statistic such as network packet counts, CPU utilization, or memory statistics. Thousands of metrics can be collected per node and each metric is sampled on intervals of seconds to minutes [1]–[3]. Analyzing this data is invaluable for management and debugging [1], [4], but extensive manual analysis of this data is not feasible.

Recently, researchers have begun using machine learning (ML) to help understand HPC system telemetry data and gain useful insights. ML approaches can process vast volumes of data and, in addition, ML-based frameworks benefit from the versatility of models that generalize to various systems and even previously unseen situations. ML frameworks have been shown to diagnose performance variations [5]–[8], improve scheduling [9] or improve system security by detecting unwanted or illegal applications on HPC systems [10] using multivariate time series data. In this paper, we refer to such ML frameworks that use multivariate time series HPC system telemetry data as *HPC ML frameworks*.

The growth in ML frameworks indicates that ML is becoming a key part of HPC system analysis and management. Widespread adoption of such methods in production systems, however, depends on solving important challenges in accountability, accuracy, and explainability. These frameworks commonly have a taciturn nature, e.g., reporting only the final classification result "`memory leakage on nodeid:599`" without providing reasoning relating to the underlying data. Furthermore, black-box ML models can perform several data transformations before classification is achieved, and thus are often difficult to understand. The black-box design of these frameworks creates a multitude of disadvantages such as degrading user trust and making debugging challenging.

Researchers have proposed a variety of methods to resolve this ML *explainability* problem. Some classifiers such as linear regression and decision trees are *inherently interpretable*. However, in our experience, even decision trees can become overly complex when trained with HPC data sets. There are also methods that explain black-box classifiers [11], based on whether they explain a single prediction or the complete classifier. However, most of the current explainability approaches are not designed for multivariate time series data and do not produce sufficiently clear explanations when it comes to explaining the HPC ML frameworks.

There are several reasons that existing explainability methods fail to provide satisfactory explanations for HPC time series data. One differentiating factor is the complexity of the data. Figure 1 shows a *sample* where 199 metrics are obtained via the system telemetry during an application run. Current sample-based methods [12] are built on the premise that one sample is self-explanatory and users can visually differentiate between two samples; however, additional sample HPC time series data with hundreds of metrics similar to the one shown in Fig. 1 is not a sufficient explanation. On the other hand, existing feature-based methods [13], [14] provide a collection of features and require users to know normal and abnormal values for features, as well as their meanings.

‡The authors have contributed equally to this work.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

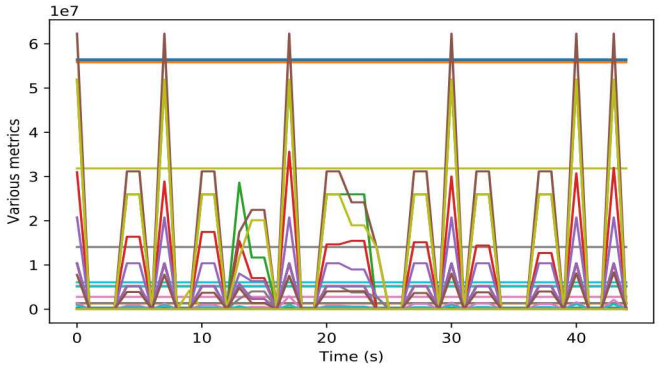

Fig. 1. A 45-second time window sample from a single node of a multi-node application execution on Voltrino, a Cray XC30m supercomputer with 56 nodes.

In this paper, we evaluate a novel *time series explainability method* [15] that provides *counterfactual* explanations for time series and compare it against state-of-the-art explainability methods, LIME [13] and SHAP [14], using various HPC system telemetry datasets. 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 these metrics were different in the given sample, the classification label would have been different." The *counterfactual time series explainability method* generates explanations by selecting a minimum number of time series from the training set and substituting them in the sample under investigation to obtain different classification results. By analyzing a limited range of substituted metrics, system administrators or users may understand the expected behavior of the model. It is also possible to use these explanations to debug misclassifications, understand how the classifier makes decisions, provide adaptive dashboards that explain critical metrics from ongoing application runs and learn about the essence of normal or anomalous behavior of a system.

