9-2023

AutoConf: Automated configuration of unsupervised learning systems using metamorphic testing and Bayesian optimization

Lwin Khin SHAR
Singapore Management University, lkshar@smu.edu.sg

GOKNIL Arda

Erik Johannes HUSOM

Sagar Sen SEN

Naing Tun YAN
Singapore Management University, yannaingtun@smu.edu.sg

See next page for additional authors

Citation

SHAR, Lwin Khin; GOKNIL Arda; HUSOM, Erik Johannes; SEN, Sagar Sen; YAN, Naing Tun; and KIM, Kisub. AutoConf: Automated configuration of unsupervised learning systems using metamorphic testing and Bayesian optimization. (2023). *2023 38th IEEE/ACM International Conference on Automated Software Engineering: Luxembourg, September 11-15: Proceedings*. 1326-1338.
Available at: https://ink.library.smu.edu.sg/sis_research/8405

AutoConf: Automated Configuration of Unsupervised Learning Systems using Metamorphic Testing and Bayesian Optimization

Lwin Khin Shar*, Arda Goknil †, Erik Johannes Husom‡, Sagar Sen§, Yan Naing Tun¶ and Kisub Kim ‖
School of Computing and Information Systems, Singapore Management University and SINTEF Digital, Norway
Email: *lkshar@smu.edu.sg, †arda.goknil@sintef.no, ‡erik.johannes.husom@sintef.no, §sagar.sen@sintef.no, ¶yannaingtun@smu.edu.sg, ‖kisubkim@smu.edu.sg

***Abstract*—Unsupervised learning systems using clustering have gained significant attention for numerous applications due to their unique ability to discover patterns and structures in large unlabeled datasets. However, their effectiveness highly depends on their configuration, which requires domain-specific expertise and often involves numerous manual trials. Specifically, selecting appropriate algorithms and hyperparameters adds to the complexity of the configuration process. In this paper, we propose, apply, and assess an automated approach (*AutoConf*) for configuring unsupervised learning systems using clustering, leveraging metamorphic testing and Bayesian optimization. Metamorphic testing is utilized to verify the configurations of unsupervised learning systems by applying a series of input transformations. We use Bayesian optimization guided by metamorphic-testing output to automatically identify the optimal configuration. The approach aims to streamline the configuration process and enhance the effectiveness of unsupervised learning systems. It has been evaluated through experiments on six datasets from three domains for anomaly detection. The evaluation results show that our approach can find configurations outperforming the baseline approaches as they achieved a recall of 0.89 and a precision of 0.84 (on average).**

I. Introduction

Unlike supervised learning, which relies on labeled data, unsupervised learning requires no explicit supervision to learn meaningful representations of the input data. It can identify hidden structures, such as clusters or low-dimensional manifolds, in massive datasets, e.g., time-series sensor data obtained from Cyber-Physical Systems (CPS), or discover novel and unexpected relationships between data points. This capability has made unsupervised learning an attractive alternative for many applications, ranging from anomaly detection [1] to predictive maintenance [2]. For instance, unsupervised anomaly detection techniques can be preferred over supervised ones [3] when anomalies are rare or unknown, data distribution is not well-defined, or real-time anomaly detection is needed.

The effectiveness of systems using unsupervised learning methods, e.g., clustering, highly depends on the configuration of these methods, e.g., the selection of data preprocessing hyperparameters, features, and appropriate clustering algorithms and their hyperparameters. For instance, the hyperparameters govern various aspects of the learning process, such as the number of clusters, the dimensionality of the learned representations, or the regularization strength. Finding the optimal configuration of an unsupervised learning system for a specific dataset and application is often challenging due to the large configuration space and requires numerous manual trials and errors. Therefore, there has been a growing need for developing automated approaches to configure unsupervised learning systems. On the other hand, the absence of ground truth labels due to the systems' unsupervised nature poses a challenge for automating the configuration process.

Considerable research has been devoted to developing automation support for several machine learning (ML) steps (including data preprocessing, feature engineering, model generation, and model evaluation) in the context of automated machine learning (AutoML) [4]–[6]. AutoML aims to simplify and accelerate the ML process, allowing non-experts to use ML effectively and efficiently. Existing AutoML approaches primarily focus on supervised learning systems [4]–[6], with limited support for unsupervised learning methods such as clustering. The approaches for unsupervised learning support clustering algorithm selection or hyperparameter tuning without considering other critical ML steps such as data preprocessing and feature engineering [7]–[13]. To deal with the lack of ground truth labels in unsupervised learning, they mostly rely on internal validity metrics like the silhouette score [14], which may not adequately capture the "dynamic" nature of a dataset (i.e., the effect of changing the input datasets on the clustering results) or account for data characteristics like noise, density, and skewed distribution [15], [16].

In this paper, we propose, apply, and assess a novel approach, *AutoConf*, which leverages metamorphic testing [17]–[19] and Bayesian optimization [20] to automate the configuration of clustering-based unsupervised learning systems. Metamorphic testing is a software testing technique that involves verifying the behavior of a system by applying a series of transformations to the inputs and observing whether the output is consistent with the expected behavior. Our approach utilizes metamorphic testing to address the lack of ground truth while evaluating the performance of unsupervised learning configurations. Metamorphic testing applies dynamic transformations to the dataset, such as adding white noises with respect to different clusters or adding/removing new samples around cluster centers, to validate the robustness of the configurations against various dataset characteristics such as noise and density. Our approach uses Bayesian optimization [20] guided with metamorphic testing output as the objective function to automatically determine the optimal configuration of unsupervised learning systems. To perform the optimization, we employ the Tree Parzen Estimator (TPE) approach [21], [22], a widely-used optimization method for expensive black-box functions. TPE focuses on identifying the promising regions of the search space (which includes clustering algorithms, their hyperparameters, and hyperparameters related to other ML steps such as data preprocessing and feature engineering).

We present six metamorphic relations (MRs) for the metamorphic testing part of our approach. Our MRs are designed for anomaly detection since detecting anomalies holds significant importance in the fields of Software Engineering and CPS. Moreover, depending on the surrounding environment, CPS involves more components than regular software systems. Continuing along this train of thought, automating the configuration of ML algorithms for anomaly detection is crucial for several compelling reasons: (1) to fine-tune the systems without excessive manual intervention, (2) to search through various combinations for optimal performance, and (3) to rapidly prototype or test. By modifying our six MRs or providing new MRs, we can apply *AutoConf* to other types of unsupervised learning systems, such as predictive maintenance. In addition, *AutoConf* employs five generic (application-agnostic) MRs proposed by Xie et al. [16] for testing clustering algorithms.

AutoConf was subjected to evaluation via experiments on six datasets sourced from three distinct CPS domains (i.e., Drones, Electrocardiogram machine, and Computer Numerical Control machine) for anomaly detection. These datasets included naturally occurring anomalies and those induced artificially, with the latter generated through utilizing the industry partner's environment. The results demonstrate that *AutoConf* can identify configurations capable of detecting anomalies, surpassing the baseline approaches (based on recall and precision metrics). On average, *AutoConf* recorded recall and precision scores of 0.89 and 0.84, respectively.

Our contributions can be summarized as follows:

1) Formulation of metamorphic testing to address the lack of ground truth labels for automated configuration of unsupervised learning systems.
2) Configuration support not only for clustering algorithms and their hyperparameters but also for other ML steps, such as data preprocessing and feature engineering.
3) Novel metamorphic relations for testing unsupervised anomaly detection, utilized for automated configuration.
4) Extensive empirical evaluation based on multiple experiments utilizing six different datasets from three domains.
5) A tool of our approach and the datasets in our experiments, available at [23].

II. Background

A. Unsupervised Learning

Unsupervised learning refers to ML identifying patterns in datasets without any label or human guidance. Its ability to discover similarities and differences in datasets makes it feasible for anomaly detection, exploratory data analysis, behavior profiling, predictive maintenance, etc. While our approach is applicable to various use cases, in this paper we limit its scope to anomaly detection. Unsupervised learning for anomaly detection techniques are as follows.

