Lessons Learned from Anomaly Detection in Chameleon Cloud

S. M. Qasim*, C. Hankendi*, M. Sherman†, K. Keahey†‡, G. Stringhini*, A. K. Coskun*
*Boston University, †University of Chicago, ‡Argonne National Lab

***Abstract*—Cloud computing has become integral to modern technology infrastructure, supporting a wide range of services from e-commerce to AI applications. Chameleon is a large-scale, configurable testbed designed to enable edge-to-cloud research through full bare-metal provisioning, virtualization, and diverse hardware resources, which is built on a leading open source cloud platform OpenStack. However, monitoring Chameleon's heterogeneous infrastructure is challenging, particularly across OpenStack services and hardware components. Traditional threshold-based alerting methods struggle to keep up with the scale and complexity of such environments. In this work, we present an anomaly detection framework for OpenStack services in the Chameleon Cloud. We curate and publish the first dataset of resource usage metrics collected from OpenStack control plane services. We evaluate four state-of-the-art unsupervised multivariate time series models, namely TranAD, Prodigy, USAD, and OmniAnomaly, on this dataset and share key insights from deploying them. Our findings indicate that for our use case, while all models achieve high F1 scores, training with three days of healthy data effectively balances training cost and detection accuracy.**

***Index Terms*—Cloud, Anomaly Detection, Machine Learning.**

I. Introduction

Cloud computing systems have become integral to modern technology infrastructure, supporting countless applications and services while driving significant business innovation and growth. These systems power everything from online retail [1] and ride-hailing services [2] to AI applications like ChatGPT [3]. Even brief outages can result in substantial losses for users running applications in the cloud. For instance, a brief outage of Amazon's S3 service in March 2017 is estimated to have cost $150 million to the S&P 500 companies [4]. Therefore, cloud system operators must continuously monitor both hardware and software for any anomalies or errors to ensure robust service delivery.

To enable research that enhances the reliability, efficiency, and scalability of such cloud systems, general-purpose experimental platforms like Chameleon [5] play a vital role. Chameleon is a configurable experimental environment for large-scale edge-to-cloud research. It enables computer science research through a deeply reconfigurable testbed that supports projects in operating systems, virtualization, power management, networking, and artificial intelligence. Chameleon offers bare-metal provisioning, giving researchers full control over the software stack, including root access and kernel customization. A smaller portion of the testbed is virtualized using Kernel-based Virtual Machine (KVM), supporting use cases that benefit from finer-grained resource sharing.

Since its launch in 2015, Chameleon has supported over 10,660 users and 1269 research projects. It continues to evolve with added hardware and capabilities, serving as a key testbed for advancing cloud and systems research [6]–[11]. Chameleon infrastructure includes various hardware, including nearly 15,000 cores and 5 petabytes of storage across affiliate sites and two major sites, the University of Chicago and the Texas Advanced Computing Center (TACC), connected via a 100 Gbps network. Hardware diversity includes GPUs, FPGAs, InfiniBand, reconfigurable switches, various storage types, and non-x86 processors, such as ARMs.

Chameleon uses OpenStack [12], a leading open-source cloud platform that continues to demonstrate its value through widespread global adoption and rapid growth. According to the 2022 User Survey [13], OpenStack now powers over 40 million compute cores across more than 300 public cloud data centers. Its strong support for hybrid cloud deployments and seamless Kubernetes integration (used by over 85% of deployments) make it a foundation of modern infrastructure. Backed by a vibrant community and a stable net promoter score (NPS) of 41, OpenStack remains a scalable, production-ready, and highly valuable open-source solution for cloud computing. Chameleon itself has contributed to improve OpenStack's Blazar project and made it an official top-level OpenStack component, further integrating research with mainstream infrastructure development.

However, monitoring Chameleon is challenging due to the heterogeneous nature of its components, each of which can fail in distinct ways [14]. For instance, bare-metal servers may signal hardware anomalies through their baseboard management controllers, such as voltage fluctuations or unplugged cables. OpenStack services may generate HTTP error codes or log diagnostic messages, while nodes running Chameleon services may encounter infrastructure-level issues like disk partitions reaching capacity or resource overutilization [15]. To address these challenges and develop a more comprehensive understanding of system stability, the Chameleon operations team relies on Prometheus's [16] built-in query language and custom database queries on log data to configure targeted alerts. However, manually setting and managing alerts quickly becomes challenging, as demonstrated by the outages documented on the Chameleon website [17], and is discussed in greater detail by Keahey et al. [15].

Recent cloud-monitoring approaches employ machine-learning models that learn normal system behavior and flag deviations as anomalies [18]–[23]. For instance, Islam et al.

describes using a GRU-based autoencoder for anomaly detection in IBM cloud and shows that it can detect anomalies up to 20 minutes earlier than their existing system [20]. However, training effective models is a challenging task, as metric distributions drift over time, necessitating frequent retraining. Training on very large datasets is computationally expensive and time-consuming, whereas training on small datasets risk inadequate accuracy. There is a need for more research in this area. Additionally, to the best of our knowledge, no publicly available dataset currently captures resource-usage metrics for OpenStack services. This gap makes it difficult to balance training-dataset size, model accuracy, and training time.

We address this limitation by releasing a comprehensive dataset of resource-usage metrics for every OpenStack service running on the control-plane node, which hosts all management and orchestration services of Chameleon Cloud at the University of Chicago, together with an anomaly-detection framework tailored to those services. In this paper, we present our deployment of state-of-the-art unsupervised multivariate time-series anomaly detection techniques, namely TranAD [24], Prodigy [25], USAD [26], and OmniAnomaly [27], on the Chameleon Cloud and discuss the key lessons learned from this process.

Our contributions and highlights are as follows:

- ⋆ We present the development of an anomaly detection framework for OpenStack services deployed on the Chameleon Cloud, leveraging resource usage metrics. Our approach is designed to be replicable across other OpenStack-based clouds.
- ⋆ We publicly release the dataset[1] used to train and test above machine-learning models. To the best of our knowledge, this is the first publicly available dataset comprising resource usage metrics for OpenStack services, facilitating broader benchmarking and evaluation of anomaly detection techniques in similar cloud platforms.
- ⋆ Lastly, we share some insights and observations from building this system, which will help others in making informed decisions when selecting models, their training cost and deployment planning. Some of our notable observations and findings include:
  - A relatively short training data window (on the order of a few days) can provide a pragmatic balance between model training cost and prediction accuracy, reducing compute overhead without sacrificing quality. We observed almost comparable F1 scores with 3 and 30 days of our training data. For example for TranAD we observed 0.978 vs 0.985 of weighted F1 score, and 0.957 vs 0.970 macro F1 score for 3 and 30 day respectively, at 10% threshold, which can reduce the training time by up to 8-9x (14 minutes vs. 125 minutes).
  - Relying solely on low-level resource metrics is effective for surfacing hardware or network problems but tends to miss application-layer faults—such as HTTP 500 errors, elevated request latency, or rate-limiting responses, highlighting the importance of incorporating higher-level signals for comprehensive anomaly detection.
  - We rank OpenStack services based on their resource usage we observed, which can support focused capacity planning and performance tuning of OpenStack based clouds.

