SAM: Subseries Augmentation-based Meta-learning for Generalizing AIOps Models in Multi-Cloud Migration

Xi Yang[1], Paulito Palmes[1], Saurabh Jha[1], Bekir Turkkan[1], Gerard Vanloo[1], Frank Bagehorn[2], Chandra Narayanaswami[1], Larisa Shwartz[3], Naoki Abe[1], Yu Deng[1], Daby M. Sow[1]
IBM Research[1], IBM Software[2], IBM Consulting[3]
xi.yang@ibm.com, paulpalmes@ie.ibm.com, {saurabh.jha, b.turkkan, gerard.vanloo}.ibm.com, fba@zurich.ibm.com, {chandras, lshwart, nabe, dengy, sowdaby}.us.ibm.com

***Abstract*—In the context of cloud computing, enterprises are increasingly adopting multi-cloud strategies to enhance performance, ensure cost efficiency, and avoid vendor lock-in. This trend presents a significant challenge for the migration of AI for IT operations (AIOps) models across different cloud providers due to variations in architecture, performance, and data distribution. Traditional methods of re-training AIOps models for new cloud environments are labor-intensive and delay deployment. To address this issue, we introduce a novel framework called SAM (Subseries Augmentation-based Meta-learning), which facilitates seamless model migration between clouds without the need for re-training from scratch. SAM leverages data augmentation and meta-learning to efficiently adapt AIOps models to new cloud environments. It has proven effective in adapting anomaly detectors across various configurations over both public and simulated datasets. We believe that SAM can also be adapted to other AI models used for automating IT tasks such as alerting and resource scaling.**

***Index Terms*—multi-cloud migration, model generalization, anomaly detection**

I. Introduction

Multi-cloud computing is gaining importance in the IT landscape and among businesses [**?**], [**?**]. From a business perspective, it empowers organizations to avoid vendor lock-in and provides the flexibility to choose from different cloud providers. In spot markets, it facilitates the economical migration of "bursty" applications. From a technical standpoint, it ensures resiliency, availability, and flexibility. It can safeguard against cloud provider outages, optimize capacity via regional cloud providers, and enable the use of best-of-breed services, which promotes productivity and offers opportunities for various applications to utilize the cloud platform that best matches their requirements.

When operating in cloud computing, AIOps models are developed using IT operational data (*e.g.*, metrics, logs, traces) to automate and streamline operational workflows. Such models can help reduce significantly the cognitive load and enhance the productivity of Site Reliability Engineers (SREs) by exploiting extensive and diverse operational data [**?**]. For example, AIOps models can efficiently detect anomalies, locate root causes, automatically resolve problems, and manage cloud resources, all without requiring manual operational intervention. The integration of data-driven AIOps models plays a pivotal role in accelerating and automating the resolution of complex IT issues, thereby simplifying the management burden for SREs.

When migrating IT applications from a source cloud to a target cloud, the observability data is susceptible to experiencing distribution drifts. It can be caused by the diverse compute, network, storage, software, and hardware configurations across different cloud providers [**?**]. Thus, directly applying the AIOps model learned from one cloud to another would likely result in suboptimal performance (*e.g.*, accuracy, false positive rate) due to the discrepancies in data distributions. Though it is possible to re-train the model from scratch using newly observed data from the target cloud, this can be expensive as the data collection is usually time-consuming and labor-intensive. The objective of the present paper, therefore, is to develop strategies for seamlessly migrating AIOps models to new cloud environments, by formulating it as an out-of-distribution generalization problem [**?**] and solving by leveraging machine learning techniques. Since different IT operational data can be represented as or converted to time-series, with similar format to metrics, we focus on migrating models based on time-series metrics.

Towards this goal, in this paper we propose a Subseries Augmentation-based Meta-learning (*SAM*) framework for multi-cloud migration, aiming to learn generalized AIOps models from a source cloud, which can be readily adapted to target clouds with zero-shot learning. *SAM* consists of two key steps: (i) data *augmentation from subseries*, which partitions and clusters the time-series metrics collected from source cloud into different groups with varying distributions, and then augment data from each distribution to generate diverse representations; and (ii) *meta-learning domain generalization*, which captures shared patterns across varying distributions to learn a single model with generalization capability. The effectiveness of *SAM* was validated on a public dataset and a simulated dataset, each exhibiting significant distribution drifts across different configurations. The key contributions of this paper can be summarized as follows:

1) A novel framework, *SAM*, was proposed to facilitate

the migration of AIOps models in multi-cloud environments. This framework can capture generalized patterns across different distributions through the combination of data augmentation and meta-learning.

2) The effectiveness of *SAM* was validated via extensive experiments over both public and simulated datasets. *SAM* significantly outperformed direct model migration across different distributions and achieved comparable performance to scenarios in which the model was retrained with sufficient data collected from the target cloud. It indicates *SAM*'s capability to maintain decent performance while avoiding the costs associated with data collection and the delays in deployment.
3) The effectiveness of *SAM* in migrating state-of-the-art anomaly detectors was evaluated. Our results indicate that *SAM* can successfully adapt anomaly detectors across various experimental settings without the need for re-training. We argue that *SAM* can be applied to a broader class of AIOps models used for managing IT tasks such as root cause analysis and resource scaling.

II. Problem Statement

A. Drifts in Multi-Cloud Computing Environment

In multi-cloud computing, various cloud providers offer diverse compute, network, storage, software, and hardware configurations, which often leads to altered behaviors when serving the same application. Figure 1 illustrates the average 90th percentile latency in seconds for five services of a containerized application deployed on two systems with different configurations. System A is equipped with 12 VCPUs and 48 GiB of memory, while System B has 24 VCPUs and 96 GiB of memory. As expected, the latency decreases for most of the services on System B due to its greater resources, while this is not the case for all services. The distribution of latency for each service also varies. For example, Service A exhibits a much larger standard deviation in latency on System A compared to System B.

Another example of behavioral changes due to differences in configurations is depicted in Figure 2. A larger system is configured with 24 VCPUs and 94 GiB of memory, while a smaller system has 24 VCPUs but only 46 GiB of memory. We observe that the smaller system's Requests Per Second (RPS) in Configuration 1 is comparable to that of the larger system but in Configuration 2 the smaller system's RPS drops significantly in contrast to the larger system.

The variations in configurations lead to distribution drifts in the gathered observability data during the migration of applications and associated AIOps models from a source cloud to target clouds. A model that performs well on the source cloud may not necessarily perform as effectively on the target cloud due to differing distributions. Although the model can be re-trained using newly collected data from the target cloud, data availability remains a significant obstacle, as the process of collecting and labeling sufficient data is inherently time-consuming and can lead to delays in model re-learning. Additionally, there is often a strong

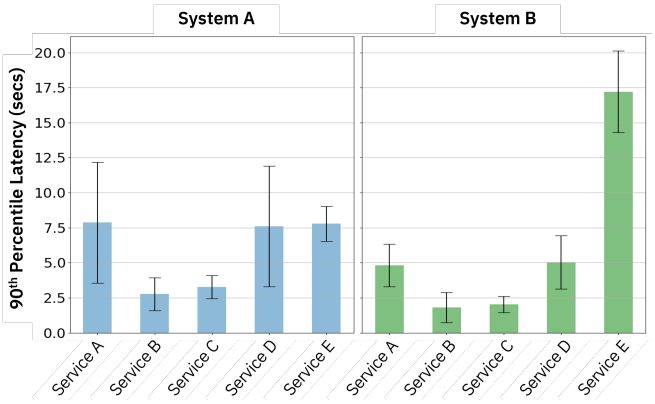


Fig. 1: 90th percentile latency of services for two systems.

