

# McUDI: Model-Centric Unsupervised Degradation Indicator for Failure Prediction AIOps Solutions

Lorena Poenaru-Olaru<sup>1</sup>, Luis Cruz<sup>1</sup>, Jan Rellermeyer<sup>2</sup> and Arie van Deursen<sup>1</sup>

<sup>1</sup>Software Engineering, TU Delft, Netherlands

<sup>2</sup>Dependable and Scalable Software Systems, Leibniz University Hannover, Germany

{L.Poenaru-Olaru, L.Cruz, arie.vandeursen}@tudelft.nl, rellermeyer@vss.uni-hannover.de

## Abstract

Due to the continuous change in operational data, AIOps solutions suffer from performance degradation over time. Although periodic retraining is the state-of-the-art technique to preserve the failure prediction AIOps models' performance over time, this technique requires a considerable amount of labeled data to retrain. In AIOps obtaining label data is expensive since it requires the availability of domain experts to intensively annotate it. In this paper, we present McUDI, a model-centric unsupervised degradation indicator that is capable of detecting the exact moment the AIOps model requires retraining as a result of changes in data. We further show how employing McUDI in the maintenance pipeline of AIOps solutions can reduce the number of samples that require annotations with 30k for job failure prediction and 260k for disk failure prediction while achieving similar performance with periodic retraining.

## 1 Introduction

Given the large amount of operational data generated by large-scale software systems, manual inspection for abnormalities or failures in the systems becomes impractical. This led to the appearance of the AIOps (Artificial Intelligence for IT Operations) research field. AIOps is a specific machine learning (ML) application that applies machine learning techniques to operational data to identify or predict failures or anomalies that occur in a large-scale software system.

As observed in other real-world ML applications, such as healthcare, manufacturing, finance, and agriculture [Vela *et al.*, 2022], one of the biggest challenges in building AIOps models is the temporal quality degradation (AI aging) [Dang *et al.*, 2019], [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b]. The model quality (performance/accuracy) degradation is a consequence of the evolving character of operational data, also known as concept drift. In some situations, concept drift leads to severe model degradation [Bayram *et al.*, 2022], which impacts the reliability and trustworthiness of AIOps models. Moreover, a considerable degradation in the quality of AIOps models generates service interruptions [Dang *et al.*, 2019],

which causes substantial monetary losses (between \$300,000 and \$400,000 according to [Lyu *et al.*, 2021b]).

To overcome the challenge of model performance degradation, previous work proposes to periodically retrain (update) AIOps models [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b]. However, in a practical scenario, periodic retraining is sometimes unfeasible due to the hidden deployment costs [Haakman *et al.*, 2021]. Once a model is updated, it undergoes compliance verification, which is time-consuming [Haakman *et al.*, 2021], and afterward, engineers need to ensure its smooth integration with the software system that incorporates the AIOps model [Amershi *et al.*, 2019]. Thus, a solution that reduces the number of necessary updates without harming the performance of the model is desired.

Previous work [Lyu *et al.*, 2023] proposed a more systematic solution to periodic model retraining, namely retraining only when necessary. This solution involves updating an AIOps model when a model degradation indicator detects a significant drop in its performance. However, this solution requires having true labels available in real-time, which implies that operational engineers continuously manually annotate data, making it unfeasible for real-world scenarios. Thus, a label-independent model degradation indicator is required.

To overcome these limitations, we propose McUDI a novel model-centric label-independent performance degradation indicator, which indicates when an AIOps failure prediction model requires retraining. McUDI computes the feature importance ranking from the trained AIOps failure prediction model and selects the most important features according to the average importance rank. To identify drift, McUDI applies the Kolmogorov-Smirnov (KS) statistical test on the data distribution computed from the most important features. Our contributions are the following:

1. We propose a model-centric unsupervised failure prediction AIOps models degradation indicator and evaluate it on two operational datasets.
2. We propose a maintenance pipeline that incorporates McUDI which reduces the need for high-quality labels necessary for retraining while preserving the performance of AIOps solutions.
3. We evaluate the capability of the state-of-the-art data monitoring practices to identify performance degradation on failure prediction data.

4. We publicly share our reproduction package<sup>1</sup>.

## 2 Related Work

### 2.1 Concept Drift Definition and Evaluation

In literature, the term concept drift generally refers to changes in the data that occur over time [Bayram *et al.*, 2022]. Concept drift detectors are monitoring tools developed to capture these changes and raise alerts when they occur [Lu *et al.*, 2019]. Plenty of drift detectors were proposed for classification problems [Bayram *et al.*, 2022]. Some of the existing concept drift detection techniques monitor the changes in accuracy over time and they are, therefore, label-dependent, while others identify drifts only in the characteristics of the dataset, such as the data distribution [Lu *et al.*, 2019]. Although the label-dependent drift detection techniques are reliable model performance degradation indicators, they are expensive to employ in real-world scenarios since true labels are not immediately available. Thus, a better alternative to real-world applications is the usage of data distribution-based drift detectors, which do not rely on labels. However, previous work [Poenaru-Olaru *et al.*, 2022] has shown that these types of detectors are quite sensitive to small changes in the data that do not necessarily affect the accuracy of ML models.