- *Clustering methods* include KMeans [24], [25], MeanShift [26], [27], DBScan [28], and Optics [29].
- *Outlier detection methods* include One-Class SVM [30], Local Outlier Factor [31] and Isolation Forest [32].

One unsupervised learning method is clustering observations in a data set based on their characteristics. It aims to find a cluster configuration with the maximum similarity between in-cluster observations and the maximum dissimilarity between different clusters. Measuring the Euclidean distance between observations gives observation similarity. There are several clustering algorithms (e.g., KMeans [24], [25], Mini-batch KMeans [33], MeanShift [26], [27], DBScan [28], Optics [29], and affinity propagation [34]) in the literature.

Outlier detection methods are classification-like as these methods only consider the binary classification of data (inliers and outliers) without considering clusters. In our approach, we focus on configuring clustering algorithms because they serve other purposes, such as behavior profiling and predictive maintenance (requirements of our industry partner), on top of anomaly detection. Assume a CPS with a CNC (Computer Numerical Control) machining having several phases. We can map clusters to different phases of its time-series sensor data, perform data analysis of distinct clusters, and reason about the process behaviors. For example, outliers persistently occurring around a particular cluster may indicate the maintenance need of the CNC parts involved in the corresponding phase.

B. Metamorphic Testing

Metamorphic testing is an effective testing technique to address the oracle problem [17]–[19]. It is based on the notion that it may be easier to analyze the relations (metamorphic relations) between outputs of different test runs, rather than to define the expected input-output behavior [35].

Metamorphic testing relies on metamorphic relations (MRs), which are fundamental properties of the program being tested, applicable to multiple inputs and their expected outputs [36]. The testing methodology involves running the system with different inputs for the same test case and verifying the outputs against the metamorphic relation (MR) to determine pass/fail. The three key steps in metamorphic testing are as follows [35]:

- *Construction of MRs.* Program properties to be tested are identified and represented as MRs for multiple test case inputs and their expected outputs.
- *Generation of source test cases.* Source test cases could be generated using various testing techniques (e.g., Boundary testing [37] or Equivalence partitioning [38]).
- *Execution of MRs.* After generating follow-up test cases based on the source test cases, both are executed to verify the described MRs.

C. Bayesian Optimisation

Bayesian optimization [20] is a sequential design strategy used to globally optimize black-box functions without assuming any functional forms. It is utilized to optimize functions expensive to evaluate. It provides a promising approach to explore the search space effectively and efficiently by minimizing the number of function evaluations required. Due to this property, it is a favorable option for hyperparameter tuning in ML as well as for optimization problems in other domains.

III. Configuration of Unsupervised Learning

This section exemplifies unsupervised learning systems and their configuration. Figure 1 illustrates an example of an unsupervised learning system [39], i.e., an ML pipeline that automatically discovers reference patterns for process behavior in sensor data for AI-enabled IIoT. The pipeline consists of three main steps: *data preprocessing*, *unsupervised learning of clusters*, and *labeling and validating new data*.

The pipeline splits training time series data into subsequences and extracts statistical features from them in Step 1 (❶). Each feature vector represents a subsequence. With the obtained feature vectors, it performs cluster analysis on them in Step 2 (❷). Each vector is assigned to a cluster/category. The output is a model, consisting of several cluster centers. The feature vectors of the data are being validated in Step 3 (❸). The subsequences are labeled corresponding to the cluster information that they are assigned to in the model. It finally calculates a *deviation metric* giving how far a given feature vector is from the cluster centers in feature space. The options and algorithm parameters in these three steps particularly affect the pipeline output for an input time series.

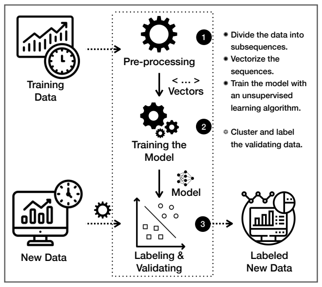

Fig. 1: An example unsupervised learning system.

Data preprocessing (Step 1). This step has three parameters —- sliding window size, overlap, and features (Table I). Information in raw time series exists as relations between consecutive time-varying data points. Clustering high volumes of raw time series is computationally expensive. Features (e.g., mean and variance) are extracted from data subsequences, thus removing the temporal dimension. Lubba et al. [40] discuss several features we can extract from time series data, while each one may have a different impact on the pipeline output. The sliding window technique [41], [42] requiring more computing time provides the finer granularity needed (sliding window size).

The optimal window size (the length of a sliding cutout of a time sequence of data) depends on the sampling frequency of the time series and the order of magnitude of the labels and segments produced. We can choose to overlap between the subsequences/windows. For instance, with a window size of ten and an overlap of nine, we can get a maximum of 190 subsequences from a time series of two-hundred data points.

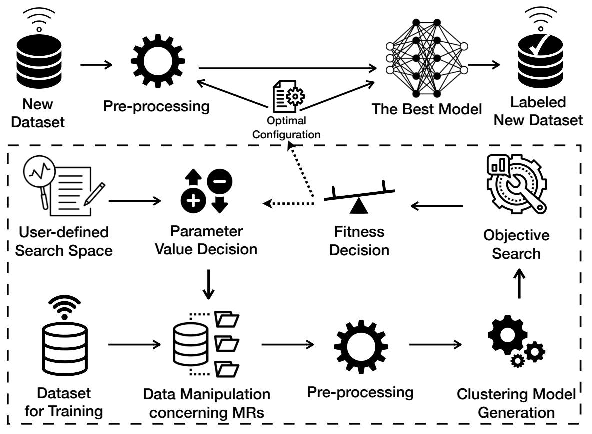

Fig. 2: Overview of our approach *AutoConf*

Unsupervised learning of clusters (Step 2). An unsupervised learning system is configurable for several clustering methods, such as KMeans, DBScan, and Optics, which require different parameters. For instance, the standard KMeans is used for its efficiency when the number of clusters is specified. It has the following configuration parameters: (i) the number of clusters, (ii) predefined centroids (`true` or `false`), (iii) the initialization method (`k-means++` and `init`), (iv) the number of runs with different centroid seeds, (v) the maximum number of iterations for a single run, (vi) the relative tolerance with regards to the Frobenius norm of the difference in cluster centers of two consecutive iterations to declare convergence, and (vii) KMeans algorithm (e.g., lloyd and elkan).

Labeling and validating new data (Step 3). The purpose is to validate time series data and detect anomalies in an early stage using a deviation metric. There are a few metrics for detecting deviation/anomalies. One such metric is to compute the (Euclidean) distance of an incoming new data to each cluster centriod and identify the nearest cluster; and if the distance to the nearest cluster is greater than a given threshold (e.g., two standard deviation of the mean distances-to-centriods of all the samples in the nearest cluster), then the new data is flagged as an anomaly.

IV. Approach

Figure 2 presents an overview of our approach, where the core component is enclosed by a dotted box. Our approach endeavors to identify the optimal configuration for accurately clustering the input data by considering the user-defined search space and input dataset. The output configuration includes the optimal parameters for preprocessing the input dataset to fit it into the best model determined by our approach.

