An Empirical Study of the Impact of Data Splitting Decisions on the Performance of AIOps Solutions

YINGZHE LYU, Queen's University, Canada
HENG LI, Polytechnique Montreal, Canada
MOHAMMED SAYAGH, ETS - Quebec University, Canada
ZHEN MING (JACK) JIANG, York University, Canada
AHMED E. HASSAN, Queen's University, Canada

AIOps (Artificial Intelligence for IT Operations) leverages machine learning models to help practitioners handle the massive data produced during the operations of large-scale systems. However, due to the nature of the operation data, AIOps modeling faces several data splitting-related challenges, such as imbalanced data, data leakage, and concept drift. In this work, we study the data leakage and concept drift challenges in the context of AIOps and evaluate the impact of different modeling decisions on such challenges. Specifically, we perform a case study on two commonly studied AIOps applications: (1) predicting job failures based on trace data from a large-scale cluster environment, and (2) predicting disk failures based on disk monitoring data from a large-scale cloud storage environment. First, we observe that the data leakage issue exists in AIOps solutions. Using a time-based splitting of training and validation datasets can significantly reduce such data leakage, making it more appropriate than using a random splitting in the AIOps context. Second, we show that AIOps solutions suffer from concept drift. Periodically updating AIOps models can help mitigate the impact of such concept drift, while the performance benefit and the modeling cost of increasing the update frequency depend largely on the application data and the used models. Our findings encourage future studies and practices on developing AIOps solutions to pay attention to their data-splitting decisions to handle the data leakage and concept drift challenges.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **Maintaining software**; **Operational analysis**;

Additional Key Words and Phrases: AIOps, machine learning engineering, failure prediction, data leakage, concept drift, model maintenance

ACM Reference Format:
Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. 2021. An Empirical Study of the Impact of Data Splitting Decisions on the Performance of AIOps Solutions. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2021), 37 pages. https://doi.org/10.1145/3447876

Authors' addresses: Yingzhe Lyu, ylyu@cs.queensu.ca, Queen's University, Software Analysis and Intelligence Lab (SAIL), Kingston, ON, Canada; Heng Li, heng.li@polymtl.ca, Polytechnique Montreal, Department of Computer Engineering and Software Engineering, Montreal, QC, Canada; Mohammed Sayagh, mohammed.sayagh@etsmtl.ca, ETS - Quebec University, Department of Software Engineering and IT, Montreal, QC, Canada; Zhen Ming (Jack) Jiang, zmjiang@cse.yorku.ca, York University, Department of Electrical Engineering & Computer Science, Toronto, ON, Canada; Ahmed E. Hassan, ahmed@cs.queensu.ca, Queen's University, Software Analysis and Intelligence Lab (SAIL), Kingston, ON, Canada.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

1 INTRODUCTION

Large-scale software systems like Amazon Web Service are proliferating in both size and complexity. Meanwhile, operations of these large-scale systems are generating large volume of monitoring data, such as alerting signals [16], events [24], logs [31], and resource usage monitoring data [91]. It has become increasingly challenging for practitioners to collect, manage, analyze, and leverage such big operation data. Therefore, AIOps [17, 67] has been proposed to help practitioners address the challenges in the operations of large-scale software systems. AIOps solutions leverage machine learning (ML) techniques and operation data to support various goals in software and system operations, such as node failure predictions [49, 51], job failure predictions [24, 70], disk failure predictions [5, 24, 89], and service outage predictions [16]. Many of the prior works are already adopted and used in practice. For example, Lin et al. [51] successfully applied their model on a large-scale cloud service system in Microsoft to predict failure-proneness of nodes, and Xu et al. [89] deployed their disk error forecasting approach to select healthier disks for VM allocation and live migration.

Despite the breakthroughs in ML models and their applications in AIOps, there are still challenges preventing the integration of such ML models into software systems [41], such as the challenges in model evaluation and model evolution [3, 73]. One of the main reasons is that ML experts usually focus on tuning the ML model performance instead of maintaining model behavior after deploying in the field [41]. Hence, software engineering for machine learning has become an emerging topic that aims to manage the lifecycle of machine learning models (i.e., training, testing, deploying, evolving, etc.) [3, 41, 63]. Within the lifecycle of ML models, making appropriate decisions for data splitting (e.g., splitting data into training and validation sets) is particularly challenging, even for ML experts [38, 63]. For example, ML experts highlight the importance of data splitting in ML modeling [63] and advocate the introduction of engineering processes for data splitting [38]. In particular, in the context of AIOps, ML modeling faces three data splitting (DS)-related challenges during the process of developing AIOps solutions, as shown in Figure 1.

- **Imbalanced data:** Operation data is often very imbalanced [5, 24, 49, 51, 57], which challenges AIOps modeling as the models tend to make a more accurate prediction on the majority class while performing poorly on the minority class [44, 80, 89]. Such a challenge requires the application of data rebalancing techniques (e.g., over-sampling, under-sampling, SMOTE [13], ROSE [54])) to make the modeled classes more balanced (i.e., splitting the data of different classes to achieve a better balance between the classes) [44, 80] or using cost-sensitive models [1, 15, 35].
- **Data leakage:** Prior studies (e.g., [5, 24, 57]) in AIOps randomly split operation data into training and validation data. However, such a splitting strategy may risk data leakage, i.e., leak information in the validation data that should not be available for model training into the training data, which may introduce bias and result in misleading evaluation results [39, 40, 65, 72]. For example, in a recent Kaggle competition [74], the leakage of the future information into the training features cause the model to make unrealistic good predictions that could not reflect the actual model performance in a practical setting.
- **Concept drift:** Over time, the distribution of the operation data and the relationship between the variables in the data may be constantly evolving [17, 49, 51] (a.k.a., concept drift [64, 84–86]). Concept drift may lead to `obsolescence` of the models trained on historical data, i.e., a model trained on outdated data may perform poorly on new data.

In this work, we study the data leakage and concept drift challenges in the context of AIOps and evaluate the effectiveness of different techniques in addressing such challenges. The challenge of imbalanced data has been extensively

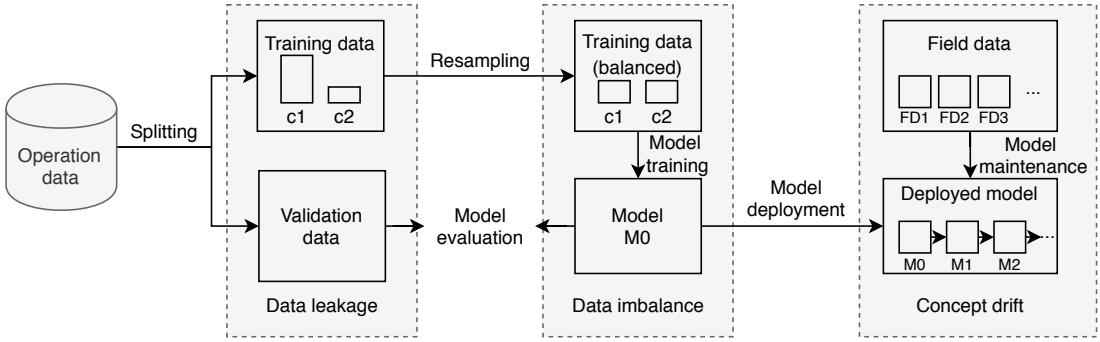


Fig. 1. A general AIOps modeling process and the associated data splitting challenges.

discussed in prior work [80] in the context of defect prediction, and the main findings are also applicable in the AIOps context [51]. In particular, we perform a case study on two public operation datasets: the Google cluster trace dataset [88] and the Backblaze disk stats dataset [34], which are used in many prior AIOps studies [5, 24, 57, 70, 71]. Based on the two datasets, we systematically evaluate the impact of data splitting decisions on prior AIOps solutions for predicting job failures [24] and disk failures [5, 57]. We replicated the previous works' approaches but varying the choices in different data splitting contexts (e.g., during the training or maintenance phases). We organize our work by answering the following four research questions (RQs):

RQ1: *Can existing AIOps solutions suffer from data leakage issues?*
Data leakage is the introduction of information in the training data that should not be available for model training and can lead to the bias of model evaluation [39, 40, 65, 72]. In this RQ, we investigate the existence of data leakage issues in the context of AIOps.

RQ2: *What is the impact of different strategies for splitting data into training and validation datasets on mitigating the risk of data leakage?*
Prior work (e.g., [5, 24, 57]) randomly splits operation data into training and validation data, which may risk data leakage and lead to bias in model evaluation. In this RQ, we thoroughly evaluate the impact of two approaches for splitting training and validation data (i.e., random splitting vs. time-based splitting) on the validity of the model evaluation.

RQ3: *Are there concept drift issues facing AIOps solutions?*
Over time, the change in the distribution of the data and the relationship between the variables (i.e., concept drift [64, 84–86]) can negatively impact the performance of AIOps models. In this RQ, we investigate the existence of concept drift issues in the context of AIOps.

RQ4: *How should we maintain and update models that are deployed in practice to mitigate the risk of concept drift?*
Concept drift may lead to *obsolescence* of AIOps models. In this RQ, we evaluate the impact of periodically updating the model on mitigating the concept drift issue. We also investigate the impact of different model updating frequency on the performance and cost of AIOps models.

There are prior works that handle the data leakage and concept drift problems. However, the proposed solutions for these two problems have never been examined in the context of AIOps. As the operational environment is constantly changing [46, 49, 51], operation data may be more likely to subject to the data leakage and concept drift issues. In

addition, the operation data is usually a mixture of temporal (e.g., log data) and spatial data (e.g., hardware configurations or software versions) [49, 51], among which the data types are generally heterogeneous (e.g., numeric, ordinal, and nominal values [24, 49, 51, 88]). The complexity of data types and distributions makes AIOps an interesting area to validate these proposed solutions. Therefore, we conduct our study to check: (1) whether the problems can exist in the AIOps context; (2) if they can, how much impact these problems would have on the AIOps applications; and (3) how existing techniques could mitigate the corresponding issues in the AIOps context. Similar to various AI4SE-related problems (e.g., bug prediction [29, 37], change recommendation [62, 68]), we are not re-inventing ML/AI techniques. Instead, we want to see whether existing ML/AI techniques can work in the AIOps context and which techniques work best in the studied problem context.

In addition, we notice that recent AIOps solutions are still using techniques that can suffer from the data leakage and concept drift problems (see Section 2.2 below for details). For example, recent studies are still randomly splitting the data to evaluate their models [5, 24, 57, 66] without considering the data leakage issue. Furthermore, even AI experts argue that a comprehensive evaluation of the different data splitting is an area to be explored from an SE perspective [38, 63]. Therefore, we reiterate over AIOps solutions to provide scientific evidence that AIOps solutions suffer from data leakage and concept drift challenges, and whether using alternative approaches (e.g., time-based splitting) could improve existing practices in the AIOps context.

The contributions of this paper are:

(1) This is the first work that assesses the performance impact of various data splitting decisions on AIOps solutions. The findings and the proposed techniques in this paper can be useful to machine learning engineers or software engineering researchers interested in improving the quality and maintainability of AIOps solutions.
(2) Our results show that problems like data leakage (caused by decisions during model training and evaluation) and concept drift (caused by the evolution of data) can easily appear in AIOps solutions if not being careful while deciding on various data splitting strategies. Such problems may severely impact the performance of AIOps solutions while being deployed in the field.
(3) To mitigate the risks of the various problems arising from data splitting decisions, we also proposed suggested techniques and demonstrated their effectiveness in our case studies. In particular, we observe that using a time-based splitting of training and validation datasets can reduce data leakage and provide a more reliable evaluation. We also observe that periodically updating AIOps models can help mitigate the impact of concept drift, while the frequency of model updating should be cautiously considered.

Paper organization. The rest of the paper is organized as follows: Section 2 provides background information and discusses related work; Section 3 presents our experimental design, outlining our approach and our preparation steps for the case study; Sections 4 through 7 present our approaches and results for answering our RQs; Section 8 discusses the threats to the validity of our findings; finally, Section 9 concludes our paper.

2 BACKGROUND AND RELATED WORK

ML models are widely used in AIOps solutions (e.g., [5, 16, 24, 49, 51, 57, 70, 89]). In this section, we first discuss the data splitting-related challenges (i.e., data imbalance, data leakage, and concept drift) in the general area of Machine Learning, then we examine how prior AIOps solutions handle these data splitting-related challenges. Finally, we discuss prior studies in the area of software analytics that evaluate the impact of different modeling decisions.

2.1 Background on data splitting-related challenges

The Challenge of Imbalanced Data: Imbalanced data is a common problem in the machine learning community. It arises when one of the classes is severely underrepresented in the dataset. [32]. Imbalanced data could cause models to focus on the majority class and ignore the rare events, which heavily compromises the process of learning [44].

The machine learning community usually addresses the issue in two ways. One way is to apply data rebalancing techniques, most simply, oversampling the minority class or under-sampling the majority class. There are also approaches that blend the two sampling strategy like SMOTE (Synthetic Minority Over-sampling TEchnique) [13], and approaches that combine oversampling with the generation of artificial data like ROSE (Random OverSampling Examples) [59]. As a result, the modeled classes are more balanced and may produce more predictive models.

Other than resampling techniques that balance the sample classes, researchers also design ML models specially optimized for the imbalanced data issue by assigning distinct costs to the training samples. For example, Arya et al. [35] propose a cost-sensitive support vector machine algorithm that provides superior generalization performance compared to conventional SVM on imbalanced data; deep learning approaches can also tackle the imbalanced data problem with a weighted back-propagation [15] or a weighed form of categorical cross-entropy [1]. Besides, updatable classification algorithms can also be a viable approach in handling imbalanced data. Ming et al. [78] report that updatable classification algorithms, which update the training set incrementally to take advantage of the feedback from each run, improve the precision under some circumstances when handling imbalanced data.

The Challenge of Data Leakage: Data leakage is the introduction of information in the training data that should not be available for model training and can lead to the bias of model evaluation [39, 40, 65, 72]. The creation of such unexpected additional information in the training data would enable the models to use the future data to predict the past data [49, 51, 89], and therefore causing it to make unrealistically good predictions that could not reflect the practical performance. Leakage is a pervasive challenge in applied machine learning, causing models to over-represent their generalization error and often rendering them useless in the real world. For example, leakage of the future information into the training features are reported in many Kaggle competitions, including a recent one in a prostate cancer dataset [74]. Prior works [39, 72] suggest that when there are risks of such data leakage, time-based splitting of training and validating data splitting (i.e., splitting the data based on their time sequence) should be used over a random-based splitting strategy.

In this work, we study the existence of data leakage in the context of AIOps, which has not been explored before. We also evaluate the impact of different splitting strategies (e.g., time-based splitting) on data leakage.

The Challenge of Concept Drift: In machine learning and data mining, the distribution of the data and the relationship between the variables may evolve over time, which is known as concept drift [64, 84–86]. Concept drift may lead to obsolescence of models trained on previous data and negatively impact the performance. In order to mitigate the impact of concept drift, prior works propose approaches for detecting concept drift [26, 30, 64, 87] and handling concept drift [9, 12, 21, 28, 32, 60, 61, 77, 85]. For example, Nishida et al. [64] propose a concept drift detection method using statistical testing. It assumes that the prediction accuracy on the data from a recent time window would be equal to the overall accuracy if the target concept is stationary, and a significant decrease in the recent accuracy suggests a concept drift. When there is concept drift, aside from retraining a model from scratch, online learning updates the current model using the most recent data incrementally. Such model process input examples one-by-one and update the model after receiving each example [28]. For example, CVFDT [33] is a decision tree model that incrementally updates itself when new data becomes available and can adapt to the drifting concept.