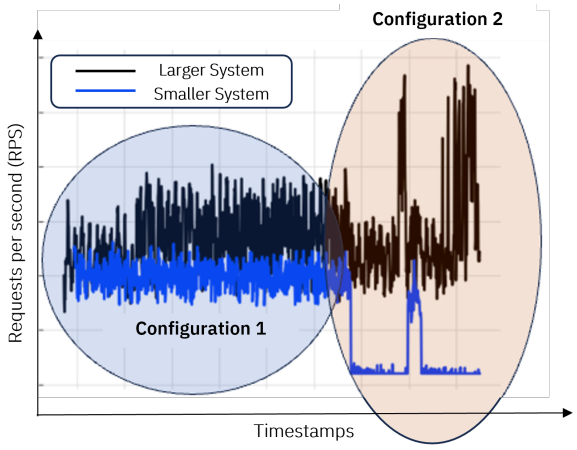


Fig. 2: RPS for two systems under different configurations

preference for having a model readily applicable before any observations become available from the target cloud, which poses further challenges for seamless and efficient deployment of AIOps solutions in multi-cloud environments.

B. Out-of-distribution Generalization

Addressing the drift in AIOps models in multi-cloud computing can be modeled as a problem of *out-of-distribution generalization*. Recently, there has been a growing body of research motivated by the need to deal with the susceptibility of machine learning models to data distribution shifts [**?**]. It can generally be addressed by enhancing data generalization [**?**], [**?**], [**?**], model generalization [**?**], [**?**], or both [**?**], [**?**].

1) Data Generalization: Data augmentation techniques can enhance generalization by generating populations with more diverse patterns in a more cost-effective manner compared to traditional data collection, which is especially beneficial when access to the data of unknown target distributions is limited [**?**]. The recent advancement in data augmentation has enabled the generation of synthetic data that closely resembles real operational environments. This capability has become a key driver behind the generalization capability of industrial AI solutions to real-world scenarios [**?**]. Despite the enriched data after augmentation, directly taking mixed

distributions as input for a single model might still have a limited capability in fully capturing their shared patterns. To handle this issue, model generalization can be employed.

2) Model Generalization: To bolster the model's capacity to tolerate unseen distributions, we aim to facilitate model generalization through meta-learning, which is also known as "learning to learn." By training the model with diverse learning tasks, meta-learning enables it to swiftly adapt to new tasks with limited observations [**?**]. To address the distribution drifts, we model each distribution in the training data as an individual learning task. Our primary goal is to derive a model that exemplifies robust generalization capabilities through adept adaptation to new tasks characterized by various distributions. In the context of multi-cloud computing, where the target cloud lacks prior information on its data distribution, we expect the model to adapt through zero-shot observations from the target cloud. To tackle this challenge, we employed a Meta-Learning Domain Generalization (MLDG) [**?**] technique in SAM. Instead of using a specific model tailored for generalization, MLDG serves as a model-agnostic algorithm that enhances the robustness of various AIOps models (*e.g.*, supervised, unsupervised, reinforcement learning). By capturing shared patterns across different distributions, MLDG can filter out the impact of infrastructure and software over the application performance, allowing us to learn various generalized models and transfer them across different configurations. However, meta-learning can only effectively capture distribution drifts and achieve a more robust model if the input data admits a certain level of generalization. To address this issue, incorporating data generalization becomes essential.

3) Data & Model Generalization: Based on the above analysis, it is evident that relying solely on either data or model generalization is insufficient. Therefore, the objective of this work is to leverage the strengths of both approaches by effectively combining them. While there have been some prior works that proposed similar goals [**?**], [**?**], they are designed for specific tasks, *e.g.*, only for unsupervised tasks, and may not generalize well for different models. To overcome this issue, in this paper, we harness the capabilities of *both data augmentation and meta-learning in SAM* to achieve data and model generalization simultaneously, aiming to obtain a model-agnostic framework with the potential to be readily applicable on a wide range of tasks.

III. Methodology

In this paper, we propose a Subseries Augmentation-based Meta-learning (SAM) framework, as illustrated in Fig. 3. The framework consists of two key steps: (i) *Augmentation from subseries*, which involves partitioning time-series data into subseries clusters with different distributions and then augmenting data from each distribution; and (ii) *Meta-learning for domain generalization*, which captures shared patterns across different distributions to achieve model generalization. These two steps will be detailed in what follows.

Algorithm 1 Augmentation from Subseries

Input: Time-series $\mathbf{X} = \{\mathbf{x}_t | t \in [1, T]\}$
Initial: Cluster number k
Employ GMM to learn $\{(\mu_k, \sigma_k) | k \in [1, K]\}$
for each cluster $k \in K$ **do**
  Get subseries $\{\bar{\mathbf{x}}^k\}$ belonging to k
  Initialize an empty time-series $\tilde{\mathbf{X}}^k = \{\}$
  while $\tilde{\mathbf{X}}^k$ has fewer timestamps than $\mathbf{X}$ **do**
    Randomly sample a subseries $\{\bar{\mathbf{x}}_p^k\}$ from $\{\bar{\mathbf{x}}^k\}$
    Randomly replace timestamps in $\{\bar{\mathbf{x}}_p^k\}$ to be the data sampled from (μ_k, σ_k) and get $\{\tilde{\mathbf{x}}_p^k\}$
    Concatenate $\{\tilde{\mathbf{x}}_p^k\}$ to the end of $\tilde{\mathbf{X}}^k$
  end while
end for
Output: The augmented time-series $\{\tilde{\mathbf{X}}^k, k \in [1, K]\}$

A. Augmentation from Subseries

Given a time-series, its observations are denoted as $\mathbf{X} = \{\mathbf{x}_t | t \in [1, T]\}$, with $\mathbf{x}_t \in \mathbb{R}^m$ being the t-th multivariate observation with m features (metrics). Our motivation is to simultaneously *partition and cluster* the time-series into subseries clusters, by learning a mapping from each observation $\mathbf{x}_t$ to a certain cluster $\{k | k \in [1, K]\}$.

Gaussian mixture model (GMM) can be employed to learn such mapping in an unsupervised manner [**?**]. The GMM decomposes all observations optimally into K Gaussians, with the k-th fitted distribution corresponding to the k-th cluster. Each cluster can then be characterized by its mean and standard deviation (std). For all K distributions, determining the optimal mean vectors $\{\mu_k\}$ is equivalent to matching each observation to a cluster, and determining the optimal $\{\sigma_k\}$ is to estimate K stds for measuring how dispersed the observations are in relation to cluster centers.

Given the subseries clusters learned from GMM, we then conducted cluster-wise data augmentation. For a certain cluster k, we first gathered all subseries $\{\bar{\mathbf{x}}^k\}$ belonging to it. An empty augmented time-series was initialized as $\tilde{\mathbf{X}}^k = \{\}$, which we would augment iteratively. Specifically, while $\tilde{\mathbf{X}}^k$ had fewer timestamps compared to $\mathbf{X}$, the following steps were conducted repeatedly: (i) A subseries $\{\bar{\mathbf{x}}_p^k\}$ was randomly sampled from $\{\bar{\mathbf{x}}^k\}$; (ii) Randomly selected timestamps in $\{\bar{\mathbf{x}}_p^k\}$ were replaced with the data sampled from (μ_k, σ_k), converting it to $\{\tilde{\mathbf{x}}_p^k\}$; (iii) $\{\tilde{\mathbf{x}}_p^k\}$ was concatenated to the end of $\tilde{\mathbf{X}}^k$, with an interpolated timestamp to ensure smoothness. Consequently, for each cluster k, we obtained an augmented time-series $\tilde{\mathbf{X}}^k$, from a specific distribution with diverse potential patterns.

