PreGAN+: Semi-Supervised Fault Prediction and Preemptive Migration in Dynamic Mobile Edge Environments

Shreshth Tuli, Giuliano Casale and Nicholas R. Jennings

Abstract—Typical mobile edge computing infrastructures have to contend with unreliable computing devices at their end-points. The limited resource capacities of mobile edge devices gives rise to frequent contentions, node overloads or failures. This is exacerbated by the strict deadlines of modern applications. To avoid failures, fault-tolerant approaches utilize preemptive migration to transfer active tasks across nodes and prevent nodes running at capacity. However, prior work struggles to dynamically adapt in settings with highly volatile workloads or even accurately detect and diagnose anomalies for optimal remediation. To meet the strict service level objectives of contemporary workloads, there is a need for dynamic fault-tolerant methods that can quickly adapt to changes in edge environments while having parsimonious remediation in the form of preemptive migration to avoid stressing the system network. This work proposes PreGAN, featuring a Generative Adversarial Network (GAN) based approach to predict contentions, pinpoint specific resource types with high chance of overload, and generate migration decisions to proactively avoid system downtime. PreGAN leverages coupled-simulations to train the GAN model at run-time and a few-shot fault classifier to update decisions of an underpinning scheduler. We also extend it to PreGAN+ that also periodically tunes the decision model using semi-supervised training and a Transformer based neural network for low tuning time, albeit with higher memory overheads. Experiments on a Raspberry-Pi based edge environment demonstrate that both models outperform state-of-the-art baselines in fault detection and diagnosis scores by up to 12.5% and 31.2% respectively. This also translates in improvements in Quality of Service against baseline approaches.

Index Terms—Fault Tolerance; Preemptive Migrations; Edge Computing; Generative Adversarial Networks; Semi-Supervised Learning.

1 Introduction

The paradigm of mobile edge computing follows the data gravity principle, wherein the processing occurs close to the source of data, which typically are the sensors and actuators in an Internet of Things (IoT) framework [1]. This entails having compute devices geographically spread to cater to the large number of users, say within a smart-city or a smart-grid like environment with a large number of interconnected nodes [2]. In such contemporary use cases, workloads typically demand low latency execution and usually have non-stationary resource requirement characteristics exacerbating the challenge further [3]. When performing computation at such scales it becomes financially infeasible to have computationally heavy nodes as edge devices. In such settings, practitioners and service providers prefer instead to deploy resource limited nodes at the edge. Within such constraints it is common to have resource contentions that lead to system downtimes, which could lead to significant financial losses for service providers. The increasing amounts of data requiring immediate processing and the high level of volatility in modern mobile systems is pushing edge resources to their limits [4]. This gives rise to the need for developing fault-tolerance mechanisms that not only are able to quickly adapt to changes in edge systems, but also accurately predict contentions for optimal remediation.

• *S. Tuli and G. Casale are with the Department of Computing, Imperial College London, United Kingdom. N. R. Jennings is with Loughborough University, United Kingdom. E-mails: {s.tuli20, g.casale}@imperial.ac.uk, n.r.jennings@lboro.ac.uk.*

Manuscript received —; revised —.

Challenges. The problem of fault-tolerant computing is challenging. A part of fault-tolerance is the ability to proactively predict faults and diagnose root-cause issues to run apt remediation steps. For mobile edge computing, this entails the prediction of the specific nodes where resource contention might occur and also identifying the specific resource type (such as CPU, memory or disk) that would get overloaded. Fault predictions can then be utilized to inform resource management and decision-making to proactively avoid overloads or faults. These steps should be carried out in near real-time for such systems to be effective [5]. Further, to ensure efficacy, such systems need to be able to predict and resolve diverse types of faults. However, this is challenging due to the volatile nature of workloads and resources, which have characteristics that may change unexpectedly [6]. Additionally, in mobile computing environments, the network bandwidth is limited. Performing remediation in the form of task migration may cause long transfer times, consequently increasing the average response times. Thus, we need approaches with low overheads to effectively trade off between remediating faults and having high overheads. Specifically, this requires fault predictions to have low false positives to avoid running unnecessary remediation. Avoiding significant adverse effects of contentions in the form of system downtimes additionally requires such methods to have low false negatives, requiring such prediction models to be extremely accurate. To do this, some existing methods leverage machine learning models due to their high accuracy [7], [8]. However, in such cases,

the annotated log traces available to brokers are limited, leading to restricted size of the labelled fault classification data. This makes it hard to train supervised machine learning models for fault detection and classification.

Existing solutions. Several approaches have been proposed in the past few years for fault-tolerant computing in edge or cloud platforms [4], [9]. One of the common approaches is to provide node duplication or redundancy and network contingencies to avoid service downtime of any element of a mobile edge framework. However, in large-scale resource-constrained deployments, having a redundant node for each edge host is infeasible [10]. Another approach is to replicate the running instances of the tasks on separate nodes [11]; however, this doubles the compute requirements in an already resource limited setting [12]. A solution that is commonly used by prior work is instead *preemptive migration* [13] where the execution state of a running task is saved as a *checkpoint* which is then transferred to another node to resume execution. This is enabled by containerization like virtualization services. However, in volatile environments with several tasks running in the same edge system, frequently checkpointing and transferring tasks could impose excessive stress on both system and network. We need parsimonious migration strategies that migrate only the critical tasks. In recent years, many different methods have been proposed to decide the appropriate migration decisions to enhance service reliability. These range from Particle Swarm Optimization (PSO) [4], Integer Linear Programming [9], [14] and Bayesian Classification [7] to using Neural Network and clustering algorithms like k-means [8]. However, such methods are often not generalizable or struggle to adapt in volatile settings quickly and accurately, as discussed later in Section 2. Thus, we propose a novel fault-tolerance model and demonstrate its efficacy against these state-of-the-art baselines [7], [8].

New Insights and Contributions. For effective fault-tolerance, it is critical to accurately detect, diagnose and classify faults online for a mobile edge environment. To do this, we utilize in this work, a neural network based state reconstruction model. A prototypical network is utilized to classify faults into the specific resource type that could be overloaded, such as CPU, memory or disk [15]. To efficiently model the temporal trends and the cross-correlations of performance and resource utilization metrics among different edge nodes, we employ graph attention network (GAT) and gated recurrent units (GRU) [16], [17]. GAT allows the model to assign different weights to different host nodes. We then use a Generative Adversarial Network (GAN) to generate informed preemptive migration decisions that are utilized to update the task placement decision from an underpinning scheduler [18]. To train the GAN model, we leverage a simulator, which emulates scheduling decisions in a virtual replica of the mobile edge computing environment. We ensure that the dynamic changes in the physical environment are also reflected in real-time within the virtual replica, which is referred to as having a tight coupling between the physical and virtual counterparts. We refer to this as a *coupled* simulator in our work [19]. We call the model with a static policy over task execution as PreGAN, which has been first introduced in a preliminary version of this paper [20]. Further, to adapt the model in dynamic settings, we extend PreGAN to PreGAN+ that leverages semi-supervised learning to periodically tune the classification model and leverages a Transformer [21] based model for time-efficient tuning. PreGAN+ can be configured to run identically as PreGAN, *i.e.*, without online fine-tuning. This work leverages co-simulation and self-supervised learning to predict preemptive migration decisions for fault-tolerant edge computing. We perform extensive empirical experiments on real-life edge testbed to compare and analyze PreGAN and PreGAN+ against the state-of-the-art methods. Our experiments show that both methods outperform baselines in terms of QoS metrics, reducing the energy consumption, response time and SLO violations by up to 8%, 7% and 15%, respectively for PreGAN+. It achieves this by up to 12.5% more accurate fault detection, 13.9% lower task migrations and 23.8% lower overhead than the most accurate baseline. PreGAN too outperforms baselines, but the improvements are lower compared to PreGAN+. Although PreGAN+ gives better QoS scores compared to PreGAN, it imposes higher overheads (see Section 4.8) and is more suitable in settings with high capacity broker nodes.

The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 outlines the PreGAN and PreGAN+ for model training, preemptive migration prediction and execution. A performance evaluation of the proposed method is developed in Section 4. Finally, Section 5 concludes the paper and discuses future directions.

2 Related Work

Fault-tolerance deals with developing systems that have an ability to withstand failures and faults in the workloads and efficiently manage resources to maintain optimum QoS. Most methods can be divided into two categories: reactive and proactive. Table 1 summarizes the prior work. The former take action after observing a system fault by typically checkpointing, replicating or resubmitting tasks affected by the fault [29]. The latter aim to avoid expensive fault recovery by predicting failures in advance and taking appropriate steps for remediation by preemptive migration or fault-aware scheduling [29]. As reactive schemes often lead to poor QoS in highly dynamic setups [4], we focus on proactive methods in this work.

Most contemporary state-of-the-art techniques employ some form of specialized algorithms or machine learning models. Dynamic Fault Tolerant Migration (DFTM) [9] is a recent method that uses an integer linear programming (ILP) formulation to analyze workload traffic, select the tasks running on the hosts that should be migrated and the target hosts for restoration. Other approaches such as Threshold-Based Adaptive Fault Tolerance (TBAFT) [22] migrate tasks proactively from overloaded devices to devices with low resource usage. It uses an ARIMA model to predict resource utilization metrics of host machines and compares them against human-set thresholds and executes task migration to avoid contention. Another method proposes a preference based fault management technique (PBFM) [23] that tries to balance between the QoS improvement and the migration cost using a multi-objective integer linear programming formulation. Methods such as Multi-stage Co-flow Scheduling (MCS) [14] also consider task co-dependencies

Table 1: Comparison of related works with different parameters (✓means that the corresponding feature is present).