Time-based ensembles combine individual base models trained on data from small time periods. The intuition is that the base models trained from such small time periods can better capture the relationship between the variables, as the concept drift in a smaller period is relatively small. For example, Steet and Kim propose the Streaming Ensemble Algorithm (SEA) [77], which is a majority-voting ensemble approach that constantly replaces the weakest classifier in the ensemble with a quality measure that considers both the accuracy and diversity of classifiers in the ensemble. Cano and Krawczyk propose the Kappa Updated Ensemble (KUE) [12], which is a combination of online and block-based ensemble approaches. KUE uses the Kappa statistic for dynamic weighing and selection of base classifiers.

Other advanced techniques in handling concept drift include an enhancement of the time-based ensemble methods by Krawczyk et al. [43] that improves the model's robustness to drift and noise by adding abstaining options to classifiers, allowing classifiers in the ensemble to refrain from making a decision if they have a confidence level below a specified threshold. Cano et al. [11] propose a rule-based classifier for drifting data streams using grammar-guided genetic programming. The model, namely evolving rule-based classifier for drifting data streams (ERulesD^{2}S), can provide accurate predictions and adapt to concept drift while offering the full interpretability based on classification rules.

In this work, we study the existence of concept drift in the AIOps datasets using the statistical testing method proposed by Nishida et al. [64]. We also evaluate the impact of periodically updating AIOps models on the model performance, as model retraining is model-agnostic and applies to general models. We focus on the impact on the general models used in existing AIOps solutions. On the other hand, the above-mentioned approaches for handling concept drift usually only apply to specific models. For example, online models only apply to a few tree-based models (e.g., CVFDT [33], Hoeffding Tree [20]), while none of them have been applied in existing AIOps solutions for the studied applications.

2.2 Prior research on AIOps that deals with data splitting-related challenges

Prior work proposed many AIOps solutions to address various problems in the operations of large-scale software and systems, such as incident prediction [5, 16, 24, 49, 51, 57, 70, 89], anomaly detection [31, 50], ticket management [90, 91], issue diagnosis [55], and self healing [18, 19, 52, 53]. For example, Lin et al. [51] and Li et al. [49] leverages temporal data (e.g., CPU and memory utilization metrics, alerts), spatial data (e.g., rack locations), and config data (e.g., memory size) to predict node failures in large-scale cloud computing platforms. El-Sayed et al. [24] and Rosa et al. [70] learned from the trace data to predict job failures in the Google cloud computing platform. Botezatu et al. [5], Mahdisoltani et al. [57], and Xu et al. [89] leveraged disk-level sensor data and system-level events to predict disk failures in operations of large-scale cloud platforms. As illustrated in Figure 1, ML modeling, in particular, supervised learning, in the context of AIOps usually faces three data splitting-related challenges: the imbalanced data challenge in model training, the data leakage challenge in model training and evaluation, and the concept drift challenge in model maintenance. Table 1 and Table 2 list prior AIOps work that leverages supervised learning and unsupervised learning techniques, respectively. For the works using supervised learning, we summarize how they handle the three challenges in different ML modeling phases. Below, we discuss prior AIOps solutions that rely extensively on supervised ML models.

Handling imbalanced data. Operation data is often very imbalanced [5, 24, 49, 51, 57]. Therefore, AIOps solutions usually apply data rebalancing techniques (e.g., over-sampling, under-sampling, SMOTE, ROSE) to make the modeled classes more balanced and produce more accurate models [44, 80]. For example, Botezatu et al. [5] and Mahdisoltani et al. [57] use under-sampling approaches (i.e., randomly reducing the samples of the majority class) to balance the samples of failed disks and normal disks in their tasks of disk failure prediction. El-Sayed et al. [24] uses an over-sampling approach (i.e., making random duplication of the minority class) to balance the samples of failed jobs and normally

terminated jobs in their tasks of job failure prediction. Xu et al. [89] and Chen et al. [16] use the SMOTE over-sampling approach [13] to balance their classes in the tasks of disk failure prediction and service outage prediction, respectively.

This work does not explore the impact of data rebalancing techniques on AIOps solutions, as data rebalancing has been extensively discussed in prior work (e.g., [49, 51, 80]). Instead, we use an under-sampling approach to balance the classes in both our studied datasets, as done in Botezatu et al. [5] and Mahdisoltani et al. [57].

Handling data leakage. Prior studies [5, 24, 57, 66] usually randomly split the dataset into a training set and a validation set. For example, El-Sayed et al. [24] randomly split the whole Google cluster trace dataset [88] into 70% training data and 30% validation data. Botezatu et al. [5] and Mahdisoltani et al. [57] randomly split the Backblaze disk stats dataset into 80% training data and 20% validation data, and 75% training data and 25% validation data, respectively. In comparison, some prior studies use a time-based approach to split training and validation data, which ensures that the training data always occurs before the validation data [49, 51, 71, 89].

In this work, we analyze the existence of data leakage in the studied operation datasets (**RQ1**). Then we evaluate the impact of using a time-based splitting (i.e., considering the temporal order in the data) instead of random splitting on model evaluation (**RQ2**).

Handling concept drift. Existing AIOps studies usually train a static model regardless of potential concept drift [5, 16, 24, 57, 71, 91] without respecting that the operation data is constantly evolving [17, 49, 51]. However, concept drift may lead to the obsolescence of such static models trained on previous data. In order to mitigate the impact of concept drift, other prior works suggest that AIOps models need to be retrained periodically to ensure that the models are not outdated [49, 51].

In this work, we first analyze the existence of concept drift in the studied operation dataset (**RQ3**), then we evaluate the impact of periodically updating a model instead of using a static model on the model performance, and how the model update frequency might impact the performance and cost of AIOps models (**RQ4**).

2.3 Prior research on the impact of modeling decisions in the area of software analytics

Prior studies explored the impact of different modeling decisions (e.g., model evaluation decisions) on the performance of ML models used in software analytics.

Data rebalancing. Prior work studies the impact of the different data rebalancing techniques on the performance of ML models for software analytics (e.g., defect prediction) [2, 80]. Agrawal et al. [2] proposed and evaluated a data re-sampling technique, called SMOTUNED (an automatically-tuned version of SMOTE), in order to tackle the imbalanced classes in the training data. They observed a significant performance improvement when using SMOTUNED. Tantithamthavorn et al. [80] studied the impact of different class rebalancing techniques (i.e., over-sampling, under-sampling, SMOTE [13], and ROSE [54]) on the performance and interpretation of defect prediction models. They observed that data rebalancing techniques result in a higher recall and a lower precision in defect prediction, while such techniques do not impact the AUC measure. Besides, they do not recommend using data rebalancing when interpreting the models.

Model training and maintenance. Prior studies evaluate the impact of different model training decisions on the performance of ML models for software analytics [29, 81, 83]. Ghotra et al. [29] observed that using different classifiers can statistically significantly impact the performance of defect prediction models. Tantithamthavorn et al. [81, 83] showed that using different hyperparameters can lead to a statistically significant difference in the performance of defect prediction models. Ekanayake et al. [22, 23] found that software systems are subject to considerable concept drift in their evolution history, which impacts the stability of defect prediction models. In this work, we investigate the

Table 1. AIOps studies that leverage supervised learning and their data splitting-related techniques used at different modeling phases.

Category	Models[1]	Data type	Handling imbalanced data	Handling concept drift	Handling data leakage	Public availability	Ref
Job failure prediction	RF	Temporal and config data	Over-sampling	Static model	Random-based splitting (70%/30%)	Public	[24]
	LDA, QDA, LR	Temporal and config data	Unknown	Static model	Time-based splitting	Public	[71]
	NN	Temporal and config data	Unknown	Retraining model	Time-based splitting	Public	[70]
Disk failure prediction	RGF, GBDT, RF, SVM, LR, DT	Temporal and config data	Under-sampling	Static model	Random-based splitting (80%/20%)	Public	[5]
	RF, CART, LR, NN, SVM	Temporal data	Under-sampling	Static model	Random-based splitting (75%/25%)	Public	[57]
	FastTree[2]	Temporal and config data	Over-sampling (SMOTE)	Retraining model	Time-based splitting	Private	[89]
Node failure prediction	MING[3]	Temporal and spatial data	Under-sampling	Retraining model	Time-based splitting	Private	[51]
	MING, RF, LSTM	Temporal, spatial, and config data	Mixture of under-sampling and over-sampling	Retraining model	Time-based splitting	Private	[49]
Service outage prediction	GBDT	Temporal data	Over-sampling (SMOTE)	Static model	Unknown	Private	[16]
Ticket management	NN	Temporal and spatial data	Clustering	Static model	Time-based splitting	Private	[91]
	NN	Temporal data	Clustering	Static model	Time-based splitting	Private	[90]

[1] Abbreviation for model names: RF (Random Forest), LDA (Linear Discriminant Analysis), QDA (Quadratic Discriminant Analysis), LR (Logistic Regression), NN (multi-layer Neural Network), RGF (Regularized Greedy Forests), GBDT (Gradient Boosting Decision Tree), SVM (Support Vector Machine), DT (Decision Tree), CART (Classification And Regression Trees), and LSTM (Long Short Term Memory networks).

[2] FastTree is a form of "Multiple Additive Regression Trees" (MART) gradient boosting algorithm.

[3] MING is a hybrid approach which combines LSTM, random forest, and a learning to rank model.

Table 2. AIOps studies that leverage unsupervised learning and their used techniques.

Category	Model[1]	Data type	Public availability	Ref.
Identifying incident indicators	CARs	Temporal data	Private	[52, 53]
Mining execution patterns	FCA	Temporal data	Private	[52, 53]
Identifying performance issues	HMRF	Temporal and config data	Private	[50, 53]
Events correlation	Nearest neighbor	Temporal data	Private	[55]
Identifying service problems	Cascade clustering	Temporal and config data	Private	[31]
Healing action recommendation	FCA	Temporal data	Private	[18, 19]

[1] Abbreviation for model names: FCA (Formal Concept Analysis), CARs (Class Association Rules), and HMRF (Hidden Markov Random Field).

concept drift problem in the AIOps context and study the impact of periodically updating AIOps models on handling such concept drift.

Model evaluation. Prior work also evaluates the impact of different modeling decisions on the evaluation of machine learning models [79, 80, 82]. Tantithamthavorn et al. [79, 82] evaluate different model evaluation approaches in defect predictions. They argued that k-fold cross-validation, which is commonly used in software defect predictions, can lead to significant bias in the model evaluation. Tantithamthavorn et al. [80] also evaluated the impact of data rebalancing approaches on model evaluation. They found that data rebalancing approaches impact the threshold-sensitive evaluation metrics (e.g., precision, recall, and F1 metrics) while such data rebalancing approaches do not impact the threshold in-sensitive metrics (e.g., AUC). Therefore, they claimed that threshold in-sensitive metrics are more appropriate when comparing the performance of different models. In this work, we evaluate the impact of different model evaluation approaches (i.e., using a random-based splitting or time-based splitting) on the evaluation of AIOps models.

3 CASE STUDY SETUP

We perform a case study on two large-scale public operation datasets to understand the challenges of data leakage and concept drift in the context of AIOps and evaluate data splitting strategies in handling such challenges. We first present our studied datasets in Section 3.1, then describe our experimental design, including how we preprocess the operation data, construct and evaluate the AIOps models, in Section 3.2.

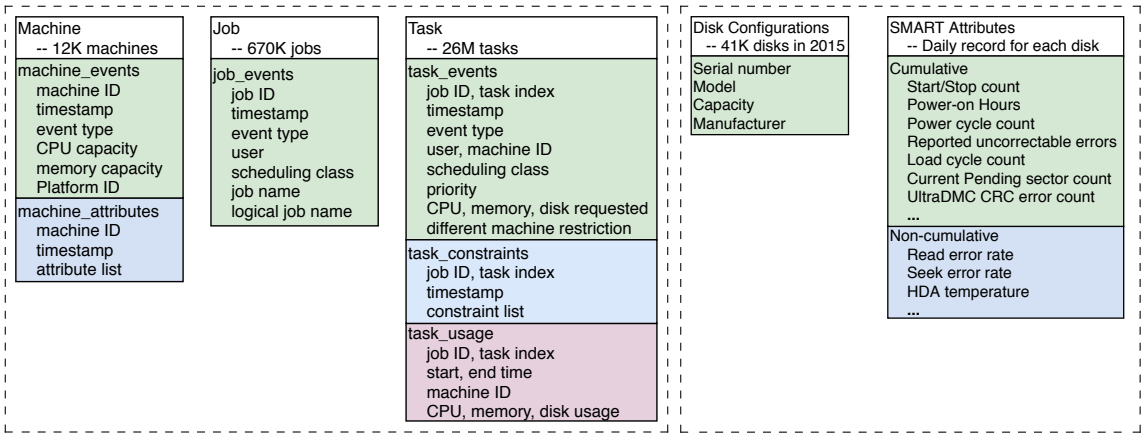
3.1 Studied Operation Datasets

In this work, we perform our case studies on two operation datasets: the Google cluster trace dataset [88] and the Backblaze disk stats dataset [34]. For the Google cluster trace dataset, we build machine learning models to predict job outcomes (i.e., failure or not). For the Backblaze disk stats dataset, we build models to predict disk failures. We focus on these datasets in particular since they are widely studied by prior work as a case study of operation datasets [5, 24, 57, 71, 89] and they are publicly available (see Table 1). Note that we do not consider the other AIOps tasks that are shown in Table 1, such as node failure prediction, since no public dataset is available for these tasks. We also do not consider the AIOps tasks that rely on unsupervised learning techniques shown in Table 2 due to the difficulty of obtaining oracles or domain experts to evaluate the results of such tasks.

3.1.1 Google Cluster Trace Dataset. The cluster data released by Google contains the trace information on a production cluster of about 12K machines in a period of 29 days for 670K jobs and 26M tasks [14]. The data features workload

arrives at a cell (i.e., a set of machines that share a common cluster-management system) in the form of jobs, and each job comprises one or more tasks, each of which is scheduled on a single machine. For confidentiality reasons, the data is anonymized in several ways, i.e., many fields are randomly hashed, all timestamps are relative to an unknown time point, resource sizes have been linearly transformed with an unknown function, and the semantic information about jobs is obfuscated. Figure 2a shows the dataset schema and information provided in the Google cluster trace dataset. The categories of metrics that we extract from the Google dataset and use in our study are further described in Table 3.

In this paper, we replicate the prior work [24] which learns from the Google cluster data to predict job failures (i.e., terminated for any reason before successfully completed) based on the information from job submission and the monitoring data during the first five minutes of the job execution. The Google cluster data records an event flag (e.g., FINISH, KILL, or FAIL) when the job status changes. We consider a job fails if its final status is FAIL, same as prior work [24]. We then formulate the prediction of job failures on the Google data as a binary classification problem. Further details on how we build and evaluate different classification models are provided in Section 3.2.



(a) Google data schema. (b) Backblaze data schema.