The remainder of the paper is organized as follows. Section II reviews the related literature. Section III describes the observability infrastructure at Chameleon. Section IV details the major challenges we faced while building this framework. Section V discusses the proposed solution, our training and inference pipelines. Section VI describes the experiments, our evaluation methodology and results. Finally, Section VII concludes the paper.

II. Related Work

There is a significant amount of prior research on anomaly detection in cloud computing environments, driven by the increasing complexity, scale, and dynamic behavior of cloud infrastructures. This includes the development of OpenStack-native tools such as Rally [28] and Tempest [29], which are used for load testing and integration testing, respectively. Tools such as Ceilometer [30], Monasca [31], and Aodh [32] were once integral components of OpenStack's telemetry and monitoring stack. However, their adoption has declined in favor of Prometheus [16] and Alertmanager [33], especially in deployments using Kolla-Ansible [34].

Early approaches to anomaly detection emphasized statistical analysis and resource monitoring. Huang et al. [35] developed a relaxed linear programming variant (RLPSVDD) to improve computational efficiency and reduce false alarms when applied to time-series data from Yahoo's cloud performance logs. DriftInsight [36] advanced this paradigm by employing convergence-based state modeling and unsupervised clustering to dynamically detect anomalies in highly dynamic cloud environments. Wang et al. designed a self-adaptive cloud monitoring system based on correlation analysis and principal component analysis (PCA), which dynamically adjusts monitoring parameters based on estimated anomaly degrees to reduce overhead while maintaining accuracy [37]. While Roots [38] is a full-stack performance anomaly detection system for PaaS environments that correlates multi-layer performance data without requiring application-level instrumentation. Sauvanaud et al. implement a supervised learning-based anomaly detection system augmented with fault injection, enabling real-time diagnosis of SLA violations and service degradation [22]. Authors use Random Forests, Neural Nets and k-Nearest Neighbours and Naive Bayes to evaluate their system. PerfInsight introduces a robust clustering-based method for detecting abnormal behaviors in large-scale cloud systems using Mann-kendall test and DBSCAN for metric profiling [21]. Works such as Toslali et al. aim to find anomalous services and code by monitoring variance in response times in distributed system traces [39].

[1] https://github.com/ai4cloudops/Chameleon-Cloud-Anomaly-Detection

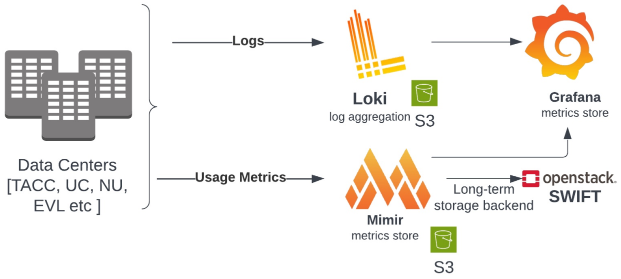

Fig. 1. Observability data is collected to a central observability cluster.

More recent works like that of Islam et al. presents a serverless GRU-based anomaly detection system for IBM Cloud's multi-dimensional telemetry [40]. The system reduces false alerts, supports online training, and improves detection accuracy by integrating DevOps feedback and automating real-time model retraining, but the work lacks specific information about the model and the metrics, which the authors published in a later work as Islam et al. [41], and share a high-dimensional anomaly detection dataset from the IBM Cloud's Console, comprising telemetry data from over 4.5 months. CloudShield is a real-time anomaly and attack detection system based on pretrained deep learning model, which identifies anomalies by statistically comparing the distribution of the unpredicted part denoted by Reconstruction Error Distribution (RED) [23]. In contrast to prior work, this paper specifically targets OpenStack, the leading open source cloud platform, and explores the trade-offs between training data duration, training cost, and anomaly detection accuracy. Furthermore, we publicly release the labeled dataset used in our evaluation to support reproducibility and future research.

III. Observability Infrastructure at Chameleon Cloud

Chameleon collects rich logs and metrics to support robust operations and enable efficient troubleshooting. Figure 1 illustrates the architecture of metrics and log flow within the Chameleon Cloud infrastructure. All observability data is centrally stored in a dedicated observability cluster hosted at TACC. Node-level performance metrics are collected using Prometheus, while container-level metrics are gathered via cAdvisor [42] and scraped every 15 seconds, providing high-resolution insight into resource usage such as CPU, memory, disk, and network. Chameleon, however, does not collect any resource usage metrics for hypervisors or compute hosts used by tenants, as it cannot run an agent due to bare-metal isolation. Additionally, Chameleon packages several stock Prometheus exporters that are not yet available in Kolla-Ansible [34] namely SNMP, IPMI, and Redfish. Also their packaging of the OpenStack ironic exporter has been merged upstream. Chameleon also generates several custom metrics using Loki's [43] recording rules, which can count log lines matching certain labels. For instance, they generate `node success rate`, by comparing `provisioning success` to `provisioning started` events, as stock Prometheus scraping of Ironic node states does not have a fine enough time resolution to capture all state transitions. All performance metrics are stored in Grafana Mimir [44], enabling scalable and long-term retention and querying.

Logs from all OpenStack services are sent to a standard location and automatically ingested into Loki using Fluentd [45]. Both logs and metrics can be queried manually through the Grafana [46] interface. Chameleon Cloud operators employ Grafana to set manual thresholds for key metrics such as disk usage, and for error logs like a large number of failed provisions that might indicate undetected faults, thereby supporting system monitoring and ongoing maintenance.

Chameleon Cloud also runs Jenkins [47] tests to validate key user workflows, such as node reservation, provisioning, public IP assignment, SSH access, and metric reporting. These tests run hourly or daily to detect system issues, uncover monitoring gaps, and correlate errors with potential underlying causes.

IV. Challenges in Developing an Anomaly Detection Framework for Cloud Systems

Building an anomaly detection system for OpenStack services poses unique challenges because of the sheer volume, high dimensionality, and heterogeneity of resource-usage metrics. Below are the main challenges we encountered:

A. Lack of Labeled Datasets

Chameleon Cloud operates an OpenStack deployment. Although OpenStack is a widely adopted cloud platform, to the best of our knowledge no public dataset offers labeled datasets of resource usage metrics for its services. While usage patterns vary across deployments, such datasets are essential for rigorously evaluating competing anomaly detection methods

B. Model Selection

The second challenge involves identifying suitable anomaly-detection models. Time-series techniques range from classical statistical tests to sophisticated machine-learning architectures. Choosing a technique that fits best to our data is critical for our system. We describe in Section VI, how we evaluate various models and choose the best performing one for our system.