Work	Supervision	Approach	Fault Detection	Fault Remediation	Remediation Type	Performance Parameters			
						Accuracy	Overhead	Energy	SLO Violations
DFTM [9]	Unsupervised	Integer Linear Programming	✓	✓	Reactive			✓	✓
TBAFT [22]	Supervised	Overload prediction and task replication		✓	Proactive		✓	✓	✓
PBFM [23]	Unsupervised	Linear Programming based load balancing		✓	Proactive		✓	✓	
MCS [14]	Unsupervised	Heuristics based optimization		✓	Reactive		✓	✓	✓
SFS [24]	Unsupervised	Heuristics based load balancing		✓	Reactive	✓	✓		
PCFT [4]	Supervised	Particle Swarm Optimization	✓	✓	Reactive		✓	✓	✓
ECLB [7]	Supervised	Bayesian Optimization and Neural Networks	✓	✓	Proactive	✓		✓	✓
AWGG [25]	Unsupervised	Autoencoder based clustering	✓		Reactive	✓			
TopoMAD [26]	Unsupervised	Topology based autoencoder and LSTM	✓		Reactive	✓	✓		
CSAVM [27]	Unsupervised	Evolutionary search and live migration		✓	Reactive	✓		✓	✓
DDQP [28]	Unsupervised	Double Deep Q Networks		✓	Reactive			✓	✓
CMODLB [8]	Unsupervised	K-Means clustering and Swarm Optimization	✓	✓	Proactive	✓		✓	✓
This work	Semi-supervised	Prototypical Networks and GANs	✓	✓	Proactive	✓	✓	✓	✓

to optimize QoS. Another ILP based method proposes a preference based fault management technique [23] that tries to balance between the QoS improvement and the migration cost using a multi-objective formulation. Such methods do not scale for real-time operations, making them unsuitable for mission critical edge applications. SFS [24] is a failover strategy that selects only those tasks that are about to violate their deadlines to reduce migration overheads. However, modeling the SLO deadlines does not consider other QoS parameters while optimizing the selection of the target hosts. This makes it often perform poorly in setups with heterogeneous edge nodes.

The Proactive Coordinated Fault Tolerance (PCFT) [4] method uses Particle Swarm Optimization (PSO) to reduce the overall transmission overhead, network consumption and total execution time for a set of tasks. This method first predicts faults in the running host machines by anticipating resource deterioration and uses PSO to find target hosts for preemptive migration decision. This approach mainly focuses on reducing transmission overheads in distributed cloud setups, but often fails to improve the I/O performance of the compute nodes [9]. The Energy-efficient Checkpointing and Load Balancing (ECLB) [7] technique uses Bayesian methods and neural networks to classify host machines into three categories: overloaded, underloaded and normal execution. This classification is used to decide appropriate task migrations to reduce the number of overloaded hosts. However, this model only considers computational overloads and does not consider other fault types. Other recent methods utilize deep neural networks to execute fuzzy clustering [25], [30]. For instance, the Adaptive Weighted Gath-Geva (AWGG) [25] clustering method is an unsupervised model that detects faults using stacked sparse autoencoders to reduce detection times. Another recent class of methods applies neural networks to reconstruct the last state of the system [26]. The reconstruction error is used as an indicator of the likelihood of the current state being anomalous. For instance, TopoMAD [26] utilizes a topology-aware neural network that is composed of a Long-Short-Term-Memory (LSTM) and a variational autoencoder (VAE) to detect faults. However, the reconstruction error is only obtained for the latest state, which limits them to use reactive fault recovery policies unsuitable for volatile environments.

CSAVM [27] uses another evolutionary search scheme to take live migration decisions for the task queues. The method is used to optimize the power consumption of a compute setup by preventing unnecessary migrations.

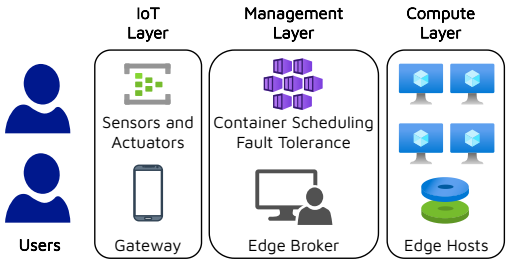


Figure 1: System Model.

DDQP [28] uses double deep Q-networks to place services on network nodes. A similar method, FTAW [31], uses deep Q-networks to decide optimal scheduling decisions for input workflow type workloads in distributed edge environments. However, such reinforcement learning schemes are known to be slow to adapt in volatile settings [19].

A recent work, CMODLB [8], uses a clustering-based multiple objective dynamic load balancing technique to avoid resource contentions in cloud computing nodes. This method uses k-means to cluster nodes and identify overloaded hosts, PSO to select tasks and deep learning and fuzzy-logic optimization to select target hosts. Tasks are selected from these hosts using the PSO approach and target hosts using deep learning and fuzzy-logic based optimization. However, slow PSO optimization leads to high recovery times that is often not helpful in highly dynamic systems [19]. Instead, in the proposed approach we directly predict resource management decisions as a single inference of a neural model, having much lower decision time compared to optimzation based strategies such as CMODLB. In our experiments, we compare PreGAN against the state-of-the-art baselines that perform both fault detection and remediation: DFTM, ECLB, PCFT and CMODLB.

3 Methodology

3.1 System Model and Problem Formulation

In this work, we target a standard heterogeneous edge computing environment where all nodes are in the same Local Area Network (LAN); see Figure 1 for an overview. Tasks are container instances generated from the sensors and actuators in the IoT layer and communicated to the compute nodes via gateway devices. The edge broker takes all scheduling, management and preemptive migration decisions for all tasks, which execute in edge hosts.

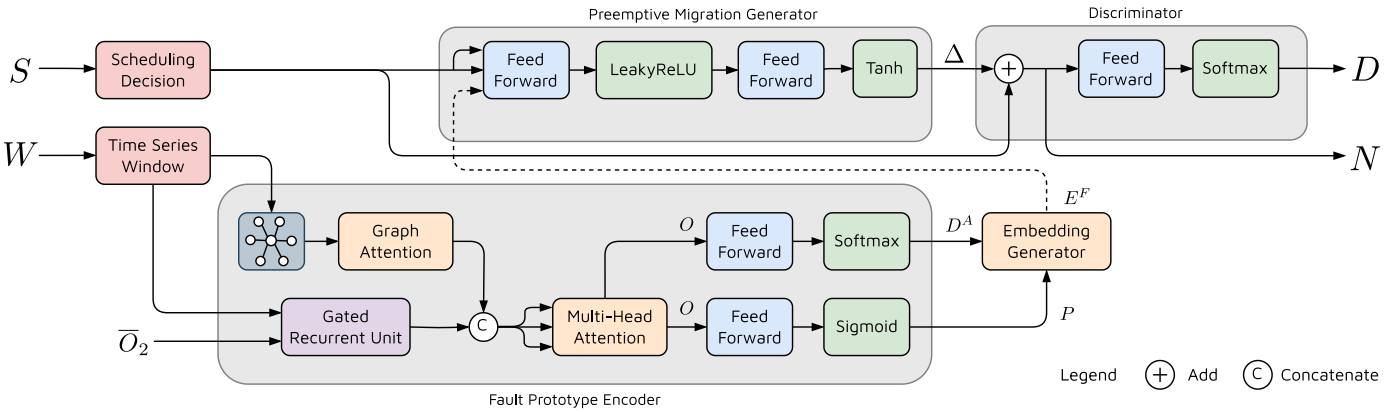


Figure 2: The PreGAN Model.

We assume that there are m host machines in the fog resource layer and denote them as $H = \{h_1, \ldots, h_m\}$. We assume the paradigm of discrete-time control, where we take scheduling and migration decisions at periodic intervals [3], [32]. We assume a scheduler is already present in the broker. The t-th interval is denoted as I_t and the scheduling decision and host characteristics (measured features) at this interval are denoted as S_t and $x_t \in \mathbb{R}^{m \times n}$; thus, each host has n features. The time-series $\{x_0, \ldots, x_{t-1}\}$ is denoted as $\mathcal{T}_t$. We divide the problem of fault-tolerance into two sub-problems, namely:

Fault Prediction: For an input multi-variate time-series $\{x_0, \ldots, x_{t-1}\}$, we need to predict fault labels for each host in I_t as $y_t \in \{0, 1, \ldots, c\}^m$. Here, $y_{t,i}$ denotes the output prediction for h_i; $y_{t,i} = 0$ means no fault and $y_{t,i} = j$ means fault of class j among the user-specified c fault classes $j \in \{1, \ldots, c\}$.

Preemptive Migration Decision: For an input scheduling decision S_t (a set of task and host pairs) and a solution of the prediction problem y_t, we wish to predict a migration decision Δ_t. To model the dependence of a data point x_t at a timestamp t, we consider a sliding window of length k, denoted as

$$W_t = \{x_{t-k}, \ldots, x_{t-1}\},$$

as routinely done in prior work [33]. Thus, in lieu of using just x_t, we use W_t as it allows us to capture the local context. For model robustness, we normalize the input windows by min-max scaling. For the sake of simplicity and without loss of generality, we drop the t index whenever not ambiguous and use x, S, W, Δ. A summary of the presented notation is shown in Table 2.