Algorithm 1 presents the high-level algorithm for our search process using metamorphic testing. It takes two input parameters: $refData$ and $searchSpace$. $refData$ is any time-series data (it may contain some outliers or white noises) collected under normal operating conditions of any cyber-physical system (CPS). $searchSpace$ is the search space of

Algorithm 1 FindBestConfig

```
Require: $refData$
Require: $searchSpace : \{window, overlap, model, modelParams\}$
 1: $losses, selectedValues \leftarrow \{\}$
 2: $meanLoss \leftarrow 1$
 3: while $\neg timeout \vee meanLoss \neq 0$ do
 4:     $selectedValues \leftarrow OptimizeSearch(searchSpace, meanLoss)$
 5:     $X \leftarrow ExtractFeatures(refData, selectedValues)$
 6:     $model \leftarrow BuildClusterModel(X, selectedValues)$
 7:     $silhouetteScore = ComputeSilhouetteScore(model)$
 8:     if $silhouetteScore < 1$ then
 9:         $losses \leftarrow |silhouetteScore|$
10:     else
11:         $losses \leftarrow 1 - silhouetteScore$
12:     end if
13:     $losses \leftarrow NumOfOutiers(model)/totalNumOfSamples$
14:     for all $mr \in BenignMRs$ do
15:         $X' \leftarrow GenerateFollowupDataset(X, mr)$
16:         $model' \leftarrow BuildClusterModel(X', selectedValues)$
17:         $losses \leftarrow EvaluateBenignMR(model, model')$
18:     end for
19:     for all $mr \in AnomalyMRs$ do
20:         $X' \leftarrow GenerateFollowupDataset(X, mr)$
21:         $model' \leftarrow BuildClusterModel(X', selectedValues)$
22:         $losses \leftarrow EvaluateAnomalyMR(model, model')$
23:     end for
24:     $meanLoss \leftarrow mean(losses)$
25: end while
```

each (hyper)parameter. For example, the 'model' parameter specifies the search space of 'clustering' algorithms, such as KMeans, DBScan, and Optics (see Section III).

The algorithm performs an iterative search for the best configuration until either a timeout occurs or the loss, which penalizes the bad configuration, becomes zero (Line 3). In each iteration, it carries out the following steps:

- $OptimizeSeach$ in Line 4 chooses a value for each parameter defined in $searchSpace$ using Bayesian optimization [21]. This method optimizes the search for promising regions of the hyperparameter space based on historical 'losses' collected through the input parameter $meanLoss$ obtained from previous iterations.
- $ExtractFeatures$ in Line 5 produces dataset X having the statistical features extracted from $refData$, according to the selected configuration values ($selectedValues$).
- $BuildClusterModel$ in Line 6 produces (source) clustering model ($model$) based on dataset X and the selected configuration values.
- We compute the silhouette score [14] that reports the intrinsic quality of $model$ (Line 7). We then calculate $loss$ with respect to the silhouette score (Lines 8-12).
- We compute $loss$ with respect to the number of outliers produced by $model$ (Line 13).
- In Line 15, the follow-up dataset X' is generated from the source dataset X based on a *benign MR* (i.e., an MR specifying the same clustering model behavior for source and follow-up models). A follow-up clustering model ($model'$) is then built using X' (Line 16) and evaluated against the source clustering model ($model$) using function $EvaluateBenignMR$ (Line 17). The function expects the two clustering models' outcomes to be the same. If not, the samples that deviate from the expected outcome are considered erroneous. The function computes $loss$ as the number of erroneous samples relative to the total number of samples.
- In Line 20, the follow-up dataset X' is generated from the source dataset X based on an *anomaly MR* (i.e., an MR specifying different model behaviors for source and follow-up models). A follow-up clustering model ($model'$) is then built using X' (Line 21) and evaluated against the source clustering model ($model$) using function $EvaluateAnomalyMR$ (Line 22). The function expects the two clustering models' outcomes to be different. If not, the samples not leading to the expected deviation are considered erroneous. The function computes $loss$ based on the number of erroneous samples relative to the number of manipulated samples (Line 22).
- We compute 13 losses, and their mean ($meanLoss$ in Line 24) is used in the next iteration to select the next best parameters by $OptimizeSearch$ (Line 4).

In the rest of the section, we first present the search space ($searchSpace$) (Section IV-A), explain the hyperparameter search optimization in $OptimizeSearch$ in Line 4 (Section IV-B), introduce our metamorphic relations (Section IV-C), and finally give the details of *loss* computation (Section IV-D).

A. Search Space

The user defines the search space as per the application's requirements, such as anomaly detection and predictive maintenance, and may omit some configuration options to expedite the search process (see Figure 2). A search space has been specified for the example system in Section III with hyperparameters including $window$, $overlap$, $feature$, $model$, and $modelParams$. $window$ denotes the subsequence size of time series and is a uniform distribution of integers between 30 and 1500, with a step size of 1. $overlap$ is a continuous hyperparameter with a uniform distribution between 0 and 1, indicating the ratio of overlap between subsequences. $feature$ is a set of engineered features extracted from subsequences (see Table I); an arbitrary number is selected for a configuration. $model$ refers to clustering algorithms (e.g., KMeans, DBScan, Optics); one is chosen for each configuration.

$modelParams$ refers to parameters defined for each clustering algorithm, e.g., *eps*, $min\_samples$, $metric$, $algorithm$ for DBScan. eps represents the maximum distance between two samples (one in the neighborhood of the other), $min\_samples$ is the number of samples in a neighborhood for a point considered a core point, $metric$ is the distance measure (e.g., $Euclidean$ and $manhattan$) for instances in a feature array, $algorithm$ refers to algorithms (e.g., *ball_tree*, $brute$) computing pointwise distances and find nearest neighbors.

B. Hyperparameter Search Optimization

Our search space is extensive and includes a variety of clustering algorithms and hyperparameters, such as the time

TABLE I: List of features for the system in Section III.

Feature name	Mathematical definition
Mean	$\mu = \frac{1}{w}\left(\sum_{i=1}^{w} x_i\right)$
Range	r = max(x)−min(x)
Gradient	$\nabla x = \frac{x_{i+1} - x_{i-1}}{2d}$
Variance	$v = \frac{1}{w}\sum_{i=1}^{w}(x_i - \mu)^2$
Frequency strength	$\nu = \|\text{DFT}(w)\|_2$
Related quantities	**Symbol**
Sliding window size	w
Overlap of sliding windows	d

window and overlap for computing statistical features over time series data. The time window can be an integer between 1 and the dataset size, while the overlap is a percentage between 0% and 100% (exclusive). Furthermore, we need to optimize hyperparameters for the clustering algorithms, such as the epsilon value in DBScan and the number of clusters in KMeans. We use Bayesian optimization to optimize our search ($OptimizeSearch$ in Line 4 in Algorithm 1), which is more efficient than random or grid searches because it uses past evaluations to guide future ones. Bayesian optimization quickly identifies promising areas of the search space and avoids areas that are unlikely to yield good results.

For Bayesian optimization, we employ TPE [21], which creates a "surrogate" probabilistic model mapping hyperparameters to a probability of a score on the objective function, i.e., $P(score|hyperparameters)$. This probabilistic model guides the selection of the next set of hyperparameters by choosing the ones that perform best on the surrogate function. As the surrogate function is easier to optimize than the objective function, Bayesian methods can efficiently find the best values for the objective function.

TPE takes *hyperparameters* and an *objective function* as inputs. It builds a surrogate probability model of the objective function, finds the hyperparameters performing best on the surrogate, applies them to the actual objective function, updates the surrogate model with new results, and repeats the process until the maximum iterations or time limit is reached.

C. Metamorphic Relations

Our approach offers a practical and accessible means for end-users to identify the best-performing configuration of their unsupervised learning system, without requiring extensive theoretical background knowledge. To do so, we apply eleven MRs in the search process including the five generic MRs proposed by Xie et al. [16] for testing clustering algorithms.

Our MRs are tailored for anomaly detection in CPS but can be customized for other unsupervised applications such as predictive maintenance [2]. They fall into two categories: *benign* and *anomaly*. Benign MRs define the same clustering model behavior for source and follow-up models, while anomaly MRs define different behaviors for the two models.

Given an anomaly detection system U and dataset D, we denote $R_s = U(D)$ as the anomaly detection result. A transformation τ applied to D leads to D^τ. Let us refer to $R_f = U(D^\tau)$ as the new anomaly detection result. An MR defines the behavior of U by transforming D by τ. In other words, an MR defines the expected relation γ between R_s and R_f after τ. We refer to the original dataset D and the result R_s as the source dataset and output, respectively. D^τ and R_f are referred to as the follow-up dataset and output, respectively.