C. Data Volume and Feature Representation

Most of the modern anomaly detection approaches infer normal behavior from historical observations before evaluating new data. However, determining the appropriate training horizon is non-trivial. Is one day of history sufficient, or is a full month required? We investigate this question by training and testing our models on windows of one, three, seven, and thirty days, thereby quantifying the trade-off between training data volume and detection accuracy.

V. Proposed Solution

Recent approaches to anomaly detection in cloud systems predominantly leverage machine learning [18]–[23], owing to their capacity to ingest large-scale telemetry, learn complex dependencies among metrics, and adapt to evolving workloads, therefore, we adopt a machine learning based strategy. Figure 2

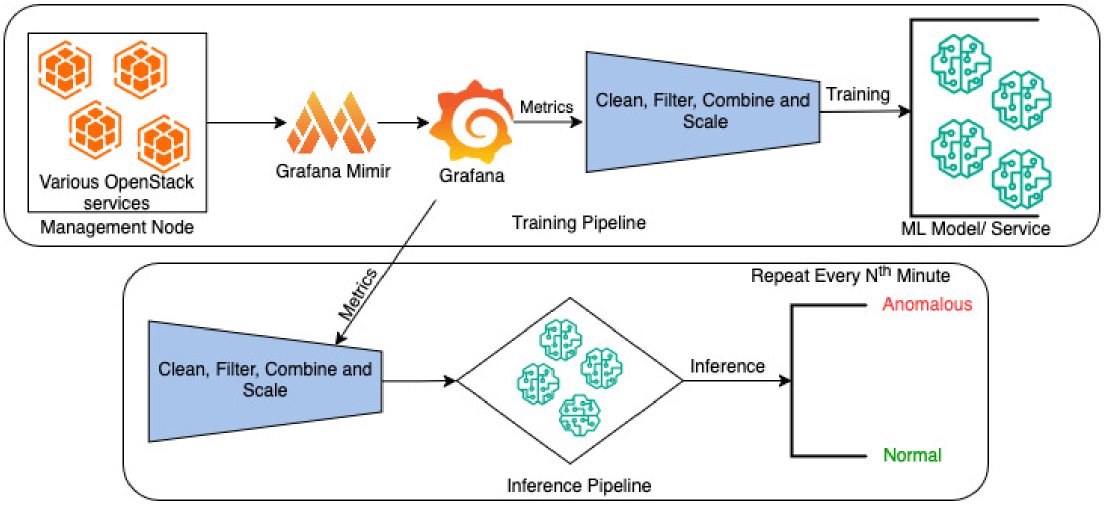

Fig. 2. Our model training and inference pipeline.

presents a high level overview of our system, which comprises two main components: (1) the Training Pipeline and (2) the Inference Pipeline. Each component supports model agnostic processing of multivariate time series data for anomaly detection.

A. Models Evaluated

Our data consists of multivariate time series, and initially, we did not have access to the labeled data necessary for using supervised models. Together, these factors lead us to pursue unsupervised multivariate time series anomaly detection techniques. Most recent unsupervised studies typically incorporate stochasticity and employ Variational Autoencoder (VAE) [48] or Transformer [49] architectures.

VAE is a deep generative model that encodes high dimensional input x into a probabilistic latent representation z. The encoder, or inference network $q_\phi(z|x)$, maps input data to a latent distribution, typically modeled as a Gaussian distribution, as shown below:

$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \text{diag}(\sigma^2_\phi(x)))$$

The decoder, or generative network $p_\theta(x|z)$, reconstructs the input from samples drawn from this latent distribution. A standard Gaussian prior is imposed over the latent variables:

$$p(z) = \mathcal{N}(0, I)$$

The model is trained by maximizing the Evidence Lower Bound (ELBO) on the marginal likelihood:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \| p(z))$$

Here, the first term encourages accurate reconstruction of the input, and the second term regularizes the latent distribution to be close to the prior. To enable backpropagation through stochastic nodes, the reparameterization technique is used:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Transformer is a deep learning model architecture designed for sequence-to-sequence tasks, based entirely on attention mechanisms [49]. Unlike Recurrent Neural Networks, it processes input sequences in parallel, making it highly efficient and scalable.

The core component of the Transformer model is the *self-attention* mechanism, which computes contextualized representations by attending to all positions in the sequence. Given an input sequence such as $X = [x_1, x_2, \ldots, x_n]$, each token is projected to a query (Q), key (K), and value (V) matrix:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

Self-attention computes attention weights using scaled dot-product:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

To enhance representation learning, Transformers use *multi-head attention*:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head is:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

The Transformer encoder consists of layers of multi-head self-attention and position-wise feedforward networks. Positional encodings are added to the input embeddings to preserve the order of input:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Each encoder or decoder layer includes residual connections and layer normalization to stabilize training.

We select some of the best-performing, open-source models namely TranAD [24], USAD [26], OmniAnomaly [27], and

Prodigy [25] as our top candidates. These models have demonstrated strong results on widely used benchmark datasets such as SMD [27], which has similar time series data such as ours, but it lacks the logical dimension of service.

All four methods are published recently in top-tier conferences, are highly cited, and have been extensively replicated, making them reliable, well-tested, and suitable for evaluation in our setting. They also share a common underlying principle: each model is trained to learn the distribution of normal system behavior. During inference, the model attempts to reconstruct the input data. If the reconstruction error or likelihood exceeds a certain threshold, the data is classified as anomalous; otherwise, it is considered healthy. These models are discussed in detail below.

OmniAnomaly [27] combines a Variational Autoencoder (VAE) with a Gated Recurrent Unit (GRU) [50] to capture both temporal dependencies and the stochastic nature of time series data. To enhance the flexibility of the posterior distributions, OmniAnomaly employs Planar Normalizing Flows [51] to transform simple Gaussian assumptions into complex non-Gaussian forms. Temporal connections among latent variables are modeled through a Linear Gaussian State Space Model (SSM), allowing the latent space to reflect historical dynamics more accurately.

Anomaly detection is performed based on the reconstruction probability of each input, and lower probabilities indicate higher chances of being anomalous. OmniAnomaly also introduces using the Peaks-Over-Threshold (POT) method from Extreme Value Theory for Threshold selection. Furthermore, OmniAnomaly supports interpretable anomaly detection by ranking individual dimensions of the input based on their reconstruction probabilities.

USAD [26] combines the architectural stability of Autoencoders with adversarial training dynamics inspired by Generative Adversarial Networks (GANs), USAD consists of a shared encoder and two decoders forming two autoencoders. The model is trained in two phases. In the first phase, both autoencoders learn to reconstruct normal time series windows. In the second phase, adversarial training is introduced where one autoencoder attempts to reconstruct original inputs, while the second attempts to distinguish whether its input came directly from the original data or from the first autoencoder. This design amplifies reconstruction errors in the presence of anomalies, making them easier to detect.

