Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring

Poenaru-Olaru, Lorena; Cruz, Luis; Rellermeyer, Jan S.; van Deursen, Arie

DOI
10.1109/DeepTest66595.2025.00006

Publication date
2025

Document Version
Final published version

Published in
Proceedings - 2025 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning, DeepTest 2025

Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring

Lorena Poenaru-Olaru*, Luis Cruz*, Jan S. Rellermeyer†, Arie van Deursen*
*Software Engineering Research Group, Delft University of Technology, Delft, Netherlands
L.Poenaru-Olaru@tudelft.nl, L.Cruz@tudelft.nl, arie.vandeursen@tudelft.nl
†Dependable and Scalable Software Systems, Leibniz University Hannover, Hanover, Germany
rellermeyer@vss.uni-hannover.de

***Abstract*—Failure prediction models can be significantly beneficial for managing large-scale complex software systems, but their trustworthiness is severely affected by changes in the data over time, also known as concept drift. Thus, monitoring these models against concept drift and retraining them when the data changes becomes crucial in designing reliable failure prediction models. In this work, we evaluate the effects of monitoring failure prediction models over time using label-independent (unsupervised) drift detectors. We show that retraining based on unsupervised drift detectors instead of periodically reduces the cost of acquiring true labels without compromising accuracy. Furthermore, we propose a novel feature reduction for unsupervised drift detectors and an evaluation pipeline that practitioners can employ to select the most suitable unsupervised drift detector for their application.**

***Index Terms*—failure prediction, machine learning monitoring, concept drift, concept drift detection**

I. Introduction

Failures in large-scale software systems generate service interruptions [1] and are challenging to manage due to their complexity [1]. Consequently, plenty of attention has been directed toward automated techniques that use Artificial Intelligence (AI) and machine learning (ML) to detect failures (AIOps) [1], which are also referred to as failure prediction models. It has been observed that failure prediction models are efficient solutions for handling system issues which can detect around 70% of a system's failure within 10 minutes [2].

Although beneficial, failure prediction models suffer from temporal quality degradation in accuracy caused by changes in data over time, also known as concept drift [1], [3], [4]. Thus, concept impacts the reliability of failure prediction models [5]. In this paper, concept drift refers specifically to data changes that degrade the accuracy of failure prediction models.

To overcome the effects of concept drift, previous work proposes to periodically retrain (update) failure prediction models [3]–[5]. However, this comes with both computational effort [6] and hidden deployment costs such as compliance verification [7] and integration with a larger software system [8]. Lyu et al. [6] propose retraining only when a label-dependent drift detector identifies a significant accuracy drop. However, this approach assumes immediate access to true labels, which is often impractical, as labels may require manual annotation by engineers once the root cause of failure is understood [2].

Continuously monitoring failure prediction models by verifying whether concept drift occurs is a promising solution [3], [4] since practitioners can understand when the model's outcome is reliable and can be used for decision making purposes [5]. However, using a label-independent drift detector could eliminate the necessity of gathering true labels. Two label-independent techniques to monitor data against concept drift in ML systems were proposed by researchers from industry [8]–[10], namely monitoring the skewness of the features and monitoring changes in data distribution over time. Similarly to [6], these techniques can be used to identify when the model requires updating [8], [11]. However, their effectiveness in preserving the accuracy of the failure prediction model has, to the best of our knowledge, not yet been assessed.

The contributions of this study are threefold. (1) We assess existing label-independent concept drift detection techniques on three popular open-source failure prediction datasets. (2) We propose a novel feature-reduction technique based on the model's feature importance ranking that can be incorporated with an existing unsupervised data distribution-based drift detector. (3) We present a replicable evaluation pipeline for identifying suitable unsupervised drift detectors for specific failure prediction models[1]. In this work, we answer the following research questions:

RQ1: Can monitoring the skewness in features individually indicate concept drift?
- a) Can changes in specific features indicate concept drift?
- b) Is the proportion of changing features an indicator of concept drift?

RQ2: Can monitoring changes in data distribution over time indicate concept drift?
- a) To what extent can data distribution-based drift detectors identify concept drift?
- b) What is the effect on the failure prediction models' accuracy and costs (retraining and label costs) of retraining based on a data distribution drift detector vs periodically?

II. Related Work

Concept Drift Definition and Evaluation: Concept drift generally refers to changes in the data that occur over time [12] and can be monitored using concept drift detectors [13].

[1]Replication Package: https://github.com/LorenaPoenaru/aiops_failure_prediction

Multiple drift detectors were proposed for classification problems [12]. Supervised drift detectors monitor model accuracy changes, while unsupervised drift detectors monitor data characteristics, such as the data distribution [13]. Although supervised detectors indicate degradation in the model's performance, they need immediate labels, which is sometimes impractical in real-world applications. Unsupervised detectors, which don't need labels, are, thus, more suitable.

Concept drift detectors are evaluated in two manners: *assessing the drift detection accuracy* and *assessing the performance of an ML model over time*. Drift detection accuracy measures the ability to distinguish between drifts and non-drifts, but requires knowledge of the exact drift occurrence [13], [14], which is often unknown in real-world scenarios. Previous work proposed a technique that uses a two-proportion Z-test on the error rate to label testing batches into drift and non-drift, which allows the evaluation. Assessing the performance of an ML model over time is usually preferred in the real world since knowledge regarding the moment of drift occurrence is not required [12]. This assessment technique compares the performance (e.g. accuracy, ROC-AUC, etc.) of periodically retrained ML models to those retrained based on drift detection to understand whether the drift detector can be used as a retraining indicator [13].