Assume that dataset $D = \{s_1,\ s_2,...,\ s_n\}$ contains n instances, and each instance $s_i = (s_1^i,\ s_2^i,...,\ s_d^i)$ has d-dimensional attributes. The transformation τ in our MRs manipulates instances (s_i) and their attributes (s_d^i) in the source dataset D transformed into the follow-up dataset D^τ:

Benign MR1.1 modifying attributes: Modifying the original raw attribute (s_j^i) of an instance (s_i) in time-series dataset D to represent white noise, which can result from electronic interference, is not expected to alter the anomaly detection result (R_f) compared to the original result (R_s). While this may seem counterintuitive, the modification is made on the raw attribute rather than the statistical attribute used in clustering. Hence, a single attribute spike should not substantially affect the statistics. This metamorphic relation is based on the assumption that machinery in CPS may produce white noises and, if infrequent, is considered normal behavior.

Anomaly MR1.2 modifying attributes: Modifying the raw attribute (s_j^i, ${s^{i+1}}_l$,..., ${s^{i+k}}_z$) of several consecutive instances (s_i, s_{i+1}, ..., s_{i+k}) to reflect anomalous behavior is expected to yield an anomaly detection result (R_f) different than the original result (R_s), which should flag the manipulated instances as anomalies. This MR considers variations observed in consecutive instances to indicate actual anomalies, rather than white noise or temporary signal inference.

For testing an example anomaly detection system in a drone flight scenario, *MR1.1* can simulate a sudden wind gust, causing a spike in speed or rotation, which should not be reported as an anomaly to avoid unnecessary investigations. However, if such a condition persists, *MR1.2* can be used to test and flag it as an anomaly, indicating an extreme environmental condition that may require aborting the mission.

Benign MR2.1 modifying clusters: We modify the raw attribute (s_j^i) of an instance (s_i) from each distinct cluster to represent white noise specific to the cluster. If the anomaly detection system is robust, the modified instances should not be detected as anomalies, i.e., the new anomaly detection result (R_f) should remain the same as the original result (R_s). For instance, a machinery process may have multiple phases that operate differently and produce distinct clusters. Each phase may experience spikes or white noise that are considered normal (it is the case with our industry partners).

Anomaly MR2.2 modifying clusters: If we modify the raw attribute (s_j^i, ${s^{i+1}}_l$,..., ${s^{i+n}}_z$) of n consecutive instances (s_i, s_{i+1}, ..., s_{i+n}) from few clusters to represent anomalous behavior with respect to distinct phases of a machinery process, the new result (R_f) should differ from the original result (R_s). The modified instances should be flagged as anomalies.

Benign MR3.1 adding new instance(s): If one new instance (s_1) that statistically significantly deviates from the core samples of the clusters (white noise) is added to D (D^τ), the new and original results (R_f and R_s) should be the same.

Algorithm 2 EvaluateBenignMR

```
Require: $model, model'$
1: $count \leftarrow GetSampleCount(model')$
2: $err \leftarrow 0$
3: for all $sample \in model'$ do
4:     if $\neg IsSameCluster(sample, model)$ then
5:         $err \leftarrow err + 1$
6:     end if
7: end for
8: $loss \leftarrow err/count$
9: return $loss$
```

Anomaly MR3.2 adding new instance(s): If n new, consecutive instances that statistically significantly deviate from all the core samples of the clusters are added to D (D^τ), the new and original results (R_f and R_s) should be different. The manipulated instances should be flagged as anomalies.

The generic MRs (proposed by Xie et al. [16]) used in our approach are: *manipulating cluster density*, *adding informative or uninformative attributes*, *removing redundant attributes*, and *scaling data*. We refer the reader to [16] for their details, noting that not all the MRs provided by Xie et al. are applicable to our context. For instance, the MR about reordering the samples in the dataset is not suitable for time-series data.

D. Objective Function

Our objective function aims to assess the 'fitness' of the currently selected algorithm and hyperparameter values and provide a 'fitness' score to guide the search optimization function in finding better values in the next iteration.

Metrics such as accuracy, precision, and recall are commonly used for assessing and tuning ML algorithms. However, these metrics require labeled data not available in our context. Therefore, we use metamorphic testing as a performance measure to address the test oracle problem. Metamorphic testing evaluates the 'dynamic' aspects of clustering systems by testing the effects of dataset transformations on clustering results. This approach ensures that the clustering model works well with interconnected datasets.

Essentially, meeting the constraint of an MR can be treated as *an objective*. We adopt the generic MRs proposed by Xie et al. [16] and introduce our MRs specific to our context of anomaly detection in CPS (see Section IV-C).

While applying *benign MRs* ($EvaluateBenignMR$ in Line 17 in Algorithm 1), we expect the same model behavior for source and follow-up test cases. Algorithm 2 gives $EvaluateBenignMR$ that evaluates the performance of a clustering model ($model$) with a follow-up model ($model'$) created by a benign MR. For each sample in the follow-up model, we check if it belongs to a different cluster in the source model (Line 4). If yes, the error count is incremented (Line 5). We divide the error count by the number of samples in the follow-up model to calculate the loss (Line 8). The loss represents the clustering results' deviation on the follow-up model from the source model under a benign MR. We return the loss as output (Line 9).

Algorithm 3 EvaluateAnomalyMR

```
Require: $model, model'$
1: $goodLabels \leftarrow ExcludeOutliers(models.labels)$
2: $count, badSamples \leftarrow GetInjectedBadSamples(model')$
3: $err \leftarrow 0$
4: for all $badSample \in badSamples$ do
5:     if $badSample.label \in goodLabels$ then
6:         $err \leftarrow err + 1$
7:     end if
8: end for
9: $loss \leftarrow err/count$
10: return $loss$
```

While applying *anomaly MRs* ($EvaluateAnomalyMR$ in Line 22 in Algorithm 1), we expect different model behavior for source and follow-up test cases. Algorithm 3 gives $EvaluateAnomalyMR$ that evaluates the performance of a clustering model ($model$) with a follow-up model ($model'$) created by an anomaly MR. For each bad sample (anomaly) injected into the follow-up model, we check if it is an outlier in the source model (Lines 4 and 5). If yes, the error count is incremented (Line 6). We divide the error count by the number of injected bad samples to calculate the loss (Line 9). The loss represents the clustering results' deviation on the follow-up model from the source model under an anomaly MR. We return the loss as output (Line 10).

In total, we use eleven MRs to evaluate configurations. We also employ two more metrics (Lines 8-13 in Algorithm 1):

- **Silhouette Score.** We incorporate the silhouette score (Lines 7-12 in Algorithm 1), i.e., an internal validation technique that assesses the effectiveness of a clustering system by measuring intercluster compactness and intracluster separation [15], [43]. Although the silhouette score has received criticism for its static approach, as it does not consider input dataset changes or the relationships between various clustering outcomes [16], we believe it could still be a valuable metric in evaluating configurations. It ranges from -1 (worst) to $+1$ (best), with values close to 0 indicating overlapping clusters and negative values indicating incorrect sample assignments. In each trial, we calculate the silhouette score of the resulting clusters. For negative scores, we penalize the model with losses equal to the absolute value of the score. For positive scores, we calculate losses as $losses \leftarrow 1 - silhouetteScore$.
- **Noise.** We assume that the training dataset contains only benign data with no anomalies and that the model should be robust against white noise or interference spikes in the data without reporting them as anomalies. We calculate the number of outliers the model produces for each trial and penalize the model for generating outliers. The loss is calculated as the ratio of the number of outliers to the total number of samples (Line 13 in Algorithm 1).

Users can assign weighted scores to each objective based on their relative importance. For instance, if the intrinsic quality of the clusters is more important than others, the silhouette

TABLE II: Configuration domain applied in our experiments

Data preprocessing	Sliding window size w=[30,1500]
	Overlap d=[0,1)
	features={mean,range,gradient,variance}
Clustering method	{KMeans, Mini-batch KMeans, DBScan, Optics}
	Hyperparameters
KMeans	n_clusters=[1,15]
Mini-batch KMeans	n_clusters=[1,15]
	max_iter={50, 100, 150}
	batch_size={256, 512, 1024, 2048}
DBScan & Optics	eps=[0.1,5]
	min_samples=[5,70]
	metric=['euclidean', 'l1', 'l2', 'manhattan']
	algo=['auto', 'ball_tree', 'kd_tree', 'brute']

TABLE III: Statistics of the datasets. Training dataset (Column '#Train') contains no known anomaly. Test dataset (Column '#Test') contains a mix of normal samples and real/injected faulty samples.