Our specific contributions are as follows:

- Demonstration of existing general-purpose explainability methods and how they are inadequate to explain HPC time series frameworks (§ V-A),
- demonstration of the *counterfactual time series explainability method* with several ML-based HPC frameworks and data sets (§ IV, V),
- comparison of the *counterfactual time series explainability method* with state-of-the-art explainability methods using a set of novel and standard metrics. The *counterfactual time series explainability method* generates comprehensible explanations for HPC time series data, and performs better than baselines in terms of *faithfulness* and *robustness* (§ V).

II. Background and Related Work

In this section, we discuss the use of ML in HPC, the need for explainable AI methods, existing explainability methods and their limitations.

A. Machine Learning in HPC

As the scale of HPC systems increased to millions of cores and thousands of nodes, automated techniques have become essential to ensure the reliability, security, and efficiency of the systems. Consequently, researchers have developed many frameworks, techniques, and algorithms to automate HPC system management using ML. Tuncer et al. design a supervised learning framework that extracts statistical features from the time series, and uses models such as the random forest to classify performance variations [5], [6]. Borghesi et al. describe an autoencoder-based unsupervised framework that detects performance anomalies using the anomalous behavior in system telemetry data [7].

ML has been applied to HPC time series data for other novel purposes as well. Ates et al. describe a feature extraction and random forest based framework that classifies applications executing on the nodes, and demonstrate successful detection of unwanted applications such as cryptocurrency miners [10]. Klinkenberg et al. describe a supervised learning framework that extracts statistical features and uses a random forest classifier to detect critical node failures before they happen [8].

B. Existing Explainability Methods

The field of explainability aims to generate explanations for existing black-box models and we need to ensure that administrators and users are able to make sense of the ML classifications and predictions, and accordingly iterate on their design, fix possible bugs, or determine additional probes for data collection.

We use the helpful classification of Arya et al. when discussing the existing literature [11]. We focus on explainable models and local sample- and feature-based explanations for black-box models.

Explainable models learn a model that is inherently understandable by untrained operators. They include simple models like logistic regression and decision trees and newer models like CORELS, which learns minimal rule lists that are easy to understand [16]. CORELS fails to learn a usable model and decision trees become too complex when HPC time series data is applied.

Sample-based explanations provide samples that explain decisions by giving supportive or counter examples, or prototypes of certain classes. Koh and Liang use influence functions to find training set samples that are most impactful for a specific decision [17]. Contrastive explanations method (CEM) provides a synthetic sample with a different classification result that is as similar to the input sample as possible, while using autoencoders to ensure the generated sample is realistic [12]. Sample-based methods do not address the inherent complexity of HPC data, since they focus on tabular data or images where the test sample and the explanation sample are easy to compare manually. HPC time series data with thousands of time series per sample is challenging for users to compare and contrast without additional computing.

Feature-based explanations highlight certain features that are impactful during classification. Global feature-based expla-

nations for some classifiers such as random forests and logistic regression use the learned weights of the classifiers [18]. Local feature-based explanations include LIME, which fits a linear model to classifier decisions for samples in the neighborhood of the sample to be explained [13]. SHAP derives additive Shapley values for each feature to represent the impact of that feature [14]. These feature-based models do not support using time series directly; however, many HPC frameworks use feature transformations and feature-based explanations can explain these models' classifications in terms of the features they use. In this case, interpreting the meaning of complex features is left to the user.

Time series specific explanations have also been foci of research. Schegel et al. evaluate different general purpose explainability methods on time series [19]. Karlsson et al. propose a method to synthetically generate sample-based explanations for time series [20]. All of these methods operate on univariate time series and do not address the problem of explaining multivariate time series. Assaf and Roy propose a method to extract visual saliency maps for multivariate time series [21]; however, their method is specific to deep neural networks. Furthermore, saliency maps lose their simplicity when scaled to hundreds or thousands of time series.