B. Meta-learning for Generalized Model

In the context of meta-learning, we consider each augmented time-series with a specific distribution as a "domain". Our goal is to develop domain-agnostic models from training domains $\mathcal{S}$ (*i.e.*, $\{\tilde{\mathbf{X}}^k\}$ with different distributions in our case), which can be readily generalized to unseen domains. It

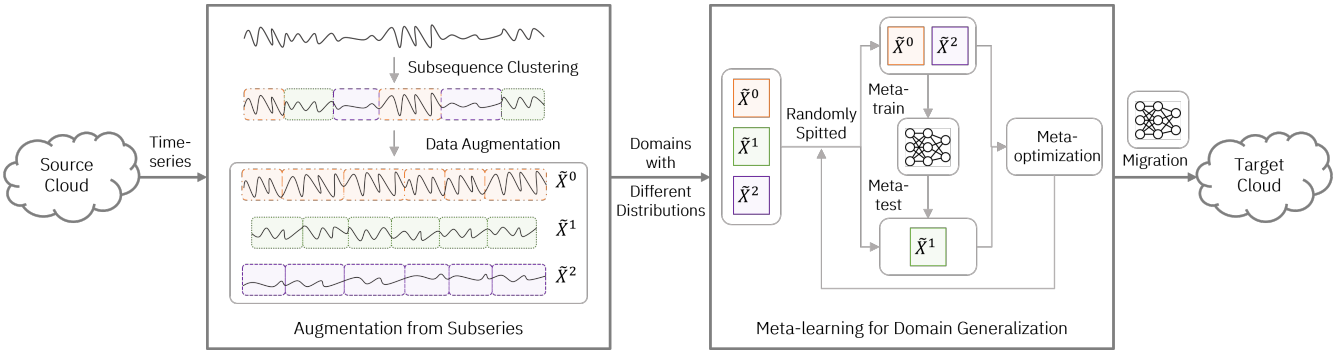


Fig. 3: Illustration of the SAM Framework.

can be modeled as a Meta-Learning Domain Generalization (MLDG) problem [**?**].

Algorithm 2 Meta-learning for Domain Generalization

1: Input: Source domains $\{\mathcal{S}\}$ and initialized Θ
2: **for** ite **in** iterations **do**
3: **Splitting** $\mathcal{S}$ for *meta-train* and *meta-test*
4: **Meta-train**: updating Θ to Θ'
5: **Meta-test**: calculating loss over meta-test by Θ'
6: **Meta-optimization**: Updating Θ via combined loss from both *meta-train* and *meta-test*
7: **end for**
8: Output: A generalized model readily migrating to $\mathcal{T}$

Given a set of source domains $\{\mathcal{S}\}$ and a target domain $\mathcal{T}$ (for pursuing a common task, *e.g.*, anomaly detection, while supplied with different statistics), MLDG aims to learn a single model parameterized with Θ from $\{\mathcal{S}\}$, which can be generalized to $\mathcal{T}$ with zero-shot observation. It consists of four iterative steps: domain splitting, meta-train, meta-test, and meta-optimization.

In *domain splitting*, the source domains $\{\mathcal{S}\}$ are divided into two sets (*i.e.*,, $\{\mathcal{S}^{tr}\}$ and $\{\mathcal{S}^{te}\}$) for *meta-train* and *meta-test*, respectively. In *meta-train*, based on $\{\mathcal{S}^{tr}\}$, the gradient $\nabla_\Theta = f'_\Theta(\{\mathcal{S}^{tr}\};\Theta)$ is calculated and Θ is updated with a one-step gradient: $\Theta' = \Theta - \alpha\nabla_\Theta$, where α indicates the step size. Then in *meta-test*, given the updated model parameters Θ', the loss over $\{\mathcal{S}^{te}\}$ is calculated, *i.e.*, $g(\{\mathcal{S}^{te}\};\Theta')$. Finally, in *meta-optimization*, Θ is optimized with a combined loss from both meta-train and meta-test:

$$\Theta = \Theta - \gamma\frac{\partial(f(\{\mathcal{S}^{tr}\};\Theta) + \beta g(\{\mathcal{S}^{te}\};\Theta - \alpha\nabla_\Theta))}{\partial\Theta} \quad (1)$$

The above four steps will be conducted iteratively until convergence, *e.g.*, reaching a pre-defined iteration number. Through meta-learning, the model gains the generalization capability across different domains, enabling it to be effectively migrated to an unseen domain.

IV. Experiments

A. Public Dataset

1) Data Description: To validate the efficacy of our proposed framework, *SAM*, we applied it to a public Server Machine Dataset (SMD) [**?**]. SMD[1] is collected from a large Internet company over five weeks, containing the metrics data of 28 server machines from three groups, with each machine being named as `machine-<group_index>-<index>`. There were 38 metrics collected over time, including CPU load, network usage, memory usage, etc. The SMD has been extensively employed to evaluate anomaly detection performance in prior works [**?**], [**?**], [**?**]. Recently, Tuli et al. indicated that anomaly detection for some of the machines in SMD are trivial [**?**], *i.e.*, the anomalies can be easily captured by naive methods [**?**]. Following their suggestions, in this paper, we included only those machines in SMD for which detecting anomaly was non-trivial using simple heuristics, specifically the data from `machine-1-1`, `2-1`, `3-2`, and `3-7`, denoted as m_1, m_2, m_3, and m_4, respectively.

Table I depicts the detailed data size and anomalies for each selected machine. The data from each machine was divided into two subsets with roughly equal size, with the first half for training and the second half for testing. The training set is entirely normal, while the testing set contains both normal and abnormal data. The anomalies were tagged by domain experts referring to incident reports. The average anomaly rate among the four machines is around 2.56%. The last column in Table I indicates the timestamp where the first anomaly occurs in testing data, which can be early (*e.g.*, 135 for m_3). Thus, it would be difficult to obtain sufficient data for re-training the model to capture the anomalies in time.

To verify the distribution drifts in this data, we employed Maximum Mean Discrepancy (MMD) [**?**] with a Gaussian RBF kernel to measure the distance between training and testing sets across pairs of machines. To make the permutation test more efficient, the metrics were first projected to 5-dim by PCA, and p-value used for measuring the significance of the permutation test is 0.05. The MMD results indicate that all training and testing pairs have distribution drifts, even for data from the same machine, where the distribution can be varying over time. In addition, for each of the 38 metrics, we conducted Kolmogorov-Smirnov (KS) tests [**?**] between training and testing sets across pair-wise machines, with the p-value being 0.05. Table II summarizes the percentage

[1]https://github.com/NetManAIOps/OmniAnomaly

TABLE I: Data size and anormalies in SMD.

Machine	Train Size	Test Size	Anomaly (%)	First Anomaly
machine-1-1 (m_1)	28,479	28,479	4.73	15,849
machine-2-1 (m_2)	23,693	23,694	2.47	6,506
machine-3-2 (m_3)	23,702	23,703	2.34	135
machine-3-7 (m_4)	28,705	28,705	0.76	10,609

TABLE II: Percentage of metrics with drifts in SMD.