Dataset	CPS	Anomaly	#Train	#Test
DJI-Windy	Drone	Extreme wind	10,016	20,000
DJI-VelFault	Drone	Faulty Sensor	5,000	2,000
PX4-Vibrate	Drone	Anomalous vibrations	43,547	10,887
Ardu-GyroFault	Drone	Faulty Sensor	144,176	2,000
Sleep-Apnea	ECG	Sleep Apnea	50,000	5,000
Bosch-CNC	CNC	Anomalous vibrations	59,393	99,400

score is given more weight (w_s) to calculate the loss as $losses \leftarrow w_s \times (1 - silhouetteScore)$ in Algorithm 1.

V. Empirical Evaluation

We evaluate our approach on six datasets from three distinct domains to address the following Research Questions (RQ)s:

- ***RQ1.*** *How does unsupervised learning systems perform with* AutoConf*?*
- ***RQ2.*** *Is clustering-based anomaly detection configured by* AutoConf *more effective than baseline anomaly detection approaches?*
- ***RQ3.*** *How does Bayesian optimization boost the efficiency of the search process in* AutoConf*?*

A. Experiment Design

Configuration domain. Table II shows the configuration search space in our experiments. It is determined based on the discussions with our industry partners. Note that our approach is not limited to this particular space.

Datasets. Our evaluation used six datasets, including simulated and actual time-series log data from three CPS domains: Drones, Electrocardiogram (ECG) machine, and Computer Numerical Control (CNC) machine. The datasets consist of four simulated Drone datasets, acquired from various drone control programs (DJI, Ardupilot, and PX4), and two real ECG and CNC datasets (Sleep-Apnea and Bosch-CNC). Table III lists the dataset statistics. All datasets contain real or realistic anomalies, including those specified by our industry partner. The Sleep-Apnea dataset represents actual ECG measurement data of the sleep-apnea condition of a patient, while Bosch-CNC contains real sensor data from the Bosch CNC machine.

DJI-Windy includes drone log data under harsh weather conditions, causing the pilot to abort the mission. The anomaly was motivated by a real-world accident caused by wind gusts making the drone descend at the wrong angle [44]. We simulated the wind using the DJI simulator and collected acceleration data from the DJI Matrice 300 [45] during a mission without anomalies (train dataset) and under strong wind halfway through the mission (test dataset).

Ardu-GyroFault and *DJI-VelFault* simulate faulty sensors, Gyroscopic and Velocity, respectively (motivated by Son et al. [46]). We injected sensor failures to simulate cases with abnormal sensor readings (very low or high). We used the Ardupilot simulator [47] for Ardu-GyroFault and the DJI simulator for DJI-VelFault to conduct the standard flight mission AVC2013 [48]. In the test dataset, the "Gyroscopic Y-axis" and "Velocity Y-axis" are set to abnormally high values.

PX4-Vibrate simulates a flight for a PX4-powered drone. The fault was injected into acceleration Z-and Y axes for the heavy vibration drone behavior.

Sleep-Apnea is obtained from a sleep heart health study [49] (determining *sleep apnea* based on the patient's breathing behavior) with 6000 patients. We randomly selected one of the patient's records, which contains measurements of nasal/oral airflow (thoracic respiratory) and the sleep apnea condition (dependent variable). According to their report, apneas start if the nasal/oral airflow amplitude decreases below approximately 25% of the baseline identified during regular breathing with stable oxygen levels for more than 10 seconds.

Bosch-CNC contains vibration behavior captured from three different CNC milling machines executing 15 processes [50], including normal and anomalous conditions (see Figure 5(a)). Tri-axial accelerometers were mounted in each machine, measuring acceleration in X, Y, and Z-axes at a sampling rate of 2 kHz. For the training set, we selected a sequence with no anomalies, while for the test set, we chose a data sequence containing both normal and anomalous data.

Setup. We implemented *AutoConf* using Python 3, Scikit-Learn libraries [51], and HyperOpt [22]. The experiments were done on a Linux machine with 40 cores Intel CPU E5-2640 2.40GHz and 330GB RAM. For each training dataset, we ran 10,000 trials to find the optimal configuration. Depending on the dataset size, the time to complete the trials ranges from one to seven days. After finding the optimal configuration, the clustering model is built and is used to validate the test data (see Figure 2 and Step 3 in Section III).

B. Results

RQ1: How does unsupervised learning systems perform with* AutoConf*?: To address RQ1, we compared the anomaly detection results of the configurations produced by *AutoConf* and a baseline method utilizing an internal validity metric, since current approaches [8], [12], [13] use such metrics (e.g., silhouette score, Calinski-Harabasz score, and Davies-Bouldin score [15]) to optimize clustering systems. Initially, we performed a preliminary experiment on a sample dataset to assess these metrics and identified the silhouette score as the best-performing one. We thus chose it for the baseline approach in our evaluation. The baseline approach is simply

TABLE IV: Best Configurations found by *AutoConf*

Dataset	w	d	Model	Model Params	Min. Loss
DJI-Windy	42	28	DBScan	eps=0.3 min_samples=19 metric=l1 algo=brute	0.052
DJI-VelFault	48	30	KMeans	n_clusters=3	0.067
PX4-Vibrate	49	40	DBScan	eps=4.77 min_samples=13 metric=l2 algo=brute	0.278
Ardu-GyroFault	47	27	DBScan	eps=2.88 min_samples=20 metric=euclidean algo=brute	0.283
Sleep-Apnea	250	10	DBScan	eps=1.35 min_samples=15 metric=euclidean algo=auto	0.042
Bosch-CNC	1100	330	DBScan	eps=0.53 min_samples=6 metric=euclidean algo=auto	0.107

AutoConf using only the silhouette score (*Silhouette-Only-Approach*). Therefore, we also implicitly assessed to what extent metamorphic testing improves the configuration search.

Table V shows the anomaly detection results of *AutoConf* and the *Silhouette-Only-Approach* baseline using the internal validity metric. *AutoConf* consistently outperformed *Silhouette-Only-Approach*. The F1-score is above 0.7 for all datasets, indicating that *AutoConf* is generally effective in finding the best configurations for anomaly detection in diverse CPS. The configurations provided by *AutoConf* achieved high recall and precision, particularly for DJI-VelFault, PX4-Vibrate, and Ardu-GyroFault having distinct anomalies, such as constantly high sensor values for a prolonged period.

Table IV lists the configurations delivered by *AutoConf* for the datasets (w is the window size, and d is the overlap). Different CPS datasets require unique configurations. An anomaly detection model built from the "best" configuration should ideally represent the reference training data well and detect anomalies more effectively than models that utilize default hyperparameters. Figure 3 presents the DJI-Windy time-series training dataset, its clustering of principal components (pca) with the "best" configuration, and its clustering with default DBScan hyperparameter values from the Scikit-learn library. Each color represents a cluster. The best model correctly captured roughly seven patterns in the training data with seven clusters (excluding the outlier cluster), while the default model produced four clusters. The default model had an F1 score of 0.55 whereas the best model had an F1 score of 0.71 (Table V).

The hyperparameter w (window) in Table IV depends on the data sampling rate and the duration of the behavior we want to capture. For example, the Sleep-Apnea dataset requires a large window size (250) to capture the breathing patterns, which last several seconds. In contrast, the drone log data, which has a sample rate in milliseconds, requires a smaller window size.