Fig. 2. Data schema for our studied datasets. Each colored box represents a data table: the first line is the table name, and the following lines describe the data fields. For the Google dataset, each table (e.g., machine_events) is one or multiple CSV files containing the fields described in the box. For the Backblaze dataset, the tables are organized in a logical view while the physical data is stored as daily snapshots of each disk's attributes.

3.1.2 Backblaze Disk Stats Dataset. The Backblaze dataset describes the statistics of the hard drives in the Backblaze data center and is publicly available [34]. The dataset contains daily snapshots of each operational hard drive in the data center, including basic drive information (e.g., hard drive model, disk capacity) and SMART (Self-Monitoring, Analysis and Reporting Technology) statistics, where SMART is a system for monitoring and early detection of errors implemented by manufacturers on HDDs or SSDs. Figure 2b shows the dataset schema and information provided in the Backblaze disk stats dataset. Note that the original data tables are organized as daily snapshots of the operational disks, and we split the fields of those tables logically into configurations and SMART attributes, which are further described in Table 3.

In this paper, we also replicated the prior work [5, 57] which learns the Backblaze dataset to predict hard drive failures (i.e., sector error) within a given time period (i.e., one week) based on the monitoring data captured during

a period of time (i.e., one week) in the past. We consider a disk fails if it has an increase in the "sector error count" SMART attribute (i.e., observe a sector error) in the given time period, same as prior work [57]. We then formulate the prediction of disk failures on the Backblaze data as a binary classification problem. The details on how we build and evaluate different classification models are provided in Section 3.2.

3.2 Experimental Design

3.2.1 Preprocess Operation Data. In this work, we focus on the prediction of Google job failures and the prediction of Backblaze disk failures. We describe below how we preprocess the data for these two tasks.

Table 3. The types of metrics that are considered for predicting job failures in the Google dataset and disk failures in the Backblaze dataset.

Dataset	Type	Number of metrics we used	Description
Google May 2011 (29 days) 670K jobs (67 GB)	Temporal	6	Metric values that change during the execution of a job, such as the mean and standard variation of CPU, memory, and disk space usage by a job's task over the first 5 minutes since job submission.
	Config	9	Configuration values that are available at the time when a job is submitted such as the requested CPU, memory and disk space, which are determined during the creation of a job.
Backblaze 2015 (12 months) 18M records (4.5 GB)	Temporal	19	Features that change during the usage of an HDD Disk. Two types of SMART attributes exist for Backblaze dataset: attributes whose values are accumulated over time, such as the "reallocated sectors count" and noncumulative attributes from prior time periods such as the "read error rate". We extract both the change and the value of the last day of a one-week window as features for accumulative attributes and only the value of the last day for noncumulative attributes.
	Config	0	Features that are related to the characteristics of a disk, such as the disk model, manufacture, and disk volume. Note that we do not use any config related feature to closely replicate prior work [57].

Google Job Failure Prediction: Similarly to El-Sayed et al. [24], we predict the job failures based on the config and temporal features described in Table 3. In the Google cluster trace data, each job has several events, which is associated with a transit (e.g., submit, schedule, evict, fail, kill, finish, lost, update) among the states (e.g., unsubmitted, pending, running, dead) in the job's life cycle. Our target variable (i.e., job failures) is not directly available on the dataset, and we label a job as a failing job if its last record (last event) is a fail transit to obtain our target variable. Note that we remove the jobs that are not completed or whose records are lost during execution, as these jobs' final states are unknown. We further remove the job samples that start from the last day (i.e., the 29th day), as these jobs are more likely to fail than complete before the data cutoff time (the completed jobs last longer than the failed ones), which may cause data collection bias on the distribution of failed and completed jobs. In fact, we observed a much higher job failure rate from

the jobs starting on the last day. In the end, we successfully extracted 627K (out of a total of 670K) job samples from the first 28 days' trace data.

Backblaze Disk Failure Prediction: Although Backblaze provides HDD hard drive stats from 2013 to 2019, we decided to focus on the 2015 snapshot since it is already a large dataset (over 18M lines of records). Also, newer snapshots have a different set of smart attributes, while older data (from 2013 and 2014) does not have the necessary smart attributes. We choose the same features as in prior work [57] that are described in Table 3. Similar to prior work [57], we use features in one week to predict whether there would be disk failures in the next week. For noncumulative attributes (i.e., the attributes that are not accumulated from previous time periods, like the "read error rate" attribute), we use their value on the last day of the training period as explanatory variables. For cumulative attributes (i.e., the attributes accumulated from previous time periods, like the "reallocated sectors count" attribute), we calculate the amount of increase in the one-week training period and their value on the last day of the training period. We obtained over 7M samples in the data of 2015. Each of the samples has 19 features.

3.2.2 Build Predictive Models. To ensure the generalizability of our experiment results, we choose a variety of models used in prior works [5, 24, 57] to predict disk failures on the Backblaze disk stats dataset and job failures on the Google cluster trace dataset. The list of models we select includes Random Forest (RF), Support Vector Machine (SVM), Classification and Regression Trees (CART), Regularized Greedy Forests (RGF), and Neural Network (NN). We select these models as they are the best-performing models in the prior studies using the same datasets [5, 24, 57]. We do not choose other models (e.g., online learning models such as CVFDT [33]) as we focus on the general approaches (e.g., model retraining) that can apply to the models used in existing AIOps solutions.

The decision-tree based ensemble model RF is among the best-performing models for predicting job failures on the Google cluster trace dataset [24] and predicting disk failures on the Backblaze disk stats dataset [57] in prior works. RGF [36] is a tree-based ensemble algorithm that employs a fully-corrective greedy algorithm to update the weights of all the leaf nodes iteratively. Unlike boosting algorithms (e.g., DBDT) that treat regression tree learners as black boxes, RGF introduces an explicit regularization term to utilize the underlying tree structures. Prior work shows that the RGF model achieves higher accuracy and smaller models than GBDT on several datasets [36]. The RGF model is also among the best-performing models for disk failure prediction in prior work [5].

We configure the models the same way as prior studies. For example, we use the configuration of 50 trees for the RF model similar to El-Sayed et al. [24], and we set 100 nodes in the hidden layer for the NN model similar to Mahdisoltani et al. [57]. We manually select the hyperparameter settings for other models (i.e., CART, RGF, and SVM) that do not have explicitly reported configurations in the prior studies. We choose the implementation in the `scikit-learn`[1] Python package for all our models expect RGF, for which we choose an implementation from the `rgf-python`[2] package. Due to the skewness of our datasets (only 1.5% of the jobs on the Google cluster trace dataset are related to job failures, and only 0.07% of Backblaze samples are related to disk failures), we apply under-sampling on the training set for all of our five models for a ratio of 1:10, as done in Mahdisoltani et al. [57].

3.2.3 Evaluate Model Performance. We evaluate the performance of our models using F-1 measure (a.k.a., F1 score), Area Under Curve (AUC), and Matthews correlation coefficient (MCC), which are standard and universally used machine learning metrics. F1 score is a harmonic mean of precision and recall (we use the default threshold of 0.5). AUC measures the area under the true positive rate (TPR) false positive rate (FPR) curve and is recommended in the prior study than

[1]https://scikit-learn.org/stable/
[2]https://pypi.org/project/rgf-python/

threshold-dependent measures (i.e., precision and recall) when measuring the model performance [79]. The MCC considers true and false positives and negatives and is generally regarded as a balanced measure that can be used even if the classes are of different sizes [6], which is the case of often highly skewed operation data. Table 4 shows the evaluation metrics and their descriptions.

Table 4. Evaluation Metrics and Their Descriptions.

Metric	Description
AUC	Area under the ROC curve (TPR vs. FPR at different classification thresholds)
F1 score	$2 \times \frac{P \times R}{P+R}$, where P and R stands for the precision (i.e., $\frac{TP}{TP+FP}$) and recall (i.e., $\frac{TP}{TP+FN}$), respectively.
MCC	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

4 RQ1: CAN EXISTING AIOPS SOLUTIONS SUFFER FROM DATA LEAKAGE ISSUES?

4.1 Motivation

Data leakage is the introduction of information in the training data that should not be available for model training and can lead to the bias of model evaluation [39, 40, 65, 72]. The goal of this research question is to understand whether data leakage exists in AIOps problems. In particular, we investigate whether randomly splitting operation data for training and validating a model, which is widely used in existing AIOps research [5, 24, 57], can lead to the data leakage problem.

4.2 Approach

Randomly splitting the data into a training set and a validation set may risk data leakage in the context of AIOps (i.e., when the data samples have a temporal order), as future data (e.g., traces of 2019) may be used to train the model that predict the past (e.g., traces of 2018) [39, 40]. To investigate whether data leakage issue exists in AIOps, we compare the evaluation results of a model using a random splitting strategy with a baseline splitting strategy that predicts each sample with all the available training data in the past. Intuitively, any valid model derived from our current evaluation strategy should not achieve a better result than this baseline splitting strategy.

On the other hand, the time-based splitting strategy splits the data into the training set and the validation set based on their temporal order, as shown in Figure 3. Kaufman et al. [39, 40] suggested that time-based splitting should be used when there is a risk of data leakage. Therefore, we also compare the evaluation results of models using the time-based splitting with the models using the random splitting and the baseline splitting strategy.

The different model evaluation scenarios are illustrated in Figure 3 and detailed below.

- **Baseline splitting**: We use a baseline splitting strategy that trains a model using all the past data before predicting each data sample, which intuitively should yield better performance than other valid splitting strategies. Prior work [48] uses a similar approach to evaluate the performance of predicting log changes. Specifically, we first randomly choose N samples as the testing data. For each testing sample, we build a model with all available samples (i.e., samples that have finished before the testing sample) and test the model on the current testing sample. We then combine the prediction results of all the testing samples to calculate the performance. For the Google cluster trace data, we set N as 15,000; for the Backblaze disk stats data, we set N as 150,000. We choose these values to ensure that we have enough samples from the minority classes.

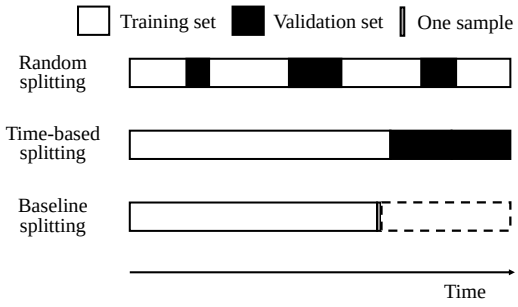


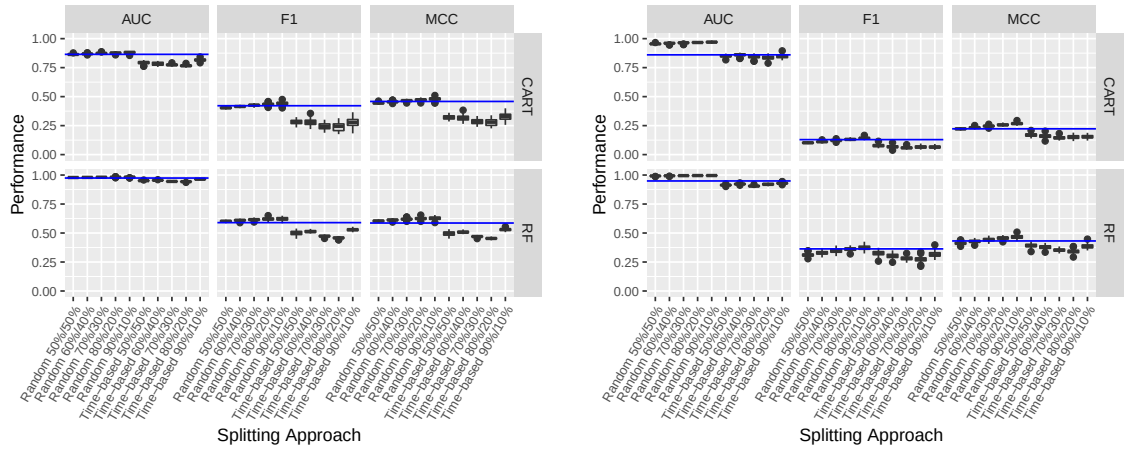
Fig. 3. Random splitting vs. time-based splitting vs. baseline splitting strategy.

- **Random splitting**: In this scenario, we first randomly split the data into a training set and a validation set, then we train a model using the training set and evaluate the model on the validation set. We consider five training/validation split ratios that range from 50%/50% to 90%/10%.
- **Time-based splitting**: In this scenario, we split the data into the training set and testing set based on the temporal order, then we train a model using the training set and evaluate the model on the validation set. We also consider the five training/validation split ratios that range from 50%/50% to 90%/10%.

Our experiment considers the prediction of job failures on the Google cluster trace dataset and disk failures on the Backblaze disk stats dataset. In order to avoid bias caused by random noise, we repeat the process ten times. As our purpose is to demonstrate the existence of data leakage in the AIOps context, we only consider the Random Forest (RF) and decision tree (CART) models. As we need to train up to 150,000 times (for the Backblaze dataset) for each model using the baseline splitting strategy, it will take a very long time (e.g., weeks to months) to train the other three models for such a large number of times. Therefore, we only consider RF and CART which can be trained faster than other models. However, we expect that the results for other models will be similar.

4.3 Results

Data leakage could exist in AIOps solutions that use a random splitting of training and validation datasets, as random splitting achieves a higher model performance than the baseline splitting strategy that leverages all the available past data. We observe that overall, models that are trained and evaluated on a random splitting have a higher performance than the baseline splitting, which indicates that the random splitting could lead to over-estimation of model performance than the baseline splitting that uses all the available past data to train a model for each data sample, as shown in Figure 4. Nevertheless, the baseline splitting strategy is usually not practical in the context of AIOps, as the AIOps datasets are usually very large, and it takes a very long time (e.g., weeks to months) to train a model for each data sample. On the Google dataset, we observe that on the RF model, all five random splitting strategies have higher performance than the baseline splitting for all three performance metrics (i.e., AUC, F1 score, and MCC). Similarly, three of the random splitting strategies (with training/validation splitting ratio 70%/30%, 80%/20%, and 90%/10%) have a higher performance than the baseline splitting for all three performance metrics on the CART model. On the Backblaze data, we observe that for both the RF and CART model, the random splitting strategies always achieve a higher AUC than the baseline splitting, while the F1 score and MCC are higher using the random splitting with splitting ratios that have a larger proportion of training data (e.g., training/validation splitting ratio 90%/10%).



(a) Google (b) Backblaze

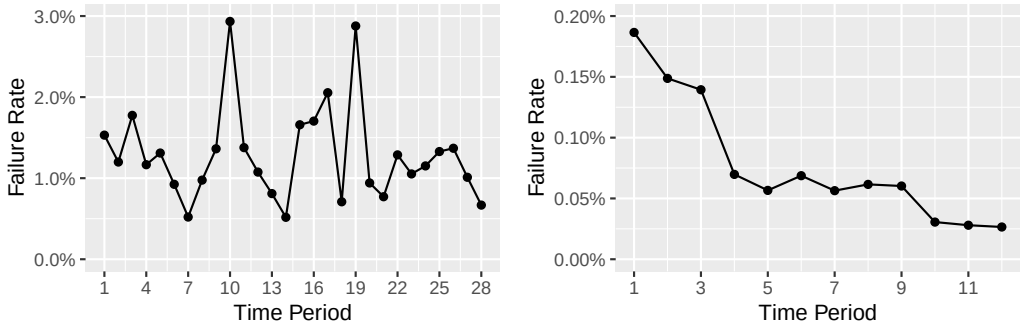
Fig. 4. Comparing the model performance when using random splitting strategies with the baseline splitting strategy. The blue horizontal line indicates the median AUC performance of the baseline splitting strategy.