When it comes to evaluation, concept drift detectors are evaluated in two manners: *assessing the drift detection accuracy* and *assessing the performance of a machine learning model over time*. The former assessment technique measures drift detectors' capability to correctly label drifts and non-drifts. Given that in real-world scenarios the moment when drift occurs is difficult to determine, this technique is employed when experimenting with synthetic data where the drift occurrence is fixed [Lu *et al.*, 2019], [Poenaru-Olaru *et al.*, 2022]. However, previous work presented a technique to label testing batches into drift and non-drift respectively using a Z test on the error-rate [Lyu *et al.*, 2021a]. The latter assessment technique is usually preferred in real-world cases since knowledge regarding the moment of drift occurrence is not required [Bayram *et al.*, 2022]. In this assessment technique, to understand the impact of the detector, the performance (e.g. accuracy, ROC-AUC, etc) of an ML model that is retrained based on a drift detector is compared to the performance of a model that is never retrained [Lu *et al.*, 2019].

### 2.2 AIOps Models Degradation due to Concept Drift

AIOps models positively influence software delivery acceleration and software systems quality and security [Dang *et al.*, 2019]. Despite their potential, it has been demonstrated that the performance of these models is not generalizable over time due to concept drift. AIOps models developed for defect prediction trained on current data cannot generalize well on future data [Bangash *et al.*, 2020]. Operational data is constantly changing/evolving (concept drift occurs) due to uncontrollable factors such as user workloads or hardware/software upgrades. Concept drift on operational data

negatively impacts the failure prediction AIOps models performances, resulting in an increase in their error rates [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b], [Li *et al.*, 2020].

Previous work highlights the importance of periodically updating AIOps models to mitigate the effect of concept drift and the necessity of monitoring AIOps models against concept drift [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b]. Research has been done on the effects of retraining failure prediction models based on a degradation indicator monitoring tool that requires labels [Lyu *et al.*, 2023]. However, there is currently no research on unsupervised model degradation indicators.

### 2.3 Data Monitoring

Model monitoring refers to a set of tests performed after the model is deployed into production [Schröder and Schulz, 2022]. One essential part of monitoring machine learning models is data monitoring against concept drift.

One data monitoring technique is monitoring the number of features that become skewed over time [Breck *et al.*, 2017]. This technique checks whether one feature in the training data behaves significantly differently than the same feature in the testing data and was previously used to assess data quality [Schelter *et al.*, 2018]. Another popular data monitoring technique is continuously checking if the distribution of data changes over time [Schröder and Schulz, 2022], [Breck *et al.*, 2017], [Polyzotis *et al.*, 2017], [Sculley *et al.*, 2015]. A change in the data distribution between training and testing implies that the model needs to be rebuilt [Amershi *et al.*, 2019] or retrained with newer samples [Arpteg *et al.*, 2018]. Although these monitoring techniques have been proposed, their capability of identifying the degradation of AIOps models has not yet been assessed.

## 3 Proposed Method

### 3.1 Background Feature Importance

Feature importance (FI) analysis has been used previously as a technique to explain the output of a certain machine learning model in both scientific and industrial settings [Rengasamy *et al.*, 2022], [Rengasamy *et al.*, 2021], [Li *et al.*, 2019]. The FI is a feature ranking technique that shows how much each feature contributes to the predictive capability of a classifier [Rajbhadur *et al.*, 2022].

According to [Li *et al.*, 2019] although the FI ranking computation is beneficial for understanding the behavior of classifiers, there is a high interest in developing techniques suitable for specific classifiers such as Random Forests [Breiman, 2001] and other tree-based classifiers [Zhou and Hooker, 2019]. The reason for this is wide applicability within different research areas of tree-based classifiers. Thus the most common metric to compute the FI ranking from tree-based classifiers is the mean decrease in impurity (MDI) [Sandri and Zuccolotto, 2008], [Huynh-Thu *et al.*, 2010], [Breiman *et al.*, 1984], [Li *et al.*, 2019], [Zhou and Hooker, 2019], also known as Gini importance. For each feature, this technique calculates the total decrease in impurity (loss) computed for each random split [Li *et al.*, 2019].