AIOps Models Degradation due to Concept Drift: AIOps models enhance software delivery and quality [1] but suffer from performance degradation over time due to concept drift [1], [3], [4], [6], [15]. For example, defect prediction models trained on past data do not generalize well to future data [3], [16]. This is a consequence of changes in the operational data (concept drift) caused by uncontrollable factors such as user workloads or hardware/software upgrades. Monitoring and periodically updating these models is necessary to mitigate concept drift [3], [4], [15]. The effects of retraining failure prediction models based on supervised detectors have been previously researched [6] while retraining failure prediction models based on unsupervised drift detectors received less attention.

Unsupervised Concept Drift Detection: Industry practitioners use unsupervised data monitoring techniques to identify data changes/concept drift [8]–[10]. Best practices include **monitoring the skewness of features** over time [9] and **monitoring data distribution** over time [8]–[10]. The skewness of features can be monitored by assessing changes from training to testing data for each individual feature or the percentage of skewed features [9]. Industry practitioners also recommend monitoring the data distribution over time by comparing the estimated data distribution from the training and testing features [9], [10], [17], [18]. However, the effectiveness of these techniques has not been assessed previously in failure prediction models. Research on unsupervised drift detection [12]–[14] identifies two types of change detection techniques: statistical tests and distance-based drift detectors. Statistical tests identify drift by checking whether there is a significant difference between the data distribution from the train set and the test set. Distance-based detectors measure the distance between these distributions and detect drift if the distance exceeds a predefined threshold [14].

III. Data and Failure Prediction Models

We employ three publicly available AIOps datasets, the Backblaze Disk Stats Dataset, the Google Cluster Traces Dataset, and the Alibaba GPU Cluster Trace Dataset. These datasets were previously used to build failure prediction models [3], [4], [6].

The **Backblaze Disk Stats Dataset** [19] contains information about various types of hard disk drives from different manufacturers [20]. It was used to design disk failure prediction models [3], [20], [21]. Similar to previous work [3], [4] we are using around 7M data samples corresponding to 12 months of data collected during 2015.

The **Google Cluster Traces Dataset** [22] contains information about traces extracted from real-world large cluster systems. It was used [3], [23] to design job failure prediction models. The dataset contains 625K samples and was collected for 29 days (May 2011).

The third dataset, the **Alibaba GPU Cluster Trace Dataset** [24], was recently publicly released (2021) by the Alibaba Group and contains workload traces collected from a production cluster containing over 6,000 GPUs. It was used to build a job failure prediction model [6]. The data was collected for two months, July to August 2020 [25], and contains approx. 701K samples.

Model and Features. Random Forests is a commonly used tree-based classifier in failure prediction models [3], [4], [6], [20], [21], [23]. In our experiments, we employ the Random Forests classifier to build failure prediction models for all three analyzed datasets. Furthermore, we include Random Forests in our experiments since they are well-researched in terms of feature importance ranking extraction [26], [27], which is a crucial part of our proposed drift detection method. We build a monthly disk failure prediction model using the Backblaze dataset, a weekly model using the Alibaba dataset, and a daily model using the Google dataset.

For all three datasets, we employ the same features as previous work [3], [4], [6], [20], [21], [23]. The exact features are described in our replication package.

Model Building Pipeline. All the failure prediction models studied in this paper are built by replicating previous works' approaches [3], [4], [6], [20], [21], [23]. The first step in creating the failure prediction models is data preprocessing through scaling using StandardScaler[2]. Scaling is performed since features have varying degrees of magnitude, which affects the classification training. To mimic a realistic scenario, we fit the scaler every time on the period of data corresponding to the training data and only then apply it to the testing data. We further apply undersampling with a ratio of 1:10 to reduce the severe class imbalance in the Google and Backblaze

[2]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

datasets. We do not apply it to the Alibaba dataset, since the classes are relatively balanced (1:3 imbalance ratio). The last steps of building the failure prediction model are training and hyperparameter tuning, for which we use Randomized Search [6]. To avoid bias we repeat all experiments using 10 different random seeds.

Detecting Drift/Changes in Data. To detect data changes, we select the Kolmogorov-Smirnov statistical test given its popularity in unsupervised drift detection [12]–[14], [28]. This statistical test verifies the similarity between two data distributions. We did not consider distance-based drift detectors since they require users to predefine drift thresholds for each dataset.

IV. Experimental Design

This section describes the experimental design used to answer the two research questions. RQ1 and RQ2.a. require a drift detection accuracy assessment, which can only be performed after extracting the ground truth regarding the batches that are labeled as drift and non-drift. RQ2.b. requires a model performance preservation assessment.