Counterfactual explanations have been used to explain ML models. Wachter et al. are among the first to use the term "counterfactual" for ML explanations [22]. DiCE is an open source counterfactual explanation method for black-box classifiers [23]. To the best of our knowledge, there are no existing methods to generate counterfactual explanations for high-dimensional multivariate time series. Furthermore, many of these methods generate synthetic data without specifically addressing the difficulties of generating highly-correlated multivariate time series data, which is the type of data used in HPC ML frameworks.

III. Selected Explainability Methods for HPC ML Frameworks

A. Counterfactual Time Series Explainability Method

We use the *counterfactual time series explanation method* proposed in our recent work to generate explanations for a black-box ML model that uses multivariate time series data as input [15]. Our method operates by finding a *minimum set of feature substitutions* from a *distractor*, x_{dist}, that will change the classification result of a test sample, x_{test}, such that the resulting sample is predicted as the class of interest, c. At a high level, the method provides an explanation like the following, "if feature X was not exceeding Y over time, x_{test} would not be classified as memory leakage". The format of the counterfactual explanation is identical to the format of the x_{test} with the same number of time series and same window length. The optimum counterfactual explanation can be constructed by finding x_{dist} among the training set and A which minimizes

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

where

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

τ is the target probability for the classifier, δ is the desired number of features in an explanation, λ is a tuning parameter, I_m is the $m \times m$ identity matrix, and $x^+ = \max(0, x)$, which is the rectified linear unit (ReLU). ReLU is used to avoid penalizing explanations shorter than δ. We set $\delta = 3$, as it is shown to be a suitable number of features in an explanation [24].

A is a binary diagonal matrix where $A_{j,j} = 1$ if metric j of x_{test} is going to be swapped with that of x_{dist}, and 0 otherwise. While forming A, we replace each feature in x_{test} by the corresponding feature from x_{dist}. In each iteration, we choose the feature that leads to the highest increase in the prediction probability. After we replace a feature in x_{test}, we continue the greedy search with the remaining feature set until the prediction probability exceeds τ, which is the predefined threshold that will flip the prediction when it is exceeded.

B. LIME

LIME stands for local interpretable model-agnostic explanations [13]. LIME operates by fitting an interpretable linear model to the classifiers predictions of random data samples. The samples are weighted based on their distance to the test sample, which generates local explanations. When generating samples, LIME produces samples within the range found in the training set. In our evaluation, we use the open-source LIME implementation.

LIME does not directly apply to time series as it operates by sampling the classifier using randomly generated data. Randomly generating HPC time series data while still following the physical constraints in the data as well as preserving representative behavior over time is a challenging open problem. In our evaluation, we apply LIME to frameworks that perform feature extraction, and LIME interprets the classifier that takes features as input. LIME requires the number of features in the explanation as a user input. Generally, it is hard for users to know how many features in an explanation are adequate. In our experiments, we use the number of metrics in the *counterfactual time series explanation method*'s explanation as LIME's input.

C. SHAP

The Shapley additive explanations (SHAP), presented by Lundberg and Lee [14], propose 3 desirable characteristics of explanations, local accuracy, missingness and consistency.

Using the model parameters, SHAP calculates feature importance values. We use the open-source KernelSHAP implementation, which we refer to as SHAP in the remainder of the paper. SHAP also suffers from one of the limitations of LIME; it is not directly applicable to time series, so we apply SHAP to frameworks that perform feature extraction. SHAP does not require the number of features in the explanation as an input.

*A sample from the dataset, which results in a new classification result.

IV. Experimental Setup

In this section, we describe the data sets, comparison metrics and ML frameworks we use to evaluate the *counterfactual time series explainability method* [15] along with other baseline explainability methods.

A. Comparison Metrics

There is no consensus on metrics for comparing explainability methods in academia [25], [26]. In this work, we use 4 metrics proposed in our recent work to compare the *counterfactual time series explanation method* with baseline methods [15].

Faithfulness to the original model: An explanation is faithful to the classifier if it reflects the actual reasoning process of the model and it is a first-order requirement of any explainability method [13]. To test the faithfulness of the methods, we explain a simple model with a known reasoning process and report the precision and recall of the explanations.