Training Set	Testing Set			
	m_1	m_2	m_3	m_4
m_1	0.763	0.816	0.789	0.816
m_2	0.816	0.684	0.763	0.763
m_3	0.789	0.816	0.632	0.816
m_4	0.816	0.737	0.737	0.684

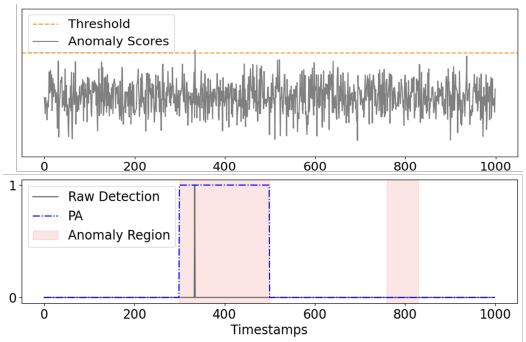
among all metrics with significant drifts. From each row, we found the training and testing data from the same machine generally have the least percentage of metrics with drifts.

2) Experimental Settings: Based on SMD, we designed extensive experiments to evaluate the *SAM* framework when migrating anomaly detection models across different server machines. The experimental settings are detailed as follows.

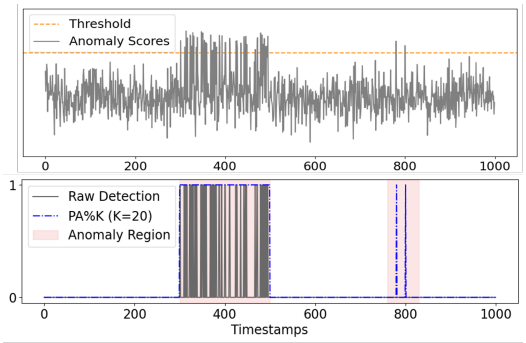
• *Baselines*: Given the four server machines in SMD, we consider two different scenarios – when a model has been trained from a certain machine, applying it to test data either 1) *Within* the same machine; or 2) *Across* different machines.

For the scenario *Within* the same machine (*e.g.*, training and testing data are both collected from m_1), since the testing set was collected posterior to training set and subtle changes might have happened along the way, there could be some drifts happening, as indicated in the MMD test. However, there would still be some patterns in the testing data shared with the training data, considering the temporal property of time-series. For example, the latter part of the training data would be highly informative for judging whether the earlier part in the testing data is normal or not. Hence, we treated this scenario as a benchmark (denoted as **Within**) and expected that migrating the model to another machine could in the best case achieve comparable performance with it. Such *Within* benchmark would be generated for each machine, thus we have 4 settings in this scenario.

For the scenario *Across* different machines (*e.g.*, training over m_1 while testing over m_2/m_3/m_4), as indicated by KS tests in Table II, there would be greater drifts compared to *Within*, thus it would be more challenging to achieve satisfactory model performance. Taking each machine as a target to be migrated to, we treated each of the other three machines as well as their combination as the training data, respectively. For example, when testing over m_1, we separately trained the model over m_2, m_3, m_4, together with the concatenated $m_{2,3,4}$. Therefore, there are 16 different train-test settings. We then compared two ways of migrating the models *Across* different machines, each with these 16 settings: 1) *Direct* migration of the model learned from the training data; and 2) Employing *SAM* for subseries cluster



(a) PA



(b) PA%K

Fig. 4: PA vs. PA%K

learning to capture different distributions, augmenting the data per distribution, treating each distribution as a domain input for the meta-learning domain generalization, and then migrating the generalized model to the target machine.

To sum up, there were in total 36 settings, with 4 for the *Within* scenario, and 16 for the *Direct* and *SAM*, respectively.

• *Evaluation*: Existing anomaly detection approaches generally employ F1 score for evaluation [**?**], [**?**], [**?**]. F1 is calculated as a harmonic mean between precision and recall, where precision quantifies what proportion of data identified to be anomalies are actually abnormal, and recall indicates what proportion of truly abnormal data can be correctly found. In time-series data, it is common that the anomalies will be lasting over a period of time [**?**]. As a result, prior works usually measure F1 after applying a Point Adjustment (PA) protocol [**?**]. PA is illustrated in Fig. 4(a) – If at least one point in a contiguous anomaly region is detected, then the entire region is considered to be accurately detected, and the prediction over the region will be adjusted accordingly.

Recently, Kim et al. demonstrated PA has a high potential to overestimate the model performance [**?**]. Specifically, it can simultaneously increase true positive and decrease false negative, while maintaining false positive. Hence, precision, recall, and consequently F1 score will always increase after PA. Taking Fig. 4(a) as an example – If the anomaly scores were randomly generated, and there was one point within the anomaly region that happened to be detected "correctly", then the whole region would be adjusted to be true positive, and such random guess would also get decent evaluation results. To address this issue, [**?**] suggested using F1 with a

PA%K protocol (**F1_PA%K**) instead. As shown in Fig. 4(b), only when the ratio of correctly detected anomalies within the anomaly region exceeds a certain threshold of K%, the PA will be carried out. Within the right-side of the anomaly region, since there are merely two points detected, the labels will not be adjusted. Herein, we set K as 20 by default.

In addition to the F1_PA%K, we also employed **AUC** and **Range-AUC** for evaluation. AUC measures the area under ROC curve, which is a plot for true positive vs. false positive at different detection thresholds. Higher AUC indicates more accurate model, with higher true positive rate and lower false positive rate. The range-AUC is adapted from AUC [**?**], considering the regional property of anomalies. When calculating precision and recall for an anomaly region, it considers (i) *existence*: whether there exist detected anomalies in the region; (ii) *size*: a larger size of correctly predicted portion of anomalies within the region is preferred; (iii) *position*: the relative position for the correctly predicted anomalies is usually the earlier the better; and (iv) *cardinality*: detecting the anomalies as a single prediction range can be more valuable compared to multiple fragmented ranges.

• *Anomaly Detectors*: Though we mainly focus on migrating anomaly detectors for multi-cloud, note that SAM can be readily adapted to other AIOps models. The anomaly detection is one of the fundamental IT tasks, with new approaches continuously being proposed (*e.g.*, GPT4TS [**?**], TimesNet [**?**], FEDformer [**?**], Autoformer [**?**], Transformer [**?**], DLinear [**?**], PatchTST [**?**], MICN [**?**]) in past years. Different approaches were compared in a recently built Time Series Library[2], where it is suggested that the Top-3 approaches are TimesNet, FEDformer, and Autoformer [**?**]. However, it is not clear whether some state-of-the-art methods were included in this ranking, and it was based on precision, recall, and F1 score with PA.

Utilizing the more rigorous evaluation measurements (*i.e.*, F1_PA%K, AUC, and Range-AUC), we compared all the above mentioned state-of-the-art methods together with a Random baseline (LSTM with random parameters [**?**]) under 36 settings (discussed in the earlier section) with SMD data. Fig. 5 shows box plots for the rankings among these methods. The smaller the ranking, the better the performance of the method is. We can clearly see that TimesNet is undoubtedly the Top-1, which agrees with the finding from the Time Series Library. It is the only method without any overlapping interquartile range (IQR) compared to the Random. According to the median of rankings, the other two approaches in the Top-3 are PatchTST and GPT4TS. Therefore, in this paper, we mainly focus on migrating **TimesNet**, as well as **PatchTST** and **GPT4TS**.

• *Parameter Settings*: To determine the optimal number of subseries clusters in GMM, we employed the Bayesian information criterion (BIC) [**?**], which aims at maximizing the likelihood of observing the data, subject to a penalty term to avoid complex models with a larger number of clusters to control the overfitting. Across different server machines, we found BIC usually converged with 4-6 clusters. Therefore, we set the cluster number default as 5 in all experimental settings. In meta-learning domain generalization, we followed the default settings described in [**?**], with the size of one-step update, *i.e.*, α, being 5e-1, the factor for balancing the meta-train loss and meta-test loss, *i.e.*, β being 1, and the learning rate γ being 1e-4. Other parameters involved in anomaly detection followed the default settings in the Time Series Library [**?**]. We utilized a Nvidia Tesla P100 with 16GB GPU memory. The training took less than 5 minutes of processing and less than 10 seconds for testing.