TranAD [24] consists of a Transformer encoder-decoder architecture used in two sequential passes. The first pass reconstructs the input time window, while the second uses the output of the first as input to produce a refined reconstruction. Adversarial training is incorporated via a critic network that helps the generator learn more discriminative reconstructions, enhancing the sensitivity to subtle anomalies. During training, TranAD uses a reconstruction loss and an adversarial loss to optimize the model. At inference time, it computes anomaly scores based on the difference between the input and its reconstructions from both passes. A higher discrepancy indicates a higher probability of anomaly.

Prodigy [25] is also an unsupervised anomaly detection framework designed for practical deployment in production High Performance Computing (HPC) systems. Prodigy aims to detect performance anomalies at the compute node level using multivariate telemetry time series data. It focuses on identifying anomalies that degrade performance without causing system failures—scenarios that are often subtle and difficult to detect.

During training, Prodigy assumes access to only healthy-labeled samples, which is a reasonable assumption for production systems where anomalies are infrequent. It uses statistical feature extraction (via TSFRESH [52]) and Chi-square feature selection [53] to derive the most informative features. These features are then used to train a VAE that learns the normal behavior of compute nodes. Anomalies are detected at inference time by comparing reconstruction error to a statistically determined threshold (e.g., 99th percentile of training errors). Prodigy also integrates counterfactual explainability through the CoMTE [54] method, which identifies the minimal set of metric changes required to reclassify an anomalous sample as healthy. This aids in root cause analysis and increases interpretability for HPC administrators.

B. Dataset Creation

To evaluate our system, we create a labeled dataset by collecting details of outages from the Chameleon Cloud outages page [17]. We write a script to parse the outage name, start and end times, and the reason for each outage.

Next, we query and process the relevant metric data, as described in the next section of Training Pipeline (Section V-C). We collect data for 14 outages affecting Chameleon site at the University of Chicago listed in Table II.

For each outage, we gather resource usage metric recorded during the outage, along with one month of same data preceding the outage start date. The data is organized into folders by outage, with each folder containing CSV files, one per service, combining all metrics indexed by timestamps. These files differ in number of columns, columns representing individual metric, and ranged anywhere from 30 to 3200 columns, with most files having around 43,200 (60x24x30) rows.

To label anomalous services, we first evaluate all services across all outages. We then manually verify and label based on whether the services flagged as anomalous exhibited metric patterns that deviated from normal behavior observed over the 30-day baseline period.

C. Training Pipeline

The management node in the Chameleon Cloud is responsible for running all OpenStack services required for managing and operating the site. Each service operates within its own container, and their performance metrics are stored in Grafana Mimir. These metrics are queried using Grafana's query language via the Grafana interface. We maintain a dictionary that maps each metric to its corresponding query. For example: *container_fs_writes_bytes_total:*

*rate(container_fs_writes_bytes_total{
hostname='mgmt01.uc.chameleoncloud.org'}[5m])*

This query filters the metric for a specific hostname and computes the per-second average rate of increase over a 5-minute window, which is appropriate for counter-type metrics. The time range for these queries can be adjusted as needed, allowing us to fetch metrics across all containers in a single query.

We develop Python scripts to parse and organize the retrieved data by device and service. For instance, the parsed results of the above metric are saved in a file named container_fs_writes_bytes_total.csv, a snippet of which is shown in Table I.

#	device	service_name	Timestamp	Value
1	/dev/dm-0	nova_serialproxy	2024-04-16 14:30:00	0

TABLE I
SAMPLE METRIC ROW IN FILE *container_fs_writes_bytes_total.csv*

After collecting all the cAdvisor metrics, we aggregate them on a per-service basis. The outage dataset we share consists of these final, service-wise combined metrics for each of the outages listed in Table II. The dataset contains the CSV for the outage and 30 days of data before the outage. We have manually analyzed each outage and marked the individual services as anomalous or healthy.

Finally, we train a separate machine learning model for each of the 65 services running in individual containers on the management node. For instance, to train a USAD model for the blazar_manager service, we first import its corresponding CSV file as a DataFrame. We then remove any columns where over 99% of the values are zeros, which almost always leaves us with fewer than 60 columns. After that, we scale all remaining features using a MinMaxScaler [55]. The model is then trained, and we save the following artifacts: the trained model, the scaler, the 90th and 99th percentile thresholds for the mean reconstruction error during training, and the names of the metrics used during training. These are later used during inference. We train each model according to its specific architecture and input requirements. For example, TranAD processes the 2D time-series data by converting it into overlapping sliding windows with a fixed window size of 10. It applies replication padding for early indices and preserves the 2D structure, resulting in a tensor of shape `[num_windows, window_size, num_features]`, which enables contextual input for anomaly detection.

In contrast, USAD requires a flattened 1D representation of each window, removing explicit temporal structure. We use a window size of 5 for this model, which they used for the SMD dataset. For Prodigy and OmniAnomaly, we train the models directly on raw time-series datapoints using the original feature vectors, simply removing the timestamp column without any windowing.

D. Inference Pipeline

During the inference process, the procedures for data collection and processing remain consistent. The only difference is that we adjust the query's time range to match the specific period we want to analyze. Once we obtain the test data in CSV format, we retrieve the stored artifacts, such as the trained model, scaler, and thresholds. We then load the test data and ask the model to reconstruct the input data. Any input window or data point (depending on the model) with a mean reconstruction error exceeding the 99th percentile is classified as anomalous; otherwise, it is considered healthy.

For our evaluation on Chameleon outages, the test data corresponds to the entire duration of the outage. In the case of online deployment, inference can be performed every hour. We run inference separately for each service and classify all the windows or data points as either healthy or anomalous. We then set a threshold on the percentage of test data that was marked as anomalous, to classify a service as anomalous, which is discussed in more detail in Section VI.

VI. EXPERIMENTS AND RESULTS

The primary goal of our experiments is to evaluate the models on the outage dataset and determine which one trains the fastest, detects the most anomalies and minimizes alert fatigue for the operations team.

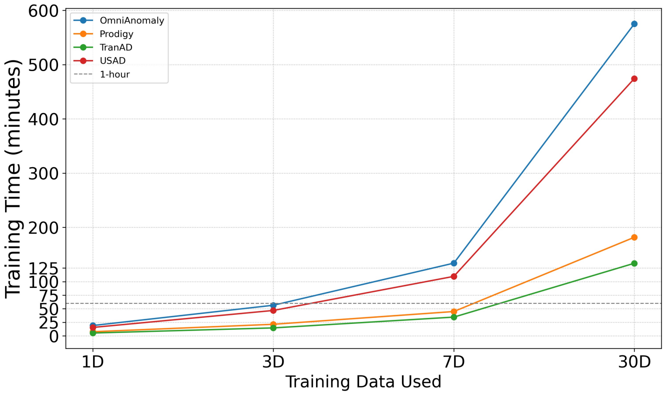

Fig. 4. Comparison of total training times for different models (65 models) with different number of days of healthy training data. TranAD trains almost 4.5x faster than OmniAnomaly.