Comprehensibility by human operators: According to a survey by Miller [24], humans prefer only 1 or 2 causes instead of an explanation that covers the actual and full list of causes and comprehension of an explanation should not require advanced ML knowledge. This is particularly important for HPC time series data, because each time series represents a different metric and each metric usually needs research to understand the meaning. Thus, to evaluate comprehensibility, we compare the *number of time series* that are returned in explanations by different explainability methods.

Robustness to changes in the sample: A good explanation would not only explain the given sample, but provide similar explanations for similar samples [27], [28]. A method that have been used to measure robustness is the *local Lipschitz constant* [27]. Intuitively, the Lipschitz constant measures the ratio of change of explanations to changes in the samples.

Generalizability of explanations: Each explanation should be generalizable to similar samples; otherwise, human operators 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 classes of misclassifications instead of a single mishap. We measure generalizability by applying an explanation's substitutions to other samples. If the same metric substitutions from the same distractor can flip the prediction of other samples, that means the explanation is generalizable.

B. Data Sets

We use three multivariate HPC system telemetry time series data sets. For all data sets, we normalize the data such that each time series is between 0 and 1 across the training set. We use the same normalization parameters for the test set. We use normalized data to train classifiers, and provide normalized data to the explainability methods. However, the real values of metrics are meaningful to users (e.g., CPU utilization %), so we provide un-normalized data in the explanations given to users and our figures in the paper.

HPAS data set: We use the HPC performance anomaly suite (HPAS) [3] to generate synthetic performance anomalies on HPC applications and collect time series data using LDMS [1]. We run our experiments on Voltrino at Sandia National Laboratories, a 24-node Cray XC30m supercomputer with 2 Intel Xeon E5-2698 v3 processors and 125 GB of memory per node [29]. We run the Cloverleaf, CoMD, miniAMR, miniGhost, and miniMD from the Mantevo Benchmark Suite [30], proxy applications Kripke [31] and SW4lite [32], and MILC [33] which represents part of the codes written by the MIMD Lattice Computation collaboration. We run each application on 4 nodes, with and without anomalies. We use the cpuoccupy, memorybandwidth, cachecopy, memleak, memeater and netoccupy anomalies from HPAS.

Each sample has 839 time series, from the `/proc` filesystem and Cray network counters. We take a total of 617 samples for our data set, and we divide this into 350 training samples and 267 test samples. One sample corresponds to the data collected from a single node of an application run. After this division, we extract 45 second time windows with 30 second overlaps from each sample.

Cori data set: We collect telemetry data from Cori [34] to test explainability methods' performance with real applications in a large-scale system. The goal of this data set is to use monitoring data to classify applications. Cori is a Cray XC40 supercomputer with 12,076 nodes. We run our applications in compute nodes with 2 16-core Intel Xeon E5-2698 v3 processors and 128 GB of memory. We run 6 applications on 64 nodes for 15-30 minutes: 3 real applications, LAMMPS [35], a classical molecular dynamics code with a focus on materials modeling, QMCPACK [36], an open-source continuum quantum Monte Carlo simulation code, HACC [37], an open-source code uses N-body techniques to simulate the evolution of the universe; 2 proxy applications, NEKBone and miniAMR from ECP Proxy Apps Suite [38]; and HPCG [39] benchmark, which is used to rank the TOP500 computing systems.

We collect a total of 9216 samples and we divide this sample set into 7373 training and 1843 test samples. Each sample represents the data collected from a single node of an application run and has 819 time series collected using LDMS from the `/proc` filesystem and PAPI [40] counters.

Taxonomist data set: This data set, released by Ates et al. [41], was collected from Voltrino, a Cray XC30m supercomputer, using LDMS. The data set contains runs of 11 different applications with various input sets and configurations, and the goal is to classify the different applications.

We use all of the data, which has 4728 samples. We divide it into 3776 training samples and 952 test samples. Each sample has 563 time series. Each sample represents the data collected from a single node of an application run.

C. Machine Learning Techniques

We evaluate the explainability techniques by explaining 3 different ML pipelines that represent different HPC frameworks proposed by researchers.