We also use statistical tests to evaluate the significance of the performance difference between the random splitting strategies and the baseline splitting strategy. On the Google dataset, we find that all the performance metrics (i.e., AUC, F1 score, and MCC) for the RF models have a statistically significantly (Wilcoxon test; p-value < 0.05) higher performance on the random splitting than the baseline splitting; while on the CART model, the random splitting has a statistically higher performance only on the 80%/20% and 90%/10% training/validation ratios. On the Backblaze dataset, we find a significantly higher AUC performance for all random splitting ratios on both the RF and the CART models; for the F1 score and MCC metrics, we observe an insignificant difference on some splitting ratios. Overall, the difference indicates that randomly splitting data for training and validating a model could have introduced data leakage and over-estimated the performance of the models.

Random splitting of training and validation datasets has higher performance than time-based splitting. We observe that all the performance metrics (i.e., AUC, F1 score, and MCC) for the models that are trained and evaluated on a random sample are statistically significantly (Wilcoxon test; p-value $\ll 0.05$) higher than the performance metrics for the models that are trained and evaluated on a time-based sample with the same training/validation split ratio. For example, the average AUC of the model with 70%/30% random splitting is 0.98 and 0.99 for the Google and Backblaze datasets, respectively, while the average AUC of the model obtained using the 70%/30% time-based splitting strategy is 0.94 and 0.91, respectively. We also observe that the evaluated performance using the time-based splitting is lower than the performance of the baseline splitting. The result fits the suggestion by Kaufman et al. [39, 40] that time-based splitting is a legitimate approach to deal with data leakage issue in the context of AIOps.

4.4 Discussion

When the data is not identically and independently distributed (i.i.d), a data splitting that does not consider the temporal order in the data would lead to data leakage [39]. Indeed, we observe that the distribution of the dependent variable (i.e., the daily Google job failure rate and the monthly Backblaze disk failure rate) changes over time, as shown



(a) Google (b) Backblaze

Fig. 5. Failure rates in different periods in the studied datasets.

in Figure 5a and Figure 5b. The Google job failure rate is fluctuating between 0.5% to 3% in most of the time span, and the Backblaze disk failure rate decreases from 0.19% in the first month to 0.02% (i.e., about 90% decrease) in the last month. The results indicate that, when using a random splitting, the training data can randomly sample from the overall distribution of the failure rate; however, in a real scenario, one can only have the knowledge of the distribution of the failure rate before the testing time. The exposure of the distribution of the failure rate after the training point can lead to data leakage.

Summary of RQ1

Randomly splitting operation data for the training and validation of a model may cause data leakage problems in AIOps solutions that impact a model's realistic evaluation.

5 RQ2: WHAT IS THE IMPACT OF DIFFERENT STRATEGIES FOR SPLITTING DATA INTO TRAINING AND VALIDATION DATASETS ON MITIGATING THE RISK OF DATA LEAKAGE?

5.1 Motivation

In the last research question (Section 4), we proved that randomly splitting data for AIOps solutions do suffer from the data leakage problem. In this research question, we further study the impact of different training/validation splitting strategies on mitigating the data leakage challenge in AIOps solutions. In particular, we analyze the impact of random splitting and time-based splitting strategies with different splitting ratios across different models.

5.2 Approach

AIOps models are usually trained and evaluated on existing data, then deployed in the field to predict future samples that are unseen when training the models [49, 51]. When data leakage exists, the evaluated model performance would be higher than the real model performance on the field unseen data [39, 40, 65, 72]. In order to study the impact of different splitting strategies on mitigating data leakage, we simulate the real usage scenario of AIOps models and create three disjoint subsets from the available data: (1) a **training dataset** for training the model, (2) a **validation dataset**

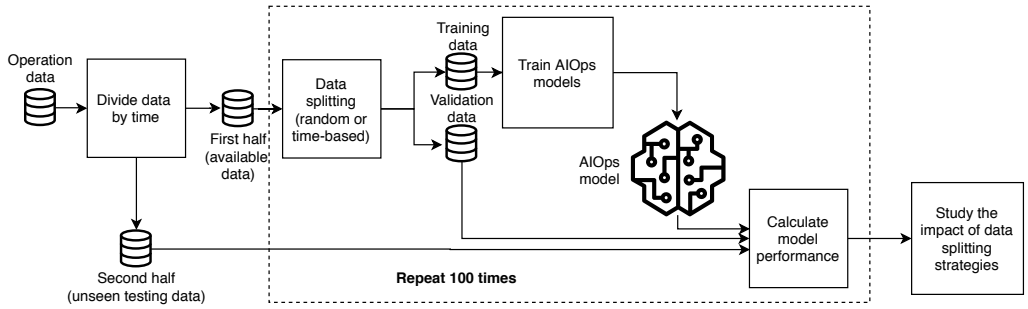


Fig. 6. An illustration of our experiment on studying the impact of different strategies for splitting the training and validation datasets.

for evaluating the performance of the model, and (3) an **unseen testing dataset** for testing the model on the simulated field unseen data. We divide the whole data into two halves in the middle of the time span. For example, for the Google dataset, the first half includes data in the first 14-day, while the second half includes data in the latter 14-day. We fix the second half as the *unseen testing dataset*, and use different splitting strategies to split the first half data (i.e., the simulated *available data*) into the *training dataset* and the *validation dataset*.

Figure 6 illustrates our experiment of splitting the data, training, and evaluating the models. Similar to RQ1, our case study considers the prediction of job failures on the Google cluster trace dataset and the prediction of disk failures on the Backblaze disk stats dataset. For both time-based and random splitting, we choose the training/validation ratios that range from 50%/50% to 90%/10%. Then, we use the training data to build the machine learning models described in 3.2.2. Finally, we leverage the metrics described in 3.2.3 to evaluate the performance of the models on the validation dataset and the unseen testing dataset. The process from data splitting to model evaluation is executed 100 times for each model with each splitting strategy and splitting ratio.

Comparing the model performance on the validation dataset and the unseen testing dataset. For each splitting strategy and splitting ratio, we compare the performance of the models on the validation dataset with the performance of the same models on the unseen testing dataset. Ideally, if no data leakage exists, the evaluated performance of a model on the validation dataset should be identical to its performance on the unseen testing data. Since there is no standard approach to quantifying the magnitude of the differences between the performance metrics, we also compute Cliff's δ between the performance on the validation set and the unseen testing data. Cliff's δ [56] is a non-parametric effect size measure that quantifies the magnitude of difference between two groups of observations. We apply the thresholds provided by Romano et al. [69] to assess the magnitude of Cliff's δ: Negligible, $\delta < 0.147$; Small: $\delta < 0.33$; Medium: $\delta < 0.474$; Large: $\delta \geq 0.474$.

We also consider a baseline model that is trained on the whole first half data (i.e., the simulated available data) and evaluated on the second half data (i.e., the simulated unseen data), as one may deploy a model that is trained on all the available data and use it to predict future unseen samples [82].

Calculating and ranking the performance difference of different splitting strategies. We also calculate the performance difference between the validation set and the unseen testing sets on each splitting strategy and splitting ratio for each model. We name the performance difference *optimism*, which have been used in prior studies to represent the difference in two observations [47, 58]. For example, if the evaluated AUC of a model is 90% on the validation dataset and 80% on the unseen testing dataset, then the AUC optimism is 10%. As we repeat our experiments 100 times, we end-up with 100 optimism values for each splitting strategy and splitting ratio for each model. We then statistically

compare the 100 optimism values of each combination of splitting strategy, splitting ratio, and model. Specifically, we sort the optimism values using the Scott-Knott clustering technique [75], which ranks all the biases into statistically distinct groups with hierarchical clustering analysis. The biases within a group have no statistically significant difference (i.e., p-value ≥ 0.05), while the biases across different groups have a statistically significant difference (i.e., p-value < 0.05).

5.3 Results

The time-based splitting strategy shows a more consistent performance between the validation and the unseen testing datasets compared to random splitting. Figure 7 and Figure 8 show the performance of the models that are trained and evaluated using different data splitting strategies and splitting ratios. Under a random splitting, the evaluated model performance (i.e., on the validation dataset) is less consistent with the performance of the same model on the unseen testing dataset; while when a time-based splitting is used, the evaluated model performance is more consistent with its performance on the unseen testing dataset. For example, for the Google dataset, the evaluation of the NN model receives an average AUC optimism of 0.045 with a 70%/30% random splitting. In contrast, the evaluation of the same model only receives an average AUC optimism of 0.004 with a 70%/30% time-based splitting. For models that leverage random splitting, we observe that the effect size of all the differences between the performance on the validation versus the unseen testing dataset is large. That holds for all the models, training and validation splitting ratios, and the performance metrics (i.e., AUC, F1 score, and MCC). For models that leverage time-based splitting, we observe four and three models that show a negligible to a medium effect size on at least one metric and at least one training/validation splitting ratio on the Google and Backblaze dataset, respectively.

Our Scott-Knott analysis of the performance optimism (Figure 9) shows that across all models and splitting ratios, for both the Google and Backblaze datasets, the least biased evaluation strategies (e.g., in the first ten groups) are all time-based splitting, while the most biased evaluation (e.g., in the last five groups) are all random splitting strategies. Our results indicate that the time-based splitting is more appropriate for AIOps model evaluation since it produces more consistent performance between the unseen testing data and the validation set.

The time-based splitting strategy provides a more realistic evaluation of an AIOps model when it is retrained on all the available data and applied to future unseen data. The estimated performance of a model (obtained on the validation dataset) is closer to the performance of the baseline model when leveraging the time-based rather than the random-based splitting strategy, as shown by the red line in Figure 7 and 8. For example, for the Backblaze dataset, the evaluation of the RF model with a 70%/30% random splitting has an average AUC difference of 0.08 from the baseline model, while the evaluation of the same model with a 70%/30% time-based splitting has an average AUC difference of only 0.02 from the baseline model. In some scenarios, a model may be retrained on all the available training data (i.e., the baseline model in our context) after the model evaluation [82]. Therefore, the time-based splitting strategy also provide a more realistic evaluation of an AIOps model when retrained on all the available data and applied to future unseen data. On the other hand, the evaluation using the random splitting strategy may show unrealistically high performance.

Some models (e.g., CART) are more likely to produce large evaluation optimism than other models. As shown in Figure 9, the CART model with the random splitting always produces the largest optimism across all the models and splitting methods (i.e., with the biggest ranks) for both case studies. For the Google dataset, the SVM and NN models with the random splitting also produce the evaluation results that are among the most biased. In contrast, the SVM and NN models with the time-based splitting produce the least biased evaluation results. For the Backblaze

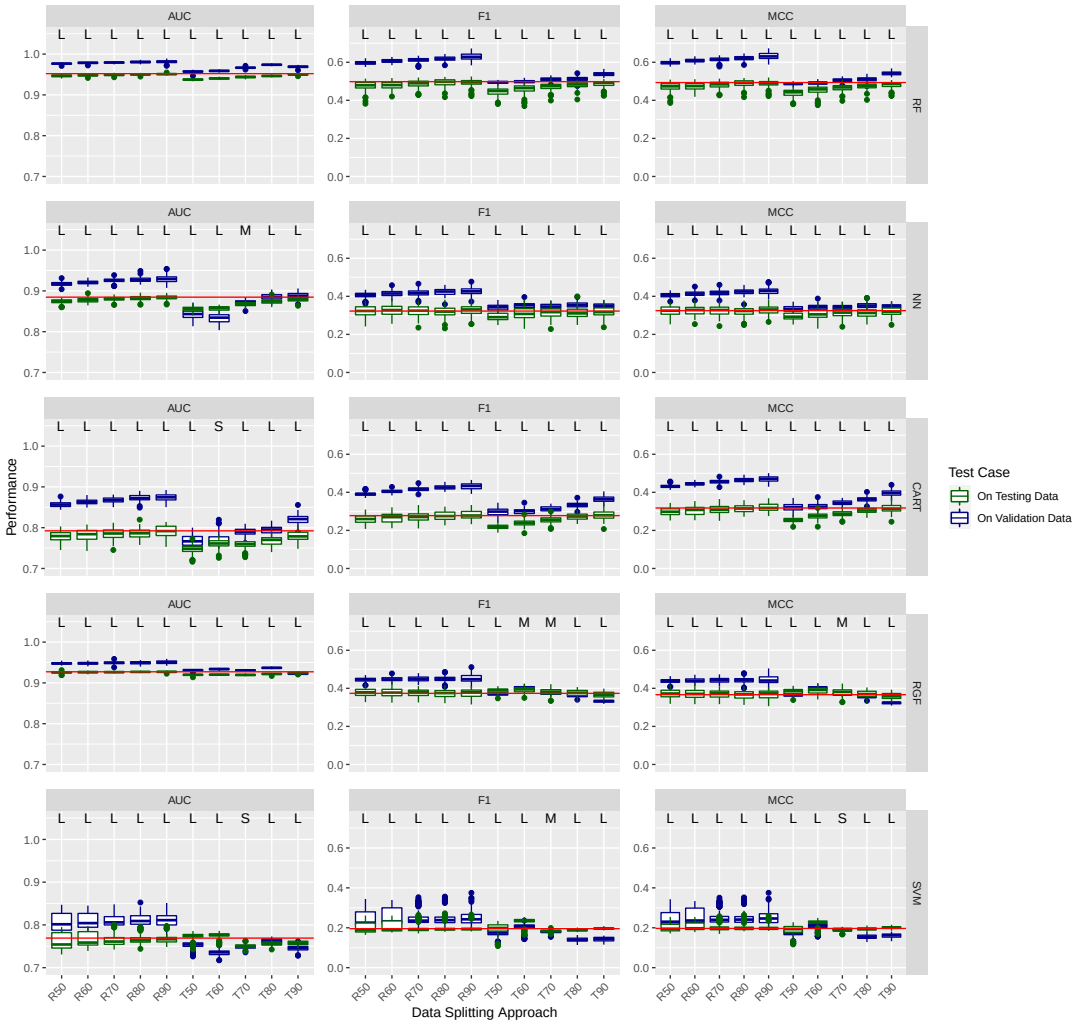


Fig. 7. Performance of the models for Google job failure prediction under different data splitting strategies. The red line indicates the performance of a model trained on the first half data and tested on the second half data (i.e., the baseline model). The text above the plots shows the effect size between the performance on the validation data and the unseen testing data (letter N, S, M, and L represents negligible, small, medium, and large effect sizes, respectively). "R" in the x-axis labels stands for random splitting, and the following number is the ratio of the training data. For example, R70 stands for a random splitting with 70% of the data as the training data and 30% as the validation data. "T" in the x-axis labels stands for time-based splitting, and the following number is the ratio of the training data. For example, T70 stands for a time-based splitting with 70% of the data as the training data and 30% as the validation data.

dataset, the RF and RGF models with the random splitting are among the most biased, while the NN model with the time-based splitting is the least biased.