<sup>1</sup>Replication Package: [https://github.com/LorenaPoenaru/aiops\\_failure\\_prediction](https://github.com/LorenaPoenaru/aiops_failure_prediction)

### 3.2 McUDI

McUDI is derived from one of the most commonly used unsupervised techniques to detect drift the Kolmogorov-Smirnov (KS) statistical test. This test is preferred over others since it is non-parametric, label-independent [Gama *et al.*, 2014], has low computational costs [dos Reis *et al.*, 2016], and does not require a manually adjusted drift threshold [Lu *et al.*, 2019]. The KS test is usually applied to assess whether the data distribution in the training set is different than the one in the testing set. Despite its popularity, one major limitation of KS is the high number of false alarms caused by the fact that the data distributions are computed using all the available features, while any change does not influence the model’s performance in the data [Poenaru-Olaru *et al.*, 2022]. McUDI aims to overcome this limitation by combining a technique of extracting the model’s most important features with the KS drift detection method. The major difference between McUDI and KS is the fact that while KS computes the distribution using all available features, McUDI computes the distribution using solely the most important features, being, thereby, a model-centric unsupervised drift detector. McUDI is composed of 3 important steps, namely **extract important features, compute distributions, statistical test significance**.

**Extract Important Features.** We use the mean decrease in impurity to rank features according to their importance to the trained model. We extract the average importance rank among all features and consider the features with feature importance rank higher than the average relevant and discard the rest. Considering the average importance instead of a fixed percentage of important features increases the generalizability of McUDI since some models might consider 80% of the features important, while others might consider only 50%.

**Compute the Distributions.** After the less important features are filtered out, the data distribution needs to be estimated. Thus, in this step, the distribution of the training data and the distribution of the testing data on the most important features is computed.

**Statistical Test Significance.** The KS statistical test is employed to assess the similarity/dissimilarity between the distributions of the most important features from the training and testing data. The null hypothesis of this test is that the two distributions are similar. An alarm is raised (drift is detected) when the p-value is smaller than 0.05 (95% confidence interval).

## 4 Research Questions

This section highlights the research questions we aim to answer in our study.

1. How well do state-of-the-art (SOTA) concept drift monitoring techniques identify model degradation in failure prediction AIOps solutions (job & disk failure)?
2. How does our proposed model-centric approach (McUDI) compare to traditional SoTA monitoring practices?
  - (a) How does McUDI compare to traditional SOTA monitoring practices in terms of drift detection accuracy?
  - (b) How does McUDI compare to traditional SOTA

monitoring practices in terms of model performance over time?

3. What is the effect of including McUDI in the maintenance pipeline of failure prediction AIOps solutions in terms of label costs compared to existing practices?

## 5 Evaluation Methodology McUDI

In this section, we present the employed operational datasets, the baseline methods, and performance evaluation.

### 5.1 Data & Failure Prediction AIOps Model

#### Data

For our study, we employ two open-source AIOps datasets for which previous work has shown that concept drift occurs [Lyu *et al.*, 2021a].

The first dataset is the **Google Cluster Traces Dataset** [Reiss *et al.*, 2011] and contains information about traces extracted from real-world large cluster systems. This data was collected for 29 days (May 2011). In total, there are around 625K samples. Previous work [Lyu *et al.*, 2021a], [El-Sayed *et al.*, 2017] used this dataset to design job failure prediction models.

The second dataset is the **Backblaze Disk Stats Dataset** [Inc., ] and contains information about HDD hard drive stats from 2013. Similar to previous work [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b] we are using 12 months of data corresponding to the year 2015, which contains around 7M samples of data. This dataset was used to design disk failure prediction models [Mahdisoltani *et al.*, 2017], [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b], [Botezatu *et al.*, 2016].

#### Data Preprocessing and Feature Selection

To build the AIOps model that predicts disk failure, we employ 19 features out of which some of them have values accumulated over time, such as the “reallocated sectors coun” features, while others contain non-accumulated values over time, such as “read error rate”. When building the AIOps model that predicts job failure, we employ a total of 15 features, out of which 6 are temporal features, such as the “standard deviation of CPU”, and 9 are configuration features, such as “memory and disk space”.

Unlike the original approach presented in [Lyu *et al.*, 2021a], we use a slightly different scaling method. Instead of fitting the scaler on the entire dataset and therefore transforming it, we use the scaler fitted on the first period of data only to transform the datasets from both the first and the second period. We argue that this data scaling method is more realistic for model deployment because using the whole dataset in this context would assume that we have access to future data.