[2]https://github.com/thuml/Time-Series-Library?tab=readme-ov-file

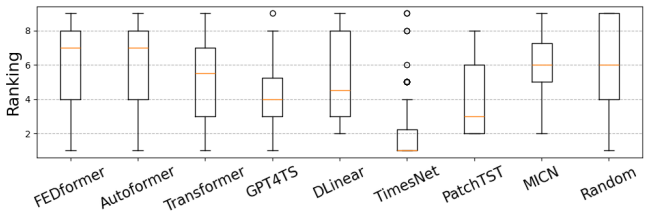


Fig. 5: Ranking of the AD methods (The smaller the better).

3) Results: Table III and Fig. 6 show the results over the SMD data. Table III focuses on the Top-1 anomaly detector (*i.e.*, TimesNet) while Fig. 6 covers more comprehensive comparisons for all Top-3 (*i.e.*, TimesNet, PatchTST, and GPT4TS). For each method, there were 36 settings (with different training and testing data) compared. Regarding the two migration manners in the scenario of *Across* different machines (*i.e.*, *Direct* and *SAM*), we calculated the mean(std) for each evaluation measurement learned based on the four training data (*i.e.*, training by the other three machines respectively and their concatenation), which is shown in bottom row of *Direct* and *SAM* cells in each sub-table of Table III. In Fig. 6, the bar charts for the mean(std) of *Direct* and *SAM* are compared to the *Within*.

Note that in real-world scenarios the target data (*i.e.*, where the model is migrating to) can be fully unobservable, and hence, *Across* would be a more realistic while challenging task compared to *Within*. When evaluating the results, we took *Within* as the benchmark, expecting that *our SAM (learned from data more likely to face drifts than Within) performs comparable to Within, while better than Direct*.

• *SAM vs. Direct*: From Table III, we can see in the majority of settings that *SAM* outperformed *Direct*. Among the four sub-tables, the average F1_PA%K, AUC, and Range_AUC for *Direct* are 0.459, 0.717, and 0.743, respectively, while *SAM* achieves 0.580, 0.791, and 0.808, with clear improvements of 0.121, 0.074, and 0.065 compared to *Direct*. Another observation is that *SAM* has much smaller std compared to *Direct*. For *Direct*, the average std across the four sub-tables for F1_PA%K, AUC, and Range_AUC are 0.068, 0.075, and 0.077, respectively, while *SAM* has the average std of 0.008, 0.011, and 0.011, which indicates a significantly more stable performance compared to *Direct*.

Among the 16 settings in *Across* (*i.e.*, taking each machine for testing, while every other machine and their combination for training), there are 14 out of the 16 settings

TABLE III: Results over SMD (with Top-1 anomaly detector, *i.e.*, TimesNet). The best performing model is highlighted in bold face under the *Across* configuration.

(a) Testing over $\mathbf{m_1}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	m_1	0.852	0.956	0.943
Across	Direct	m_2	0.649	0.854	0.879
		m_3	0.635	0.823	0.843
		m_4	0.782	0.888	0.919
		$m_{2,3,4}$	0.616	0.796	0.794
		mean(std)	0.670(.065)	0.840(.034)	0.859(.046)
	SAM	m_2	0.852	0.943	0.938
		m_3	0.833	0.943	0.940
		m_4	0.850	0.948	0.940
		$m_{2,3,4}$	0.843	0.942	0.937
		mean(std)	**0.844(.007)**	**0.944(.002)**	**0.939(.001)**

(b) Testing over $\mathbf{m_2}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	m_2	0.427	0.745	0.805
Across	Direct	m_1	0.213	0.495	0.536
		m_3	0.365	0.723	0.783
		m_4	0.391	0.708	0.768
		$m_{1,3,4}$	0.378	0.723	0.788
		mean(std)	0.337(.072)	0.662(.097)	0.719(.106)
	SAM	m_1	0.419	0.743	0.804
		m_3	0.419	0.727	0.787
		m_4	0.423	0.751	0.814
		$m_{1,3,4}$	0.429	0.735	0.796
		mean(std)	**0.422(.004)**	**0.739(.009)**	**0.800(.010)**

(c) Testing over $\mathbf{m_3}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	m_3	0.249	0.562	0.543
Across	Direct	m_1	0.304	0.578	0.582
		m_2	0.217	0.497	0.493
		m_4	0.398	0.705	0.704
		$m_{1,2,4}$	0.253	0.563	0.566
		mean(std)	**0.293(.068)**	**0.586(.075)**	**0.586(.076)**
	SAM	m_1	0.260	0.566	0.550
		m_2	0.263	0.571	0.552
		m_4	0.259	0.561	0.549
		$m_{1,2,4}$	0.279	0.621	0.602
		mean(std)	0.265(.008)	0.580(0.024)	0.563(.022)

(d) Testing over $\mathbf{m_4}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	m_4	0.791	0.920	0.943
Across	Direct	m_1	0.358	0.616	0.676
		m_2	0.579	0.826	0.842
		m_3	0.578	0.842	0.869
		$m_{1,2,3}$	0.622	0.836	0.855
		mean(std)	0.534(.065)	0.780(.095)	0.810(.078)
	SAM	m_1	0.790	0.900	0.932
		m_2	0.776	0.915	0.945
		m_3	0.807	0.895	0.927
		$m_{1,2,3}$	0.776	0.889	0.917
		mean(std)	**0.787(.013)**	**0.900(.010)**	**0.930(.010)**

that *SAM* outperformed *Across* for F1_PA%K, AUC, and Range_AUC, respectively. The only two exceptions happen when migrating the model from m_1 to m_3 and from m_4 to m_3. Comparing across the four sub-tables, in general the performance when taking m_3 for testing (*i.e.*, results in Table III(c)) is worse. Even when considering the *Within* scenario (*i.e.*, the training data is also collected from m_3), the performance is not good, with the F1_PA%K, AUC, and Range_AUC being 0.249, 0.562, and 0.543, respectively. It indicates that there might be some drastic drifts occurring in the testing data of m_3, with some patterns that happen to be captured by m_1 and m_4. However, when taking m_2 for training in *Direct*, the luck is not repeated and the performance turns out to be much worse. Furthermore, Table III(c) indicates that *SAM* using the training data of $m_{1,2,4}$ outperforms m_1, m_2, and m_4. Hence, broader sources of training data is helpful to cover more diverse distributions, from which *SAM* can learn more generalized patterns.

Similar findings can be drawn from Fig. 6, considering the Top-3 anomaly detectors. Again, *SAM* in general performs better compared to *Direct*, especially with higher mean values when taking m_1 and m_4 for testing across the three models. The exceptions happen when taking m_3 for testing. However, in all these exceptions, the differences between the mean value of *Direct* and *SAM* are negligible; while the *Direct* has much larger std compared to *SAM*. In real-world cases, where it is not always possible to acquire data with a similar distribution to the unobservable target before migration, a more stable model with decent performance, such as *SAM*, would be preferred.

• *SAM vs. Within*: *SAM* is expected to achieve comparable performance to *Within*, in which both the training and testing data are collected from an identical machine. Due to the homology between training and testing data, *Within* may have already observed some disclosed distribution patterns from the testing data beforehand. In *SAM*, the target distribution is fully unobservable, and there is no guarantee that the available data has any similar patterns to the test data. So when taking *Within* as the benchmark, it will be particularly impressive if *SAM* can obtain similar or even better performance compared to *Within*.