The performance of the configuration produced by *AutoConf* is slightly lower for the DJI-Windy dataset due to inconsistencies in the test data, which can lead to confusion in the anomaly detection model. Figure 4(a) presents the acceleration sensor values in the training and test data. During some periods in the test data, the flight direction of the drone aligns with the wind direction, resulting in low acceleration values that are similar to those in the training data. However, the training data contains high acceleration data for a certain duration, indicating a strong-wind situation (the wind condition was to match the current weather as closely as possible). These situations can result in false alarms and false negatives (see Figure 4(a) for the actual labels and the labels predicted with *AutoConf*), affecting recall and precision.

Figure 5(a) displays a time series of the Bosch-CNC training and test datasets, while Figure 5(b) depicts the actual and predicted labels (with AutoConf) for the same dataset. The anomalous data in the test dataset has larger amplitudes than the normal data in the training dataset, aiding in distinguishing the two types of behavior. However, statistical properties such as the mean and range of the values show similar characteristics in certain portions of both the anomalous and normal data, which could be the reason for the relatively low performance of *AutoConf* on the Bosch-CNC dataset.

Answer to RQ1: As each CPS may produce datasets with different characteristics, unsupervised learning systems should be configured differently for each CPS dataset. Thanks to metamorphic testing, *AutoConf* provides configurations yielding better results than the ones delivered by the baseline approach using an internal validity metric.

RQ2: Is clustering-based anomaly detection configured by* AutoConf *more effective than baseline anomaly detection approaches?: To address RQ2, we performed a comparison between the anomaly detection outcomes derived from the clustering-based approach that was configured by *AutoConf* and the corresponding results produced by three baseline anomaly detection techniques, i.e., One-Class SVM (OneSVM), Isolation Forest (IF), and Local Outlier Factors (LOF) [52]. Our aim was to assess if *AutoConf* is capable of discovering configurations that lead to superior anomaly detection than the competing anomaly detection techniques.

The last three columns in Table V list the F1 scores of the three baseline approaches. The *AutoConf* configurations outperform these approaches except for DJI-VelFault and Ardu-GyroFault datasets. Overall the baseline approaches failed to achieve high accuracy, whereas the *AutoConf* configurations attain above an F1-score of 0.7 for all the datasets.

Answer to RQ2: *AutoConf* can identify configurations that yield similar or better anomaly detection results than the baseline anomaly detection approaches.

RQ3: How does Bayesian optimization boost the efficiency of the search process in* AutoConf*?: To address RQ3, we

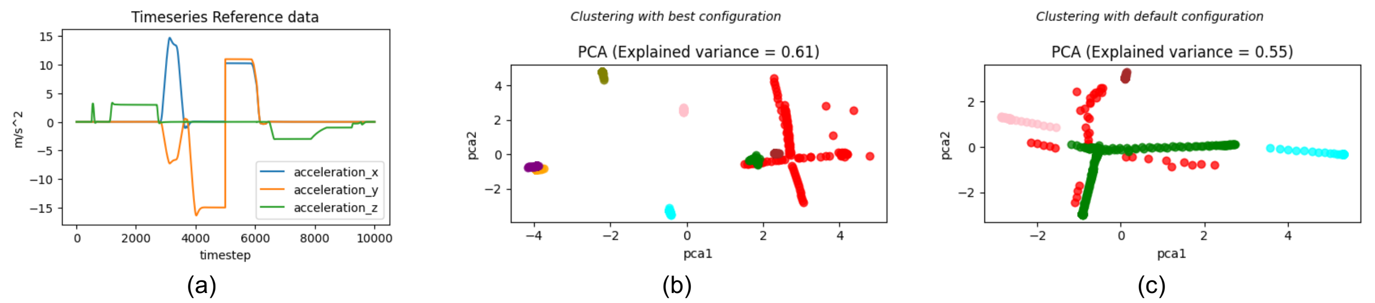

Fig. 3: (a) Timeseries reference data of DJI-Windy dataset; (b) Clustering using the best configuration found by *AutoConf*; (c) Clustering using default configuration.

TABLE V: Comparison of *AutoConf* and baseline approaches

	AutoConf			*Silhouette-Only-Approach*			*OneSVM*	*IF*	*LOF*
Dataset	**Recall**	**Precision**	**F1-score**	**Recall**	**Precision**	**F1-score**	**F1-score**	**F1-score**	**F1-score**
DJI-Windy	0.76	0.66	0.71	0.18	0.43	0.26	0.51	0.65	0.49
DJI-VelFault	1	1	1	1	1	1	1	1	1
PX4-Vibrate	1	0.92	0.96	1	0.87	0.93	0.64	0.64	0.4
Ardu-GyroFault	1	0.9	0.95	1	0.87	0.93	0.93	0.9	0.9
Bosch-CNC	0.76	0.72	0.74	0.5	0.66	0.57	0.56	0.55	0.49
Sleep-Apnea	0.84	0.85	0.85	1	0.65	0.78	0.39	0.39	0.39

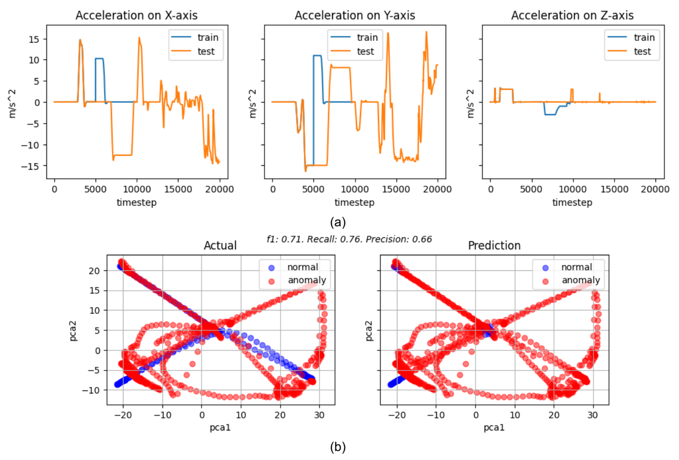

Fig. 4: (a) Time series plot of DJI-Windy train and test dataset. (b) Comparison of actual labels vs predicted labels.

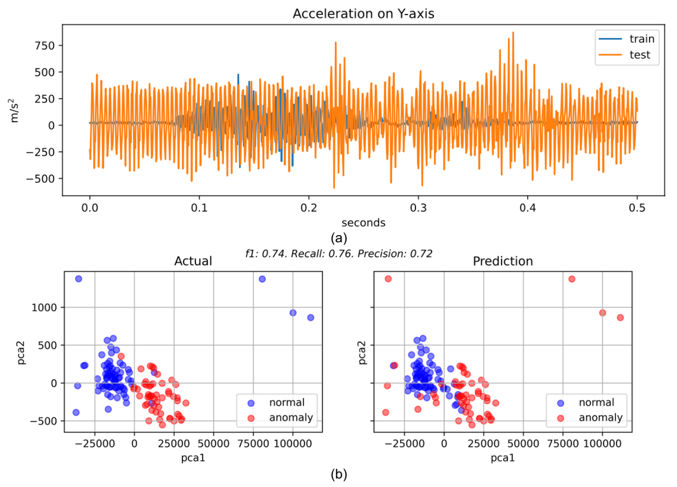

Fig. 5: (a) Time series plot of Bosch-CNC train and test dataset. (b) Comparison of actual labels vs predicted labels.

conducted a comparative analysis of the hyperparameter selection and exploration strategies of two versions of *AutoConf*- one using the TPE algorithm and the other using random search. The results of our analysis were consistent across all six datasets; therefore, we present the outcomes solely for the DJI-Windy dataset. Figure 6(a) and (b) present the scatter plots of the hyperparameter values (eps and $min\_samples$) selected by *AutoConf* using the TPE algorithm for each experiment trial. TPE initially selects values from the entire range with equal probability. As it gains more knowledge about the impact of the hyperparameter on the objective function, it gradually concentrates on areas it anticipates the highest benefit. It still explores the whole solution space, but less frequently. Figure 6(c) shows the scatter plot of loss vs. trial, with the search space concentrated around the minimum loss of 0.05.

Figure 7(a) and (b) present the scatter plot of hyperparameter values (eps and $min\_samples$) selected by random search, which randomly explores the entire solution space without focusing on any particular area. Consequently, the resulting loss values are not concentrated at the lower end (see Figure 7).