#### Failure Prediction AIOps Model

We build the failure prediction AIOps model using a Random Forests Classifier. We specifically choose a tree-based classifier since these types of classifiers are very well researched in terms of feature importance ranking extraction [Rengasamy *et al.*, 2022], [Rengasamy *et al.*, 2021], [Rajbahadur *et al.*, 2022] which is an important step in McUDI. Out of the tree-based classifiers we specifically choose Random Forests since it is a commonly used tree-based classifier in failure prediction AIOps solutions [Lyu *et al.*, 2021a], [Lyu *et al.*,

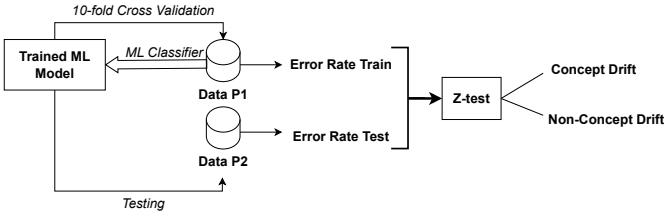


Figure 1: Obtaining the ground truth. Pipeline to assess the presence of concept drift between two batches (Data P1 and Data P2).

al., 2021b], [Botezatu *et al.*, 2016], [El-Sayed *et al.*, 2017], [Mahdisoltani *et al.*, 2017], [Lyu *et al.*, 2023]. The hyperparameters of the Random Forests classifier are chosen according to previous work [Lyu *et al.*, 2021a].

## 5.2 Baseline Model Degradation Indicators

We begin our experiments by understanding how well the two data monitoring techniques, monitoring the number of features that change and monitoring changes in data distribution, capture model performance degradation. When monitoring the number of changing features we count the number of features that are different from training and testing using the KS statistical test. We further assess how well KS captures drift.

## 5.3 Performance Evaluation

To evaluate McUDI we employ the two aforementioned concept drift evaluation techniques, the drift detection accuracy assessment, and the performance over time of failure prediction model assessment.

### Evaluation based on Drift Detection Accuracy

To evaluate drift detectors based on their accuracy in correctly distinguishing between drifts and non-drifts, we first need to obtain the ground truth in terms of which testing batch is a drift and which one is a non-drift.

**Extracting the Ground Truth.** To extract the ground truth regarding when concept drift occurs we follow the same technique as previous work [Lyu *et al.*, 2021a]. This technique implies that machine learning data is trained on data corresponding to the previous time period and evaluated on data from the current time period. The assumption is that the performance of the model on the training data (past data) is not statistically different than the performance of the model on the testing data (current data). If this assumption does not hold, then there is concept drift. In our paper, we assess the performance of the model on training and testing by computing the prediction error rate on training and testing, respectively, similar to previous work [Lyu *et al.*, 2021a].

In Figure 1 we depict the pipeline of identifying whether concept drift occurs 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). As it can be observed from Figure 1 we compute the error rate on training by performing a 10-fold cross-validation on Data P1 and the error rate on testing 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 or not.

The formula of this test is the following:

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

where  $\epsilon_{train}$  is the prediction error rate on the training set,  $\epsilon_{test}$  is the prediction error rate on the testing set,  $\epsilon$  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 machine learning model's performance on datasets extracted from the two different periods, implying that there is no concept drift. The null hypothesis is rejected when the p-value of the Z-test is lower than 0.05. The severity of concept drift is calculated as  $((\epsilon_{test} - \epsilon_{train}) / \epsilon_{train})$ .

**Assessing the Drift Detection Accuracy.** Given that in each dataset, the ratio between the number of drifts and non-drifts is highly imbalanced, we employ three metrics that take into account the imbalance between the two, namely the *Balanced Accuracy*, the *Specificity* and the *Sensitivity*. The Specificity shows how many drifts are correctly identified out of the total number of drifts. The Sensitivity shows how many non-drifts are correctly identified out of the total number of non-drifts. The Balanced Accuracy is the arithmetic mean between the Sensitivity and Specificity and it shows how many drifts and non-drifts are overall correctly classified.

$$\text{Specificity} = \frac{TN}{TN + FP} \quad (2)$$

; where  $TN$  corresponds to True Negatives and  $FP$  corresponds to False Positive

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (3)$$

; where  $TP$  corresponds to True Positive and  $FN$  corresponds to False Negatives

$$\text{BalancedAccuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2} \quad (4)$$

### Evaluation based on Model Performance Preservation

In this drift detection evaluation technique, we study the effects on the model performance of not updating the model at all (static), updating the model periodically (the most commonly used technique to mitigate the effects of concept drift on AIOps data), and updating the model based on a drift detection technique. By comparing updating a static model with updating a model based on a drift detection technique, we aim to understand whether including a drift detection technique as a retraining indicator significantly increases the performance of a model compared to never updating it. By comparing a periodically updated model with a model updated based on a drift detector we aim to investigate whether the drift detection techniques manage to capture enough drifts to achieve comparable model performance with periodic updates.