According to Table III, *SAM* generally has similar results to *Within*. Among four sub-tables, the average F1_PA%K, AUC, and Range_AUC for *Within* are 0.580, 0.796, and 0.809, respectively, while *SAM* achieves 0.580, 0.791, and 0.808, with the differences being merely 0.000, 0.005, and 0.001 compared to *Within*. Besides, *SAM*'s low std indicates stable performance across different scenarios. In a few scenarios when the testing data may have significant drifts, like when testing over m_3 as we discussed in a prior section, *SAM* can even get slightly better performance compared to *Within* by capturing more generalized patterns among the

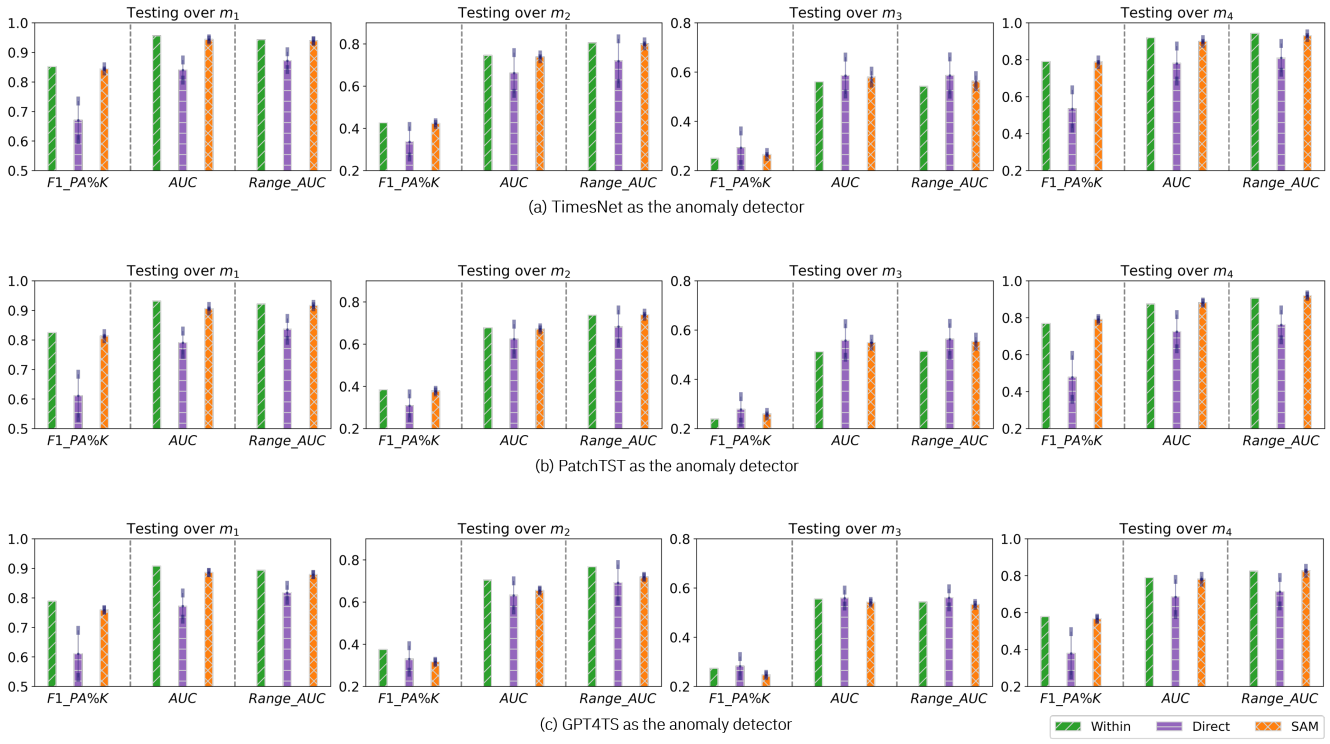


Fig. 6: Results over SMD (with Top-3 anomaly detectors, *i.e.*, TimesNet, PatchTST, and GPT4TS).

different distributions from the training data. From Fig. 6, we get similar findings. It can be readily seen in the figure that in the majority of cases *SAM* has similar heights to *Within* (with small stds) across different evaluation measurements under varying settings for all three anomaly detectors.

B. Our Dataset

1) Data Description: We further validated *SAM* over our own generated dataset that mimics real-world application migration from a source system to a target system with significant differences in the system specification. Herein, we used a modified version of Instana's Robot Shop application [**?**] deployed on two Kubernetes systems. This application simulates the process of buying a robot from an online store, which involves 13 services and multiple dependencies between them. Each Kubernetes cluster system ran on virtual machines from IBM Cloud. The first system (*i.e.*, s_1) was configured with five machines with 8 vCPUs, 32 GB of memory, and 600 GB of storage; The second one (*i.e.*, s_2) was configured with ten machines with 8 vCPUs, 8 GB of memory, and 600 GB of storage. For each system, a single control node is used and all nodes are configured as worker nodes. To ensure the scalability, KEDA autoscaler was used to decide the number of pods created for each service. Instana agents were deployed over both systems to monitor the application and collect the metric data.

We ran the application for both systems for 24 hours while changing the workload (*i.e.*, changing request arrival distribution) on the application and collected the metric data through Instana agents. For the second 12 hours, we injected periodic faults in 30 minutes intervals for a service which mimics HTTP 500 server not available error codes [**?**]. We considered 4 metrics of calls per second, erroneous calls per second, average occurrence of errors, and end to end latency for each of the 13 services. Thus, we have 52 metrics in total. The anomaly detectors can be trained over these metrics to understand when the application is not running efficiently. In our dataset, we have access to the ground truth labels for start and end of anomalies through the fault injection logs. The data size for the two systems, as well as their respective anomaly rate and the timestamp when the first anomaly occurs in testing data are detailed in Table IV.

2) Experimental Settings: We designed various experimental settings to evaluate *SAM* when migrating anomaly

TABLE IV: Data size and anormalies in our dataset.

System	Train Size	Test Size	Anomaly (%)	First Anomaly
`system-1` (s_1)	1,200	1,200	17.67	371
`system-2` (s_2)	1,200	1,200	17.58	365

TABLE V: Percentage of metrics with drifts in our dataset.

Training Set	Testing Set	
	s_1	s_2
s_1	0.196	0.536
s_2	0.714	0.589

detection models across systems detailed as follows.

• *Baselines*: Given the two systems in our dataset, we consider two different scenarios, *i.e.*, after training a model from one system, applying it to test either 1) *Within* the same system; or 2) *Across* different systems.

For the scenario *Within* the same system, when conducting MMD tests, we found both s_1 and s_2 do not have significant drifts between their training and testing data. Thus, **Within** is expected to perform well by knowing more prior knowledge about the test distribution, and we took it as a benchmark and expected *SAM* to achieve comparable performance. We generated benchmark from *Within* for each of the 2 systems.

For the scenario *Across* different systems, as indicated by KS tests shown in Table V, the drifts are greater compared to *Within* (higher percentage of metrics have significant drifts when training and testing sets are different). Thus, it would be more challenging to achieve decent model performance in *Across*. Taking each system as a target to be migrated to, we treated the other system as the training data. Therefore, we have two different train-test settings, i.e, from s_1 to s_2 and from s_2 to s_1. We then compared two ways, *i.e.*, **Direct** migration of the model or employing **SAM** to learn more generalized model for migration, across different systems. To sum up, there were in total 6 settings, with 2 for the *Within*, and 2 for the *Direct* and *SAM*, respectively.