We begin our experiments by evaluating the training times of each model using varying durations of healthy data: 1 day, 3 days, 7 days, and 30 days immediately preceding the outage for all 65 OpenStack services running on the control plane node. As shown in Figure 4, TranAD consistently exhibits the fastest training times. This efficiency, attributed to its transformer-based architecture, makes it a compelling choice for scenarios in which rapid model retraining is essential. In contrast, OmniAnomaly shows the longest training time, taking nearly ten hours to train 65 models on one month of data. We also note that the TranAD models trained on 30 days and 3 days of data take about 125 minutes and 14 minutes, respectively.

We then compare the storage footprint of the trained models. On average, TranAD models are approximately 10 times larger than the other methods, whereas Prodigy yields the smallest model sizes of around 150Kb.

Outage	Date	Explanation
Chincar Maintenance	Jan 26–29, 2024	The outage affected Chameleon deployment at NCAR, but not at UC. We marked this test dataset as normal.
Chameleon Portal, Chi-Edge, and JupyterHub	Jan 30–31, 2024	Caused by datacenter power maintenance. It affected the managed VM service, Chameleon portal, edge, and JupyterHub services. Chameleon at UC was unaffected.
CHI@UC Site Maintenance	Feb 06–08, 2024	Recabling and replacement of failing network hardware. All services were affected. All services in the dataset are marked as anomalous.
Partial Authentication Outage	Mar 06–07, 2024	Some users could not authenticate to CHI@UC, TACC, and Edge. Using a different browser resolved the issue. Dataset marked as normal.
CHI@UC Network Uplink Maintenance	Mar 11–12, 2024	Horizon dashboard and uplink traffic to UC were affected. **Horizon and Neutron L3 agent are labeled anomalous.**
KVM@TACC System Maintenance	Mar 27, 2024	Outage affected KVM at TACC. Chameleon at UC is marked as normal.
Planned Help Desk Outage	Apr 29, 2024	Ticketing system was affected. Chameleon at UC is marked as normal.
CHI-Edge Outage	May 16–17, 2024	Chameleon at Edge experienced issues. Chameleon at UC is marked as normal.
TACC Certificate Expiry	May 21, 2024	Automation failure in replacing certificates at TACC. CHI@UC is marked as normal.
CHI@UC Uplink Networking	May 29, 2024	Networking issue blocked access to Horizon dashboard and authentication server. **Blazar Manager, Fluentd, and Neutron L3 agent are labeled anomalous.**
CHI@UC OpenStack Upgrade	Jul 22–23, 2024	Version upgrade caused metrics to shift significantly. **All services are marked as anomalous.**
Smoke Test – Bare Metal Reserve Failed (No IPs)	Jan 22, 2025	Smoke test failed due to lack of public IPs. No major metric changes. No services labeled anomalous.
Smoke Test – Object Upload Failed	Feb 08, 2025	Smoke test failed to upload to object store. No significant metric change. No services labeled anomalous.
Smoke Test – Node Reserve Failed	Feb 19, 2025	Smoke test failed in reserving a node. No significant metric change. No services labeled anomalous.

TABLE II
SUMMARY OF OUTAGES AND THEIR IMPACT ON CHAMELEON SERVICES

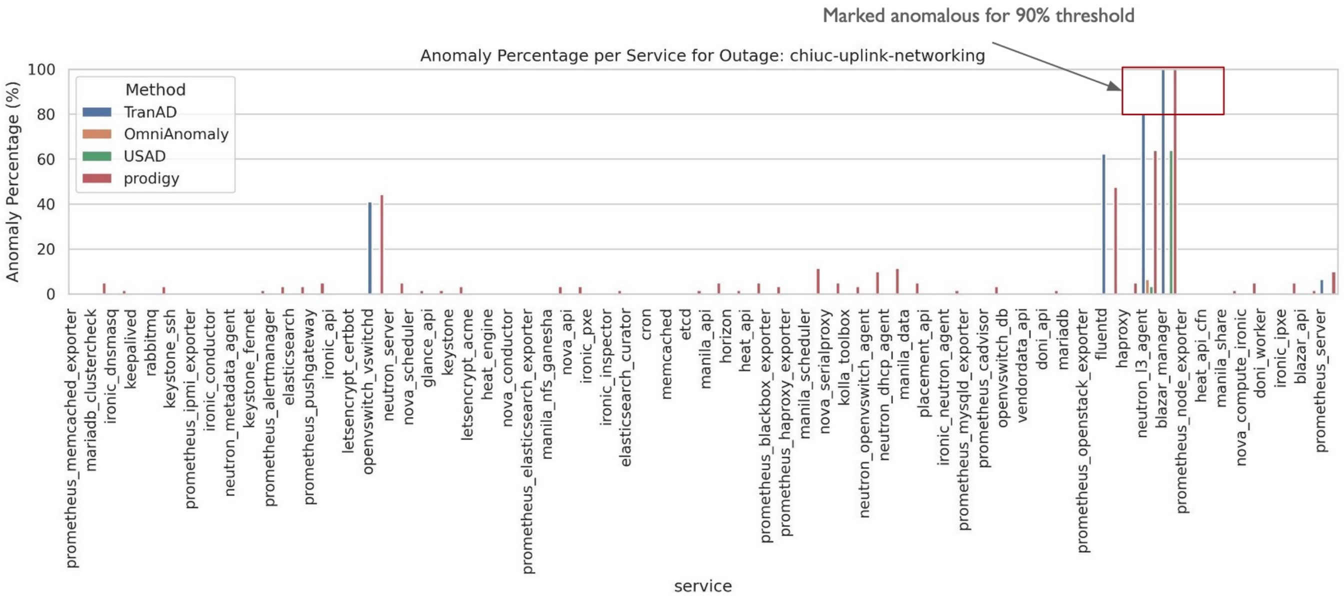

Fig. 3. Percentage of data in the outage marked as anomalous by different models. TranAD and Prodigy predict that almost 100% of Blazar manager data is anomalous, hence a 75% or 90% threshold would predict them as anomalous.