In Figure 2 we depict the difference between periodic re-training and retraining based on drift detection. The former

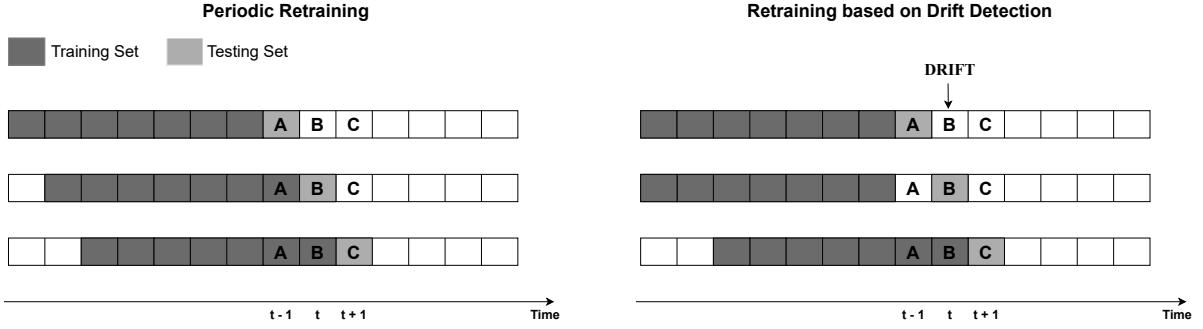


Figure 2: Difference between retraining periodically and retraining based on drift detection.

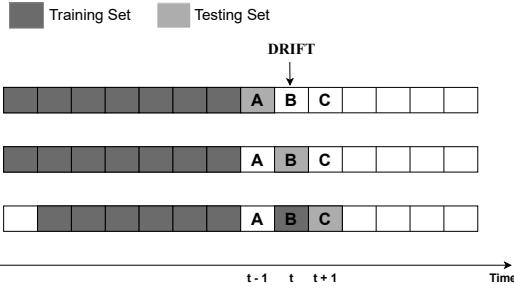


Figure 3: Label cost-efficient maintenance pipeline including McUDI for failure prediction models.

updates a model after each period, while the latter only updates a model when a drift is detected. In our approach, we employ the same sliding-window-based model retraining technique as previous work [Lyu *et al.*, 2021b], [Lyu *et al.*, 2021a] where the most recent data is used when retraining.

#### Evaluation Label Cost

To understand the benefits of a drift detector in reducing the number of times a model is retrained and redeployed and the cost of obtaining good-quality labels, we propose a new evaluation strategy depicted in Figure 3. In this evaluation strategy, the retraining is performed solely when drift is indicated by McUDI and only batch B, where the drift was identified, is included in the training set instead of including the most recent data (both batch A and B) as previous work [Lyu *et al.*, 2021a], [Lyu *et al.*, 2021b]. If no drift is detected in batch A, the data distribution of batch A is similar to the data distribution in the training data. Therefore, retraining on batch A comes with the cost of obtaining labels for this data without including new information in the training set.

## 6 Experiments

We begin the experimental section by presenting how we extracted the testing batches when concept drift impacts the model performance. We further show our results on the performance of different concept drift indicators in identifying these testing batches.

### 6.1 Extracting the Ground Truth

To avoid bias in our experiments, we extract the error rate of the employed classifier for 10 random seeds and share them in our open-source package to facilitate reproducibility.

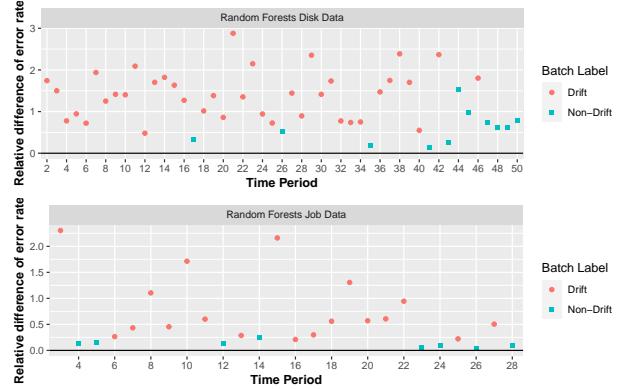


Figure 4: Ground Truth. Batches that contain drift and non-drift for disk and job datasets.

In our experiments, we initially employ the same intervals as previous work [Lyu *et al.*, 2021a], namely one-day periods for the Google dataset and one-month period for the Backblaze dataset. However, while performing the experiments, in most situations drift occurs in all batches of the latter dataset. Since we aim to understand the ability of a drift detector to both identify drift and not raise false alarms, we decreased the size of the period from one month to one week. The reason for this is that from our experiments, the data is significantly different from one month to another and a model trained on the previous month cannot generalize well on the data corresponding to the following month. The same results were not reported in the original work [Lyu *et al.*, 2021a], but this can be a consequence of the difference in data scaling methods.