A. Extracting Drift/Non-Drift Batches

To extract the ground truth regarding when concept drift occurs, we follow the same technique as previous work [3]. In Figure 1, we depict the pipeline used to identify concept drift between datasets extracted from two different periods, P1 and P2. We train an ML model using the data from the first period (P1) and test it on the data from the second period (P2). The training error rate is computed by performing a 10-fold cross-validation on Data P1 and the testing error rate is obtained by testing the model on Data P2. On the two error rates, we apply a two-proportion Z-test to assess whether there is concept drift between the datasets from two different periods:

$$Z = \frac{\epsilon_{test} - \epsilon_{train}}{\sqrt{\epsilon(1-\epsilon)(\frac{1}{n_{train}} + \frac{1}{n_{test}})}} \quad (1)$$

where ϵ_{train} is the prediction error rate on the training set, ϵ_{test} is the prediction error rate on the testing set, ϵ is the overall prediction error rate, n_{train} is the length of the training set and n_{test} is the length of the testing set.

The null hypothesis of the Z-test is that there is no significant difference between the ML model's performance on datasets extracted from the two different periods, thus it is a *non-drift batch*. The null hypothesis is rejected when the p-value of the Z-test is lower than 0.05, suggesting that there is a *drift batch*.

B. Monitoring the Skewness of Features Individually

In this work, we monitor the skewness of features by evaluating whether either *monitoring changes in features individually* or *the percentage of features that change over time* can indicate that a batch is labeled as drift or non-drift.

Monitoring changes in features individually. We apply the Kolmogorov-Smirnov statistical test on two consecutive periods of the same individual feature to assess how each feature is changing over time. We determine whether every

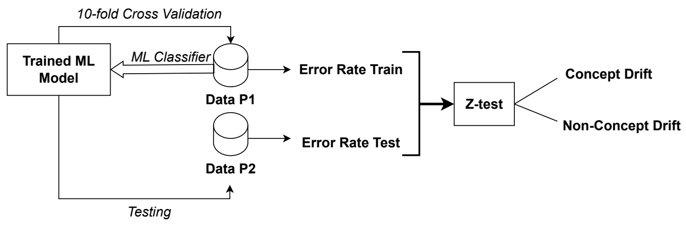

Fig. 1. Obtaining the ground truth. Pipeline to assess the presence of concept drift between two batches (Data P1 and Data P2).

two consecutive periods corresponds to a drift batch or a non-drift batch. We further count the number of times the feature changes in a drift batch and non-drift batch respectively. With this experiment, we aim to understand whether changes in specific features are more associated with drift or non-drift batches. If changes in one feature are more linked to drift batches than non-drift batches, monitoring that feature could indicate unsupervised model performance degradation for failure prediction models.

Monitoring the percentage of features that change over time With this evaluation strategy, we aim to understand whether the number of features that change from one period to another can be an indicator that the model's performance is degrading. Therefore, we apply the Kolmogorov-Smirnov statistical test on two consecutive periods of each individual feature. We count how many features have changed between these two periods and we determine whether the associated batch is labeled as drift or non-drift. This evaluation strategy assumes that, if more features are changing between two periods, there is a higher chance of having a drift. Thus, we aim to understand whether AIOps practitioners can use the number of features that change as an unsupervised model degradation indicator. We further support our observation by analyzing the correlation coefficients between the number of features that change and the batch label (drift/non-drift), which should be close to 1 or -1 to suggest that the number of features that change is a good indicator of drift.

C. Monitoring Changes Data Distribution

1) Assessing the Drift Detection Accuracy of Drift Detectors: The state-of-the-art techniques identify drift by monitoring how the distribution of the data in the training set is changing compared to the distribution of the data in the testing set [14]. In terms of the number of features included to derive the data distribution, we analyze two data distribution drift detection techniques. The first technique we refer to as *KS_ALL* identifies drift by estimating the data distribution from all the features used to train the failure prediction model [14]. Concept drift detection research suggests that feature reduction through Principal Component Analysis (PCA) should be initially applied to the features before estimating the distribution for a more accurate drift detection [29], [14]. Therefore, the second evaluated technique referred to as *KS_PCA* initially reduces the dimensionality of the features using PCA and then estimates the data distribution.

We propose a *model-driven unsupervised drift detector* that takes into account the most relevant features of the model while making predictions. Our technique includes a feature reduction that computes the data distribution of solely the features that are relevant to the model. With every model training, we compute the feature importance (FI) ranking, sort the features based on their ranking in importance, and select the features whose importance value is higher than the mean importance values. Therefore, we estimate the data distribution of solely the most important features and apply the KS statistical test to identify drift. Similar to previous techniques, drift is detected when the null hypothesis of the KS statistical test is rejected. We are referring to this technique as *KS_FI*. To extract the feature importance ranking we employ techniques previously used for classifier predictions explainability [26], [30], [31]. The most common metric to compute the feature importance (FI) ranking is the mean decrease in impurity (MDI) [30], [32], [33] also known as Gini importance. For each feature, this technique calculates the total decrease in impurity (loss) computed for each random split [30].

Evaluation Metrics. We employ three metrics in our evaluation strategy, the *True Positive Rate (TPR)*, the *True Negative Rate (TNR)*, and the *Balanced Accuracy*. The TPR shows the percentage of correctly identified drifts, the TNR shows how many non-drifts are correctly identified and the balanced accuracy shows the overall correctly classified drifts and non-drifts. The best value of each metric is 1.0.