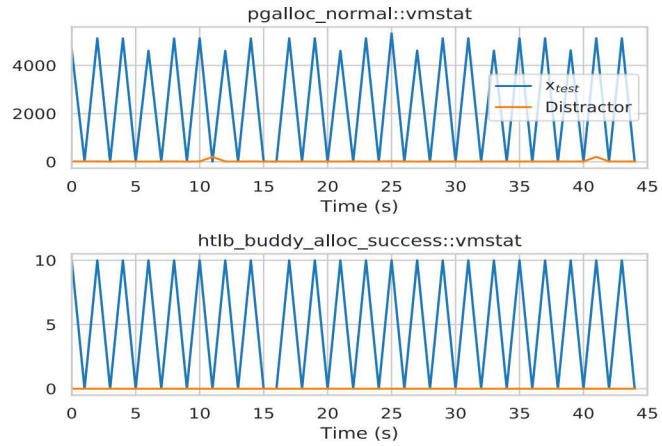

Fig. 2. The explanation of the *counterfactual time series explanation method* for correctly classified time window with the "memleak" label. The method provides two metrics as an explanation to change classification label from "memleak" to "healthy." The metrics are shown in the y-axes and the metric names are above the plots.

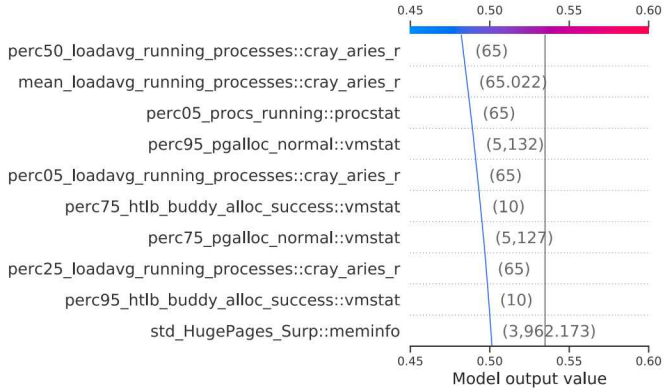

Fig. 3. The explanation of SHAP for a correctly classified time window with the "memleak" label. SHAP provides 187 features with non-zero SHAP values to explain the characteristics of "memleak" anomaly of which we show the top 10 in the figure. It is very challenging to understand how many features are sufficient for an explanation, whether these features are relevant to "memleak" or other anomalies, or how to interpret the feature values in the explanation.

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 [5], [6], [8], [10]. For example, Tuncer et al. [5] diagnose performance anomalies at runtime by collecting time series data with different types of anomalies, and train a random forest to classify the type, or absence, of anomalies using statistical features extracted from time series.

We extract 11 statistical features including 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. Then, we train scikit-learn's random forest classifier based on these features.

Autoencoder: Borghesi et al. has proposed an autoencoder architecture for anomaly detection using HPC time series data [7]. The autoencoder is trained using only "healthy" data, and it learns a compressed representation of this data. At runtime, data is reconstructed using the autoencoder and the mean error is measured. A high error means the new data deviates from the learned "healthy" data; thus it can be classified as anomalous. We implement the architecture described by Borghesi et al. and use it for our evaluation. 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. This autoencoder model is a deep neural network, and deep neural networks are known to be one of the least explainable ML methods.

Feature Extraction + Logistic Regression: The logistic regression classifier is inherently interpretable, so we use this pipeline for sanity checks 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 using the formula:

$$y = S(w \cdot x),$$

where $S(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. Thus, the classifier only learns the weight vector w during training*. Furthermore, it is possible to deduce that 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 use the same features as the random forest pipeline.

V. Evaluation

In this section, we evaluate the *counterfactual time series explanation method* and compare it with other explainability methods based on qualitative comparisons and the metrics described in Sec. IV.

A. Qualitative Evaluation

Our first-order evaluation is to use the *counterfactual time series explanation method* and the baselines to explain a realistic classifier. we use the random forest classifier with feature extraction [5], [6] and the HPAS dataset, which includes different types of performance anomalies. From HPAS data set, we choose "memleak" anomaly, which makes increasing memory allocations without freeing to mimic memory leakage. Our goal is to better understand the classifier's understanding of the "memleak" anomaly. We select a correctly classified time window with the memleak label as x_{test}, and the "healthy" class as the class of interest. We run the *counterfactual time series explanation method*, LIME, and SHAP with the same x_{test} and compare the results.