Table 2: Summary of the Main Notation

Symbol	Meaning
I_t	t-th scheduling interval
H	Set of available hosts
S_t	Scheduling decision in I_t
x_t	Host characteristics in I_t
$\mathcal{T}_t$	Time-series $\{x_0, \ldots, x_{t-1}\}$
$y_{t,i}$	Fault label for host $h_i \in H$ in I_t
W_t	Time-series window $\{x_{t-k}, \ldots, x_{t-1}\}$
D_i^A	Predicted fault class for host $h_i \in H$
E_i^F	Embedding vector of host $h_i \in H$
Δ	Preemptive migration decision
S	Scheduling decision of the underpinning scheduler
N	Proposed scheduling decision formed as $S + \Delta$
D	Discriminator output

3.2 PreGAN Model

We start with describing the PreGAN model that uses prototypical networks [15] and GANs [34] for fault-tolerance. Figure 2 provides an overview of the PreGAN model. We divide our model in two parts: (1) Fault Prototype Encoder (FPE) that aims to solve the fault prediction problem and (2) Preemptive Migration GAN for decision making (see the Preemptive Migration Generator and Discriminator in Figure 2). The multi-variate time-series is composed of the host characteristics x, including CPU, RAM, Disk utilization. We choose these in our experiments, but these can be chosen arbitrarily. We form a graph using the schedule S, such that there is an edge from h_i to h_j if there is a task migration from h_i to h_j in S_t from S_{t-1}. The n characteristics of each host are then used to populate the feature vectors of the nodes in the graph. We use in particular a graph attention network (GAT) in parallel with Gated Recurrent Units (GRUs). GAT allows us to extract the multi-host feature correlations with the migration information encoded as graph edges. GRU allows us to extract temporal trends in time-series log data for prediction of faults in the next interval. This, in conjunction with the migration decision from the underlying scheduler, is used to detect the hosts with a high chance of faults and predict the class prototypes for each fault. The outputs of GRU and GAT are then passed to a multi-head attention module. The detection and classification results are combined to create an embedding for the generator network that predicts the preemptive migration decision to amend the input schedule. The decision is forwarded to a discriminator network to compare against the original scheduling decision.

We now describe the complete pipeline in detail. The input time-series window W is first converted to a tensor of size $k \times m \times n$. The scheduling decision for each task in S_t is its mapping to a specific host and thus it is converted to a one-hot decision vector of size m, these one-hot vectors are stacked to form a matrix of size $p \times n$.

Fault Prototype Encoder. To infer over all hosts in the graph, we create a new global node connected to each host

node to capture global dependencies [35]. This gives the graph attention operation as

$$O_1 = \text{Sigmoid}\left(\frac{1}{m}\sum_{i\in\{1,\ldots,m\}} \theta_{\text{GAT}} W^i\right), \tag{1}$$

where θ_{GAT} is the weight matrix for the GAT network and W^i is obtained from W retaining only the time window of features for host h_i. The time-series W is also passed through a GRU, with $\overline{O}_2$ as the output of the previous interval

$$O_2 = \text{GRU}(W, \overline{O}_2). \tag{2}$$

For any three input tensors Q, K and V, we define Multi-Head Self Attention [21] as passing it through h (number of heads) feed-forward layers to get Q_i, K_i and V_i for $i \in \{1, \ldots, h\}$, and then applying attention as

$$\begin{aligned} \text{MultiHeadAtt}(Q, K, V) &= \text{Concat}(H_1, \ldots, H_h) \\ H_i &= \text{Attention}(Q_i, K_i, V_i). \end{aligned} \tag{3}$$

Here $\text{Attention}(Q_i, K_i, V_i)$ is the scaled-dot product attention operation [21] that allows the model to highlight specific aspects of the temporal and structural dependencies in the input that are critical to identifying faults. We apply this to compute self-attention on the GRU and GAT outputs (with task SLO requirements and host characteristics) to obtain

$$O = \text{MultiHeadAtt}([O_1; O_2], [O_1; O_2], [O_1; O_2]). \tag{4}$$

The late-fusion of the two outputs allows the downstream models, including the multi-head attention, to exploit them independently and generate O [35]. The output O is a collection of attention based encodings for each attention head. We pass this encoding through two decoders for each host h_i,

$$\begin{aligned} D_i^A &= \text{softmax}(\text{FeedForward}(O_i)), \\ P_i &= \text{sigmoid}(\text{FeedForward}(O_i)), \end{aligned} \tag{5}$$

where $D_i^A \in \mathbb{R}^2$ denotes the fault prediction as binary classification and hence a softmax operation. The model predicts a fault in h_i if $D_i^A[1] \geq D_i^A[0]$. $P_i \in \mathbb{R}^E$ is a prototype embedding for each host of size E that models the fault class, irrespective of whether a fault is detected or not (we use a 2-element tensor to allow flexible thresholds). To constrain this to a [0, 1] space, we apply a sigmoid operation. This factored prediction for each host allows our model to be applicable to an arbitrary number of hosts in the system, under computational limits. We define the embedding for host h_i as

$$E_i^F = \begin{cases} P_i, & \text{if } D_i^A[1] \geq D_i^A[0] \\ [0]_{1\times E}, & \text{otherwise.} \end{cases} \tag{6}$$

Here, for a host where fault is detected, we use the generated prototype embedding P_i and a zero-vector otherwise. We stack all host embeddings to generate E^F. This style of using fault predictions allows us to generate a representation of the fault prediction that only consists of class prototypes for the hosts where a fault is detected. This also enables the preemptive migration generation to be informed of the fault class and take more informed decisions.

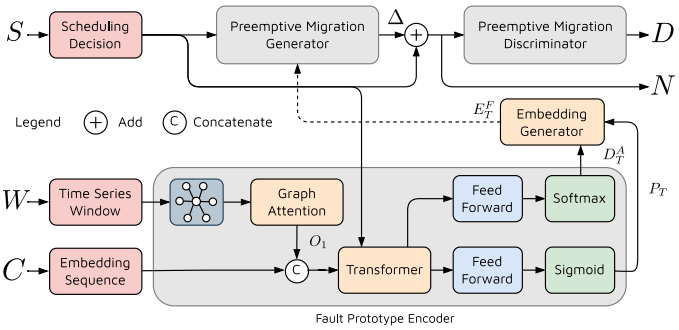


Figure 3: The PreGAN+ Model. Changes in the FPE. Generator and Discriminator models are same as in Figure 2.

Preemptive Migration GAN. The embedding produced in output by the FPE is passed through the Generator to obtain

$$\begin{aligned} \Delta_1 &= \text{LeakyReLU}(\text{FeedForward}([S, E^F])), \\ \Delta &= \text{Tanh}(\text{FeedForward}(\Delta_1)), \end{aligned} \tag{7}$$

where the $\Delta \in \mathbb{R}^{p\times n}$ denotes the preemptive migration decision. Here, the sharp transition between -1 and +1 in $\tanh$ facilitates the generation of a migration action for a given job in the output. Further, LeakyReLU gives a non-zero value for negative inputs, allowing it to perform better than ReLU for noisy inputs. The generator gets S from an underlying scheduler. Further, the output of the FPE is used to leverage the fault predictions and identify the specific task migrations that would reduce chances of contentions and subsequently faults in the fog environment. The output Δ is added to the original scheduling decision to get a new candidate schedule $N = S + \Delta$. The generated scheduling decision for each task p is calculated as $\text{argmax}(N_p)$, which is the target host. The output of the generator is then passed to a discriminator decoder

$$D = \text{Softmax}(\text{FeedForward}(N)), \tag{8}$$

where $D \in \mathbb{R}^2$ denotes predicted likelihood scores for using S and N as schedules. Unlike the FPE, we use a fully-connected network to identify the key aspects of the scheduling decision and faults, required for informed generation of the scheduling decision [36]. The final scheduling decision of PreGAN is defined as

$$\text{Final Schedule} = \begin{cases} N, & \text{if } D[1] \geq D[0] \\ S, & \text{otherwise.} \end{cases} \tag{9}$$

Summarizing, the FPE detects faults in hosts and generates class prototype embeddings for each host. These embeddings are stacked (E^F) and sent to a generator network that outputs a preemptive migration vector (Δ). The discriminator then provides a likelihood score of choosing the new (N) and the original scheduling decisions (S). The final output is N if $D[1] \geq D[0]$.

3.3 PreGAN+ Model

The PreGAN model assumes fault labels and classes to be given to tune its FPE. However, obtaining supervision labels may be expensive in many settings as we describe later in Section 4.3. Further, recurrent models like GRUs

are expensive to fine-tune as they require data to be given in a recurrent fashion at every timestep t. This makes it difficult to frequently tune PreGAN to dynamically adapt its decision making policy. However, once trained, PreGAN is a lightweight model, that when kept fixed, imposes very low decision making overheads (see Section 4.8 for broker load comparison). To allow using partial or no supervision and facilitate frequent model tuning, we modify PreGAN's FPE in PreGAN+, as shown in Figure 3.