AutoConf aims to find the optimal configuration, which does not necessarily converge to zero loss due to trade-offs between different loss functions. For example, while the silhouette score loss prioritizes *compact* clusters, the *noise* loss focuses on the inclusion of white noises or spikes in the clusters, which may degrade their compactness. *AutoConf* seeks the best possible loss value by considering all relevant loss functions.

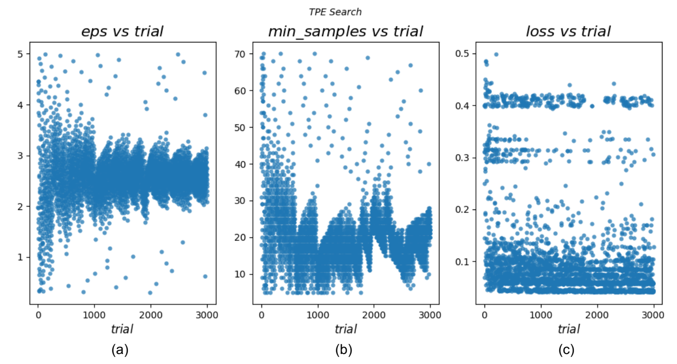

Fig. 6: Bayesian Optimization search: (a) eps vs trial (b) min_samples vs trial (c) loss vs trial

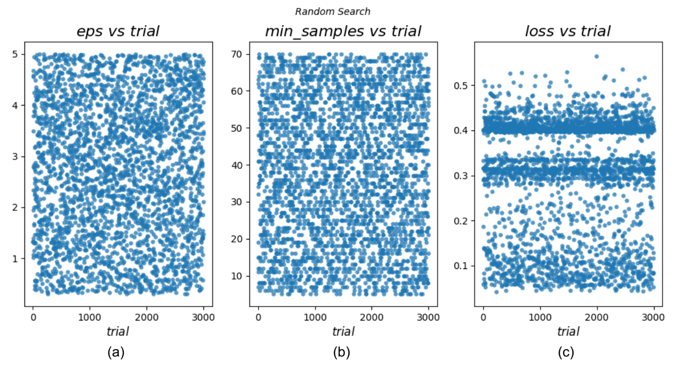

Fig. 7: Random search: (a) eps vs trial (b) min_samples vs trial (c) loss vs trial

Answer to RQ3: *AutoConf* leverages Bayesian optimization to achieve a superior guarantee of identifying the optimal configuration within a time budget. It efficiently guides the search toward the input space that minimizes the loss as determined by the prescribed loss functions. In doing so, it ensures that the entire search space is explored, thus avoiding being trapped in local optima. The pursuit of minimal loss is vital to finding configurations that enable the construction of effective anomaly detection models.

C. Threats to Validity

One threat to the validity of our results pertains to the accuracy of the drone flight simulator in replicating actual physical and environmental conditions. As a result, the logged sensor data may not be entirely realistic. To mitigate this threat, we conducted experiments with both a widely-used simulator (Ardupilot) and a proprietary simulator (DJI).

The validity of our results may be jeopardized by the relevance and authenticity of the injected faults. We address this threat by ensuring that these faults align with well-documented faults and by confirming the fault scenarios with our industry partner. Note that only drone datasets are simulated.

We determined the configurations using 10,000 trials in our experiments. Different results may emerge when fewer or more trials are conducted. To address this threat, it is necessary to repeat the experiments with different numbers of trials.

For RQ3, we compared our approach with three baseline approaches using predefined hyperparameters in Scikit-learn [51] without fine-tuning. Tuning these hyperparameters necessitates ground truth labeled data. Future work will investigate hyperparameter tuning of these approaches without labeled data.

VI. Related Work

AutoML Solutions. Most AutoML approaches primarily focus on supervised learning systems [4]–[6], while some others [7]–[13] concentrate on either clustering algorithm selection or hyperparameter tuning, with an emphasis on cluster numbers. Few approaches [8], [12], [13] consider both algorithm selection and hyperparameter tuning, while they exclude hyperparameters for other ML steps such as data preprocessing. To deal with the lack of ground truth labels in unsupervised learning, they rely on internal validity metrics (e.g., the silhouette score). However, these metrics do not capture the dynamic nature of the dataset, and their performance is sensitive to data characteristics such as noise, density, and skewed distribution [15], [16]. In contrast, *AutoConf* employs metamorphic testing to address the lack of ground truth labels.
Testing Unsupervised Learning. While most ML testing research concentrate on supervised ML, few studies investigate testing for unsupervised learning [53]. One such approach, introduced by Murphy et al. [54], utilizes MRs applicable to both supervised and unsupervised learning algorithms. Ramanathan et al. [55] combine symbolic and statistical techniques to test KMeans, while Lu et al. [56] introduce a mutation testing approach to simulate unstable situations and potential errors that may arise in unsupervised learning systems. Xie et al. [16] propose generic MRs for unsupervised learning. *AutoConf* uses metamorphic testing with custom MRs to identify unsupervised learning configurations yielding the best results.
Metamorphic Testing. Considerable research has been conducted on metamorphic testing [53] to test various aspects of ML, including deep learning models [57]–[60], their domain-specific applications [61]–[64], and supervised learning classifiers [65]–[67]. Xie et al. [16] introduce METTLE, a metamorphic testing approach for testing unsupervised learning. METTLE includes eleven MRs that manipulate instance order, distinctness, density, and attributes, and introduce outliers to the data. Our approach utilizes some of these general MRs and the MRs we introduce for anomaly detection. Unlike the MRs Xie et al. designed, our approach is tailored to address the complex behaviors exhibited by machinery within systems. This emphasis arises due to the notable differences between standard software and physical systems. For instance, consider the effect of a sudden gust of wind on a drone, causing a rapid increase in speed or rotation. Additionally, a CPS often involves multiple phases, each with distinct operations that lead to different clusters of behaviors.
Deep Learning Methods. Although deep learning methods such as LSTM, CNN, and autoencoders have proven effective in detecting anomalies in CPS (a focus on residual error-based detection) [3], we did not incorporate them in our experiments. The rationale behind this decision stems from their significant

differences in implementation for anomaly detection compared to our example anomaly detection system and the approaches we compared. Thus, we consider them as potential avenues for future research. Despite their effectiveness, our industry partner cannot yet adopt deep learning methods as a replacement for clustering methods due to their lack of interpretability and the need for a substantial amount of high-quality data [3].

VII. Discussion and Future Work

Multi-objective Search. TPE is a single-objective Bayesian optimization algorithm. However, recent research has extended it to handle multi-objective optimization. To address the configuration problem as a multi-objective problem, we intend to apply the extended TPE. Doing so will allow us to analyze the configurations in the Pareto front for multiple objectives.

VIII. Conclusion

In this paper, we introduced *AutoConf*, an automated approach that leverages metamorphic testing and Bayesian optimization to configure clustering-based unsupervised learning systems. We demonstrated the effectiveness of *AutoConf* in detecting anomalies through experiments conducted on various datasets. Thanks to metamorphic testing, *AutoConf* outperformed the baseline approaches, achieving an average recall of 0.89 and a precision of 0.84. The custom and generic metamorphic relations employed enabled *AutoConf* to address the lack of ground truth labels while capturing the "dynamic" nature of datasets together with data characteristics, such as noise and density, which the internal validity metrics are sensitive to. Bayesian optimization helped to achieve a superior guarantee of identifying the optimal configuration within a time budget.

Acknowledgment

The work of Arda Göknil, Erik Johannes Husom, and Sagar Sen has been conducted as part of the InterQ project (958357) and the DAT4.ZERO project (958363) funded by the European Commission within the Horizon 2020 research and innovation programme. The work of Lwin Khin Shar, Yan Naing Tun, and Kisub Kim is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme, National Satellite of Excellence in Mobile Systems Security and Cloud Security (NRF2018NCR-NSOE004-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

[1] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[2] N. Amruthnath and T. Gupta, "A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance," in *2018 5th international conference on industrial engineering and applications (ICIEA)*. IEEE, 2018, pp. 355–361.