Figure 4 depicts the identified drifts based on the error rate for both datasets and their severity reported as the relative difference of error rate. We consider a data batch status of either drift or non-drift. A drift label is given to batches that were labeled as drift for more than half of the number of random seeds, while the others are labeled as non-drift.

When it comes to the Google dataset, due to high-class imbalance, the data batch corresponding to the first period (first day out of 29 days) contains only samples from the majority class, which makes it impossible for the classifier to learn to distinguish between failure and non-failure. Therefore, we cannot analyze whether there is a concept drift between the first and second periods and we exclude it from the plot.

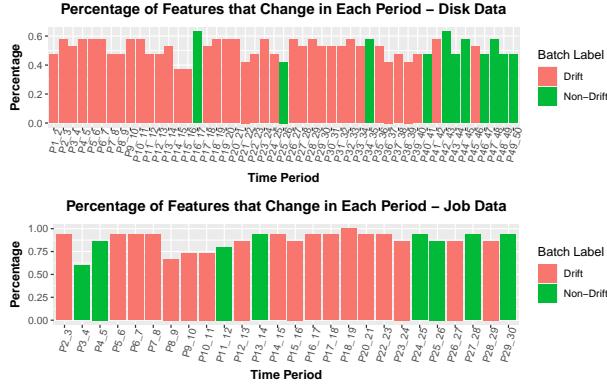


Figure 5: Percentage of features that change in each period and their corresponding batch label (drift/non-drift).

## 6.2 SoTA Degradation Indicators

This set of experiments aims to answer the first research question and understand how well state-of-the-art model degradation indicators (number of features that change and changes in data distribution) manage to capture the drifts in the two AIOps datasets.

### Monitoring Features Individually

We aim to understand whether the number of features that change is a good indicator of drift. In Figure 5 we display the number of features that are changing in each consecutive two testing batches. It can be noticed that the batches that are not labeled as drift do not necessarily contain fewer features that change, and sometimes fewer features that change are in batches that are labeled as drift. Thus, model degradation is not linked to the number of individual features that change.

### Monitoring the Distribution of the Data

To understand how accurately KS detects drifts, we compute the three aforementioned accuracy metrics and display the results in Table 1. We can observe that the sensitivity is close to 0.0 for both job and disk datasets, which shows that KS does not manage to capture non-drifts. However, we can see that KS identifies plenty of drifts, especially on the job dataset, where the specificity is 90%. Thus, KS can accurately capture drift with the cost of raising plenty of false alarms.

## 6.3 Model-Centric Degradation Indicator

With this experiment, we answer our second research question and investigate the performance of a model-centric degradation indicator (McUDI) compared to SoTA practices. We evaluate McUDI based on the drift detection accuracy (RQ2a) and the performance preservation over time (RQ2b).

### Evaluation based on Drift Detection Accuracy

We evaluate the drift detection performance of McUDI in comparison with a degradation indicator corresponding to periodically updating a failure prediction model (Periodic) and to the degradation indicator that is not model-centric (KS).

We show our results in Table 1. The Periodic degradation indicator identifies all drifts (specificity of 1.0) and does not identify any non-drifts (sensitivity of 0.0). It can be noticed

Table 1: Drift detection accuracy metrics for Periodic, KS, and McUDI. With **bold** we depict the highest value for each metric for each dataset.

	Metric	Periodic	KS	McUDI
Disk	Balanced Accuracy	0.5	0.41	<b>0.57</b>
	Specificity	<b>1.0</b>	0.72	0.69
	Sensitivity	0.0	0.09	<b>0.44</b>
Job	Balanced Accuracy	0.5	<b>0.45</b>	0.42
	Specificity	<b>1.0</b>	0.90	0.82
	Sensitivity	0.0	0.00	<b>0.01</b>

from Table 1 that the sensitivity of KS is similar to the one of Periodic for disk data (0%), suggesting that the KS degradation indicator closely resembles periodic model retraining.

On the other side, McUDI achieves the highest sensitivity for both datasets, showing its ability to accurately identify non-drifts and raise fewer false alarms compared to Periodic and KS. The highest sensitivity obtained by McUDI is 0.44 for disk data. In a more rigorous analysis, we noticed that for the disk dataset, the Random Forest classifier considers around 7 out of 19 features as important based on the mean importance threshold that is used within McUDI. This may indicate that the high number of non-important features that change severely impact the data distribution and can justify the high number of false alarms raised by KS and why McUDI is beneficial for this dataset. When it comes to the job dataset, around 9 out of 15 features are considered important, which can indicate that changes in non-important features do not considerably impact the data distribution. Therefore, we demonstrated that McUDI achieves good results in properly detecting non-drifts when the classifier considers less than half of the employed features as important. Furthermore, McUDI identifies more than 65% of the occurring drifts for both datasets, obtaining a specificity of around 69% and 82% for disk and job data, respectively.