Akin to PreGAN, in PreGAN+ we perform the graph attention operation on the input time-series window W to obtain O_1. However, instead of just providing the current graph embedding, we also provide a sequence of past (up to 100) embeddings of O_1 up to the last timestep $t-1$, which we denote as C, to re-use GAT inferences from past intervals. Unlike PreGAN, this enables us to provide the historic context, and not just the host characteristics for the last k intervals as in W, for more informed decision making. We concatenate the historic embedding sequence and GAT output $C_1 = [C, O_1]$, and pass it through a Transformer Encoder [21]. Unlike a recurrent neural network such as GRU or LSTM, a Transformer can take the complete sequence as input. It then uses attention operations to infer patterns across timesteps [37]. Specifically, we position encode the complete sequence C_1 using the trigonometric $\sin$ function to obtain C_2 [21]. The encoder then infers over C_2 and the scheduling decision S defined as

$$\begin{aligned} C_3 &= \text{LayerNorm}(C_2 + \text{MultiHeadAtt}(C_2, C_2, S)), \\ C_4 &= \text{LayerNorm}(C_3 + \text{FeedForward}(C_3)). \end{aligned} \tag{10}$$

Both the multi-head attention block and feed-forward model are embedded within the Transformer shown in Figure 3. We then pass this encoding (C_4, which is the Transformer output) through two decoders for each host h_i,

$$\begin{aligned} D^A_{T,i} &= \text{softmax}(\text{FeedForward}(C_{4,i})), \\ P_{T,i} &= \text{sigmoid}(\text{FeedForward}(C_{4,i})), \end{aligned} \tag{11}$$

where $D^A_{T,i}$ denotes the fault prediction and P_{T_i} is a prototype embedding for each host. Finally, we define the embedding for host h_i as

$$E^F_{T,i} = \begin{cases} P_{T,i}, & \text{if } D^A_{T,i}[1] \geq D^A_{T,i}[0] \\ [0]_{1\times E}, & \text{otherwise.} \end{cases} \tag{12}$$

We stack all host embeddings to generate E^F_T as the output of the FPE of PreGAN+. Unlike PreGAN that has GRU based temporal inference, a Transformer based model allows us to train FPE in a data and time efficient manner. Specifically, compared to recurrent models (GRUs and LSTMs), we do not need a lot of data to train such models [21]. Transformers also enables us to give a broader context as C_1 of the historic host characteristics in the edge environment albeit with higher memory overheads. This allows us to frequently tune the FPE in an online fashion, possibly at the start of each interval. Unlike GRUs that need data corresponding to several timesteps to fine-tune, Transformers can adapt to dynamic scenarios with limited data.

Algorithm 1 Offline FPE training algorithm

Require:
Fault Prototype Encoder E
Dataset used for training $\{W_t, S_t, \hat{y}_t\}_{t=1}^T$
Step size α, Evolutionary hyperparameter ϵ
Iteration limit L

1: Initialize weights in E. Set $l \leftarrow 0$
2: Randomly initialize class prototypes $\{P^C_0, \ldots, P^C_c\}$
3: **do**
4: **for**($t = 1$ to T)
5: $D^A, P \leftarrow E(S_t, W_t)$
6: $L_1 = \sum_{i=1}^m \big(\mathbb{1}(\hat{y}_{t,i} = 0)\log(D^A[0]) + \mathbb{1}(\hat{y}_{t,i} > 0)\log(D^A[1])\big)$
7: $L_2 = \sum_{i=1}^m \mathbb{1}(\hat{y}_{t,i} > 0)\big(\|P_i - P^C_{\hat{y}_{t,i}}\|_2 - (\sum_{j\neq\hat{y}_{t,i}}\|P_i - P^C_j\|_2)\big)$
8: **for**($i = 1$ to m)
9: **if**($\|P_i - P^C_{\hat{y}_{t,i}}\|_2 = \min_j \|P_i - P^C_j\|_2$)
10: $P^C_{\hat{y}_{t,i}} \leftarrow (1-\alpha)\cdot P^C_{\hat{y}_{t,i}} + \alpha \cdot P_i$
11: Update weights of E using L_1, L_2
12: $\alpha \leftarrow (1-\epsilon)\cdot\alpha$
13: $l \leftarrow l + 1$
14: **while** $l < L$

3.4 Offline FPE Training in PreGAN and PreGAN+

We now describe the training process for the FPE to detect and classify faults, summarized in Algorithm 1. We utilize a supervised training approach assuming sufficient compute time available at training time. To train the encoder, we first collect a dataset of input windows, scheduling decisions and fault class labels $\{W_t, S_t, \hat{y}_t\}_{t=1}^T$ by running the system without any preemptive migration. Here, $\hat{y}_{t,i}$ is the class label for h_i, with 0 indicating no fault. Now, the FPE encoder (denoted as E) generates the outputs D^A and P from the inputs W_t and S_t (line 4 in Alg. 1). We use the cross-entropy loss for fault detection (line 6 in Alg. 1). Here, $\hat{y}_{t,i}$ is used as the ground-truth labels and the loss is summed over all hosts:

$$L_1 = \sum_{i=1}^{m} \Big(\mathbb{1}(\hat{y}_{t,i}=0)\log(D^A[0]) + \mathbb{1}(\hat{y}_{t,i}>0)\log(D^A[1])\Big).$$

This loss allows us to modify the weights of the FPE encoder such that, at convergence, the model correctly assigns $D^A[1] > D^A[0]$ that denotes the presence of a fault as assumed in equations (6) and (12).

For fault-classification, we use a triplet loss. We first define class prototype vectors for each fault class $\{P^C_0, \ldots, P^C_c\}$, that are randomly initialized from $[0,1]^E$. Just like the means in a k-means clustering approach, these vectors denote the centroids of the embeddings of their respective classes [15], [38]. As is common in prototypical networks used in few-shot learning, the new embedding generated by the model belongs to the class which has the least Euclidean distance from the class prototypes. The triplet loss aims at reducing the distance of the output embedding from the true class prototype and increasing it from the false class ones (line 7 in Alg. 1). Thus, for all those hosts with a true fault, the loss includes the L2-norm between the output embedding and the true class prototype

and negative L2-norm between the output and false class prototypes:

$$L_2 = \sum_{i=1}^{m} \mathbb{1}(\hat{y}_{t,i} > 0) \Big(\|P_i - P^C_{\hat{y}_{t,i}}\|^2 - \sum_{j \neq \hat{y}_{t,i}} \|P_i - P^C_j\|^2 \Big). \quad (13)$$

Another step in the training process is to update the class prototypes. If an output embedding belongs to the correct class, we can update the prototype of that class as

$$P^C_{\hat{y}_{t,i}} \leftarrow (1-\alpha) \cdot P^C_{\hat{y}_{t,i}} + \alpha \cdot P_i, \quad (14)$$

where α is the step-size. We exponentially decay our step-size to ensure that the model learns class prototypes quickly initially and converges to a stable prototype set using a hyperparameter based decay (line 11 in Alg. 1). The triplet loss in conjunction with the prototype updates allow the prototypes to be distant from one another, making each prototype a characteristic identifier for that class.

3.5 Online FPE Tuning in PreGAN+

We now describe how we periodically fine-tune the FPE in the PreGAN+ model, summarized in Algorithm 2. We leverage a semi-supervised approach, where we generate labels for a fraction of the datapoints (denoted by ξ) to guide unsupervised clustering. Our approach is motivated by *constrained K-Means* [39]. We assume that within the ξ fraction of labeled datapoints, there is at least one from each class. We impose an assumption, common in prior work [4], [7], [22], [37], that the fraction of faulty datapoints is consistent across data. Thus, for all labelled data, we find the fraction of no-fault datapoints as

$$\Gamma = \frac{\mathbb{1}(\hat{y}_{t,i} = 0)}{\mathbb{1}(\hat{y}_{t,i} > 0)} \quad (15)$$

Now, for each protoype vector P_u without any label, we calculate the sum distance with each class prototype.

$$f_d(P_u) = \sum_{j=1}^{c} \|P_u - P^C_j\|. \quad (16)$$

We then take the top Γ fraction of the datapoints with the highest f_d output. We assign to these the fault-label 0 (no-fault class) and assign to the rest fault-label 1 (fault class). This function is represented as $f_l(P_u)$ for a vector P_u. The intuition is as follows. Considering that Γ fraction of datapoints in a dataset have no faults, the prototype vector for such datapoints should not belong to any fault class. This has been shown to translate to having the highest distances from the class representatives, namely, class prototypes [39]. Thus, the top Γ fraction of datapoints with the highest distance as calculated in equation (16) denote datapoints with fault label 0. We assume that at each timestep the hosts with known labels have the index set L and those with unlabelled classes U, such that $L \cup U = \{1, \ldots, m\}$. Thus, in such setting, the loss function for fault-detection becomes

$$L_1 = \sum_{i \in L} \Big(\mathbb{1}(\hat{y}_{t,i} = 0) \log(D^A[0]) + \mathbb{1}(\hat{y}_{t,i} > 0) \log(D^A[1]) \Big) + \sum_{i \in U} \Big((1 - f_l(P_i)) \log(D^A[0]) + f_l(P_i) \log(D^A[1]) \Big).$$

Algorithm 2 Online PreGAN+ FPE Tuning algorithm

Require:
Fault Prototype Encoder E
For timestep t, labeled L and unlabeled U datasets
Step size α, Evolutionary hyperparameter ϵ
Iteration limit L

1: Initialize weights in E. Set $l \leftarrow 0$
2: Initialize class prototypes $\{P^C_0, \ldots, P^C_c\}$ to labelled vectors
3: **do**
4: $D^A, P \leftarrow E(S_t, W_t)$
5: $L_1 = \sum_{i \in L} \big(\mathbb{1}(\hat{y}_{t,i} = 0) \log(D^A[0]) + \mathbb{1}(\hat{y}_{t,i} > 0) \log(D^A[1]) \big) + \sum_{i \in U} \Big((1 - f_l(P_i)) \log(D^A[0]) + f_l(P_i) \log(D^A[1]) \Big)$
6: $L_2 = \sum_{i \in L} \mathbb{1}(\hat{y}_{t,i} > 0) \big(\|P_i - P^C_{\hat{y}_{t,i}}\|_2 - (\sum_{j \neq \hat{y}_{t,i}} \|P_i - P^C_j\|_2) \big) + \sum_{i \in U} f_l(P_i) \Big(\|P_i - P^C_{f_c(P_i)}\|_2 - \sum_{j \neq f_c(P_i)} \|P_i - P^C_j\|_2 \Big)$
7: **for**($i = 1$ to m)
8: **if**($\|P_i - P^C_{\hat{y}_{t,i}}\|_2 = \min_j \|P_i - P^C_j\|_2$)
9: $P^C_{\hat{y}_{t,i}} \leftarrow (1-\alpha) \cdot P^C_{\hat{y}_{t,i}} + \alpha \cdot P_i$
10: Update weights of E using L_1, L_2
11: $\alpha \leftarrow (1 - \epsilon) \cdot \alpha$
12: $l \leftarrow l + 1$
13: **while** $l < L$

Thus, we use the prototypical vectors to *estimate* the fault class of the unlabeled datapoints that are temporarily assigned fault-label 1.