The *counterfactual time series explanation method* provides two time series as explanation, and is shown in Fig. 2. The first metric to be replaced is `pgalloc_normal` from `/proc/vmstat`, which is a counter that represents the number of page allocations. The plot shows the number of page allocations per second due to the preprocessing. It is obvious that performing memory allocations in a periodic manner is one important factor contributes to memory leaks.

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

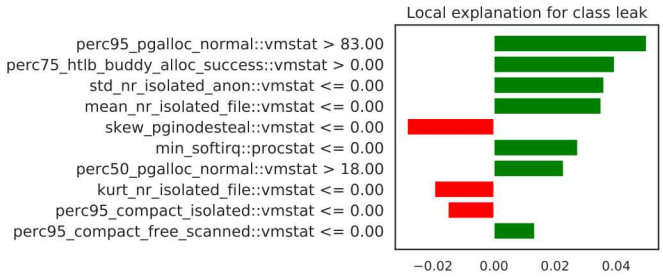

Fig. 4. The explanation of LIME for correctly classified time window with the "memleak" label. LIME provides features that positively (green) and negatively (red) affect the decision. Although the first two features are derived from the metrics in the *counterfactual time series explanation method*'s explanation, it is not straightforward to interpret the values of the features, especially the negative ones. The number of features is an input from the user.

The second metric, from the same time window, in Fig. 2 is `htlb_buddy_alloc_successes`, which shows the number of successful huge page allocations. Memory leaks do not need to cause huge page allocations, since memory leaks in a system with fragmented memory might cause failed huge page allocations. This shows that our training set is biased towards systems with less fragmented memory, most likely because many of the benchmarks are short-lived.

The LIME explanation is shown in Fig. 4. Green values indicate that the features were used in favor of memory leak, and red values were opposing memory leak. We keep the number of features in the explanation at the default value of 10. The first two features are derived from the metrics in our explanation, and a threshold is given for the features, e.g., the 95th percentile of page allocations is over 83, which causes this run to be likely to be a memory leak. Interpreting features such as percentiles, standard deviation and thresholds on their values is left to the user. Furthermore, the effect of the red features is unclear, as it is not stated which class the sample would be if it is not labeled as leak.

The SHAP explanation, in Fig. 3, contains 187 features with very similar SHAP values. Even though we can sort the features by importance, it is difficult to decide how many features are sufficient for a good explanation. Also, SHAP provides a single explanation for one sample, regardless of which class we are interested in, so the most important features are features that are used to differentiate this run from other CPU-based anomalies, which may not be relevant if our goal is to understand memory leak characteristics. 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 metric, or whether it was too high or too low.

LIME and SHAP get prediction probabilities for randomly generated data and use this for the explanations. Therefore, it is important to ensure that the randomly generated data is realistic. However, the random data generations assumes that all of the features are independent variables. In reality, many features are in fact dependent, e.g., statistical features generated from the same metric. The *counterfactual time series explanation method* does not generate synthetic data and uses the entire time series instead of just the features, and makes no assumptions on the dependence of features.

B. Comprehensibility

We use the number of metrics in the explanation to calculate comprehensibility. The *counterfactual time series explanation method* returns 2 time series for the qualitative evaluation example in Fig. 2, and in most cases the number is below 3 but can go up to 10 for some difficult cases. SHAP returns 187 features, and typically SHAP explanations contain hundreds of features for HPC time series data. LIME requires the number of features as an input and does not include instructions of how to select the number of features.

C. Faithfulness Experiments

We check if the explanations represent the model's decision-making process, 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 the classifier uses less than 10 features. In a logistics regression model, it is possible to see number of features used in the classification. For HPAS and Cori data sets, the resulting classifier uses 5 features and 8 for Taxonomist. Because we know the used features, we can rank the explanations based on precision and recall.

- **Recall:** How many of the metrics used by the classifier are in the explanation?
- **Precision:** How many of the metrics in the explanation are used by the classifier?