We next assess the models to determine which one outperforms the others in terms of overall performance in terms of the F1 scores. Figure 5 and Figure 6 present the weighted and macro F1 scores, respectively, for each model. Each line represents a different model trained using a specific duration of healthy data preceding the outage event. For instance,

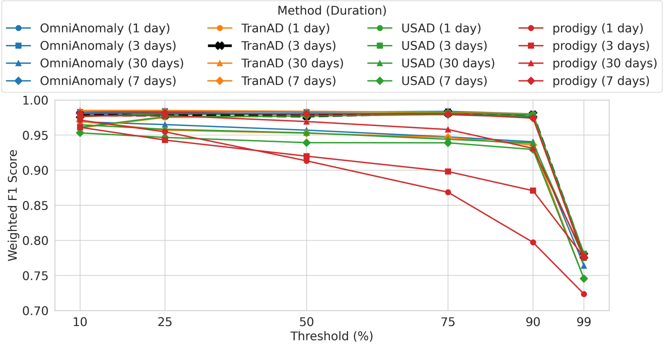

Fig. 5. Comparison of Weighted F1 scores for various models against different thresholds for marking a service anomalous. TranAD (3 days) provides almost the same F1 score as TranAD (30 days) using less training time.

OmniAnomaly (30 days) indicates that the OmniAnomaly model was trained using 30 days of healthy data preceding the outage. Similarly, USAD (7 days) refers to training the USAD model on 7 days of pre-outage healthy data. Each model is then used to generate the predictions for batches or data points based on its training configuration. We aggregate these predictions to compute the percentage of outage-period data identified as anomalous. For example, Figure 3 illustrates the proportion of data flagged as anomalous by each model for various services during the chiuc-uplink-networking outage.

For practical implementation of the model predictions, we define a threshold that represents the minimum percentage of data points that must be classified as anomalous for a service to be labeled as anomalous. For example, if a service has 1,000 data points during an outage and 800 of them exhibit reconstruction errors exceeding the 99th percentile threshold, then those 800 are classified as anomalous, resulting in an anomaly percentage of 80% for that service. This threshold serves as a tunable parameter, allowing us to adjust the model's sensitivity to anomalies.

We then compare the final binary classification whether a service is predicted as anomalous or healthy against our manually labeled outage dataset to compute the F1 scores.

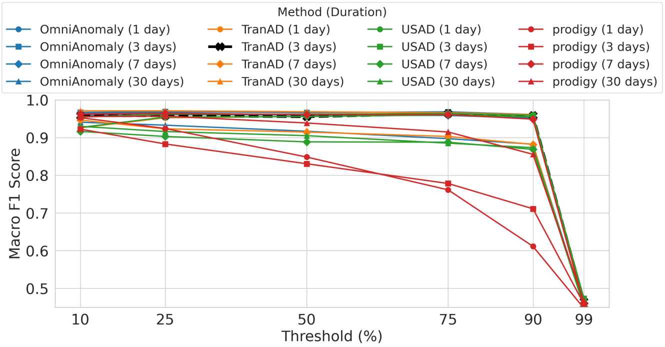

Fig. 6. Comparison of Macro F1 scores for various models against different thresholds for marking a service anomalous. TranAD (3 days) is highlighted in black.

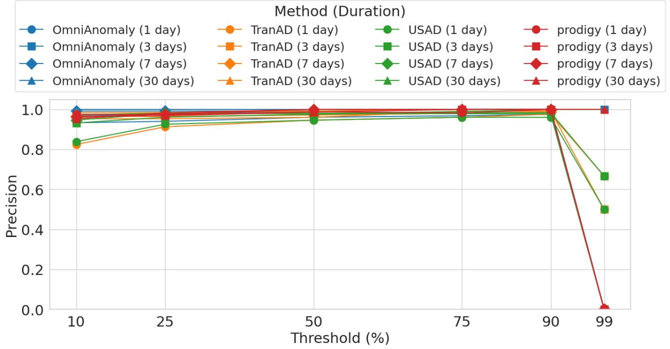

Fig. 8. Comparison of precision for different models against different thresholds. We observe a high precision in general for all models.

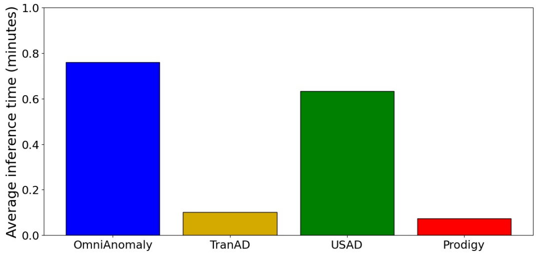

Fig. 7. Comparison of inference times for different models. Prodigy and TranAD are 6–7× faster.

We observe in Figure 5 and Figure 6 that TranAD (1, 3, and 30 days), all OmniAnomaly models, USAD (1 and 7 days), and Prodigy (7 days) achieve comparable weighted and macro F1 scores, ranging from 0.966 macro F1 for OmniAnomaly (3 day) to 0.957 and 0.970 for TranAD (3 and 30 days), respectively, at the 10% threshold. They also continue to show comparable F1 scores across different thresholds. On closer inspection, some models, such as TranAD (3 days), show a slight increase in F1 score as the threshold rises to 75%. For example, the macro F1 scores for TranAD 3 days at 10, 25, 50, 75, and 90% thresholds are 0.957, 0.957, 0.955, 0.963, and 0.958, respectively. Beyond this point, the macro F1 score drops sharply to 0.46 at the 99% threshold. Consequently, training time becomes the defining criterion for deployment. We further observe that the TranAD model trained on 3 days of data achieves the best tradeoff between high F1 scores and training time.

During our experiments, we also measure the mean inference times for these models. Figure 7 shows that TranAD and Prodigy are approximately 6-7× faster than OmniAnomaly and USAD. This measurement does not include the time required to prepare the test data (e.g., creating a 3D tensor for TranAD or collapsing 2D data into a single 1D array for USAD).

Additionally, Figure 8 presents the precision achieved by all models across their respective training data durations. We observe a high precision overall, which is preferable because greater precision reduces false alerts. Figure 9 illustrates recall. Recall steadily declines as the decision threshold increases and falls to nearly zero at a 99% threshold, demonstrating that setting the threshold too high causes the system to overlook

TABLE III
TOP-10 SERVICES FOR EACH RESOURCE METRIC (AVERAGE VALUES OVER ONE MONTH).