For classification, in lieu of initializing the class prototypes randomly, we use the labeled datapoints in the training set and sample uniformly at random to set the initial class prototype vectors. Now, for each prototype vector P_u without any label, we define the corresponding class as

$$f_c(P_u) = \arg\min_j \|P_u - P^C_j\|, \quad (17)$$

which translates to the class for which the class prototype is the closest as per L2 distance. We use this *estimated* class label for unlabeled datapoints, which gives us the classification loss as

$$L_2 = \sum_{i \in L} \mathbb{1}(\hat{y}_{t,i} > 0) \Big(\|P_i - P^C_{\hat{y}_{t,i}}\|_2 - \sum_{j \neq \hat{y}_{t,i}} \|P_i - P^C_j\|_2 \Big) + \sum_{i \in U} f_l(P_i) \Big(\|P_i - P^C_{f_c(P_i)}\|_2 - \sum_{j \neq f_c(P_i)} \|P_i - P^C_j\|_2 \Big).$$

Thus, we use the loss function as shown in equation (13) for the labeled datapoints and the estimated fault labels for the unlabeled datapoints.

3.6 Online GAN Training

We now describe how we use the FPE encoder and a co-simulation engine to train the GAN model both for the PreGAN and PreGAN+ models, summarized in Algorithm 3. Co-simulation is a technique in edge computing that allows a framework to run multiple event-based simulations with

Algorithm 3 The GAN training algorithm

Require:
Pretrained FPE Encoder E
Generator and Discriminator networks Gen, $Disc$
Co-Simulator Sim
1: Initialize weights in Gen, $Disc$.
2: **for**($t = 1$ to T)
3: $D^A, P \leftarrow E(S_t, W_t)$
4: Get E^F by (6)
5: $\Delta = Gen(S_t, E^F)$ ▷ Generative preemptive migration
6: $N = S_t + \Delta$ ▷ Form new scheduling decision
7: $D = Disc(S_t, N)$ ▷ Compare the two decisions
8: Calculate L_D using (18)
9: Update weights of $Disc$ using L_D
10: Calculate L_G using (19)
11: Update weights of Gen using L_G

Algorithm 4 The PreGAN/PreGAN+ testing algorithm

Require:
Pretrained models E, Gen, $Disc$
1: **for**($t = 1$ to T)
2: $D^A, P \leftarrow E(S_t, W_t)$
3: Get E^F by (6)
4: $\Delta = Gen(S_t, E^F)$ ▷ Generative preemptive migration
5: $N = S_t + \Delta$ ▷ Form new scheduling decision
6: $D = Disc(S_t, N)$ ▷ Compare the two decisions
7: **if** PreGAN+
8: Fine-Tune E using Algorithm 2
9: Fine-Tune G and D using Algorithm 3
10: Execute final schedule obtained from (9)
11: **if**($D[1] \geq D[0]$)
12: **return** N ▷ Return new decision if higher score
13: **return** S

different parameters like scheduling decisions to optimize the QoS of the system [19]. As event-based co-simulators can provide us QoS estimates quickly, they allow us to avoid running decisions on the physical setup, saving time and execution costs. Thus, unlike a traditional GAN model, we train our discriminator in a self-supervised manner using a co-simulator.

Our GAN training runs in an online fashion instead of using a pre-collected dataset. For an input pair (S, W), we infer using the PreGAN model to produce as final schedule as N or S. We then run the co-simulator with the two scheduling decisions S and N and obtain the QoS scores. Simulation is repeated at every invocation of PreGAN and the generated scores are then used to fine-tune online the GAN model. Specifically, these scores may be calculated using a combination of metrics such as energy consumption, response time and SLO violations [20], [32]. We denote the co-simulated QoS scores by $\mathrm{Sim}(S)$ and $\mathrm{Sim}(N)$. To make the likelihood score output of the discriminator correspond to the co-simulated scores, we use the binary cross-entropy loss as

$$L_D = \frac{1}{2}\Big(\mathbb{1}(\mathrm{Sim}(N) \geq \mathrm{Sim}(S))\Big(\log(D[1]) + \log(1-D[0])\Big) + \mathbb{1}(\mathrm{Sim}(N) < \mathrm{Sim}(S))\Big(\log(D[0]) + \log(1-D[1])\Big)\Big), \quad (18)$$

where the loss does not propagate to the generator (fixing N). This pushes the discriminator to give a higher likelihood score to the decision with a higher simulated QoS score. To train the generator, we use the adversarial loss

$$L_G = \frac{1}{2}\Big(\log(D[1]) + \log(1 - D[0])\Big), \quad (19)$$

where the loss propagates to the generator with the discriminator weights kept fixed. This is equivalent to Eq. (18) but with $\mathrm{Sim}(N) \geq \mathrm{Sim}(S)$, pushing the generator to output a migration decision Δ which gives a schedule N better than the original schedule S. To do this, the generator gets fault class embeddings from the FPE encoder. Akin to prior work [20], [32], we use a convex combination of energy consumption and average response time as the Sim score. Specifically, for a decision D, we denote the average energy consumption by AEC_D and average response time as ARC_D. Thus,

$$\mathrm{Sim}(D) = \beta \cdot AEC_D + (1 - \beta) \cdot ART_D. \quad (20)$$

This style of training a discriminator model has two benefits: it allows us to run our model inference without the co-simulation at inference time, eliminating the computationally expensive generation of simulated traces (except for fine-tuning).

3.7 Online Inference

We now describe the inference procedure using the trained PreGAN or PreGAN+ models (summarized in Algorithm 4). For any input time-series window and scheduling decision W, S, the PreGAN model outputs D, N. Based on the likelihood scores of the discriminator D, PreGAN predicts if the preemptive migration decision would improve QoS or not. If it does, *i.e.*, $D[1] \geq D[0]$, N is executed, otherwise the original decision S is executed (lines 11-13 in Alg. 4). Also, in case of PreGAN+, we fine-tune the FPE encoder using new data with fixed ξ fraction as labeled (process described in Section 4.3, where ξ is a hyper-parameter). ξ controls the overhead imposed by the fault labelling engine and the amount of labelled data available to tune the FPE encoder of PreGAN+ (more details in Section 4.7). We also tune GAN using the co-simulating both decisions S and N.

Overall, the FPE in PreGAN allows us proactively predict faults and guide the GAN model to generate a preemptive migration decision over the schedule generated by a policy oblivious to future system faults. The discriminator of the GAN then performs a cost-benefit analysis over the original scheduling decision to avoid excessive migration overheads. This allows PreGAN to optimize scheduling decisions and cut down execution costs.

Computational Complexity. Assuming that the inference of a deep neural network is an $O(1)$ operation, we provide the computational complexity in the Big-O notation. We represent the number of host machines as $|H|$. The inference using the FPE encoder is a $O(1)$ operation. To generate the host embeddings, *i.e.*, eq. (12), we have $O(|H|)$ operations. The final scheduling decision is also generated

using the comparison of $D[1]$ and $D[0]$ as in eq. (9), which is a $O(1)$ operation. Hence, the overall complexity of the model is $O(|H|)$, which makes it linear with the size of the host set.

4 Experiments

We compare the proposed methods, PreGAN and PreGAN+ with the state-of-the-art baselines: DFTM [9], ECLB [7], PCFT [4] and CMODLB [8] (more details in Section 2). Other heuristic based approaches have been omitted for brevity as these baselines outperform those in all the metrics we consider in our experiments. We use hyperparameters of the baseline models as presented in their respective papers. We train all deep learning models using the PyTorch-1.8.0 [40] library[1].

4.1 Evaluation Setup

Our evaluation testbed is a cluster with 16 Raspberry Pi 4B nodes. Our cluster contains 8 nodes with 4-GB RAM and another 8 but with 8-GB RAM. The power consumption models are taken from the commonly-used SPEC benchmarks repository [41]. We run all experiments for 100 scheduling intervals, with each interval being 300 seconds long, giving a total experiment time of 8 hours 20 minutes. We average over 5 runs and use diverse workload types to ensure statistical significance.

For our workloads, we consider the *DeFog* applications: Yolo, PocketSphinx and Aeneas [42]. DeFog is a fog computing benchmark suite that consists of various real-world application instances. The three specific applications used in our experiments were considered due to their heterogeneous resource requirements and volatile characteristics. At the beginning of every scheduling interval, we create $Poisson(\lambda)$ new workloads with $\lambda = 5$, sampled uniformly from the three applications. Poisson distribution is a natural choice for a bag-of-tasks workload model, common in edge environments [3], [32].

We emulate mobility of the edge nodes using the `NetLimiter` tool that varies the communication latency of each worker node with the broker using the mobility model described by Gilly et al. [43]. Specifically, using the mobility traces from the Simulation of Urban Mobility (SUMO) tool [44], we set the latency and bandwidth parameters of each edge host. SUMO simulates mobile vehicles in a smart-city like environment. SUMO provides us with values of the `ping` time and network bandwidth for each node in the environment that we set using `NetLimiter`.