[3] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[4] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[5] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "Automl to date and beyond: Challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–36, 2021.

[6] M. Bahri, F. Salutari, A. Putina, and M. Sozio, "Automl: state of the art with a focus on anomaly detection, challenges, and research directions," *International Journal of Data Science and Analytics*, vol. 14, no. 2, pp. 113–126, 2022.

[7] E. Ditton, A. Swinbourne, T. Myers, and M. Scovell, "Applying semi-automated hyperparameter tuning for clustering algorithms," *arXiv preprint arXiv:2108.11053*, 2021.

[8] R. ElShawi, H. Lekunze, and S. Sakr, "csmartml: A meta learning-based framework for automated selection and hyperparameter tuning for clustering," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 1119–1126.

[9] X. Fan, Y. Yue, P. Sarkar, and Y. R. Wang, "On hyperparameter tuning in general clustering problems," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2996–3007.

[10] M. C. De Souto, R. B. Prudencio, R. G. Soares, D. S. De Araujo, I. G. Costa, T. B. Ludermir, and A. Schliep, "Ranking and selecting clustering algorithms using a meta-learning approach," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 3729–3735.

[11] D. G. Ferrari and L. N. De Castro, "Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods," *Information Sciences*, vol. 301, pp. 181–194, 2015.

[12] Y. Poulakis, C. Doulkeridis, and D. Kyriazis, "Autoclust: A framework for automated clustering based on cluster validity indices," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1220–1225.

[13] M. Carnein, H. Trautmann, A. Bifet, and B. Pfahringer, "confstream: Automated algorithm selection and configuration of stream clustering algorithms," in *Learning and Intelligent Optimization: 14th International Conference, LION 14, Athens, Greece, May 24–28, 2020, Revised Selected Papers 14*. Springer, 2020, pp. 80–95.

[14] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[15] Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu, "Understanding of internal clustering validation measures," in *2010 IEEE international conference on data mining*. IEEE, 2010, pp. 911–916.

[16] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P.-L. Poon, and B. Xu, "Mettle: a metamorphic testing approach to assessing and validating unsupervised machine learning systems," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1293–1322, 2020.

[17] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," *arXiv preprint arXiv:2002.12543*, 2020.

[18] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2013.

[19] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, vol. 37, no. 3, pp. 46–53, 2018.

[20] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[21] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *NIPS*, 2011.

[22] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*. PMLR, 2013, pp. 115–123.

[23] AutoConf, 2023. [Online]. Available: https://github.com/Jesper20/autoconf

[24] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Los Angeles LA USA, 1967, pp. 281–297.

[25] H. Steinhaus *et al.*, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci*, vol. 1, no. 804, p. 801, 1956.

[26] K. Fukunaga and L. Hostetler, “The estimation of the gradient of a density function, with applications in pattern recognition,” *IEEE Transactions on information theory*, vol. 21, no. 1, pp. 32–40, 1975.

[27] Y. Cheng, “Mean shift, mode seeking, and clustering,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[28] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[29] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: Ordering points to identify the clustering structure,” *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[30] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[31] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[32] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[33] D. Sculley, “Web-scale k-means clustering,” in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 1177–1178.

[34] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[35] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, “A survey on metamorphic testing,” *IEEE Transactions on software engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[36] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, “Metamorphic testing: A review of challenges and opportunities,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–27, 2018.

[37] D. W. Andrews, “Testing when a parameter is on the boundary of the maintained hypothesis,” *Econometrica*, vol. 69, no. 3, pp. 683–734, 2001.

[38] S. C. Reid, “An empirical analysis of equivalence partitioning, boundary value analysis and random testing,” in *Proceedings Fourth International Software Metrics Symposium*. IEEE, 1997, pp. 64–73.

[39] E. J. Husom, S. Tverdal, A. Goknil, and S. Sen, “Udava: An unsupervised learning pipeline for sensor data validation in manufacturing,” in *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, 2022, pp. 159–169.

[40] C. H. Lubba, S. S. Sethi, P. Knaute, S. R. Schultz, B. D. Fulcher, and N. S. Jones, “catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis,” *Data Mining and Knowledge Discovery*, vol. 33, no. 6, pp. 1821–1852, 2019.

[41] C.-S. J. Chu, “Time series segmentation: A sliding window approach,” *Information Sciences*, vol. 85, no. 1-3, pp. 147–173, 1995.

[42] T.-c. Fu, “A review on time series data mining,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[43] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[44] S. Shankland, “Facebook drone investigation: Wind gust led to broken wing,” https://www.cnet.com/tech/services-and-software/facebook-drone-investigation-wind-gust-led-to-broken-wing, 2016.

[45] DJI, 2023. [Online]. Available: https://www.dji.com/sg/matrice-300

[46] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, “Rocking drones with intentional sound noise on gyroscopic sensors,” in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.

[47] “Ardupilot sitl simulator,” https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2022.

[48] A. Team, “Sparkfun autonomous vehicle competition 2013,” https://avc.sparkfun.com/2013, Accessed 2022.

[49] SHHS, “Sleep heart health study dataset,” https://sleepdata.org/datasets/shhs/pages/04-dataset-introduction.md, Accessed 2023.

[50] M.-A. Tnani, M. Feil, and K. Diepold, “Smart data collection system for brownfield cnc milling machines: A new benchmark dataset for data-driven machine monitoring,” *Procedia CIRP*, vol. 107, pp. 131–136, 2022.

[51] 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, 2011.

[52] C. Feng and P. Tian, “Time series anomaly detection for cyber-physical systems via neural system identification and bayesian filtering,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2858–2867.

[53] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine learning testing: Survey, landscapes and horizons,” *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[54] C. Murphy, G. E. Kaiser, and L. Hu, “Properties of machine learning applications for use in metamorphic testing,” Columbia University, Tech. Rep., 2008.

[55] A. Ramanathan, L. L. Pullum, F. Hussain, D. Chakrabarty, and S. K. Jha, “Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision,” in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 786–791.

[56] Y. Lu, K. Shao, W. Sun, and M. Sun, “Mtul: Towards mutation testing of unsupervised learning systems,” in *Dependable Software Engineering. Theories, Tools, and Applications: 8th International Symposium, SETTA 2022, Beijing, China, October 27-29, 2022, Proceedings*, 2022, pp. 22–40.

[57] X. Du, X. Xie, Y. Li, L. Ma, J. Zhao, and Y. Liu, “Deepcruiser: Automated guided testing for stateful deep learning systems,” *arXiv preprint arXiv:1812.05339*, 2018.

[58] J. Ding, X. Kang, and X.-H. Hu, “Validating a deep learning framework by metamorphic testing,” in *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*. IEEE, 2017, pp. 28–34.

[59] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. J. C. Bose, N. Dubash, and S. Podder, “Identifying implementation bugs in machine learning based image classifiers using metamorphic testing,” in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 118–128.

[60] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.

[61] Z. Q. Zhou and L. Sun, “Metamorphic testing of driverless cars,” *Communications of the ACM*, vol. 62, no. 3, pp. 61–67, 2019.

[62] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars,” in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[63] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, “Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 132–142.

[64] M. S. Ramanagopal, C. Anderson, R. Vasudevan, and M. Johnson-Roberson, “Failing to learn: Autonomously identifying perception failures for self-driving cars,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3860–3867, 2018.

[65] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, “Application of metamorphic testing to supervised classifiers,” in *2009 Ninth International Conference on Quality Software*. IEEE, 2009, pp. 135–144.

[66] S. Al-Azani and J. Hassine, “Validation of machine learning classifiers using metamorphic testing and feature selection techniques,” in *Multi-disciplinary Trends in Artificial Intelligence: 11th International Workshop, MIWAI 2017, Gadong, Brunei, November 20-22, 2017, Proceedings 11*. Springer, 2017, pp. 77–91.

[67] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, “Testing and validating machine learning classifiers by metamorphic testing,” *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.