For the same model and splitting strategy, the evaluation optimism are generally consistent across different splitting ratios. However, a very large splitting ratio (e.g., 90%/10%) may cause a bigger optimism than other splitting ratios. When a very large splitting ratio (e.g., 90%/10%) is used, there is more data for training and less

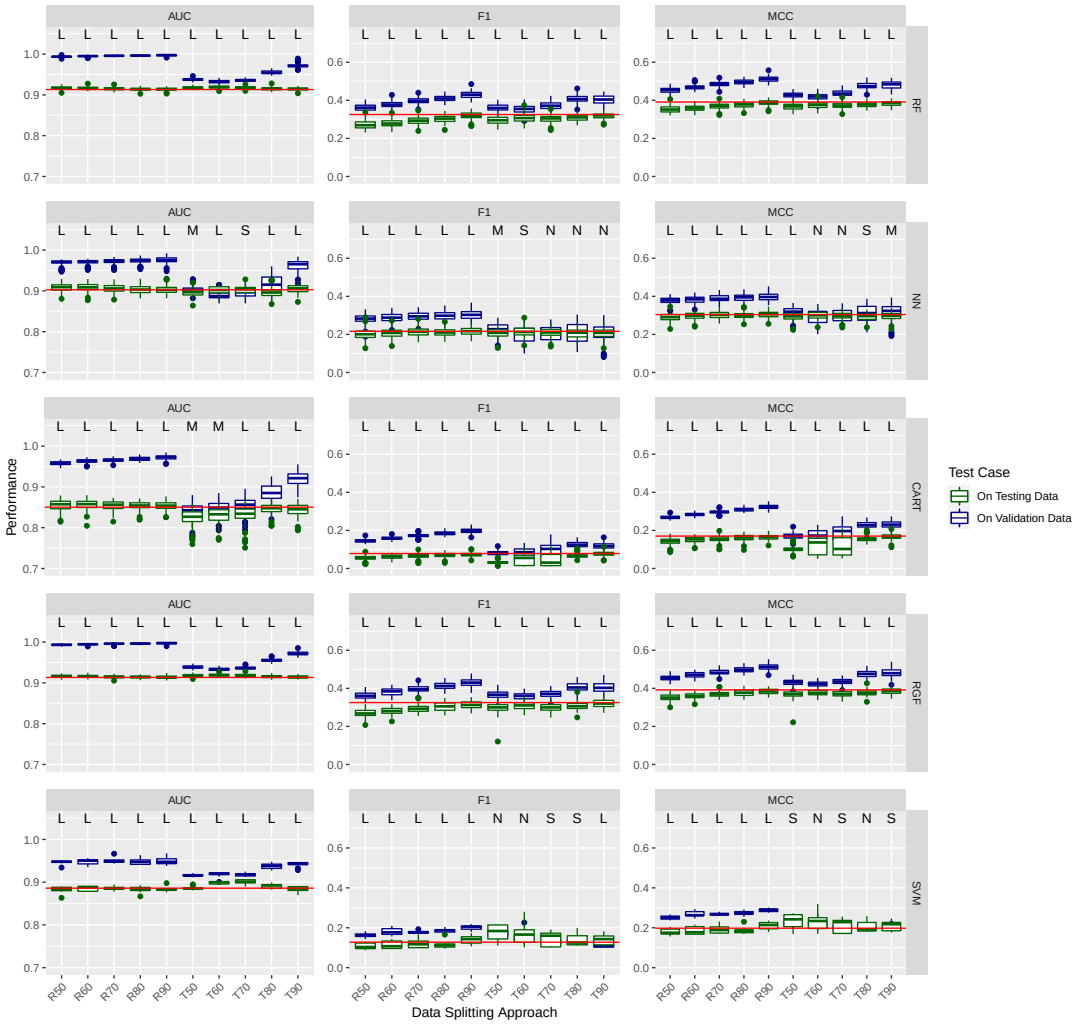
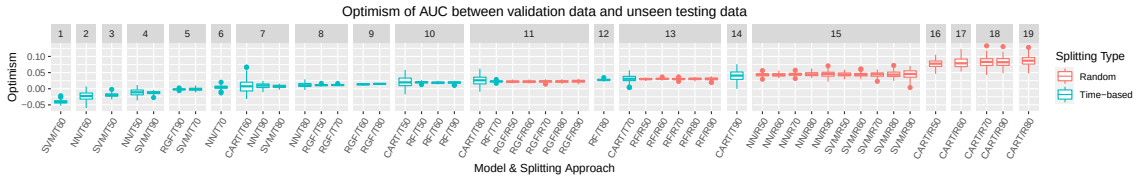
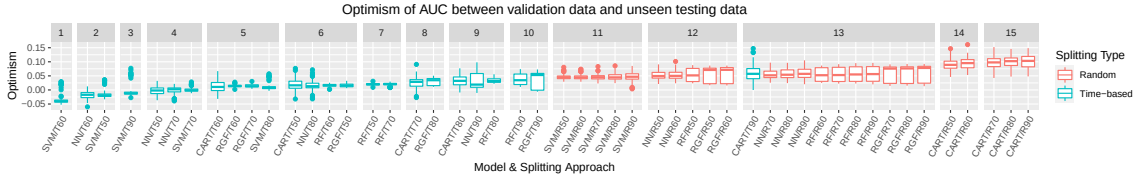


Fig. 8. Performance of the models for Backblaze disk failure prediction under different data splitting strategies. The red line indicates the performance of a model trained on the first half data and tested on the second half data (i.e., the baseline model). The text above the plots shows the effect size between the performance on the validation data and the unseen testing data (letter N, S, M, and L represents negligible, small, medium, and large effect sizes, respectively). "R" in the x-axis labels stands for random splitting, and the following number is the ratio of the training data. "T" in the x-axis labels stands for time-based splitting, and the following number is the ratio of the training data.

for evaluation, which may cause severer data leakage and result in more biased evaluation results. As shown in Figure 9, for many models and splitting strategy combinations, a very large splitting ratio (e.g., CART/R90, CART/T90, SVM/R90, RF/R90 for the Google dataset, and CART/R90, CART/T90, RFG/R90, RGF/T90, RF/R90, NN/R90 for the BackBlaze dataset) produces the largest evaluation biases in all the splitting ratios under the same setting.



(a) Google



(b) Backblaze

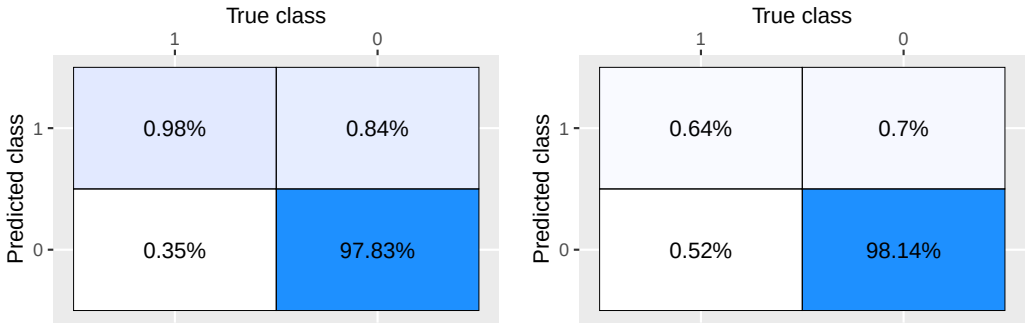
Fig. 9. Scott-Knott test of performance optimism. The red and green boxplots indicate the boxplots related to random and time-based splitting, respectively.

5.4 Discussion

Explanation of the model performance using confusion matrices. In order to understand the performance of the models and the factors that lead to the different results of the random and time-based splitting strategies, we analyze the confusion matrices of the models when evaluated on the validation datasets. Figure 10 and Figure 11 show the confusion matrices of the best-performing RF models using the random splitting and time-based splitting with a 70%/30% training/validation ratio, for the Google and Backblaze datasets, respectively.

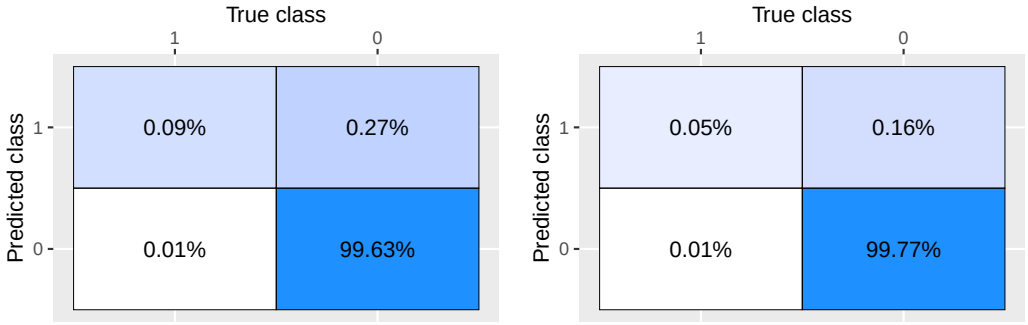
First, the confusion matrices explain why the models achieve a relatively higher AUC but lower MCC and F1 score. Our datasets are very imbalanced: only 1.30% of the samples in the Google dataset and 0.07% in the Backblaze dataset are positive cases. Under such an imbalanced data distribution, the models can still effectively distinguish the positive and negative cases. For example, as shown in Figure 11a, 90% (0.09% out of 0.1%) positive cases are correctly predicted as positive cases (true positive rate), while only 10% positive cases are wrongly predicted as negative cases (false negative rate); 99.73% (99.63% out of 99.90%) negative cases are correctly predicted as negative cases (true negative rate) while only 0.27% negative cases are wrongly predicted as positive cases (false positive rate). Therefore, the models achieve high AUC values. Although the true positive rate (90%) is much higher than the false positive rate (0.27%), the number of false positive cases (0.27%) is bigger than the number of true positive cases (0.09%), thus the F1 score and MCC values are relatively lower.

The confusion matrices also explain the difference between the results of time-based and random splitting. Both the true positive rate and the false positive rate of the model using the random splitting are much higher than the time-based splitting strategy. On the Google dataset, the true positive rate and the false positive rate on the random splitting are 0.98% and 0.94% while only 0.64% and 0.7% on the time-based splitting, respectively. On the Backblaze data, the same rates are 0.09% and 0.27% on the random splitting while only 0.05% and 0.16% on the time-based splitting. The difference can be explained by the intuition that the model using a random splitting can learn a wider range of failure patterns that evolve over time. Thus, the model tends to predict a larger percentage of positive cases. In comparison,



(a) Random splitting (b) Time-based splitting

Fig. 10. Confusion matrices for the performance of the RF model on the Google dataset using the 70%/30% random and time-based splitting strategies. "1" indicates a positive class (a failed job) while "0" indicates a negative class (a successful job).



(a) Random splitting (b) Time-based splitting

Fig. 11. Confusion matrices for the performance of 70%/30% random and time-based splitting on the RF model on the Backblaze dataset. "1" indicates a positive class (a failed disk) while "0" indicates a negative class (a functional disk).

the model using a time-based splitting can only learn the failure patterns within the time period used for training the model. In summary, random splitting may exploit the samples from the future to increase the prediction accuracy and over-estimate the model performance.

The impact of using different model configurations. In our experiments, we reuse the hyperparameters settings from prior studies on the same datasets (e.g., the same hyperparameter settings for the NN model as in Mahdisoltani et al. [57] and the same hyperparameters for the RF model as in El-Sayed et al. [24]). For the hyperparameters that are not explicitly stated in the prior studies, we manually select for our studied models. As suggested in the prior works [76, 81, 83], hyperparameter settings can have a significant impact on the performance of prediction models. For clarification, our goal is not to build the optimal models for the studied datasets, but rather to evaluate the impact of different data splitting strategies on the model performance. As our findings are consistent across different models, we expect that the findings also apply to different configurations of the same models. To verify our expectation, we vary the hyperparameters of our studied models and examine its impact on our findings. Specifically, we conduct the same

experiment in this RQ, except using hyperparameter settings obtained through a random search. We choose a random search rather a grid search for hyperparameter optimization because prior work reports that it is more efficient for hyperparameter optimization and it can find models that are as good as or better than a grid search [4].

By varying the hyperparameter settings using the random search, the models may achieve different performance. For example, the AUC, F1 score, and MCC of the CART model on the Google dataset using the 70%/30% time-based splitting change from 0.87, 0.45, and 0.41 to 0.92, , 0.40, and 0.40, respectively. However, we observe similar findings from our studied datasets and models. Overall, the random splitting has a larger gap between the performance on the validation set and the unseen testing sets than the time-based splitting, and the performance from the time-based splitting is closer to the baseline model (i.e., the model trained on the first half of data and predicted on the second half) than random splitting. For example, the RF model with the 70%/30% random splitting on the Backblaze dataset achieves an AUC of 0.986 on the validation set and an AUC of 0.923 on the unseen testing set (i.e., the optimism is 0.063). In comparison, the same model with the 70%/30% time-based splitting on the same dataset achieves an AUC of 0.934 on the validation set and 0.927 on the unseen testing dataset (i.e., the optimism is only 0.007). The baseline model has an AUC performance of 0.923, which is closer to the validation performance using the time-based splitting. In addition, all random splitting strategies still show a large effect size between the performance on the validation set and the performance on the unseen testing set, while some time-based splitting strategies show a medium, small, or even negligible performance difference.

Summary of RQ2

While prior work relies on the random splitting of training and validation sets, their reported performance on the validation set could be higher than on the unseen testing data, which is a biased evaluation. The bias is particularly larger when specific models (e.g., CART) or a very large splitting ratio (e.g., 90%/10%) is used. On the contrary, the time-based splitting is more appropriate for AIOps model evaluation since it produces more consistent performance between the validation and unseen testing data.

6 RQ3: ARE THERE CONCEPT DRIFT ISSUES FACING AIOPS SOLUTIONS?

6.1 Motivation

In machine learning and data mining, concept drift means the change in the relationships between the variables over time [84–86, 92]. Concept drift may negatively impact the performance of a model trained from the past data when applied to the new data [84–86]. Therefore, in this RQ, we analyze the studied datasets to understand whether concept drift issues exist in the context of AIOps. In particular, we leverage statistical analysis to measure the existence of concept drift in the studied datasets.

6.2 Approach

Prior work [42, 45, 64] assumes that, given a stationary data distribution (i.e., no concept drift), a model trained on a previous time period would achieve a prediction performance (when evaluated on the next time period) that has no statistical difference from the prediction performance on the training period. We follow the same hypothesis to measure the concept drift in our studied datasets. If a model trained from the previous data shows a statistically significant performance difference on the new data, then a concept drift exists.