Rank	**Memory (MB)**		**Network (MB/s)**		**Disk I/O (MB/s)**		**CPU Usage**	
	Service	Avg	Service	Avg	Service	Avg	Service	Avg
1	ironic_conductor	47186.26	manila_share	101.34	mariabackup	12.06	rabbitmq	0.758
2	ironic_ipxe	9115.98	manila_nfs_ganesha	101.34	ironic_conductor	0.64	neutron_server	0.527
3	elasticsearch	7448.36	glance_api	84.16	ironic_ipxe	0.17	mariadb	0.394
4	mariadb	7137.44	prometheus_elasticsearch_exporter	83.46	cron	0.15	mariabackup	0.357
5	horizon	3005.16	memcached	83.46	mariadb	0.06	nova_api	0.266
6	glance_api	2370.28	keystone_ssh	83.46	prometheus_server	0.01	nova_compute_ironic	0.170
7	nova_api	2105.93	doni_api	83.44	neutron_dhcp_agent	0.00	ironic_api	0.135
8	memcached	1887.35	ironic_dnsmasq	83.44	etcd	0.00	keystone	0.129
9	keystone	1831.33	ironic_neutron_agent	83.44	horizon	0.00	placement_api	0.101
10	neutron_server	1625.48	prometheus_ipmi_exporter	83.44	elasticsearch	0.00	neutron_dhcp_agent	0.089

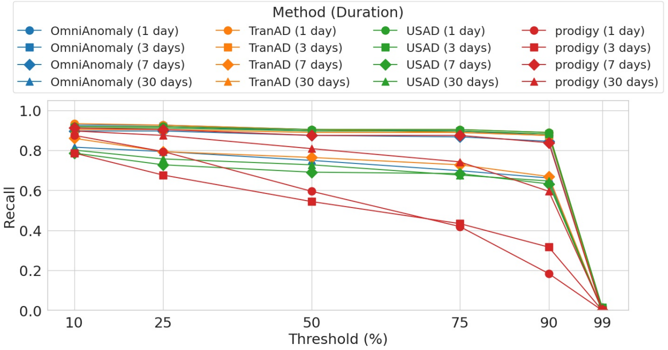

Fig. 9. Comparison of recall for different models across varying thresholds. We observe a sharp drop in recall when the threshold exceeds 90%.

most anomalies.

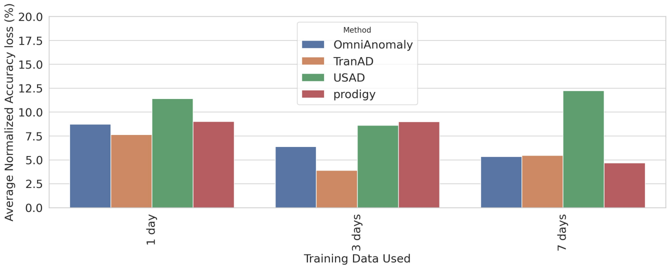

Fig. 10. Comparison of loss of accuracy for different models with different durations of healthy training data against their respective 30 day baseline.

Lastly, Figure 10 presents the average normalized accuracy loss across all outages for each model trained on varying durations of healthy data, compared against its own 30-day training baseline. We calculate accuracy as the difference in the percentage of data marked as anomalous by the model, relative to its 30-day baseline. This difference is then normalized and aggregated across all outages.

We also observe that application-level outages, like failures to reserve a node when no free IP addresses are available, surfaced by Tempest tests (smoke tests), do not cause any change in resource usage metrics and are therefore missed by our anomaly detection method.

We would also like to report the observation that using distant data, such as one month of data from six months ago, to train our model results in highly inaccurate outcomes, often marking all services as anomalous due to data drift.

In the end, using one month of data (Feb 2024), we compute the average CPU, memory, disk I/O, and network utilization for every service, we use the following metrics and methodology to compute these averages:

- **CPU usage** — We take the five-minute rate of `container_cpu_usage_seconds_total` and compute its mean to rank the services.
- **Memory usage** — We use the mean value of `container_memory_working_set_bytes`.
- **Disk I/O** — For each service we collect `reads_bytes_total` and `writes_bytes_total` for every device at five-minute rate, then sum the mean rate of each device to obtain the overall disk-I/O usage.
- **Network usage** — Analogous to disk I/O: we gather `transmit_bytes_total` and `receive_bytes_total` for all network interfaces at five-minute rate and sum the mean rate of each interface to compute the service's network usage.

Our evaluation reveals some informative findings. For instance, `ironic_conductor` consumes roughly five times as much memory as `ironic_ipxe`, which itself averages about 9GB. `manila_share` and `manila_nfs_ganesha` generate 20% more network traffic than the other services, while `mariabackup` exhibits significantly higher disk activity. Finally, `rabbitmq` and `neutron_server` consume considerably more CPU than the remaining services, offering actionable insights for capacity planning and performance tuning of OpenStack deployments. Table III lists the ten services with the highest average usage for each resource type.

VII. CONCLUSION

In this work, we presented a practical anomaly detection framework tailored for OpenStack services running on the Chameleon testbed. We publish the first publicly available dataset of OpenStack service metrics, enabling further research in cloud infrastructure monitoring and anomaly detection. Our experiments highlight several key findings. We demonstrate that using just a few days of healthy data for training provides an effective balance between training efficiency and detection accuracy. Additionally, we categorize OpenStack services based on their observed resource usage patterns (CPU, memory, disk, and network), helping inform service-specific deployment and planning. We believe these insights, along with our shared dataset and implementation experiences, can help guide future efforts in building robust and efficient monitoring systems for complex cloud environments.

Acknowledgment

We thank Fatih Acun, Efe Sencan, Zongshun Zhang and Amin Khodaverdian for their helpful discussion on this work.

References

[1] Amazon Web Services, “Amazon.com case study,” 2022. Accessed: 2025-04-15.

[2] C. Tang, Y. Wang, B. Fan, B. Wang, S. Chen, Z. Qiu, C. Liang, J. Zhao, Y. Zhu, M. Chen, and Z. Hu, “Rethinking the cloudonomics of efficient i/o for data-intensive analytics applications,” 2023.

[3] OpenAI, “Openai and microsoft extend partnership.” https://openai.com/index/openai-and-microsoft-extend-partnership/, January 2023. Accessed: 2025-04-15.

[4] T. Nguyen, “One amazon employee’s ”human error” may have cost the economy millions,” *Vanity Fair*, March 2017. Accessed: 2025-04-17.

[5] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, “Lessons learned from the chameleon testbed,” in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC ’20)*, USENIX Association, July 2020.

[6] P. Ruth and M. Cevik, “Experimenting with aws direct connect using chameleon, exogeni, and internet2 cloud connect,” in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–2, 2019.

[7] K. Keahey, P. Riteau, J. Anderson, and Z. Zhen, “Managing allocatable resources,” in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 41–49, 2019.

[8] P. Ruth, K. Keahey, M. Cevik, Z. Zhen, C. Wang, and J. Anderson, “Overcast: Running controlled experiments spanning research and commercial clouds,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2021.

[9] A. Esquivel Morel, W. Fowler, K. Keahey, K. Zheng, M. Sherman, and R. Anderson, “Autolearn: Learning in the edge to cloud continuum,” in *Proceedings of the SC’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 350–356, 2023.

[10] J. Anderson and K. Keahey, “Migrating towards single sign-on and federated identity,” in *Practice and Experience in Advanced Research Computing*, PEARC ’22, (New York, NY, USA), Association for Computing Machinery, 2022.