We get explanations for each sample in the test set, and show the average precision and recall in Fig. 5. We first run the greedy search method and get the number of metrics in the explanation to not disadvantage other explainability methods. Then, we get the same number of metrics from each method. In this way, e.g., LIME is not adversely affected by trying to provide 10 features in the explanation even though only 7 are used by the classifier.

The results show that besides random, all explainability methods are relatively strong in faithfulness. LIME has, in particular, low precision for Cori and Taxonomist data sets which show that LIME explanations contain features that the

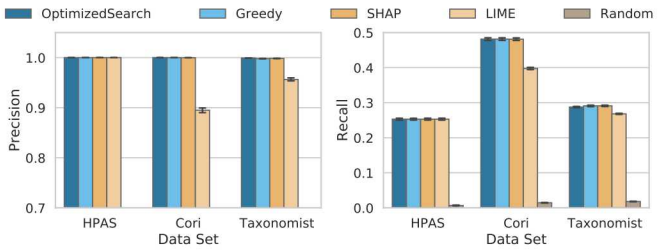

Fig. 5. Precision and recall of the explanations for a classifier with known feature importances. The *counterfactual time series explanation method* (OptimizedSearch and Greedy), and SHAP have perfect precision. LIME has lower precision for Cori and Taxonomist data sets, which indicates that although 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.

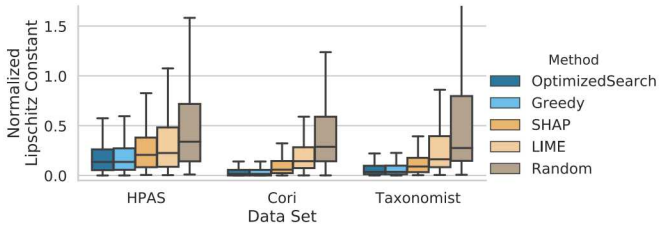

Fig. 6. Robustness of explanations to changes in the test sample. The *counterfactual time series explanation method* (OptimizedSearch and Greedy) is the most robust to small changes in the input, resulting in more predictable explanations and better user experience. Lipschitz constant is normalized to enable comparisons among data sets, and a lower value indicates better robustness.

classifier actually does not use at all. This could be due to the randomness in the data sampling stage of LIME.

D. Robustness Experiments

For robustness, we calculate the Lipschitz constant for each test sample and show average results in Fig. 6. In these results, the *counterfactual time series explanation method* is the most robust explainability technique since it does not involve random data generation for explanations, which reduces the randomness in the explanations. It is important for explanations to be robust, which ensures that users can trust the ML models and explanations.

E. Generalizability Experiments

We evaluate if the *counterfactual time series explanation method*'s explanations for one x_{test} are generalizable to other samples. We use the HPAS data set and random forest classifier with feature extraction. There are 3 classes that are confused with each other. For each misclassified test instance, we get an explanation and apply the same metric substitutions using the same distractor to other test samples with the same (true class, predicted class) pair. We report the percentage of (true class, predicted class) pairs that flip the prediction with the explanation among all possible (true class, predicted class) pairs in Fig. 7. According to the 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 check the explanation for every misprediction, and instead they can get a general understanding of the error characteristics of the classifiers from a few explanations, which is one of the main objectives of explainability.

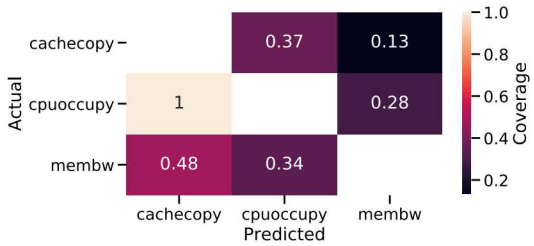

Fig. 7. The ratio of test samples that the *counterfactual time series explanation method*'s explanations are applicable to, among samples with the same misclassification characteristics. For the cpuoccupy runs that are misclassified as cachecopy, every explanation is applicable to every other sample with the same misclassification.

VI. Conclusion and Future Work

We identify the following open problems for future exploration. Such data is widely used in ML-based HPC analysis and management methods that show a lot of promise to improve HPC system performance, efficiency, and resilience. Being explainable is an important requirement for any ML framework that seeks widespread adoption.