In our study, we use the natural time intervals (i.e., one-day periods for the Google dataset and one-month periods for the Backblaze dataset) to split the data into different time periods. We choose such a time window size as prior works have applied similar update strategies. For example, Lin et al. [51] update their model deployed in a production cloud service system with data from a one-month window. Similarly, Li et al. [49] consider retraining their model periodically and they also apply a one-month window. Also, Xu et al. [89] perform a daily model update with the data in a 90-day sliding window. We conduct our experiment as follows:

(1) For each time period, we train a model using the data from that time period and test the same model using the next time period's data to measure the prediction error rate.
(2) We then compute the statistical difference between the model's prediction error rate on the training time period and its prediction error rate on the testing time period, similar to prior work [45, 64]. However, these studies [45, 64] do not explicitly explain how they measure the prediction error rate on the training time period. Thus we follow prior work [42, 86] and use 10-fold cross-validation on the training time period to measure the prediction error rate on the training time period.
(3) Similar to prior work [45, 64], we use a two-proportion Z-test to compute the statistical difference between the model's prediction error rates in the training and testing time periods, which is described as follows:

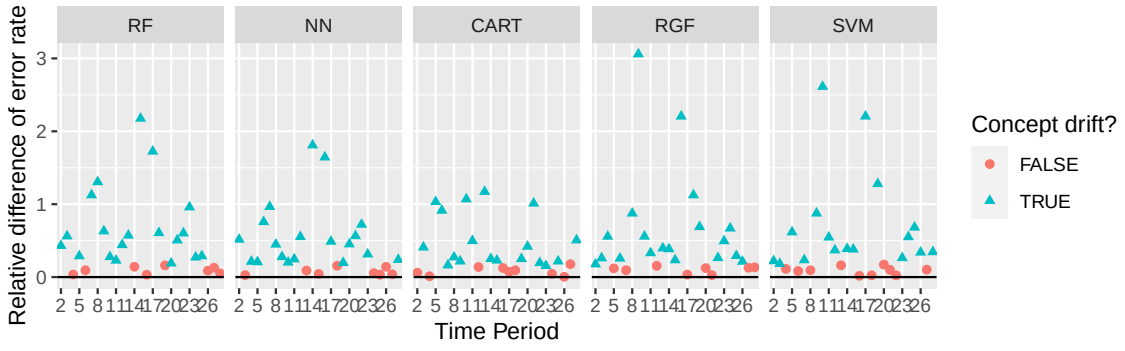
$$Z = \frac{(\hat{p_2} - \hat{p_1}) - 0}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{n_1} + \frac{1}{n_2})}} \tag{1}$$

where $\hat{p_1}$ is the prediction error rate in the training time period, $\hat{p_2}$ is the prediction error rate in the testing time period, $\hat{p}$ is the overall prediction error rate, and n1 and n2 are the number of samples in the training time period and the testing time period, respectively.
(4) We then determine the significance level (i.e., p-value) from the Z-test. When the p-value is less than 0.05, we reject the null hypothesis (i.e., $\hat{p_1} - \hat{p_2} = 0$) and consider the alternative that there is a concept drift between the two time periods.
(5) To understand the magnitude of the concept drift between two adjacent time periods, we also calculate the relative difference between the prediction error rates in the training and testing time periods ($(\hat{p_2} - \hat{p_1})/\hat{p_1}$).

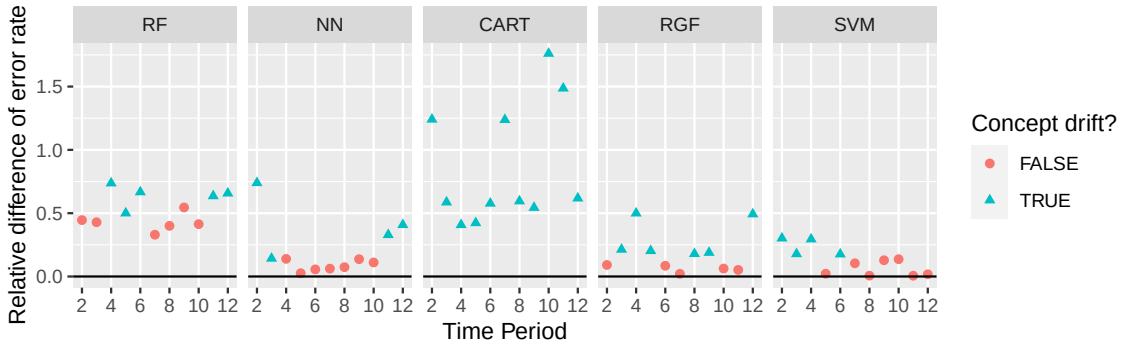
6.3 Results

Concept drift exists in the operation data. Figure 12 describes the concept drift in different time periods of the studied datasets. We observe that many of the time periods show a concept drift from its previous period. For example, on the Google dataset, the RF, NN, and CART models indicate that 70% (19 out of 27) time periods exhibit concept drift, and the CART and SVM indicate 18 and 17 periods with concept drift, respectively. On the Backblaze dataset, the CART model shows that all 11 time periods have concept drift from the previous time periods, while the other four models (i.e., RF, NN, RGF, SVM) indicate that 5, 4, 6, and 4 out of the 11 time periods exhibit concept drift, respectively. The different number of concept drift detected by different models may be because different models have different sensitivity to the evolution of the data. For example, comparing the CART and RF model, the CART model may be more sensitive to the specific patterns in each period of data (i.e., more vulnerable to over-fit), while the RF model can better learn the general patterns [7] thus it is less sensitive to the specific changes in the data across different time periods.

Aside from having a larger portion of time periods showing concept drift, we also notice that the Google dataset shows a higher level of concept drift than the Backblaze data. For example, on our studied models (i.e., RF, NN, CART,



(a) Google



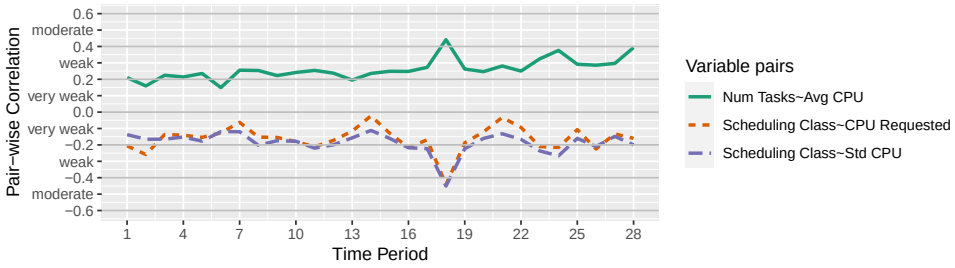
(b) Backblaze

Fig. 12. Concept drift between consecutive time periods in the studied datasets. A green symbol indicates a concept drift between a time period with the previous time period (i.e., p-value < 0.05 in the Z-test); while a red symbol indicates no concept drift between a time period and the previous time period (i.e., p-value $\geq$ 0.05 in the Z-test).

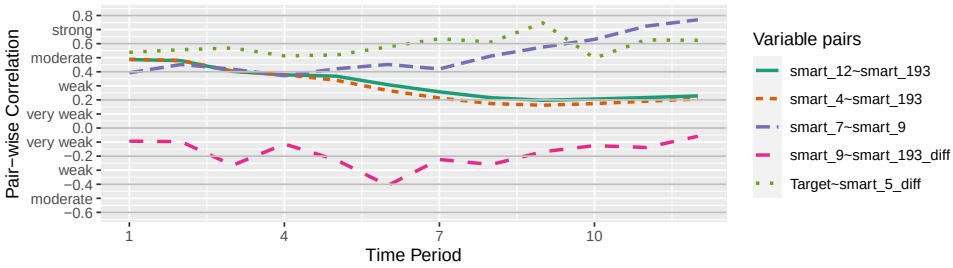
RGF, SVM), the consecutive time periods in the Google dataset exhibit a relative error rate difference of up to 2.2, 1.8, 1.2, 3.0, and 2.6 while the values are only 0.7, 0.7, 1.8, 0.5, and 0.3 on the Backblaze data, respectively.

6.4 Discussion

We also conduct experiments on the relationship between the variables and find that the existence of concept drift may be explained by the fact that the relationship between the variables in the operation data evolves over time. Figure 13 shows the Spearman correlation between each pair of variables that have at least a moderate correlation (i.e., ≥ 0.4 or ≤ -0.4) in one time period. We consider the Spearman correlation because it does not require the data to be normally distributed (the Shapiro-Wilk test of normality shows that the variables in our datasets are not normally distributed). For the Google dataset, the correlations between the explanatory variables fluctuate dramatically over time. For example, the correlation between the variable `Scheduling Class` and the variable `CPU Requested` changes from `very week` in one time period to `moderate` in another time period. For the Backblaze dataset, the correlations between the explanatory variables show more gradual changes over time. For example, the correlation between the



(a) Google



(b) Backblaze

Fig. 13. Pairwise Spearman correlation between the variables that have at least a moderate correlation on one time period. We classify the absolute values of the correlation r as follows [25]: very weak ($r < 0.2$); weak ($r < 0.4$); moderate ($r < 0.6$); strong ($r < 0.8$); very strong ($r \geq 0.8$).

variable `smart_5_diff` and the target variable evolves from `moderate` to `strong`, while the correlation between the variable `smart_12` and the variable `smart_193` evolves from `moderate` to `weak` over time.

Summary of RQ3

Concept drift exists in the operation data, which can be explained by the fact that the relationship between the variables in the operation data evolves over time. Practitioners and researchers should proactively detect and address the problem of concept drift in their AIOps solutions.

7 RQ4: HOW SHOULD WE MAINTAIN AND UPDATE MODELS THAT ARE DEPLOYED IN PRACTICE TO MITIGATE THE RISK OF CONCEPT DRIFT?

7.1 Motivation

In the last research question (Section 6), we show that *concept drift* issues do exist in our studied operation datasets. Such concept drift may lead to obsolescence of AIOps models, i.e., a model trained from the previous data may become outdated. This research question aims to evaluate the impact of updating a model on reducing the impact of the concept drift. We also study how different model update frequency impacts the performance and cost of AIOps models.

Fig. 14. Periodically updated models vs. stationary models.

7.2 Approach

We measure whether continuously retraining a model can help reduce the impact of concept drift in AIOps models. To do so, we compare how the performance of a model evolves when using a periodically updated model compared to a stationary model, which are illustrated in Figure 14 and defined as follows:

- **The stationary model.** It is a model trained on an initial time window (e.g., the first six months for the Backblaze data) and never gets updated. In our evaluation, we consider a stationary model trained on the first half of the data, which consists of the first fourteen days from the Google dataset and the first six months from the Backblaze datasets.
- **The periodically updated model.** Contrary to the stationary model built on an initial time window, the periodically updated model is updated using the latest available data (i.e., the data in a sliding window that moves forward over time). We do not use all the historical data to update the model (instead use the data in the latest sliding window), as it is difficult to scale, especially when the model needs to be updated frequently (e.g., in an online deployment environment). Besides, updating the model using all the historical data may not lead to better model performance than only using the new data in the latest sliding window [51].

In our evaluation, we split samples into multiple time periods of one-day and one-month, and initially train a model on the first fourteen days and six months from the Google and Backblaze datasets, respectively. Then, we slide the training window forward by one period, train the model using the data in the training window, and test the model on the next period. For example, we first train a model on the Backblaze dataset on the first six months, test it on the seventh month. Then, we train a model on the data between the second and the seventh months and test the model on the eighth month. We also compute the statistical difference (Wilcoxon Mann Whitney test) and the effect size (Cliff's δ) between the performance on the stationary model and on the periodically updated model on each testing time window to measure the significance and magnitude of the difference.

Analyzing the impact of time period sizes. In order to understand how frequently one should update AIOps models, we divide the whole dataset into N equal-size time periods (i.e., chunks) to train the periodically updated models. Note that we use a fixed sliding window size (i.e., half of the whole data) for different N settings. In the beginning, we train a model using the first half of the data. Then, we simulate the scenario of updating the model when the data chunk in a new time period becomes available and use the periodically updated model to predict the samples in the data chunk of the next time period. In this work, we vary N from 4 to 24. For each N setting, we evaluate the overall model performance on all the tested time periods (i.e., by comparing the predicted results and observed results on all the tested data combined) and the total training and testing time (i.e., the modeling cost).

In this research question, we consider the same models (i.e., RF, CART, NN, RGF, and SVM) and the same performance metrics (i.e., F1-measure, AUC, and MCC) that are used in RQ2.

7.3 Results

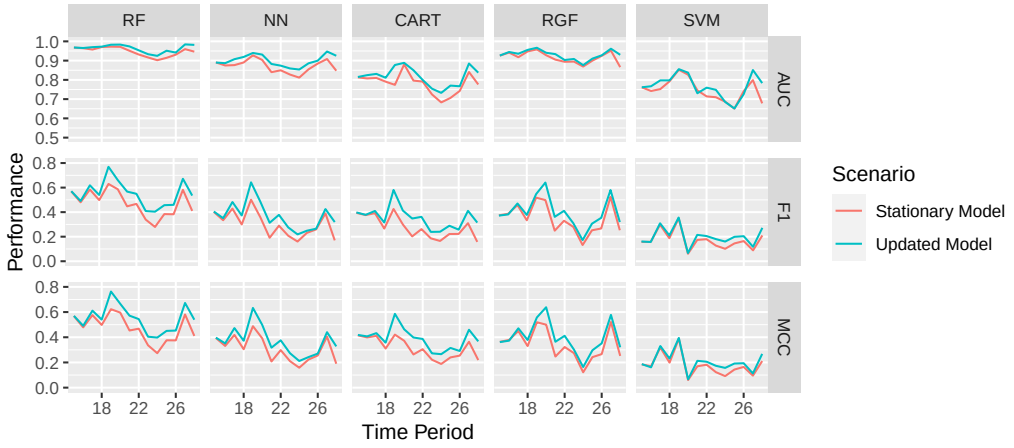
Periodically updated AIOps models provide better performance than the stationary models, which suggest modelers update their AIOps models periodically. As shown in Figure 15a and Figure 15b, periodically updating the models achieve a better performance in terms of the evaluated metrics than using a stationary model, and overall the difference becomes bigger when the distance between the training periods of the stationary model and the testing period becomes larger. We observe that the stationary models and the periodically updated models have a bigger difference on the Google dataset than on the Backblaze dataset, which may be explained by the fact that the Google dataset have a more severe concept drift issue (as discussed in Section 6).

We also use the Wilcoxon test and Cliff's δ to measure the significance and magnitude of the performance difference between the stationary and periodically updated models on each testing window after the first testing window (on which the stationary model and periodically updated model have the same training set thus no significant performance difference). On Google data, we observe that after the third testing period (i.e., the 17th time period), all models have a statistically different (Wilcoxon test; p-value < 0.05) performance with at least a medium effect size (Cliff's δ, $\delta \geq 0.474$) on all three performance metrics, except on the 26th and 27th period of NN model and the 19th period of the RGF model, where the MCC and F1 score do not have a significant difference; and the 18th, 24th to 26th time periods of the SVM model, where the performance difference of AUC metric is not significant, and the effect size is negligible or small. We observe 11, 10, 10, 10, and 7 out of 13 testing periods have a statistically different performance with at least a medium effect size on all three performance metrics using the RF, CART, NN, RGF, and SVM models, respectively.