2) Assessing the Effects of Retraining based on Drift Detection: We aim to understand the effect of retraining based on drift detection vs retraining periodically, which is the current state of practice for failure prediction models [3], [4]. In Figure 2 we illustrate the difference between the two retraining techniques. In the case of periodic retraining, every time new data becomes available the model is retrained. The new data is included in the retraining data and the old data is discarded, a retraining strategy called the sliding-window approach, also used in previous work [4]. The drift detection-based retraining strategy comes with the assumption that if no drift is detected, the new available data is similar to the one that the model is already trained on, thus retraining the model on the new data does not necessarily bring new information. Therefore, in this evaluation strategy, retraining is performed solely when drift is indicated by a drift detector. As depicted in Figure 2, only batch B, where the drift was identified, is included in the training set while batch A is not included since there is no drift identified. This evaluation strategy helps in reducing not only the number of times the model requires retraining, but also the costs of obtaining labels for retraining. Thus, from our example, the true labels from AIOps practitioners are not required for batch A. In terms of drift detection, we employ the same data distribution drift detection techniques as previously, namely KS_All, KS_FI, and KS_PCA. Furthermore, for this experiment, we use a model that is never retrained, which we refer to as *static model* as a baseline.

Evaluation Metrics We evaluate the effects of retraining based on drift detection compared to periodical retraining using three metrics. The first metric, ROC_AUC is related to the performance of the failure prediction model's performance. This metric shows how well these models distinguish between failures and non-failures. The other two metrics, the effectiveness per unit of retraining cost (ERC) and the effectiveness per unit of labeling cost (ELC) are derived from cost-effectiveness analysis [34]. The ROC_AUC takes values between [0, 1], where 1 corresponds to perfect prediction.

The ERC metric was proposed and used by previous work [6] to determine the benefits of different model retraining strategies for failure prediction models. This metric is calculated using Equation 2, where the portion of retrainings refers to the percentage of time periods that require updates, while the performance improvement is the percentage of ROC AUC improvement over the static model. A higher ERC corresponds to a more cost-effective strategy.

$$ERC = \frac{Performance\_Improvement}{Portion\_of\_Retrainings} \tag{2}$$

Although the ERC metric can be used to determine the cost-effectiveness with respect to the retraining frequency, it does not take into account the costs of labels. Therefore, we propose another cost-effectiveness metric, namely the ELC, which is defined by Equation 3. Similarly to ERC, the performance improvement shows the improvement of ROC AUC percentage over the static model, while the portion of labels represents the number of required labels to perform drift detection-based retraining divided by the total number of labels that are used to perform periodic retraining. Therefore, this metric determines how much the performance improves with respect to the label annotation costs. A higher ELC corresponds to a more cost-effective strategy.

$$ELC = \frac{Performance\_Improvement}{Portion\_of\_Labels} \tag{3}$$

V. Experiments

A. Monitoring the Skewness of Features Individually

Monitoring changes in features individually. This experiment aims to answer *RQ1.a*. In Figure 3 we depict the feature change rate for each individual feature used for each failure prediction model. This rate is calculated by dividing the number of times a feature changed in a drift/non-drift batch by the total number of drift/non-drift batches. A feature change rate of 100 shows that this feature has changed in all the drift/non-drift batches, while a feature change rate of 0 shows that this feature has never changed.

The idea behind this experiment is to discover whether specific features change only in drift batches since changes in those features can indicate drift. From Figure 3 we can observe that in the Google and Alibaba datasets, some features only change in drift batches, but this trend does not apply to the Backblaze dataset. However, the "Smart 193 Raw Diff" feature from Backblaze is changing more in drift batches compared to non-drift batches. This shows that changes in "Smart 193

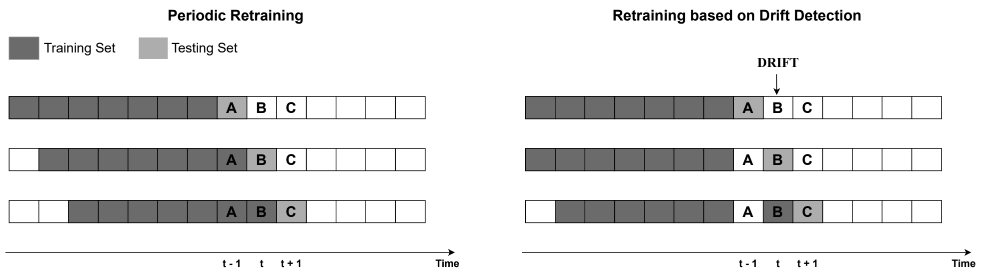

Fig. 2. Retraining periodically vs retraining based on drift detection strategies.

Raw Diff" can indicate drift, but it should not be used as the only drift indicator it might raise false positives when detecting drift. Furthermore, we can see that the majority of features do not change over time in the Backblaze dataset.

For the Google data, our experiments suggest that most of the features are changing in a similar proportion for both drift and non-drift batches. The only exception is the feature "Diff Machine" which has only changed within drift batches with a feature change rate of 0.06.

In the case of the Alibaba dataset, 6 out of 12 features only change in drift batches, namely "Avg GPU Work Mem", "Avg Mem", "CPU Usg", "GPU Work Util", "Max GPU Work Mem" and "Max Mem". These features refer mostly to the used resources when predicting job failures, such as GPU, CPU, or memory. However, due to the limited data availability in Alibaba, we only have one batch representing non-drift.