### Evaluation based on Performance Preservation

With this experiment, we aim to understand the effects of having a drift detector as an indicator of when a model should be retrained. Thus, we compare the effect of employing McUDI and KS in the maintenance pipeline of AIOps failure prediction models. We also report the performance of never retraining the model (Static) and the performance of retraining the model periodically (Periodic) as lower and upper baselines, respectively. We further report the number of times retraining is necessary with respect to the total number of possible retraining times averaged over the 10 random seeds.

We depict our results in Table 2. The lower baseline, Static achieves the lowest performance in terms of ROC AUC and requires 0/25 retraining times. The upper baseline, Periodic, achieves the highest ROC AUC and requires constant retraining (25/25). From Table 2 we can notice that, in case of disk data, Periodic, KS, and McUDI achieve similar results in terms of ROC AUC (around 91%), while McUDI requires the lowest number of retraining times (23.5). We can again observe that the performance of KS and Periodic is identical. When it comes to the job dataset, both McUDI and KS achieve similar performance in terms of ROC AUC to Peri-

Table 2: Performance of each retraining strategy with respect to the number of required retraining times. *ROCAUC* is the metric we use to assess the performance of the failure prediction model. *Drifts* refers to the number of identified drifts/number of total testing batches.

Model	Disk Data		Job Data	
	ROCAUC	Drifts	ROCAUC	Drifts
Static	0.90	0/25	0.83	0/14
KS	0.91	25/25	0.86	11.9/14
McUDI	0.91	23.4/25	0.86	12.5/14
Periodic	0.91	25/25	0.87	14/14

odic with a lower number of required retraining times.

#### 6.4 Label Costs Evaluation

In this experiment, we answer our third research question and show the effect of including a McUDI in the maintenance pipeline of AIOps solutions in terms of label costs. In other words, we compare the label costs when performing periodic retraining with the label costs when employing McUDI, as well as the performance in terms of average ROC AUC when employing each of the two types of retraining. We quantify label costs as the number of samples that require annotation to retrain the AIOps model.

Table 3: Results of label costs assessment. *ROCAUC* is the metric we use to assess the performance of the failure prediction model. *L Cost* refers to the number of necessary samples that require expert annotation to perform retraining.

	Disk Data		Job Data	
	ROCAUC	L Cost	ROCAUC	L Cost
McUDI	0.91	3,745,919	0.86	304,326
Periodic	0.91	4,005,978	0.87	333,982

From Table 3 we notice that the performance of McUDI is almost similar to the one obtained by periodically retraining models for both disk and job data. The advantage of employing McUDI is that it reduces the number of necessary retraining times as well as the required labels to perform the retraining. Thus, by employing our proposed maintenance pipeline, the number of samples that need annotation is reduced by approximately 260k and 30k samples for disk and job data respectively.

### 7 Discussions

Throughout our experiments, we showcase that one commonly used unsupervised technique to monitor concept drift, namely the number of features that change, is not a good performance degradation indicator for failure prediction AIOps models. However, monitoring the distribution manages to accurately capture the moments when concept drift affects the accuracy of the AIOps solution. Thus, we encourage the AIOps community to constantly monitor changes in the data distribution over time as a model quality indicator.

While experimenting with both McUDI and KS, we noticed the benefits of our solution on the disk dataset. McUDI

raises significantly less false alarms when identifying drifts when predicting disk failure compared to KS. This can be explained by the fact that the number of features that the classifier considers important is almost half of the total features employed. Thus, when computing the data distribution using all features, the KS statistical test might detect drift when non-important features are changing, but they don't impact the performance of the model. This makes a model centric approach such as McUDI an accurate indicator of model degradation with strong evidence on disk failure AIOps solutions.

Besides the constant retraining and model redeployment associated with current failure prediction AIOps model retraining practices (periodic retraining), there is also a high number of high-quality labels that need to be provided by domain experts periodically. In our work, we provide a model maintenance pipeline, which employs McUDI as a technique to verify when the model requires retraining. By employing our maintenance pipeline instead of periodical retraining, AIOps researchers and practitioners can lower both the times the model needs to be redeployed and the number of high-quality labels provided by domain experts, which saves annotation costs. Therefore, we recommend employing this model maintenance pipeline for AIOps applications integrated into larger systems for which redeployment is expensive or for AIOps applications for which obtaining the labels comes with a substantial demand from experts for manual data annotation

### 8 Conclusions

In this paper, we propose McUDI a novel model-centric unsupervised performance degradation indicator for classification problems that can identify when failure prediction AIOps models require retraining. McUDI is unsupervised since it does not require labels to identify changes in data that lead to model performance degradation. Furthermore, McUDI is model-centric since it checks for data changes solely among the most important features indicated by the trained model.