Production systems: The challenges faced when deploying ML frameworks on HPC systems can lead to important research questions, and it has been shown that user studies are critical for evaluating explainability methods [42]. We hope to work with administrators and deploy HPC ML frameworks with our explainability method to production systems and observe how administrators use the frameworks and explanations, and find the strengths and weaknesses of different approaches.

Explainable models: During our experiments, we conducted a preliminary experiment using CORELS, which is an interpretable model that learns rule lists [16]. We trained CORELS using the HPAS data set to classify the time windows between the 5 anomalies and the healthy class. Regardless of our experimentation with different configuration options, the model in our experiments took over 24 hours to train and the resulting rule list classified every time window as healthy regardless of the input features. Challenges in this direction include designing inherently explainable models that provide good accuracy for HPC time series data.

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] 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.

[2] 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 *Proceedings of the 15th ACM International Conference on Computing Frontiers*, ser. CF '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 303–308.

[3] 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. New York, NY, USA: Association for Computing Machinery, 2019.

[4] R. Izadpanah, N. Naksinehaboon, J. Brandt, A. Gentile, and D. Dechev, "Integrating low-latency analysis into HPC system monitoring," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: Association for Computing Machinery, 2018.

[5] 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.

[6] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "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.

[7] 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.

[8] 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 (CLUSTER)*, 2017, pp. 766–773.

[9] Q. Xiong, E. Ates, M. C. Herbordt, and A. K. Coskun, "Tangram: Colocating HPC applications with oversubscription," in *IEEE High Performance Extreme Computing Conference*, 2018, pp. 1–7.

[10] 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*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 92–105.

[11] V. Arya, R. K. E. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, S. Mourad, P. Pedemonte, R. Raghavendra, J. Richards, P. Sattigeri, K. Shanmugam, M. Singh, K. R. Varshney, D. Wei, and Y. Zhang, "One explanation does not fit all: A toolkit and taxonomy of AI explainability techniques," arXiv: 1909.03012 [cs.AI], 2019.

[12] 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 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 592–603.

[13] 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. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144.

[14] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774.

[15] 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, stat]*, Aug 2020, arXiv: 2008.10781. [Online]. Available: http://arxiv.org/abs/2008.10781

[16] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 35–44.

[17] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1885–1894.

[18] A. Palczewska, J. Palczewski, R. Marchese Robinson, and D. Neagu, "Interpreting random forest classification models using a feature contribution method," *Advances in Intelligent Systems and Computing*, p. 193–218, 2014.

[19] U. Schlegel, H. Arnout, M. El-Assady, D. Oelke, and D. A. Keim, "Towards a rigorous evaluation of XAI methods on time series," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 4197–4201.

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

[21] R. Assaf and A. Schumann, "Explainable deep neural networks for multivariate time series predictions," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6488–6490.

[22] 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.

[23] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining machine learning classifiers through diverse counterfactual explanations," in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, ser. FAT* '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 607–617.

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

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

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

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

[28] D. Alvarez Melis and T. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7775–7784.

[29] National Technology and Engineering Solutions of Sandia, LLC., "Advanced systems technology test beds," https://www.sandia.gov/asc/computational_systems/HAAPS.html, 2020, accessed: April 21, 2020.

[30] 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.

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

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

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

[34] National Energy Research Scientific Computing Center, "Cori," https://docs.nersc.gov/systems/cori/, 2020, accessed: June 5, 2020.

[35] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, p. 1–19, 1995.

[36] 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, arXiv: 1802.06922.

[37] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, "Hacc: Extreme scaling and performance across diverse architectures," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–10.

[38] "ECP proxy applications." [Online]. Available: https://proxyapps.exascaleproject.org/

[39] 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.

[40] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds., 2010, p. 157–173.

[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] F. Poursabzi-Sangdeh, D. G. Goldstein, J. M. Hofman, J. W. Vaughan, and H. Wallach, "Manipulating and measuring model interpretability," arXiv:1802.07810 [cs.AI], 2018.