4.2 Evaluation Metrics

For fault detection, we consider detection accuracy, precision, recall and F1 scores. For a test datapoint $\{W_t, S_t, \hat{y}_t\}$ with the PreGAN or PreGAN+ outputs D and N, the predicted and ground-truth labels are obtained as

$$\mathbb{1}(D[1] > D[0]) \text{ and } \bigwedge_{i=1}^{m} \mathbb{1}(\hat{y}_{t,i} > 0)$$

1. All model training and experiments were performed on a system with configuration: Intel i7-10700K CPU, 64GB RAM, Nvidia RTX 3080 and Windows 10 Pro OS. This was also the broker node in our setup.

respectively. This ground-truth label is 1 when any of the edge hosts has a fault and 0 otherwise.

For fault diagnosis, we consider two commonly used metrics: (1) $\mathrm{HitRate@100\%}$ is the measure of how many ground truth dimensions have been included in the top candidates predicted by the model [37], (2) Normalized Discounted Cumulative Gain ($\mathrm{NDCG@100\%}$) [45]. Further, to compare the overheads of the models, we also measure the ratio of the time to update input scheduling decision to the total decision making time and call this the "overhead ratio". For fault classification, we consider the classification accuracy. For test time, we also consider an "improvement ratio", which for an execution of T intervals is calculated as:

$$\text{Improvement Ratio} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}(D_t[1] > D_t[0]), \qquad (21)$$

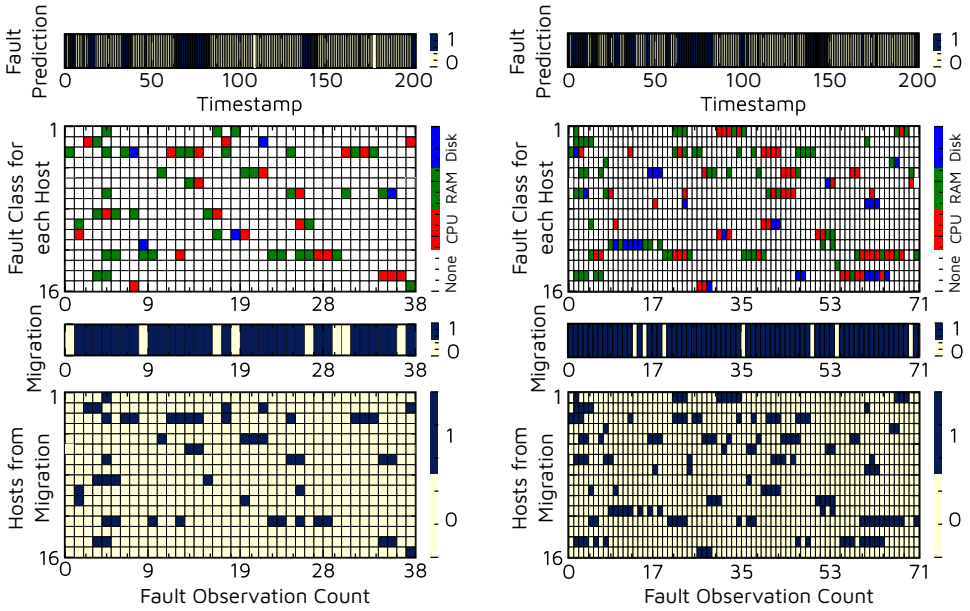
where D_t denotes the values at the t-th timestep. This metric denotes the ratio of the times the model is able to predict a better scheduling decision than the original. This gives us an indication of the how well the preemptive migration prediction performs compared to the original scheduling decision. Together with the detection accuracy this gives us a complete picture on how well the model performs compared to the case where there is no preemptive migration. To explain these metrics with a simple visualization example on 200 intervals, consider Figure 4. The top heatmap denotes the intervals at which the model predicted a fault (fault denoted with black and yellow otherwise), specifically 38 intervals out of 200 for PreGAN and 71 for PreGAN+. The second heatmap shows the fault class prediction for each host (if any). The third heatmap shows the intervals in which the likelihood score of N was higher than S (30 out of 38 for PreGAN and 64 out of 71 for PreGAN+) and the fourth heatmap shows the hosts from which there was a task migrated. Here, the improvement ratio for PreGAN is $30/38 = 0.7895$ and for PreGAN+ is $64/71 = 0.9014$. Also, we observe a strong correlation between the fault class predictions and the host selected for task migration, thanks to the factored prediction in PreGAN and PreGAN+.

We also consider standard QoS metrics including energy consumption, response time, fraction of SLO violations, migration count and resource utilization.

4.3 Implementation and Training

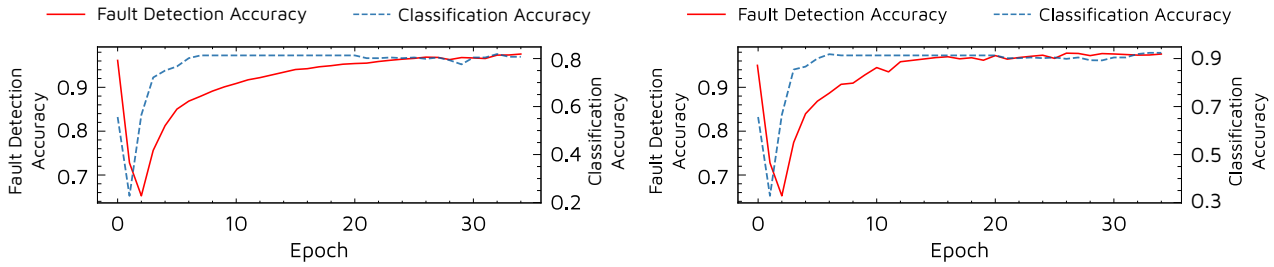
To conduct our tests, we use the COSCO framework that supports container orchestration in distributed edge clusters [19]. COSCO is at present the only framework that allows the generation of QoS scores using co-simulated traces. For a fair comparison, we use a common underlying scheduling policy, GOBI [19], that generates scheduling decisions (S_t) by optimizing QoS scores using a deep neural network based surrogate model.

To generate the ground-truth fault labels and classes, we use the Anomaly Detection Engine for Linux Logs (ADE) tool [46]. The output anomaly classes for the tool are: CPU over-utilization (CPUO), Abnormal disk utilization (ADU), Memory leak (MEL), Abnormal memory allocation (AMA) and Network overload (NOL). CPUO occurs when the CPU utilization exceed 90%. When the disk controller of a system throttles read/write operations, an ADU fault



(a) PreGAN (b) PreGAN+

Figure 4: Visualization of fault prediction, classification and migration decisions with execution intervals.



(a) PreGAN (b) PreGAN+

Figure 5: Training curves for the Fault Prototype Encoder.

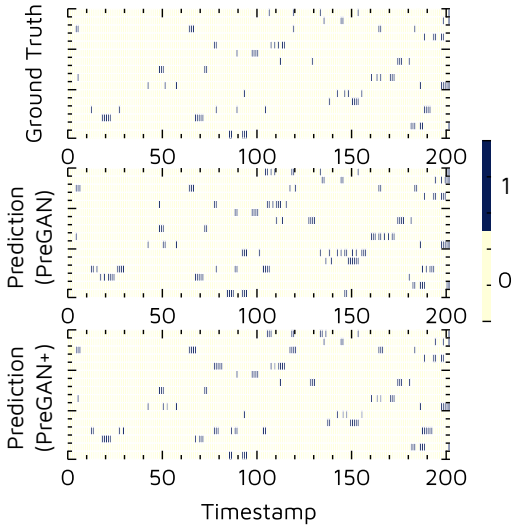


Figure 6: Fault Prediction using PreGAN and PreGAN+ models with Ground Truth labels.

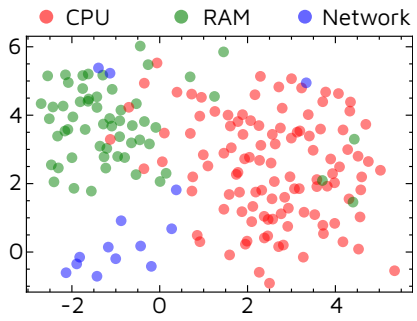
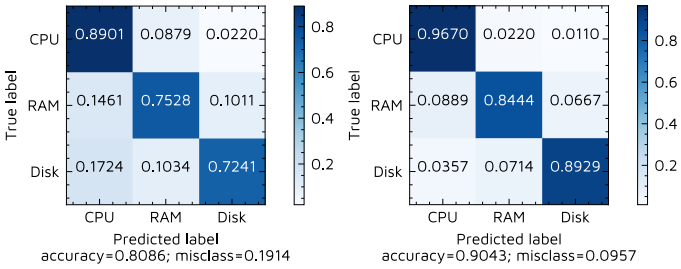


Figure 7: t-SNE plot of prototypes generated for the test set by the PreGAN+ model.

occurs. MEL occurs when there is incremental allocation of 1 MB memory every 3 seconds. When RAM I/O utilization exceed 90%, it causes an AMA fault. NOL fault occurs when network buffer overloads and temporary disk space needs to be allocated for file transfers. All faults events are raised when the above conditions hold for at least 60 seconds. For simplicity, the CPUO anomaly is classified as a CPU fault, NOL/ADU are clubbed together as a Network fault (as network buffer overloads almost always result in disk faults) and MEL/AMA are clubbed together as a RAM



(a) PreGAN (b) PreGAN+

Figure 8: Confusion matrices for fault classification on the test set.