We evaluate the performance of McUDI on two open-source operational datasets used for building failure prediction AIOps models. We empirically prove that McUDI can be beneficial when included in the maintenance pipeline of AIOps solutions since it lowers the number of times the models need to be retrained and redeployed while preserving their performance compared to the current state-of-the-art practice of periodically retraining the model. We demonstrate that McUDI surpasses KS, a commonly used unsupervised technique to detect data changes, in terms of raising false alarms. Moreover, we propose a maintenance pipeline for failure prediction AIOps solutions that lowers both the number of retraining times and the cost of getting high-quality labels.

In the future, we aim to evaluate McUDI on new operational private datasets and estimate the saved costs of including McUDI in the maintenance pipeline of existing AIOps solutions. Furthermore, McUDI will be assessed with other tree-based classifiers for disk and job failure prediction. Moreover, although solely assessed on operational data in this work, McUDI should be applicable for other types of datasets and, thus, future work can consider evaluating McUDI on datasets belonging to other domains than AIOps.

## References

- [Amershi *et al.*, 2019] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019.
- [Arpteg *et al.*, 2018] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch. Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59, 2018.
- [Bangash *et al.*, 2020] 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.
- [Bayram *et al.*, 2022] 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.
- [Botezatu *et al.*, 2016] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 39–48, New York, NY, USA, 2016. Association for Computing Machinery.
- [Breck *et al.*, 2017] 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. *2017 IEEE International Conference on Big Data (Big Data)*, pages 1123–1132, 2017.
- [Breiman *et al.*, 1984] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.
- [Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.
- [Dang *et al.*, 2019] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: Real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5, 2019.
- [dos Reis *et al.*, 2016] Denis Moreira dos Reis, Peter Flach, Stan Matwin, and Gustavo Batista. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 1545–1554, New York, NY, USA, 2016. Association for Computing Machinery.
- [El-Sayed *et al.*, 2017] 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 *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344, 2017.
- [Gama *et al.*, 2014] João Gama, Indrundefined Žliobaituundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), mar 2014.
- [Haakman *et al.*, 2021] 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.
- [Huynh-Thu *et al.*, 2010] Ván Anh Huynh-Thu, Alexandre Irthum, Louis Wehenkel, and Pierre Geurts. Inferring regulatory networks from expression data using tree-based methods. *Plos ONE*, 5(9):e12776, sept 2010.
- [Inc., ] Backblaze Inc. Backblaze hard drive stats. backblaze b2 cloud storage. <https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data>.
- [Li *et al.*, 2019] 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., Red Hook, NY, USA, 2019.
- [Li *et al.*, 2020] Yangguang Li, Zhen Ming (Jack) Jiang, Heng Li, Ahmed E. Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. Predicting node failures in an ultra-large-scale cloud computing platform: An aiops solution. *ACM Trans. Softw. Eng. Methodol.*, 29(2), apr 2020.
- [Lu *et al.*, 2019] 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.
- [Lyu *et al.*, 2021a] 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.
- [Lyu *et al.*, 2021b] 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), nov 2021.
- [Lyu *et al.*, 2023] Yingzhe Lyu, Heng Li, Zhen Ming Jiang, and Ahmed E. Hassan. Assessing the maturity of model maintenance techniques for aiops solutions. *arXiv preprint arXiv:2311.03213*, 2023.
- [Mahdisoltani *et al.*, 2017] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 391–402, Santa Clara, CA, July 2017. USENIX Association.
- [Poenaru-Olaru *et al.*, 2022] L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeyer. Are concept drift detectors reliable alarming systems? - a comparative study. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3364–3373, 2022.
- [Polyzotis *et al.*, 2017] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning.

In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1723–1726, New York, NY, USA, 2017. Association for Computing Machinery.

[Rajbahadur *et al.*, 2022] 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.

[Reiss *et al.*, 2011] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20. Posted at <http://code.google.com/p/googleclusterdata/wiki/TraceVersion2>.

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

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

[Sandri and Zuccolotto, 2008] 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.

[Schelter *et al.*, 2018] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, and Felix Biessmann. Automating large-scale data quality verification. In *VLDB 2018*, 2018.

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

[Sculley *et al.*, 2015] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2503–2511, 2015.

[Vela *et al.*, 2022] Daniel Vela, Andrew Sharp, Richard Zhang, Trang Nguyen, An Hoang, and Oleg Pianykh. Temporal quality degradation in ai models. *Scientific Reports*, 12, 07 2022.

[Zhou and Hooker, 2019] 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.