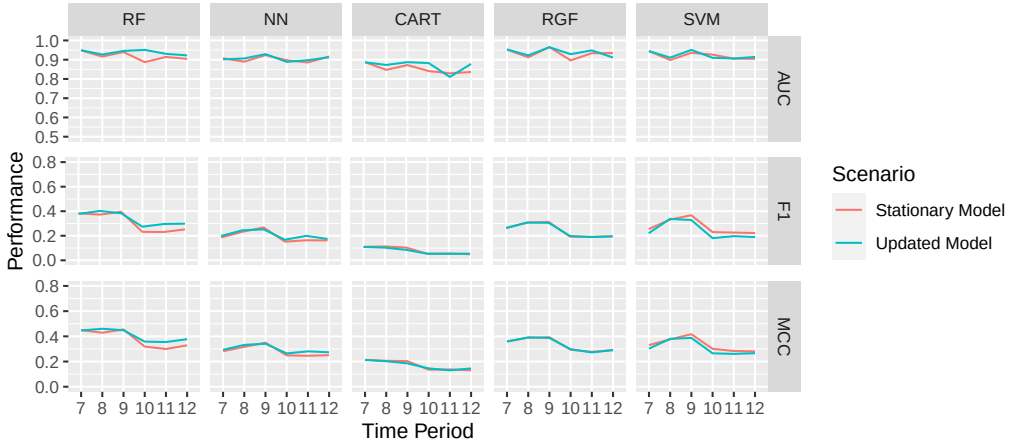
On the Backblaze data, the models generally have a statistically different (Wilcoxon test; p-value < 0.05) performance with at least a medium effect size (Cliff's δ, $\delta \geq 0.474$) on the five testing periods starting from the 8th time window but with more exceptions. On the RF and SVM models, only the performance on 1 out of 5 time period does not show a significant difference or at least a medium effect size on one of the performance metrics, while using the NN, CART, and RGF models, the performance on 2 out of 5 periods does not have a significant performance difference for at least one metric.

Finally, we observe that periodically updating some models achieves better performance improvement than updating other models. For example, updating the CART model achieves an average AUC improvement of 0.04 on the Google dataset, while updating the SVM model achieves an average AUC improvement of only 0.01. Our result on the performance difference between the stationary and periodically updated models further proves the existence of concept drift.

Increasing the frequency of updating the AIOps models can improve the performance, while the improvement shows difference across models and datasets. Figure 16 shows the overall performance of the models (in terms of AUC) when we vary the number of time periods (i.e., N). Other performance metrics show a similar trend. In most cases, increasing the model update frequency can gradually improve model performance. For example, the AUC of the CART model on the Backblaze dataset increases by 3.5% (i.e., from 0.85 to 0.88) when we increase the number of time periods from 4 to 24 (the AUC of the stationary model, i.e., when $N = 2$, is 0.84). However, in some cases, for example, when updating the SVM model on the Backblaze dataset, we did not observe any performance improvement. We infer that some models (e.g., RGF) can not learn the evolving patterns in the datasets, thus are less sensitive to the update frequency.



(a) Google



(b) Backblaze

Fig. 15. Comparison between the performance of the stationary and periodically updated models.

Increasing the model update frequency increase the overall cost of AIOps models; however, the cost increase varies significantly across models and datasets. Figure 17 shows the overall time cost (including both training and testing time) of updating different models at different update frequencies relative to the time cost of the stationary model. Updating some models (e.g., NN and CART) imposes a higher cost than updating other models (e.g., RF, RGF, and SVM). For example, changing the update frequency of the NN model from 4 to 24 leads to a 5.6 times increase of the time cost on the Backblaze dataset. In comparison, changing the update frequency of the RF model from 4 to 24 only leads to a 1.8 times increase of the time cost on the Backblaze dataset. Finally, it may not be cost-effective to increase the model update frequency for some models. For example, increasing the update frequency of the NN model

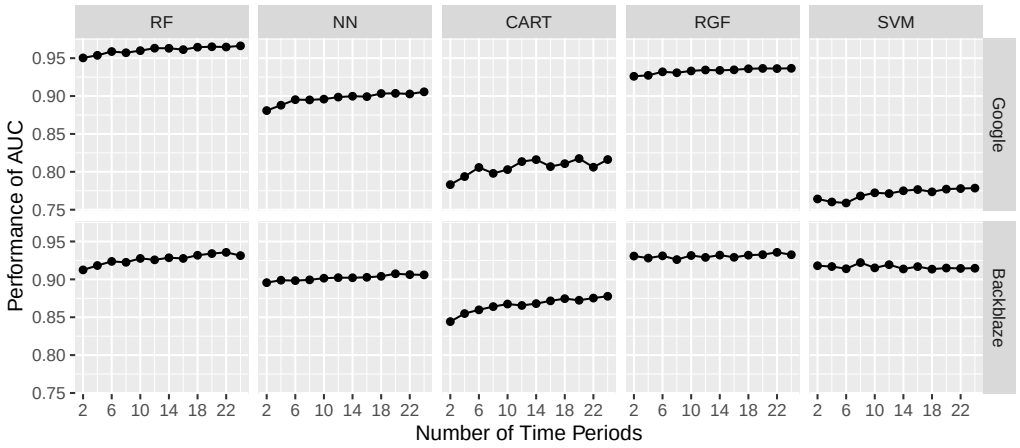


Fig. 16. Change of AUC performance on periodically updated model when varying number of chunks. The number of time periods of two indicates the stationary models.

from 4 to 24 only lead to an AUC improvement of 1.99% and 0.78% on the Google and Backblaze datasets, respectively, while it results in 6.0 and 5.6 times of modeling cost (including both training time and testing time) increase, respectively.

Updating the models more frequently increases the training cost proportionally to the update frequency. However, the overall testing time should remain consistent for different update frequencies as the models are tested on the same testing data. Therefore, the models' overall update cost is different due to the different ratios between these models' training and testing costs. For example, for the Backblaze dataset, the ratio between the total training cost and the total testing cost is 0.59, 0.48, and 0.49 for the RF, RGF, and SVM models, respectively, while the ratio is 32.90 and 3.15 for the NN and CART models, respectively. For typical machine learning applications, the training cost is far higher than the testing cost. However, when under-sampling is applied in AIOps models (as in this work), the cost of model training might be comparable to the cost of model testing, particularly for certain models (e.g., RF). Therefore, increasing the update frequency of such models come at a relatively lower cost. In general, increasing the model update frequency for the Google dataset leads to a higher ratio of the cost increase, which might be explained by the fact that the target classes in the Google dataset are relatively more balanced than that in the Backblaze dataset, as a higher data imbalance leads to a larger reduction in the training data after applying under-sampling.

7.4 Discussion

Prequential performance. To further understand the evolution of the performance of the models over time, we also evaluate the prequential AUC [8, 10] in addition to the three metrics we used to measure model performance (i.e., AUC, F1 score, and MCC). In the context of data streams, a classifier's predictive ability is usually calculated with forgetting, i.e., focusing sequentially on the most recent examples [27]. Prequential AUC [8, 10] provides an efficient way to compute AUC incrementally after each example by using a special red-black tree structure to keep the prediction results within a sliding window. Prequential AUC provides consistent results with traditional AUC [10].

The computation of prequential AUC requires a pre-determined sliding window. In our case, we choose a 10K-sample window for the Google data and a 100K-sample window for the Backblaze data to ensure sufficient samples of the minority classes in the window. We do not test other window sizes as prior work [10] reports that it only has a negligible

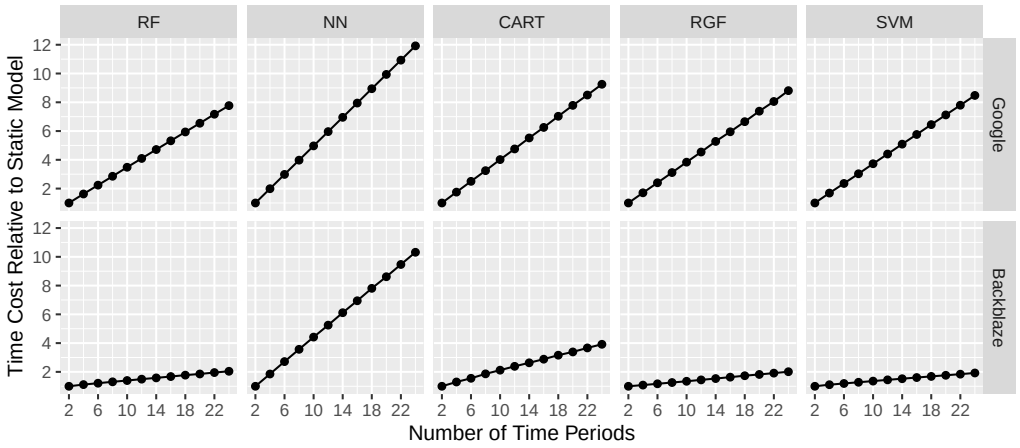


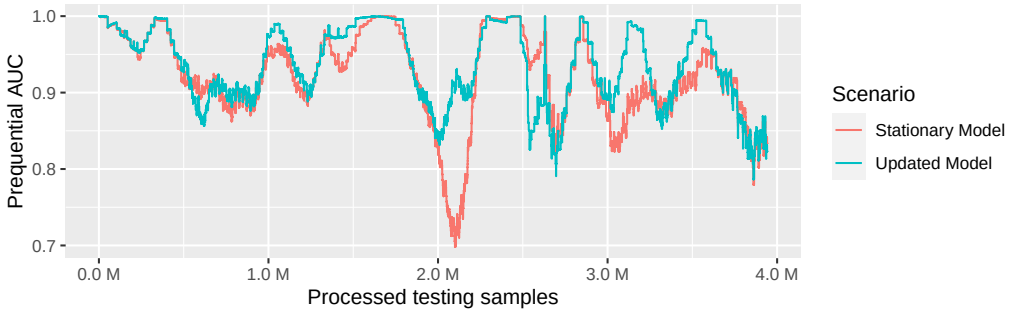
Fig. 17. The overall time cost (including both training and testing time) of updating different models at different update frequency, relative to the time cost of the stationary model. The number of time periods of two indicates the stationary models.

impact on the resulting AUC estimation. We compute the prequential AUC on the RF model for the two studied datasets to observe the evolution of the performance of the periodically updated and stationary models after each sample. As shown in Figure 18, on both datasets, the periodically updated models have a better performance than the stationary models for most of the samples, which matches the results of our block-wise AUC estimation.

The impact of using different model configurations. Similar to RQ2, in order to study the impact of hyperparameters on our findings, we vary the hyperparameters of the models. Specifically, we apply the optimized hyperparameter settings obtained from the random search on our models and carry out the same experiments in this RQ. Overall, we observe a similar trend when comparing the performance of the stationary and periodically updated models. The stationary and periodically updated models have a rather similar performance at the first testing periods, then the periodically updated models outperform the stationary ones, and the performance difference becomes bigger as the time period proceeds. For example, the AUC performance of the stationary and periodically updated CART models on the Google data both begins with 0.90 on the second testing period while ends up with 0.83 for the stationary model and 0.90 for the periodically updated model on the last period. Besides, we observe almost the same trend as our experiments above for the performance and training time impact of the update frequency.

We also conducted experiments to investigate whether highly-tuned models are more sensitive to concept drift. Specifically, we compare the performance of the same models between the default and the optimized hyperparameter settings. Our intuition is that a highly-tuned model would produce a larger performance difference from the default configuration. We find that the NN and CART models show the largest performance differences between the default and the optimized hyperparameters. For example, the stationary NN and CART models show an average relative performance difference of 13.4% and 4.0%, respectively. However, the NN and CART models are not necessarily more sensitive to concept drift (see Figure 15). In other words, we do not find any observable evidence to support that highly-tuned models are more sensitive to concept drift.

(a) Google



(b) Backblaze

Fig. 18. Comparison between the prequential AUC of the stationary and periodically updated approaches on RF model.

Summary of RQ4

Due to the existence of concept drift, AIOps models should be updated periodically, as periodically updated models outperform stationary models. In general, increasing the frequency of updating AIOps models can lead to better performance while increasing the modeling cost. However, the performance benefit and modeling cost of increasing the update frequency show very different trends across models and datasets.

8 THREATS TO VALIDITY

8.1 External Validity

An external threat to validity concerns the generalizability of our results to other datasets, models, and predicted features (target variables). While we do not generalize our results, our study considers popular datasets (i.e., the Google cluster trace dataset and the Backblaze disk stats dataset) and models (i.e., RF, CART, SVM, RGF, and NN) on two types of AIOps tasks. Future work can replicate our study on more datasets, models, and other target variables.

In this study, we focus on general models and approaches that have been used in prior AIOps works. As prior works use only traditional AI models (e.g., RF, NN, and CART), we did not assess whether our techniques and results would be generalized to other AI models (e.g., online learning models or deep learning-based models).

This work examines existing approaches from the machine learning community for handling data leakage and concept drift problems in the context of AIOps. In theory, AIOps problems can be tackled by traditional machine learning techniques. However, as the operation data is constantly evolving and usually contains heterogeneous data types, special considerations may be needed for tackling such AIOps problems.

8.2 Internal Validity

An internal threat to validity concerns the performance of the model, which might be impacted by the characteristics of the training or testing datasets. To mitigate this risk, we consider 100 different samples for random-splitting strategies, and we also consider different splitting ratios for both time-based and random-based splitting strategies.

Although the data could be partitioned into time periods of other sizes, we consider the natural time periods (e.g., days and months) such that practitioners can better leverage the tools to maintain their models (e.g., daily or monthly), similar to approaches used in prior works [49, 51, 89].

Our RQ1 and RQ2 examine the data leakage issues in the AIOps context. We would like to clarify that, while the data leakage issues impact the evaluation of AIOps models (i.e., over-estimate the model performance), they do not impact the actual performance of a model that is trained on collected historical data and applied on unseen field data.

8.3 Construct Validity

A construct threat to validity considers the threshold of 0.5 that we used when calculating the F1 score and MCC for classifying a positive outcome. The performance and conclusions may change by changing the threshold [79]. To mitigate the versatility of the conclusion from the threshold-dependent metrics, we also evaluated the model performance with AUC, a threshold-independent metric for model evaluation.

Another threat to construct validity concerns our model training process. We reuse the hyperparameters from the prior studies or manually tune the hyperparameters. Considering using other hyperparameters may lead to different model performance, in Section 5 and Section 7, we vary the hyperparameters of the models using a random search and demonstrate that our main findings can be applicable to models with different hyperparameter configurations.

There may be other potential threats concerning our model training process. Nevertheless, we replicate the machine learning workflow of the prior studies [5, 24, 57] while only changing the data splitting strategies.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we study the data leakage and concept drift challenges in the context of AIOps and evaluate the impact of different modeling decisions on such challenges. Specifically, we perform a case study on two commonly studied AIOps applications: (1) predicting job failures based on trace data from a large-scale cluster environment, and (2) predicting disk failures based on disk monitoring data from a large-scale cloud storage environment. Our results demonstrate that data leakage and concept drift issues exist in the operation data and should not be ignored in AIOps solutions. We observe that time-based splitting of training and validation data is more appropriate for AIOps model evaluation, which calls for caution on AIOps practices that attempt to evaluate their AIOps models using random-based splitting that could cause data leakage. We also observe that periodically update AIOps models can help mitigate the impact of concept drift and maintain the model performance, while the frequency of model update should be cautiously considered in the context of the dataset and the used models. Our findings motivate AIOps practitioners and software solutions that manage the AIOps life-cycle to carefully address the data splitting-related challenges (e.g., data leakage and concept drift) in developing and maintaining AIOps solutions. As prior AIOps works are still using approaches that

could induce the data leakage and concept drift issues, our work also calls for attention to the two challenges for future AIOps studies.