Monitoring the percentage of features that change over time. This experiment aims to answer *RQ1.b*. We depict our results regarding the effect of monitoring the percentage of features that change over time in Figure 3.

Our findings indicate no clear distinction between drift batches and non-drift batches when evaluating the percentage of features that change from one period to another across all three datasets. Figure 3 shows that in some cases (e.g. drift batches M3_4 of Backblaze, P8_9 of Google, and W2_3 of Alibaba) drift batches have a lower percentage of features that change compared to non-drift batches (e.g. non-drift batches M11_12 of Backblaze, P28_29 of Google and W1_2 of Alibaba). This observation is supported by low correlation coefficients obtained when correlating the percentage of features that change with the drift/non-drift batch label.

B. Monitoring Data Distribution

Assessing the Drift Detection Accuracy of Drift Detectors. With this experiment, we answer *RQ2.a*. We assess how well each drift detection technique is able to identify the testing batches corresponding to drift or non-drift. In this manner, we assess the drift detection accuracy for each detector. We show our results in Table I, where we can notice that data distribution techniques accurately detect the non-drifts in the Backblaze disk dataset, while in the job data (Google and Alibaba) they accurately identify drifts. None of the techniques detects the only non-drift batch (W1_2) in the Alibaba dataset. The KS_PCA achieves the highest score (0.90) when detecting non-drifts in both Google and Backblaze datasets. The fact that KS_PCA outperforms KS_ALL in identifying non-drifts suggests that reducing the dimensionality of the features when detecting drift prevents an excessive false positive rate.

TABLE I
DRIFT DETECTION ACCURACY METRICS FOR THE THREE DRIFT DETECTORS (KS_ALL, KS_PCA AND KS_FI). WITH **BOLD** WE DEPICT THE HIGHEST VALUE FOR EACH METRIC FOR EACH DATASET.

	Metric	KS_All	KS_FI	KS_PCA
Backblaze	Balanced Accuracy	0.65	0.01	**0.73**
	True Negative Rate	**0.09**	0.05	0.05
	True Positive Rate	0.80	0.78	**0.90**
Google	Balanced Accuracy	**0.63**	0.52	0.52
	True Negative Rate	**0.94**	0.82	0.65
	True Positive Rate	0.10	0.00	**0.30**
Alibaba	Balanced Accuracy	**0.57**	0.28	0.28
	True Negative Rate	**0.66**	0.33	0.33
	True Positive Rate	0.00	0.00	0.00

Assessing the Effects of Retraining based on Drift Detection. In this experiment, we assess the effects of including a drift detector in the maintenance pipeline of failure prediction models on their performances with the purpose of answering *RQ2.b*. We simulate a maintenance scenario in which the failure prediction model is retrained only when drift is indicated by one of the three evaluated drift detectors, saving both retraining times costs and retraining true labels costs. The results are summarized in Table II, where we present five retraining techniques, *Static*, *Periodic* and the three drift detection retraining techniques, *KS_All*, *KS_FI* and *KS_PCA*. The static retraining technique refers to the situation in which the model is never retrained. The periodic retraining technique refers to the situation in which the model is retrained periodically. We use Static and Periodic as our lower and upper baselines in terms of ROC_AUC. The other three drift detection retraining techniques refer to the situations in which the model is retrained every time the unsupervised drift detector, KS_All, KS_FI, and KS_PCA respectively, indicate the need to retrain.

From Table II we can notice that retraining based on a drift detector with feature reduction (either PCA or our proposed technique based on feature importance) preserves the performance of the model over time. These retraining strategies achieve similar performance with periodic retraining in the cases of Backblaze and Google (96% and 83% respectively) and similar performance in the case of Alibaba (61%