• *Evaluation & Anomaly Detectors*: Based on our analysis over SMD in Section IV-A2, herein, we utilized the same measurements of **F1_PA%K**, **AUC**, and **Range-AUC** for evaluating the performance of the selected Top-3 anomaly detectors, *i.e.*, **TimesNet**, **PatchTST**, and **GPT4TS**.

• *Parameters Settings*: Both systems have converged BIC with 4-6 cluster, thus we set the GMM cluster number as 5 in all experimental settings. The size of one-step update, (*i.e.*, α), the harmonic coefficient between meta-train and meta-test losses, (*i.e.*, β), and the learning rate (*i.e.*, γ) were set by default as 5e-1 and 1, respectively. The parameters in anomaly detectors are the same as [**?**]. A Nvidia Tesla P100 with 16GB GPU memory was employed in our experiments. The training is done $<$2 minutes and testing $<$5 seconds.

3) Results: Similar to SMD, in Table VI, we focused on the Top-1 anomaly detector (*i.e.*, TimesNet), and in Fig. 7, more comprehensive comparisons were carried out for all Top-3 (*i.e.*, TimesNet, PatchTST, and GPT4TS). For each method, there were 6 settings (with different data for training and testing) compared. Moreover, we took *Within* as the benchmark, and the expectation is for *SAM* performance to be comparable to *Within* and better than *Direct*.

• *SAM vs. Direct*: Table VI shows that in all settings, *SAM* outperformed *Direct*. Among the two sub-tables, average F1_PA%K, AUC, and Range_AUC for *Direct* are 0.620, 0.511, and 0.732, respectively, while *SAM* achieves 0.624, 0.607, and 0.818, with comparable F1_PA%, and the improvements for AUC and Range_AUC are 0.096 and 0.086.

Similar findings can be learned from Fig. 7. In general, *SAM* performs better than *Direct*. The average F1_PA%K, AUC, and Range_AUC for *Direct* under different settings for all three methods are 0.623, 0.510, and 0.724, respectively.

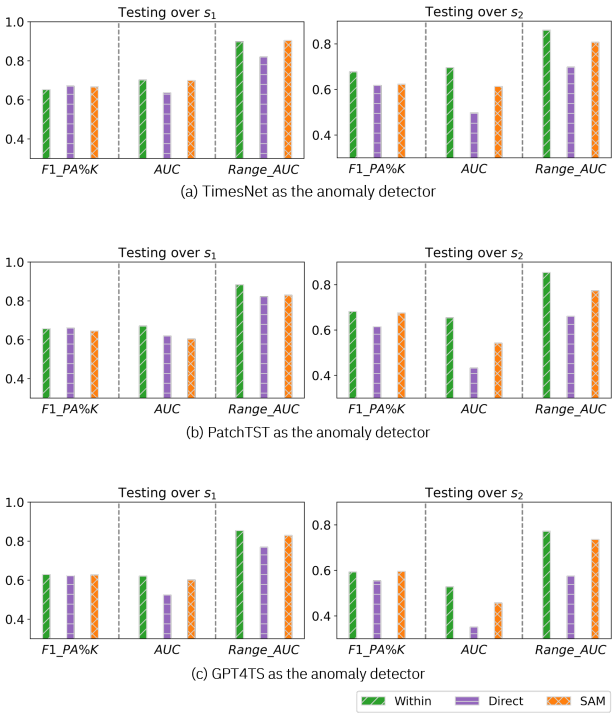


Fig. 7: Results over our dataset (with Top-3 anomaly detectors, *i.e.*, TimesNet, PatchTST, and GPT4TS).

SAM achieves 0.628, 0.586, and 0.813, with comparable F1_PA%K while AUC and Range_AUC improved by 0.076 and 0.089. Though there are few exception (*i.e.*, F1_PA%K when testing over s_1 for TimesNet and PatchTST, and AUC when testing over s_1 for PatchTST), *SAM* and *Direct* are quite close in these cases, with the differences in these exceptions being merely 0.004, 0.015, and 0.016. In general, *SAM* performs better than *Direct* in most of the settings.

• *SAM vs. Within*: Our expectation is that *SAM* achieves comparable performance to *Within*. According to Table VI, *SAM* generally performs similarly to *Within*. In particular, the average improvements of *SAM* over *Direct* for AUC and Range_AU, *i.e.*, 0.076 and 0.089, are higher than its gaps to *Within*, *i.e.*, 0.051 and 0.039. As indicated by MMD tests, the training and testing data from both s_1 and s_2 do not have significant distribution drifts. It means more information regarding the testing distributions is disclosed beforehand in the training data for *Within*, making its performance even harder to surpass. Taking all these into consideration, *SAM*'s comparable performance relative to *Within* is very promising and on par with our expectations.

V. Related Work

A. Time-series Augmentation

Machine learning models, especially deep learning models, usually rely on large amounts of data to achieve good performance. However, the data collection is usually expensive, consuming lots of time, labor, and resources. Additionally, in real-world applications, the data access is commonly

TABLE VI: Results over our dataset (with Top-1 anomaly detector, *i.e.*, TimesNet). The best performing model is highlighted in bold face under the *Across* configuration.

(a) Testing over $\mathbf{s_1}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	s_1	0.629	0.620	0.854
Across	Direct	s_2	0.622	0.524	0.768
Across	SAM	s_2	**0.626**	**0.600**	**0.829**

(b) Testing over $\mathbf{s_2}$

	Method	Training	F1_PA%K	AUC	Range_AUC
	Within	s_2	0.677	0.696	0.860
Across	Direct	s_1	0.618	0.498	0.697
Across	SAM	s_1	**0.622**	**0.613**	**0.807**

restricted and it is often not feasible to acquire sufficient data for model training purpose. To alleviate this issue, data augmentation techniques have been developed [**?**].

Recently, some data augmentation approaches specifically for time-series data have been proposed, for improving the performance in time-series classification [**?**], [**?**], forecasting [**?**], [**?**], and anomaly detection [**?**], [**?**]. In [**?**], the augmentation was conducted for anomaly detection with the purpose of generating more labeled data and handling the data scarcity and imbalance issue [**?**]. It is different from our work as their augmentation was mainly for anomalies, by modeling the anomaly detection as a supervised classification problem. Since in real-world scenarios, it is usually expensive to tag anomalies, in our work, we consider a more general case using semi-supervised learning for detection, not requiring any labeled anomalies, and augment the time-series from normal data instead. In another work [**?**], time-series augmentation was conducted for normal data, using the methods including zooming, adding random trend, reversing series, applying a random linear operation, random mutation between multiple series, etc. Though it can boost the size of time-series, it does not capture the varying distributions from the data, which is critical for the downstream meta-learning generalization task in our case.

B. Meta-learning

Meta-learning introduces a paradigm wherein a machine learning model accumulates experience across multiple learning episodes, encompassing a distribution of related tasks. It leverages experience to bolster the future learning performance [**?**]. This "learning-to-learn" concept offers a host of advantages, including enhanced data and compute efficiency, while also bearing similarity to learning strategies observed in human and animal learning where improvements occur over both lifetime and evolutionary timescales. Unlike conventional AI approaches, where tasks are tackled from scratch using a fixed learning algorithm, meta-learning focuses on enhancing the learning algorithm itself through the insights gained from multiple learning episodes.