Our study highlights the issues and presents the best practices to handle the data leakage and concept drift in the context of AIOps solutions. In the future, we plan to assess the impact of different data splitting strategies on additional AI models (e.g., online learning models and deep learning-based models). In addition, we also seek to develop automated solutions to detect and mitigate data leakage and concept drift problems in various AIOps solutions. Finally, as operation datasets are usually very large and are constantly generating, we also intend to explore more resource-efficient approaches to develop and maintain high-quality AIOps solutions over time.

REFERENCES

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. 2019. MIMETIC: Mobile Encrypted Traffic Classification using Multimodal Deep Learning. *Computer Networks* 165 (2019), 106944.

[2] Amritanshu Agrawal and Tim Menzies. 2018. Is "Better Data" Better Than "Better Data Miners"?: On the Benefits of Tuning SMOTE for Defect Prediction. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. 1050–1061.

[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*. 291–300.

[4] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13, 10 (2012), 281–305.

[5] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. 2016. Predicting Disk Replacement Towards Reliable Data Centers. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. 39–48.

[6] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. 2017. Optimal Classifier for Imbalanced Data using Matthews Correlation Coefficient Metric. *PLOS ONE* 12, 6 (2017), 1–17.

[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[8] Dariusz Brzezinski and Jerzy Stefanowski. 2014. Prequential AUC for Classifier Evaluation and Drift Detection in Evolving Data Streams. In *Proceedings of the 3rd International Conference on New Frontiers in Mining Complex Patterns (NFMCP '14)*. 87–101.

[9] Dariusz Brzezinski and Jerzy Stefanowski. 2014. Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (2014), 81–94.

[10] Dariusz Brzezinski and Jerzy Stefanowski. 2017. Prequential AUC: Properties of the Area Under the ROC Curve for Data Streams with Concept Drift. *Knowledge and Information Systems* 52 (2017), 531–562.

[11] Alberto Cano and Bartosz Krawczyk. 2019. Evolving Rule-Based Classifiers with Genetic Programming on GPUs for Drifting Data Streams. *Pattern Recognition* 87 (2019), 248–268.

[12] Alberto Cano and Bartosz Krawczyk. 2020. Kappa Updated Ensemble for Drifting Data Stream Mining. *Machine Learning* 109, 1 (2020), 175–218.

[13] Nitesh V. Chawla, Kevin Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357.

[14] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. 2014. Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW '14)*. 341–346.

[15] Yong Chen, Ruping Pan, and Alexander Pfeifer. 2017. Regulation of Brown and Beige Fat by MicroRNAs. *Pharmacology & Therapeutics* 170 (2017), 1–7.

[16] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *The World Wide Web Conference (WWW '19)*. 2659–2665.

[17] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: Real-World Challenges and Research Innovations. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '19)*. 4–5.

[18] Rui Ding, Qiang Fu, Jian-Guang Lou, Qingwei Lin, Dongmei Zhang, Jiajun Shen, and Tao Xie. 2012. Healing Online Service Systems via Mining Historical Issue Repositories. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*. 318–321.

[19] Rui Ding, Qiang Fu, Jian Guang Lou, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Mining Historical Issue Repositories to Heal Large-Scale Online Service Systems. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14)*. 311–322.

[20] Pedro Domingos and Geoff Hulten. 2000. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. 71––80.

[21] Priyanka B. Dongre and Latesh G. Malik. 2014. A Review on Real Time Data Stream Classification and Adapting to Various Concept Drift Scenarios. In *2014 IEEE International Advance Computing Conference (IACC '14)*. 533–537.

[22] Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall, and Abraham Bernstein. 2009. Tracking Concept Drift of Software Projects using Defect Prediction Quality. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories (MSR '09)*. 51–60.

[23] Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall, and Abraham Bernstein. 2012. Time Variance and Defect Prediction in Software Projects - Towards an Exploitation of Periods of Stability and Change as well as a Notion of Concept Drift in Software Projects. *Empirical Software Engineering* 17, 4-5 (2012), 348–389.

[24] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. 2017. Learning from Failure Across Multiple Clusters: A Trace-Driven Approach to Understanding, Predicting, and Mitigating Job Terminations. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS '17)*. 1333–1344.

[25] James D. Evans. 1996. *Straightforward Statistics for the Behavioral Sciences*.

[26] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with Drift Detection. In *Advances in Artificial Intelligence (SBIA '04)*. 286–295.

[27] João Gama, Raquel Sebastião, and Pedro Rodrigues. 2013. On Evaluating Stream Learning Algorithms. *Machine Learning* 90 (2013), 317–346.

[28] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Computing Surveys* 46, 4, Article 44 (2014).

[29] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. 2015. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *Proceedings of the 37th International Conference on Software Engineering (ICSE '15)*. 789–800.

[30] Maayan Harel, Koby Crammer, Ran El-Yaniv, and Shie Mannor. 2014. Concept Drift Detection through Resampling. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML '14)*. II–1009–II–1017.

[31] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying Impactful Service System Problems via Log Analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*. 60–70.

[32] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. 2012. Learning from Streaming Data with Concept Drift and Imbalance: an Overview. *Progress in Artificial Intelligence* 1, 1 (2012), 89–101.

[33] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining Time-Changing Data Streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. 97–106.

[34] Backblaze Inc. 2020. Backblaze Hard Drive Stats. Backblaze B2 Cloud Storage. Posted at https://www.backblaze.com/b2/hard-drive-test-data.html.

[35] Arya Iranmehr, Hamed Masnadi-Shirazi, and Nuno Vasconcelos. 2019. Cost-Sensitive Support Vector Machines. *Neurocomputing* 343 (2019), 50–64.

[36] Rie Johnson and Tong Zhang. 2013. Learning Nonlinear Functions Using Regularized Greedy Forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 5 (2013), 942–954.

[37] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E. Hassan. 2016. Studying Just-In-Time Defect Prediction using Cross-Project Models. *Empirical Software Engineering* 21 (2016), 2072–2106.

[38] Andrej Karpathy. 2019. "We see more significant improvements from training data distribution search (data splits + oversampling factor ratios) than neural architecture search. The latter is so overrated :)". https://twitter.com/karpathy/status/1175138379198914560. Accessed: 2020-05-03.

[39] Shachar Kaufman, Saharon Rosset, and Claudia Perlich. 2011. Leakage in Data Mining: Formulation, Detection, and Avoidance. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11, Vol. 6)*. 556–563.

[40] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in Data Mining: Formulation, Detection, and Avoidance. *ACM Transactions on Knowledge Discovery from Data* 6, 4 (2012), 1–21.

[41] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. 2018. Software Engineering for Machine-Learning Applications: The Road Ahead. *IEEE Software* 35, 5 (2018), 81–84.

[42] Ralf Klinkenberg and Thorsten Joachims. 2000. Detecting Concept Drift with Support Vector Machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*. 487–494.

[43] Bartosz Krawczyk and Alberto Cano. 2018. Online Ensemble Learning with Abstaining Classifiers for Drifting and Noisy Data Streams. *Applied Soft Computing* 68 (2018), 677–692.

[44] Max Kuhn and Kjell Johnson. 2013. *Applied Predictive Modeling*. Vol. 26.

[45] Mark Last. 2002. Online Classification of Nonstationary Data Streams. *Intelligent Data Analysis* 6, 2 (2002), 129–147.

[46] Heng Li, Tse-Hsun Peter Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2018. Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems: An Industrial Experience Report. In *Proceedings of the 40th International Conference on Software Engineering (ICSE-SEIP '18)*. 1–10.

[47] Heng Li, Weiyi Shang, and Ahmed E Hassan. 2017. Which Log Level Should Developers Choose for a New Logging Statement? *Empirical Software Engineering* 22, 4 (2017), 1684–1716.

[48] Heng Li, Weiyi Shang, Ying Zou, and Ahmed E Hassan. 2017. Towards Just-In-Time Suggestions for Log Changes. *Empirical Software Engineering* 22, 4 (2017), 1831–1865.

[49] Yangguang Li, Zhen Ming Jiang, Heng Li, Ahmed E. Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. 2020. Predicting Node Failures in an Ultra-Large-Scale Cloud Computing Platform: An AIOps Solution. *ACM Transactions on Software Engineering and Methodology* 29, 2 (2020), 1–24.

[50] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. 2014. Identifying Recurrent and Unknown Performance Issues. In *2014 IEEE International Conference on Data Mining (ICDM '14)*. 320–329.

[51] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. Predicting Node Failure in Cloud Service Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*. 480–490.

[52] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software Analytics for Incident Management of Online Services: An Experience Report. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE '13)*. 475–485.

[53] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2017. Experience Report on Applying Software Analytics in Incident Management of Online Service. *Automated Software Engineering* 24, 4 (2017), 905–941.

[54] Nicola Lunardon, Giovanna Menardi, and Nicola Torelli. 2014. ROSE: A Package for Binary Imbalanced Learning. *R Journal* 6 (2014), 79–89.

[55] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating Events with Time Series for Incident Diagnosis. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. 1583–1592.

[56] Guillermo Eduardo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. 2011. Cliff's Delta Calculator: A Non-parametric Effect Size Program for Two Groups of Observations. *Universitas Psychologica* 10 (2011), 545–555.

[57] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. 2017. Proactive Error Prediction to Improve Storage System Reliability. In *2017 USENIX Annual Technical Conference (ATC '17)*. 391–402.

[58] Shane Mcintosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2016. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[59] Giovanna Menardi and Nicola Torelli. 2014. Training and Assessing Classification Rules with Imbalanced Data. *Data Mining and Knowledge Discovery* 28, 1 (2014), 92–122.

[60] Leandro L. Minku, Allan P. White, and Xin Yao. 2009. The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift. *IEEE Transactions on knowledge and Data Engineering* 22, 5 (2009), 730–742.

[61] Leandro L. Minku and Xin Yao. 2011. DDD: A New Ensemble Approach for Dealing with Concept Drift. *IEEE Transactions on Knowledge and Data Engineering* 24, 4 (2011), 619–633.

[62] Leon Moonen, Stefano Di Alesio, David Binkley, and Thomas Rolfsnes. 2016. Practical Guidelines for Change Recommendation using Association Rule Mining. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16)*. 732–743.

[63] Andrew Ng. 2019. "The rise of Software Engineering required inventing processes like version control, code review, agile, to help teams work effectively. The rise of AI & Machine Learning Engineering is now requiring new processes, like how we split train/dev/test, model zoos, etc.". https://twitter.com/andrewyng/status/1080886439380869122. Accessed: 2020-05-03.

[64] Kyosuke Nishida and Koichiro Yamauchi. 2007. Detecting Concept Drift Using Statistical Testing. In *Discovery Science*. 264–269.

[65] Claudia Perlich. 2014. *Lessons Learned from Data Competitions: Data Leakage and Model Evaluation. Book chapter.* 303–320.

[66] Teerat Pitakrat, André van Hoorn, and Lars Grunske. 2013. A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection. In *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems (ISARCS '13)*. 1–10.

[67] Pankaj Prasad and Charley Rich. 2018. Market Guide for AIOps Platforms. Gartner Research. Posted at https://www.gartner.com/doc/3892967/market-guide-aiops-platforms.

[68] Thomas Rolfsnes, Leon Moonen, and David Binkley. 2017. Predicting Relevance of Change Recommendations. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE '17)*. 694–705.

[69] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate Statistics for Ordinal Level Data: Should We Really Be Using T-Test and Cohen'd for Evaluating Group Differences on the NSSE and Other Surveys. In *annual meeting of the Florida Association of Institutional Research*. 1–3.

[70] Andrea Rosà, Lydia Y. Chen, and Walter Binder. 2015. Catching Failures of Failures at Big-Data Clusters: A Two-level Neural Network Approach. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS '15)*. 231–236.

[71] Andrea Rosà, Lydia Y. Chen, and Walter Binder. 2015. Predicting and Mitigating Jobs Failures in Big Data Clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '15)*. 221–230.

[72] Saharon Rosset, Claudia Perlich, Grzergorz Świrszcz, Prem Melville, and Yan Liu. 2010. Medical Data Mining: Insights from Winning Two Competitions. *Data Mining and Knowledge Discovery* 20, 3 (2010), 439–468.

[73] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, Gyuri Szarvas, Manasi Vartak, Samuel Madden, Hui Miao, Amol Deshpande, et al. 2018. On Challenges in Machine Learning Model Management. *IEEE Data Engineering Bulletin* 41, 4 (2018), 5–15.

[74] Conlan Scientific. 2020. Avoiding Data Leakage in Machine Learning. Conlan Scientific. Posted at https://conlanscientific.com/posts/category/blog/post/avoiding-data-leakage-machine-learning/.

[75] Andrew Jhon Scott and M Knott. 1974. A Cluster Analysis Method for Grouping Means in the Analysis of Variance. *Biometrics* 30, 3 (1974), 507–512.

[76] Liyan Song, Leandro L. Minku, and Xin Yao. 2013. The Impact of Parameter Tuning on Software Effort Estimation Using Learning Machines. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering (PROMISE '13)*. Article 9.

[77] W. Nick Street and YongSeog Kim. 2001. A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. New York, NY, USA, 377–382.

[78] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online Defect Prediction for Imbalanced Data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE '15, Vol. 2)*. 99–108.

[79] Chakkrit Tantithamthavorn and Ahmed E Hassan. 2018. An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18)*. 286–295.

[80] Chakkrit Tantithamthavorn, Ahmed E. Hassan, and Kenichi Matsumoto. 2020. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. *IEEE Transactions on Software Engineering* 46, 11 (2020), 1200–1219.

[81] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. 321–332.

[82] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering* 43, 1 (2017), 1–18.

[83] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2018. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering* 45, 7 (2018), 683–711.

[84] Alexey Tsymbal. 2004. The Problem of Concept Drift: Definitions and Related Work. *Computer Science Department, Trinity College Dublin* 106, 2 (2004), 58.

[85] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. 2003. Mining Concept-Drifting Data Streams Using Ensemble Classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*. 226—-235.

[86] Haixun Wang, Philip S. Yu, and Jiawei Han. 2010. *Mining Concept-Drifting Data Streams*. 789–802.

[87] Shuo Wang, Leandro L. Minku, Davide Ghezzi, Daniele Caltabiano, Peter Tino, and Xin Yao. 2013. Concept Drift Detection for Online Class Imbalance Learning. In *2013 International Joint Conference on Neural Networks (IJCNN '13)*. 1–10.

[88] John Wilkes. 2020. *Google Cluster-Usage Traces V3*. Technical Report. Google Inc. Posted at https://github.com/google/cluster-data/blob/master/ClusterData2019.md.

[89] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, Murali Chintalapati, and Dongmei Zhang. 2018. Improving Service Availability of Cloud Systems by Predicting Disk Error. In *2018 USENIX Annual Technical Conference (ATC '15)*. 481–494.

[90] Ji Xue, Robert Birke, Lydia Y. Chen, and Evgenia Smirni. 2016. Managing Data Center Tickets: Prediction and Active Sizing. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '16)*. 335–346.

[91] Ji Xue, Robert Birke, Lydia Y. Chen, and Evgenia Smirni. 2018. Spatial–Temporal Prediction Models for Active Ticket Managing in Data Centers. *IEEE Transactions on Network and Service Management* 15, 1 (2018), 39–52.

[92] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. 2016. An Overview of Concept Drift Applications. In *Big Data Analysis: New Algorithms for a New Society*. 91–114.