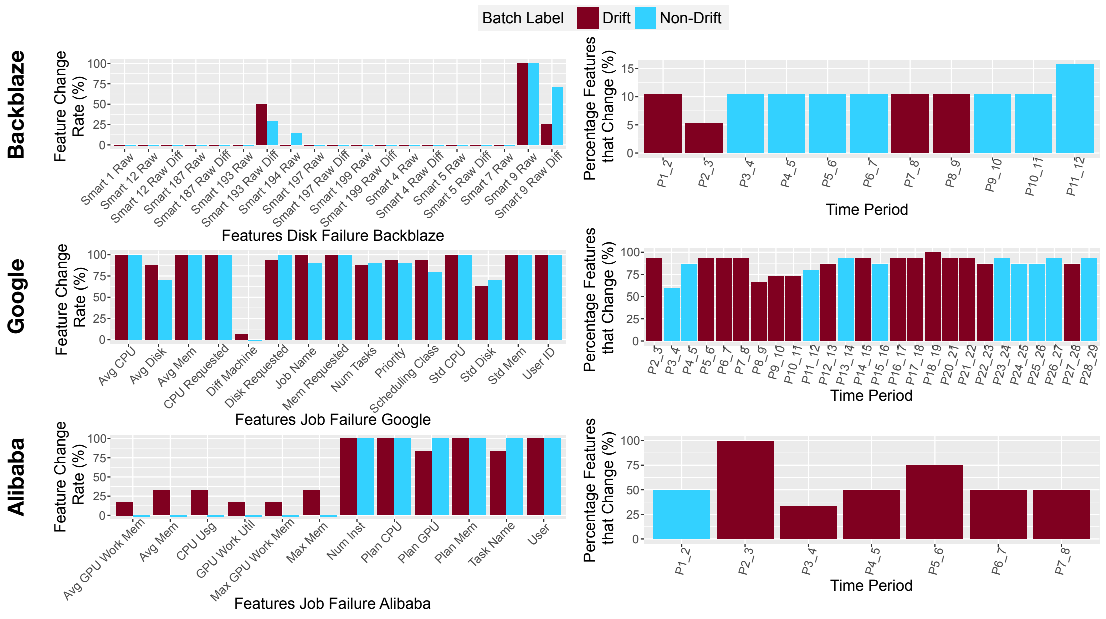

Fig. 3. (Left column) Feature Change Rate (in percentage) in Drift/Non-Drift Batches. (Right column) Ground Truth. Batches that contain drift and non-drift for all 3 datasets.

TABLE II
EFFECTIVENESS RETRAINING BASED ON DRIFT DETECTION VS STATIC (LOWER BASELINE) AND PERIODIC RETRAINING (UPPER BASELINE), MEASURED IN TERMS OF THE PERFORMANCE OF THE MODEL (ROC_AUC), PERCENTAGE OF RETRAININGS REQUIRED (RETRAININGS), EFFECTIVENESS PER UNIT OF RETRAINING COSTS (ERC), PERCENTAGE OF LABELS REQUIRED (LABELS), AND EFFECTIVENESS PER UNIT OF LABEL COSTS (ELC). BEST RESULTS (HIGHEST ROC_AUC, ERC, AND ELC AND LOWEST RETRAININGS AND LABELS) OUT OF THE 3 DRIFT DETECTION TECHNIQUES IN **BOLD**.

	Strategy	ROC_AUC	Retr.(%)	ERC	Labels(%)	ELC
Backblaze	Static	0.92	-	-	-	-
	KS_All	**0.96**	100	0.040	100	0.040
	KS_FI	0.95	**78**	**0.042**	**72**	**0.042**
	KS_PCA	**0.96**	95	**0.042**	95	**0.042**
	Periodic	0.96	100	0.040	100	0.040
Google	Static	0.77	-	-	-	-
	KS_All	**0.83**	**86**	**0.070**	**87**	**0.069**
	KS_FI	0.81	89	0.036	84	0.047
	KS_PCA	**0.83**	95	0.057	89	0.062
	Periodic	0.83	100	0.060	100	0.060
Alibaba	Static	0.58	-	-	-	-
	KS_All	0.58	0	0.000	0	0.000
	KS_FI	**0.61**	35	**0.090**	35	0.086
	KS_PCA	0.60	**25**	0.080	**20**	**0.102**
	Periodic	0.62	100	0.040	100	0.040

compared to 62% achieved by periodic retraining). KS_FI obtains the highest ERC score for Backblaze (0.042 and 0.090) Alibaba, making it the best technique for optimal performance with minimal retraining. Furthermore, it also achieves the highest ELC score for Backblaze (0.042), showing that using this retraining strategy is the best compromise between the performance and the required number of labels. KS_PCA yields similar scores for Backblaze but with higher retraining and labeling costs.

The KS_PCA retraining technique is the most efficient from the model performance perspective for Backblaze and Alibaba, achieving high ELC scores (0.042 and 0.102, respectively). For Alibaba, it requires the fewest retraining times (25%) and labels (20%), with only a 2% ROC_AUC loss compared to Periodic. KS_ALL is the best retraining technique for the Google dataset, with the highest ERC (0.070) and ELC (0.069) scores. However, KS_ALL is too sensitive when detecting drift for Backblaze, achieving similar performance with Periodic, and ineffective for Alibaba, achieving similar performance with Static.

VI. DISCUSSION

In this section, we answer each research question and discuss the findings resulting from our experiments.

Monitoring the Features Individually. Some features are linked with concept drift, but they should not be used alone as a drift indicator. However, monitoring the percentage of features that change is not a good indicator of concept drift for disk or job failure prediction models.

We demonstrate that changes in specific features (RQ1.a) can be linked with drift. Still, it cannot be used alone as a drift indicator, although considered best practice in monitoring machine learning systems [9]. Our results show that some features used to create job failure prediction models change only in drift-labeled batches. Examples are one feature, namely "Diff Machine" in Google dataset, and six features, namely "Avg GPU Work Mem", "Avg Mem", "CPU Usg", "GPU Work Util", "Max GPU Work Mem" and "Max Mem" in the Alibaba dataset. The features that change in the Alibaba dataset are related to the used resources (memory, CPU, and GPU), and for this dataset, they solely change during drift batches. However, the features related to the used resources (disk, memory, CPU) in the Google dataset change in both drift and non-drift batches. This shows that the features related to used resources are not generally an indicator of concept drift and AIOps practitioners should identify which features indicate model degradation for their AIOps models. Regarding disk failure prediction, there was no feature changing in solely drift batches. The only feature that exhibited changes in more drift batches than non-drift batches is the "Smart 193 Raw Diff" (Load Cycle Count) feature. However, our results indicate that using this feature as a concept drift indicator leads to many false alarms.