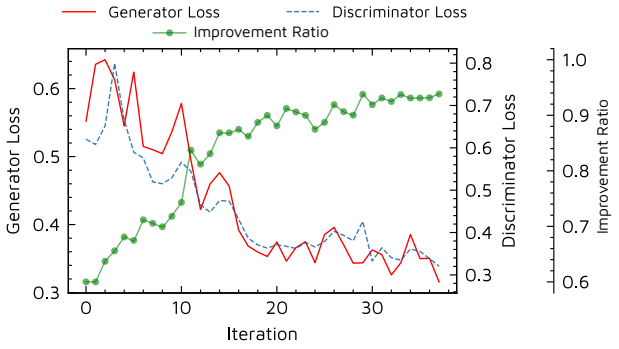


Figure 9: Training of the GAN network.

fault (due to high correlation between these two). Thus, we consider 3 fault types/classes.

To train the FPE and GAN models, we use the AdamW optimizer [47]. We train the FPE encoder and GAN using a traces of lengths 1000 and 1200 scheduling intervals respectively. These numbers were obtained using the early-stopping convergence criterion. The QoS score obtained from the co-simulations was a convex combination of the normalized energy consumption and SLO violations [19]. The hyperparameter values used in our experiments for PreGAN+ were obtained using grid search over the range of values with granularity of 0.1 as per prior work [3], [20]. We use the prototype embedding size of 8, windows size of 5, initial value of α as 0.9, β as 0.5 and decay rate ϵ as 0.05.

Figure 5 shows how the FPE encoder converges in offline training using 80% of the dataset collected from traces of 1000 intervals (rest as test set). As the model is trained, its fault detection and classification accuracies improve. We also observe that the Transformer model allows the FPE in PreGAN+ to give improved detection and classification accuracies due to the historic context being given as input for more informed decision making. Figure 6 shows the predicted and ground-truth fault labels on the test set, with hosts on the y-axis and intervals on the x-axis. We observe PreGAN+ performing better for fault diagnosis compared to PreGAN. Figure 7 shows a t-SNE plot of the class prototypes predicted on the test set using the FPE of PreGAN. Clearly, the prototype embeddings of the same class are clustered together, demonstrating that the model is able to distinguish among the different fault types. Classification confusion matrix on the test set can be seen in Figure 8. The prototype embeddings generation allows the model to

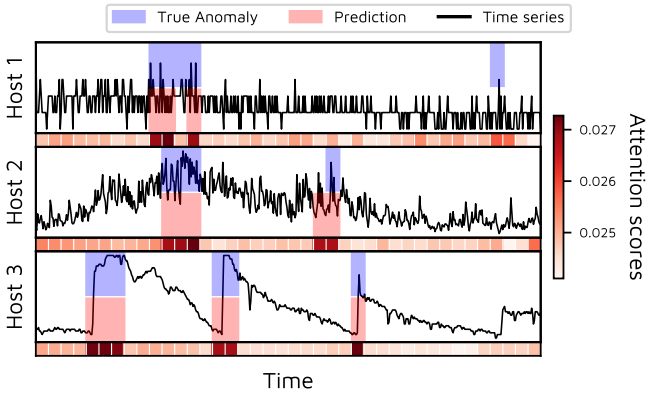


Figure 10: Visualization of Attention scores.

generalize and accurately classify even previously unseen time-series inputs into one of the three classes. After training the FPE, we train the GAN. Figure 9 shows the discriminator (L_D) and generator (L_G) loss values (for PreGAN) with the Improvement Ratio for the first 40 iterations, *i.e.*, the intervals with faults in the GAN training dataset.

4.4 Visualization of Attention Scores

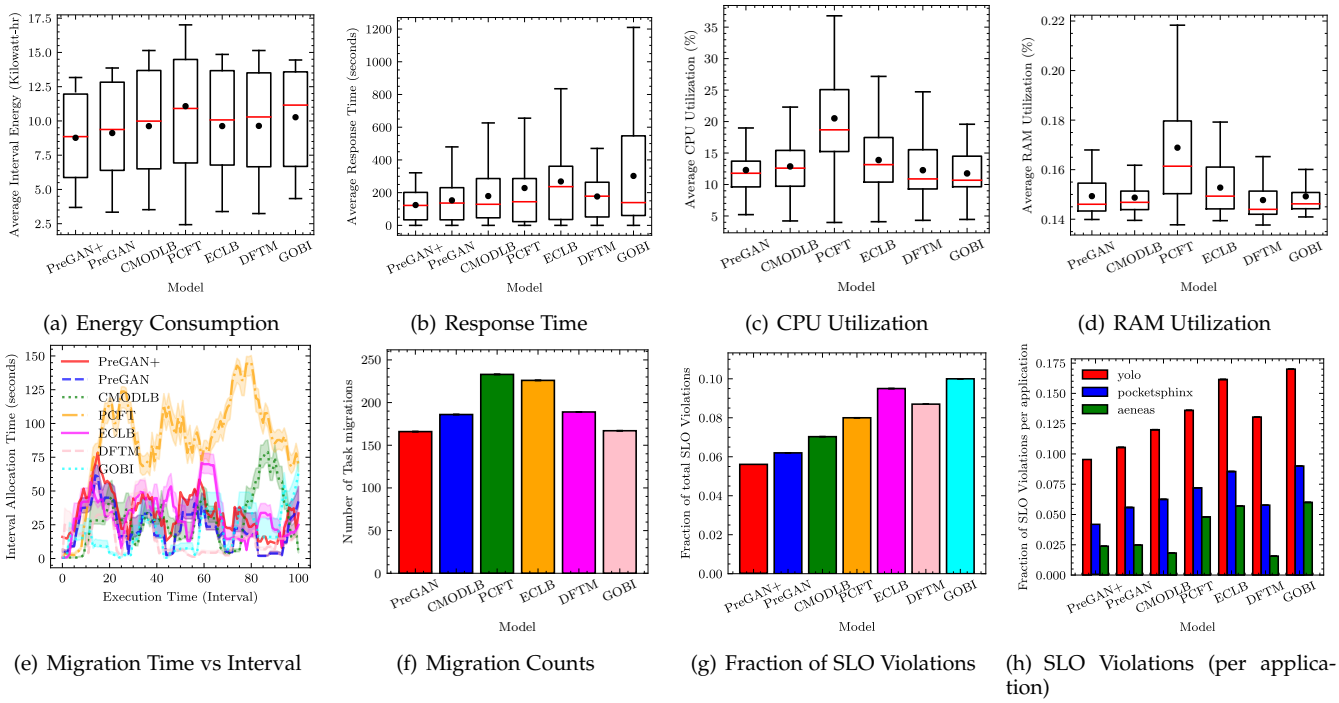
Figure 10 visualizes the attention scores for the FPE encoder of the PreGAN+ model. The model is trained on a dataset collected from a real-setup as described in Section 4.3. We show the time-series, the average attention weights for each window (averaged over multiple heads and shown by the red heatmap) for the CPU utilization of the first 3 hosts in the system. It is apparent that the attention scores are highly correlated with the peaks in the data. This allows the model to specifically detect faults in each host individually. As shown in Figure 10, the faults are detected in those hosts for which the attention scores are high.

4.5 Results

We now describe the results corresponding to 100 interval runs on the testbed using the DeFog workloads. The ground-truth labels are obtained offline, after the complete execution to compare the detection and diagnosis performance of the models. As per prior work, we give equal weights to energy consumption and average response time in Eq. 20 by having $\beta = 0.5$. The trends discussed next follow also for other values of β. We also keep $\xi = 0.4$ as per the sensitivity analysis given in Section 4.7. Table 3 shows the detection and diagnosis metrics with the improvement ratios. It also shows the overhead ratio in terms of the time it takes to run the preemptive migration model relative to the scheduling model. The overhead ratio is highest for CMODLB due to the several sequential steps in its model including k-means clustering, ANN inference, and PSO. DFTM has the lowerst overhead, but still performs poorly as it uses a basic sinusoidal thermal trend for all models and do not consider host heterogeneity. On average, the accuracy, precision, recall and the F1 score of the PreGAN and PreGAN+ models are higher than the baselines. This is due to the late-fusion of the embeddings obtained by the

Table 3: Performance scores of PreGAN and PreGAN+ and the baseline methods with standard deviation. The best scores are shown in bold.

Method	Detection				Diagnosis		Overhead Ratio	Improvement Ratio
	Accuracy	Precision	Recall	F1 Score	HR@100	NDCG@100		
DFTM	0.8731 ±0.0234	0.7713 ±0.0823	0.8427 ±0.0199	0.8054 ±0.0872	0.5129 ±0.0212	0.4673 ±0.0019	**0.0413 ±0.0021**	0.3783 ±0.1001
ECLB	0.9413 ±0.0172	0.7812 ±0.0711	0.8918 ±0.0203	0.8329 ±0.0901	0.4913 ±0.0010	0.5239 ±0.0024	0.1028 ±0.0009	0.5912 ±0.0341
PCFT	0.8913 ±0.0108	0.8029 ±0.0692	0.9018 ±0.0165	0.8495 ±0.0312	0.5982 ±0.0094	0.5671 ±0.0020	0.0913 ±0.0014	0.6824 ±0.0473
CMODLB	0.9128 ±0.0112	0.8158 ±0.0343	0.9013 ±0.0091	0.8605 ±0.0284	0.6309 ±0.0025	0.5432 ±0.0031	0.2123 ±0.0003	0.7283 ±0.0065
PreGAN+ (SS)	0.9638 ±0.0132	0.8744 ±0.0123	0.9023 ±0.0082	0.8881 ±0.0193	0.6366 ±0.0109	0.5901 ±0.0031	0.1922 ±0.0004	0.7622 ±0.0058
PreGAN+ (GT)	0.9728 ±0.0291	0.8819 ±0.0193	0.9281 ±0.0079	0.9044 ±0.0192	0.6710 ±0.0012	0.6099 ±0.0029	0.2001 ±0.0002	0.8092 ±0.0102
PreGAN+ (T)	0.9789 ±0.0099	0.8792 ±0.0727	0.9298 ±0.1018	0.9038 ±0.0193	0.6715 ±0.0061	0.6092 ±0.0010	0.2101 ±0.0010	0.7828 ±0.0070
PreGAN	0.9635 ±0.0092	0.8723 ±0.0221	0.9018 ±0.0121	0.8868 ±0.0629	0.6232 ±0.0069	0.5898 ±0.0080	0.1617 ±0.0017	0.7605 ±0.0060
PreGAN+	**0.9813 ±0.0087**	**0.8827 ±0.0059**	**0.9312 ±0.0076**	**0.9063 ±0.0384**	**0.6728 ±0.0028**	**0.6109 ±0.0041**	0.2092 ±0.0001	**0.9014 ±0.0026**