[11] D. Rosendo, K. Keahey, A. Costan, M. Simonin, P. Valduriez, and G. Antoniu, “Kheops: cost-effective repeatability, reproducibility, and replicability of edge-to-cloud experiments,” in *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*, pp. 62–73, 2023.

[12] OpenInfra Foundation, “Openstack: Open source cloud computing infrastructure.” https://www.openstack.org/, 2025. Accessed: 2025-04-17.

[13] OpenInfra Foundation, “2022 OpenStack User Survey Report,” tech. rep., OpenInfra Foundation, 2022. Accessed: 2025-04-30.

[14] W. Pourmajidi, J. Steinbacher, T. Erwin, and A. Miranskyy, “On challenges of cloud monitoring,” in *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, CASCON ’17, (USA), p. 259–265, IBM Corp., 2017.

[15] K. Keahey, J. Anderson, P. Ruth, J. Colleran, C. Hammock, J. Stubbs, and Z. Zhen, “Operational lessons from chameleon,” in *Proceedings of the Humans in the Loop: Enabling and Facilitating Research on Cloud Computing*, HARC ’19, (New York, NY, USA), Association for Computing Machinery, 2019.

[16] Prometheus Authors, “Prometheus: Monitoring system and time-series database.” GitHub repository, accessed 30 April 2025.

[17] Chameleon Cloud, “Chameleon Cloud Outages Page.” https://www.chameleoncloud.org/user/outages. Accessed: 2025-04-18.

[18] W. Pourmajidi, A. Miranskyy, J. Steinbacher, T. Erwin, and D. Godwin, “Dogfooding: using ibm cloud services to monitor ibm cloud infrastructure,” in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, CASCON ’19, (USA), p. 344–353, IBM Corp., 2019.

[19] Z. He, G. Hu, and R. B. Lee, “Cloudshield: real-time anomaly detection in the cloud,” in *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*, pp. 91–102, 2023.

[20] M. S. Islam, W. Pourmajidi, L. Zhang, J. Steinbacher, T. Erwin, and A. Miranskyy, “Anomaly detection in a large-scale cloud platform,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 150–159, IEEE, 2021.

[21] X. Zhang, F. Meng, and J. Xu, “Perfinsight: A robust clustering-based abnormal behavior detection system for large-scale cloud,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 896–899, IEEE, 2018.

[22] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. D. S. Silvestre, “Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned,” *Journal of Systems and Software*, vol. 139, pp. 84–106, 2018.

[23] Z. He, G. Hu, and R. B. Lee, “Cloudshield: real-time anomaly detection in the cloud,” in *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*, pp. 91–102, 2023.

[24] 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.

[25] B. Aksar, E. Sencan, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, B. Kulis, M. Egele, and A. K. Coskun, “Prodigy: Towards unsupervised anomaly detection in production hpc systems,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2023.

[26] 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 international conference on knowledge discovery & data mining*, pp. 3395–3404, 2020.

[27] 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 international conference on knowledge discovery & data mining*, pp. 2828–2837, 2019.

[28] “Rally: Openstack benchmarking tool,” 2014. https://docs.openstack.org/rally/latest/.

[29] “Tempest: Openstack integration test suite,” 2012. https://docs.openstack.org/tempest/latest/overview.html.

[30] “Ceilometer: Openstack telemetry,” 2013. https://docs.openstack.org/ceilometer/latest/.

[31] “Monasca: Monitoring-as-a-service,” 2015. https://wiki.openstack.org/wiki/Monasca.

[32] “Aodh: Event-based alarming for openstack,” 2015. https://docs.openstack.org/aodh/latest/.

[33] “Alertmanager: Prometheus alert management.” https://prometheus.io/docs/alerting/latest/alertmanager/, 2025. Accessed: 2025-05-08.

[34] OpenStack, “Kolla-ansible deployment guide,” 2023. https://docs.openstack.org/kolla-ansible/latest/.

[35] C. Huang, G. Min, Y. Wu, Y. Ying, K. Pei, and Z. Xiang, “Time series anomaly detection for trustworthy services in cloud computing systems,” *IEEE Transactions on Big Data*, vol. 8, no. 1, pp. 60–72, 2017.

[36] F. J. Meng, X. Zhang, P. Chen, and J. M. Xu, “Driftinsight: detecting anomalous behaviors in large-scale cloud platform,” in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 230–237, IEEE, 2017.

[37] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, “Self-adaptive cloud monitoring with online anomaly detection,” *Future Generation Computer Systems*, vol. 80, pp. 89–101, 2018.

[38] H. Jayathilaka, C. Krintz, and R. Wolski, “Detecting performance anomalies in cloud platform applications,” *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 764–777, 2018.

[39] M. Toslali, S. Qasim, S. Parthasarathy, F. A. Oliveira, H. Huang, G. Stringhini, Z. Liu, and A. K. Coskun, “Unleashing performance insights with online probabilistic tracing,” in *2024 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 72–82, 2024.

[40] M. S. Islam, W. Pourmajidi, L. Zhang, J. Steinbacher, T. Erwin, and A. Miranskyy, “Anomaly detection in a large-scale cloud platform,” in

2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 150–159, IEEE, 2021.

[41] M. S. Islam, M. S. Rakha, W. Pourmajidi, J. Sivaloganathan, J. Steinbacher, and A. Miranskyy, "Anomaly detection in large-scale cloud systems: An industry case and dataset," *arXiv preprint arXiv:2411.09047*, 2024.

[42] "cadvisor: Container advisor." https://github.com/google/cadvisor. Accessed 30 April 2025.

[43] Grafana Labs, "Grafana loki: Like prometheus, but for logs." https://grafana.com/oss/loki/, 2024. Accessed: 2025-04-15.

[44] Grafana Labs, "Grafana mimir: Open source, horizontally scalable, highly available, multi-tenant time series database." https://grafana.com/oss/mimir/, 2024. Accessed: 2025-04-15.

[45] Fluentd Project, "Fluentd: Open source data collector." https://www.fluentd.org/, 2025. Accessed: 2025-04-15.

[46] Grafana Labs, "Grafana: The open observability platform." https://grafana.com/, 2024. Accessed: 2025-04-15.

[47] "Jenkins: The leading open source automation server." https://www.jenkins.io/. Accessed: 2025-04-15.

[48] D. P. Kingma, M. Welling, *et al.*, "Auto-encoding variational bayes," 2013.

[49] 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.

[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[51] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, pp. 1530–1538, PMLR, 2015.

[52] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[53] K. Pearson, "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Series 5*, vol. 50, no. 302, pp. 157–175, 1900.

[54] E. Ates, B. Aksar, V. J. Leung, and A. K. Coskun, "Counterfactual explanations for multivariate time series," in *2021 International Conference on Applied Artificial Intelligence (ICAPAI)*, pp. 1–8, 2021.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.