Monitoring the proportion of features that changed was not linked with concept drift for either of the evaluated AIOps datasets. This shows that although considered best practice in concept drift monitoring [9], it cannot indicate drift in failure prediction datasets (RQ1.b). Therefore, we recommend practitioners investigate other unsupervised drift detection techniques.

Monitoring the Data Distribution over Time. Some data distribution-based drift detectors can accurately identify drifts, but which detector to employ is dataset-dependent. Retraining based on unsupervised data distribution drift detectors is beneficial, obtaining similar performance with periodic retraining and lowering the retraining and label costs.

In our last experiment, we simulate the scenario in which a failure prediction model is deployed into production and evaluated periodically on the upcoming batches. We compare the situation in which the model is never retrained (Static), the situation in which the model is retrained based on one of the three unsupervised drift detectors, and the situation in which the model is retrained periodically. Our results demonstrate that retraining based on unsupervised drift detectors is promising since it achieves similar performance to periodic retraining and lowers both the number of retraining times and the number of required true labels. Our findings suggest that employing unsupervised drift detectors as data monitoring tools is a promising strategy to lower the retraining labels' costs while preserving accuracy (RQ2.b).

Another important conclusion drawn from our results is that no unsupervised drift detector achieves the best results on all three analyzed datasets (RQ2.a). This shows that choosing the most suitable drift detector depends on the AIOps application and dataset. Therefore, AIOps practitioners have to experiment with their datasets to identify the most suitable unsupervised drift detection technique.

In our experiments, we evaluate both the accuracy of drift detection (RQ2.a.) and the effects of retraining based on drift detection (RQ2.b) to understand whether we can link drift detection accuracy to its effects when used as a model retraining indicator. This approach allows practitioners to identify the suitable drift detector for their datasets by evaluating drift detection accuracy on their training data. However, our results suggest that the behavior of the unsupervised drift detection techniques is different in the two evaluation scenarios. For instance, the KS_All drift detector identified 66% of the drifts on the Alibaba dataset in the former evaluation scenario (Table I), while in the latter scenario (Table II) it was not able to detect any drift. Thus, the accuracy of drift detection on smaller batches does not reflect how the detector is behaving in a production environment. Therefore, we suggest that practitioners allocate a testing period for selecting the most appropriate drift detector. In this testing period, practitioners can employ our proposed pipeline and carefully analyze the impact and required costs of retraining based on each drift detector while compared to never retraining the model (lower baseline) and periodically retraining a model (upper baseline).

VII. Threats to Validity

An **external threat to validity** is generalizability, as we used only three publicly available datasets (Backblaze, Google, Alibaba) and focused on solely existing failure prediction models without exploring techniques to improve the model (e.g. adding a new feature). However, the dataset samples are representative of real-world machines since they are published by well-known organizations for research purposes. For **internal validity**, we split data into periods and train-test sets consistently with previous work [3], [4], [6]. Regarding **construct validity**, we used Randomized Search for hyperparameter tuning, fixed iteration time to 100, applied undersampling to achieve a 10:1 ratio of non-failure to failure samples, and used 10 random seeds to minimize bias. For feature selection in KS_FI, we chose features with importance above the mean.

VIII. Conclusions and Future Work

The main goal of this article is to understand to what extent unsupervised data monitoring tools can be employed in real-world failure prediction models to identify concept drift. Furthermore, we aim to quantify the benefits of employing an unsupervised drift detector in the maintenance pipeline of a failure prediction model in terms of the number of retrainings and label costs with respect to the model performance trade-offs. To do so we extracted the best practices in unsupervised techniques for monitoring machine learning systems suggested by industry practitioners, such as *monitoring the skewness of features over time* [9] (*monitoring the percentage of features that change* and *monitoring the skewness of features over time*) and *monitoring the data distribution over time* [8]–[10]. We applied them to three failure prediction models and verified how well they indicated concept drift (model degradation). The employed datasets are representative of real-world data since they were either provided by real-world organizations (Google and Alibaba) or contain data collected from various hardware devices (Backblaze) [21].

We empirically show that monitoring the percentage of the features that change is not correlated with the presence of drift. Our experiments suggest that some features can be linked to the presence of drift, but they cannot be used alone as drift indicators. However, out of the best practices in unsupervised monitoring machine learning systems, monitoring the data distribution is the most promising technique. Therefore, unlike previous work proposing periodic model retraining [3], [4] or retraining based on supervised drift detectors [6], we demonstrate that unsupervised data distribution-based drift detectors effectively indicate when to retrain failure prediction models, reducing retraining and labeling costs. Furthermore, in this paper, we proposed integrating a feature importance technique extracted from the model into data distribution-based drift detectors. This technique extends beyond Random Forests and can be applied to any classifier where key features can be extracted, making it widely applicable. The feature importance based detector (KS_FI) was the most cost-effective retraining for Backblaze and Alibaba datasets. Furthermore, we address the model monitoring research gap [35] by proposing a pipeline that helps AIOps practitioners evaluate unsupervised drift detectors and select the most suitable one to monitor an AIOps model. Our experiments establish a foundation for evaluating drift detectors in the MLOps monitoring pipeline of failure prediction models.

Given the promising results of unsupervised drift detectors, as future work, we aim to expand our drift selection pipeline into a framework that AIOps practitioners can employ to identify the most suitable unsupervised drift detectors for their datasets or applications. However, we do not recommend a specific drift detector since some applications prioritize capturing more drifts, while others aim to reduce false alarms. We seek to expand our analysis to other AIOps applications such as node failure prediction or incident prediction.