Both domain generalization and domain adaptation approaches address situations where the source and target problems share the same objective, while the data distribution of the target task differs from that of the source task [**?**]. For instance, works like [**?**], [**?**] adopt domain adaptation methods to achieve a shared representation across different domains. This can be formulated as a k-shot learning problem, involving k-shot observations from the target cloud. However, this approach requires data collection before model training, leading to delays in model readiness. Additionally, it usually involves pair-wise adaptation, necessitating retraining of the model when adapting to another target cloud. An alternative method is domain generalization [**?**], which focuses on training a robust model using the source domain while assuming the unavailability of the target domain during the training process. This can be seen as a zero-shot learning problem.

To improve the model generalization, meta-learning methods has been utilized to perform both domain generalization and adaptation [**?**]. Combined with domain adaptation, Finn et al. proposed a model-agnostic meta-learning [**?**] for fast adaptation of deep networks. It takes meta-learning for few-shot learning by training a single model on a set of source tasks that is only a few gradient descent steps away from a good task-specific model. More recently, Li et al. proposed a meta-learning domain generalization (MLDG) [**?**], which is a model-agnostic approach for zero-shot case that is robust to domain shift. The power of MLDG not only lies in being domain-agnostic, but also being model-agnostic, which can be generalized to different tasks, feasible for supervised/unsupervised learning as well as reinforcement learning. Though some time-series foundation models have been proposed recently for zero-shot learning [**?**], they are mainly designed for forecasting or anomaly detection tasks, not being model-agnostic. Our experiments also demonstrated that our SAM framework with MLDG can further improve the performance of such models, *e.g.*, GPT4TS.

VI. Conclusions

In this paper, we present a novel SAM framework for handling the challenge in multi-cloud migration for AIOps models across different cloud providers. To achieve this goal, SAM integrates both data generalization through augmentation from subseries clusters and model generalization using meta-learning techniques. Through the synergy of data and model generalization, we foster the development of more robust AIOps models capable of seamless migration to the target cloud, avoiding the re-training of the models which usually relies on expensive data collection. The effectiveness of SAM was validated through extensive experiments on both public and simulated datasets. The decent and stable performance over different anomaly detectors shed some light on the potential for adapting SAM to other AIOps models that are used to automate IT tasks (*e.g.*, alerting and resource scaling).

VII. Acknowledgement

This work was partly funded by the SUNRISE-6G project (Grant #: 101139257), co-funded by the European Union and Smart Networks and Services Joint Undertaking (SNS JU).

References

[1] R. Jennings, *Cloud computing with the Windows Azure platform*. John Wiley & Sons, 2010.

[2] E. Di Nitto, M. A. A. da Silva, D. Ardagna, G. Casale, C. D. Craciun, N. Ferry, V. Muntes, and A. Solberg, "Supporting the development and operation of multi-cloud applications: The modaclouds approach," in *2013 15th international symposium on symbolic and numeric algorithms for scientific computing*, pp. 417–423, IEEE, 2013.

[3] H. A. Müller, L. F. Rivera, M. Jiménez, N. M. Villegas, G. Tamura, R. Akkiraju, I. Watts, and E. Erpenbach, "Proactive aiops through digital twins," in *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, 2021.

[4] S. Achar, "Enterprise saas workloads on new-generation infrastructure-as-code (iac) on multi-cloud platforms," *Global Disclosure of Economics and Business*, pp. 55–74, 2021.

[5] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[6] Z. Shen, J. Liu, Y. He, X. Zhang, R. Xu, H. Yu, and P. Cui, "Towards out-of-distribution generalization: A survey," *arXiv preprint arXiv:2108.13624*, 2021.

[7] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020.

[8] R. S. Peres, M. Guedes, F. Miranda, and J. Barata, "Simulation-based data augmentation for the quality inspection of structural adhesive with deep learning," *IEEE Access*, vol. 9, pp. 76532–76541, 2021.

[9] A. Castellani, S. Schmitt, and S. Squartini, "Real-world anomaly detection by using digital twin systems and weakly supervised learning," *IEEE Transactions on Industrial Informatics*, pp. 4733–4742, 2020.

[10] D. Li, Y. Yang, Y.-Z. Song, and T. Hospedales, "Learning to generalize: Meta-learning for domain generalization," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[11] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, pp. 5149–5169, 2021.

[12] R. Volpi, H. Namkoong, O. Sener, J. C. Duchi, V. Murino, and S. Savarese, "Generalizing to unseen domains via adversarial data augmentation," *Neurips*, 2018.

[13] T. Cao, J. Zhu, and G. Pang, "Anomaly detection under distribution shift," *arXiv preprint arXiv:2303.13845*, 2023.

[14] D. A. Reynolds *et al.*, "Gaussian mixture models.," *Encyclopedia of biometrics*, vol. 741, no. 659-663, 2009.

[15] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD*, 2019.

[16] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 841–850, IEEE, 2020.

[17] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: a comprehensive evaluation," *Proceedings of the VLDB Endowment*, vol. 15, no. 9, pp. 1779–1797, 2022.

[18] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *arXiv preprint arXiv:2201.07284*, 2022.

[19] E. Keogh, "Irrational exuberance why we should not believe 95% of papers on time series anomaly detection," in *7th SIGKDD workshop on mining and learning from time series at SIGKDD 2021. Workshop Keynote https://www. youtube. com/watch*, 2021.

[20] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[21] J. Hodges Jr, "The significance probability of the smirnov two-sample test," *Arkiv för matematik*, vol. 3, no. 5, pp. 469–486, 1958.

[22] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," in *Proceedings of the 26th ACM SIGKDD*, pp. 3395–3404, 2020.

[23] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich, "Precision and recall for time series," *Advances in neural information processing systems*, vol. 31, 2018.

[24] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, pp. 187–196, 2018.

[25] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7194–7201, 2022.

[26] T. Zhou, P. Niu, L. Sun, R. Jin, *et al.*, "One fits all: Power general time series analysis by pretrained lm," *Advances in neural information processing systems*, vol. 36, 2024.

[27] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "Timesnet: Temporal 2d-variation modeling for general time series analysis," in *The 11th international conference on learning representations*, 2022.

[28] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *International conference on machine learning*, pp. 27268–27286, PMLR, 2022.

[29] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *Advances in neural information processing systems*, vol. 34, pp. 22419–22430, 2021.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[31] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 11121–11128, 2023.

[32] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," *arXiv preprint arXiv:2211.14730*, 2022.

[33] H. Wang, J. Peng, F. Huang, J. Wang, J. Chen, and Y. Xiao, "Micn: Multi-scale local and global context modeling for long-term series forecasting," in *The Eleventh International Conference on Learning Representations*, 2022.

[34] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "Timesnet: Temporal 2d-variation modeling for general time series analysis," in *International Conference on Learning Representations*, 2023.

[35] A. A. Neath and J. E. Cavanaugh, "The bayesian information criterion: background, derivation, and applications," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 2, pp. 199–203, 2012.

[36] robot shop, "robot-shop." https://github.com/instana/robot-shop, accessed 2024-03-31.

[37] F. Bagehorn, J. Rios, S. Jha, R. Filepp, L. Shwartz, N. Abe, and X. Yang, "A fault injection platform for learning aiops models," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1–5, 2022.

[38] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, vol. 16, p. 100258, 2022.

[39] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Data augmentation using synthetic data for time series classification with deep residual networks," *arXiv preprint arXiv:1808.02455*, 2018.

[40] J. Gao, X. Song, Q. Wen, P. Wang, L. Sun, and H. Xu, "Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks," *arXiv preprint arXiv:2002.09545*, 2020.

[41] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International journal of forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[42] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," *arXiv preprint arXiv:1905.13628*, 2019.

[43] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 3068–3072, 2021.

[44] Z. Yang, I. Soltani, and E. Darve, "Anomaly detection with domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2957–2966, 2023.

[45] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.