REFERENCES

[1] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: Real-world challenges and research innovations. In *ICSE-Companion 2019*, pages 4–5, 2019.

[2] Li et al. Going through the life cycle of faults in clouds: Guidelines on fault handling. In *ISSRE 2022*, pages 121–132, 2022.

[3] Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. An empirical study of the impact of data splitting decisions on the performance of aiops solutions. *ACM Trans. Softw. Eng. Methodol.*, 30(4), jul 2021.

[4] Yingzhe Lyu, Gopi Krishnan Rajbahadur, Dayi Lin, Boyuan Chen, and Zhen Ming (Jack) Jiang. Towards a consistent interpretation of aiops models. *ACM Trans. Softw. Eng. Methodol.*, 31(1), 2021.

[5] Anas Dakkak, Jan Bosch, and Helena Olsson. Towards aiops enabled services in continuously evolving software-intensive embedded systems. *Journal of Software: Evolution and Process*, page e2592, 2023.

[6] Yingzhe Lyu, Heng Li, Zhen Ming Jiang, and Ahmed E. Hassan. On the model update strategies for supervised learning in aiops solutions. *arXiv preprint arXiv:2311.03213*, 2023.

[7] Mark Haakman, Luís Cruz, Hennie Huijgens, and Arie van Deursen. Ai lifecycle models need to be revised. *Empirical Software Engineering*, 26(5):1–29, 2021.

[8] Amershi et al. Software engineering for machine learning: A case study. In *2019 ICSE-SEIP*, pages 291–300, 2019.

[9] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. *Big Data 2017*, 2017.

[10] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *ACM International Conference on Management of Data*, 2017.

[11] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch. Software engineering challenges of deep learning. In *SEAA 2018*, pages 50–59, 2018.

[12] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 2022.

[13] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31:2346–2363, 2019.

[14] L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeyer. Are concept drift detectors reliable alarming systems? - a comparative study. In *Big Data 2022*, 2022.

[15] Lin et al. Predicting node failure in cloud service systems. In *ESEC/FSE 2018*, 2018.

[16] Abdul Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering*, 25:1–38, 11 2020.

[17] Tim Schröder and Michael Schulz. Monitoring machine learning models: a categorization of challenges and methods. *Data Science and Management*, 5(3):105–116, 2022.

[18] Sculley et al. Hidden technical debt in machine learning systems. In *NIPS'15*, NIPS'15, page 2503–2511, 2015.

[19] Backblaze Inc. Backblaze hard drive stats. https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data.

[20] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *USENIX ATC 17*, pages 391–402, 2017.

[21] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '16, page 39–48, 2016.

[22] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., November 2011. Revised 2012.03.20. Posted at http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[23] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *ICDCS 2017*, 2017.

[24] Alibaba Group. 2021. alibaba cluster trace program. https://github.com/alibaba/clusterdata.

[25] Weng Qizhen, Xiao Wencong, Yu Yinghao, Wang Wei, Wang Cheng, He Jian, Li Yong, Zhang Liping, Lin Wei, and Ding Yu. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters. In *USENIX 2022*, 2022.

[26] Divish Rengasamy, Benjamin C. Rothwell, and Grazziela P. Figueredo. Towards a more reliable interpretation of machine learning outputs for safety-critical systems using feature importance fusion. *Applied Sciences*, 11(24), 2021.

[27] Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo A. Oliva, Yasutaka Kamei, and Ahmed E. Hassan. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*, 48(7):2245–2261, 2022.

[28] Grace A. Lewis, Sebastián Echeverría, Lena Pons, and Jeffrey Chrabaszcz. Augur: a step towards realistic drift detection in production ml systems. In *SE4RAI '22*, page 37–44, 2023.

[29] Abdulhakim Qahtan, Basma Alharbi, suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams. In *KDD*, 2015.

[30] Xiao Li, Yu Wang, Sumanta Basu, Karl Kumbier, and Bin Yu. *A Debiased MDI Feature Importance Measure for Random Forests*, chapter 723. Curran Associates Inc., 2019.

[31] Divish Rengasamy, Jimiama Mafeni Mase, Aayush Kumar, Benjamin Rothwell, Mercedes Torres Torres, Morgan R. Alexander, David A. Winkler, and Grazziela Patrocinio Figueredo. Feature importance in machine learning models: A fuzzy information fusion approach. *Neurocomputing*, 511:163–174, 2022.

[32] Marco Sandri and Paola Zuccolotto. A bias correction algorithm for the gini variable importance measure in classification trees. *Journal of Computational and Graphical Statistics*, 17:1–18, 09 2008.

[33] Zhengze Zhou and Giles Hooker. Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15:1 – 21, 2019.

[34] Peter J. Neumann, Theodore G. Ganiats, Louise B. Russell, Gillian D. Sanders, and Joanna E. Siegel. *Cost-Effectiveness in Health and Medicine*. Oxford University Press, 12 2016.

[35] Meenu Mary John, Helena Holmström Olsson, Jan Bosch, and Daniel Gillblad. Exploring trade-offs in mlops adoption. In *APSEC 2023*, pages 369–375, 2023.