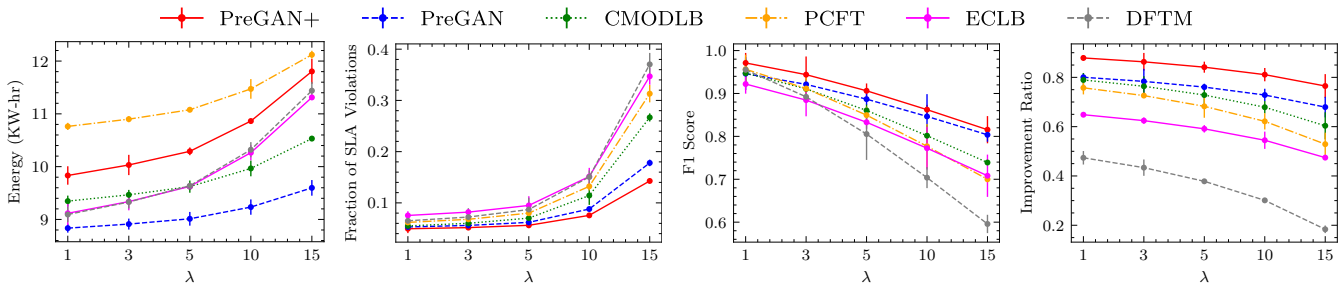
(a) Energy Consumption (b) Response Time (c) CPU Utilization (d) RAM Utilization

(e) Migration Time vs Interval (f) Migration Counts (g) Fraction of SLO Violations (h) SLO Violations (per application)

Figure 11: Comparison of QoS parameters of PreGAN against baselines.

feature (GAT) and temporal trend extraction using GRU or Transformers. This allows the proposed models to not only exploit the correlations across host devices, but also detect sudden deviations from the expected resource utilization characteristics across time. Overall, scores are highest for the PreGAN+ model, thanks to the historic context given as input to the Transformer model as well as its ability to adapt in non-stationary settings by periodic fine-tuning. CMODLB uses k-means clustering to identify a common utilization threshold, irrespective of the running workloads or the host capacities. This gives higher fault detection scores for the PreGAN and PreGAN+ models. For fault-diagnosis, the PreGAN+ method has the highest HitRate (0.6728) with CMODLB being second (0.6309). The NDCG score of the PreGAN+ model is the highest being 0.6109. This is due to the factored fault prediction in the PreGAN+ model. Finally, the improvement ratio of the PreGAN+ model is the highest, *i.e.*, 0.9014 improving from 0.7283 of the CMODLB model by 23.76%. In terms of overhead, PreGAN has lower overhead compared to PreGAN+ as PreGAN+ additionally runs the fine-tuning steps and a more sophisticated Transformer based neural network.

Figure 11 compares the QoS scores of all models, including the vanilla GOBI approach without any fault-tolerant preemptive migrations. The PreGAN+ model has the lowest average energy consumption of 8.8721 KW-hr, with PreGAN being next at 9.0121 KW-hr. This is due to the relatively lower average CPU and RAM utilizations (Figures 11(c) and 11(d)) and the energy based QoS score when training the GAN model using co-simulations (see Eq. 20). Figure 11(b) shows the average response times with response time being defined as the time between the creation of a task from an IoT sensor and the gateway receiving the response. Among the baselines, the DFTM approach has the lowest response time as it uses minimum-migration-time heuristic [9] for task migrations (Fig. 11(e)). PreGAN and PreGAN+ avoid unnecessary migrations to prevent avoidable use of network resources, improving overall system reliability (Fig. 11(f)). Even with higher response times, CMODLB and achieves lowest SLO violation rates (0.0717) among the baselines as



(a) Energy Consumption (b) Fraction of SLO Violations (c) F1 Score (d) Improvement Ratio

Figure 12: Sensitivity Analysis for all models with λ (parameter of the discrete Poisson distribution).

its migration decisions are SLO aware. PreGAN and PreGAN+ have SLO violation rates of 0.0621 and 0.0568, respectively, which are 13.4% and 20.8% lower than CMODLB. Figure 11(h) shows the SLO violation rates for each application. For all models, Yolo has the highest violation rate due to its compute heavy utilization characteristics. The migration decisions in PreGAN and PreGAN+ use the fault class labels to decide the target hosts in the migration decisions. The fine-grained classification in the proposed models allows them to migrate CPU intensive tasks from a compute constrained node to a node with low CPU utilization, even if the target node is running at capacity on RAM and Disk. Compared to the binary classification in CMODLB, this gives PreGAN and PreGAN+ more choices for the target host, allowing lower overheads in migration and more balanced resource utilization.

4.6 Ablation Analysis

To study the relative importance of each component of the PreGAN+ model, we exclude every major one and observe how it affects performance. First, we consider the PreGAN+ model without the semi-supervised fine-tuning of the FPE encoder and denote this as *PreGAN+ (SS)* model. We also disable the fine-tuning of the GAN model and denote this as *PreGAN+ (GT)* model. Finally, we replace the Transformer based encoder with GRU and refer to this as *PreGAN+ (T)* model. Results are presented in Table 3. We observe that without tuning the FPE encoder or GAN the improvement ratio drops significantly, which means that periodic fine-tuning of the neural network allows the model to adapt to dynamic scenarios and adjust to changes in the environment. Further, replacing the Transformer based encoder with a GRU, as in PreGAN, makes the training very slow as GRU requires iterative training whereas a Transformer can be given the whole sequence (C in Fig. 3) at once. Thus, periodic fine-tuning supports the adaptability of PreGAN+ and Transformer based FPE encoder makes tuning at each interval time-efficient.

4.7 Sensitivity Analysis

Sensitivity with λ. Figure 12 shows the variation of the energy consumption, SLO violations, F1 score and improvement ratio with the λ parameter in our Poisson distribution used to model the workloads. We vary λ from 1 to 15 ($\lambda = 15$ constantly gives $> 90\%$ CPU utilization for all hosts). Under a higher λ more tasks are produced, making the fault prediction harder. This is apparent from the drop in the F1 scores, leading to higher SLO violations. Even the energy consumption increases due to the increase in the average CPU utilization of the system. Overall, PreGAN shows the least relative drop in F1 scores and improvement ratio as we increase λ giving the least SLO violations even in workload heavy executions.

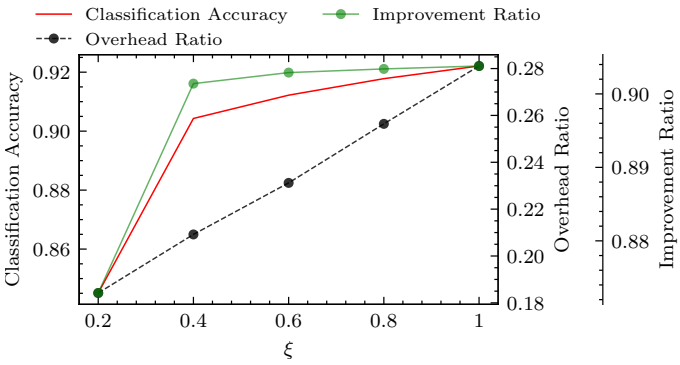


Figure 13: Sensitivity Analysis for PreGAN+ with ξ (fraction of labeled data).

Table 4: Comparison of the load on broker node.

Model	Overhead Ratio	CPU (%)	RAM (GB)
PreGAN	0.1617	28%	4.3 GB
PreGAN+	0.2902	56%	16.2 GB

Sensitivity with ξ. Figure 13 demonstrates the variation of the F1 score, overhead ratio and improvement ratio with the fraction of data that is labeled by ADE when periodically fine-tuning the FPE encoder of PreGAN+. Naturally, as we increase ξ, the amount of data to label increases, which consequently increases the overhead ratio. However, with more labeled data, the model is able to classify better and hence the classification score improves as ξ increases. Thus, there is a tradeoff between the decision making time and the performance of the model. As the classification accuracy saturates for $\xi \geq 0.4$, labelling more data does not improve significantly improve performance, but linearly increases the overhead. This is shown by the improvement ratio, which shows negligible improvement for $\xi > 0.4$. Thus, we use $\xi = 0.4$ in our experiments.

4.8 Broker Load Comparison

Table 4 compares the CPU and RAM utilizations and the overhead ratios of the PreGAN and PreGAN+ models. Even

though PreGAN+ gives better QoS scores, it imposes high overhead on the broker node, which makes it unsuitable for settings with resource constrained broker. In such settings (broker with RAM up to 5 GB), we could use PreGAN for fault-tolerant computing that too outperforms baselines.

5 Conclusions

We have presented two preemptive migration prediction models (PreGAN and PreGAN+) that can detect, diagnose and classify faults in edge computing environments. PreGAN uses GAT and GRU for feature extraction with a Multi-Head-Attention and Prototype prediction decoder to detect and classify faults. It also leverages a generator model to utilize the anomaly class prototypes to output a delta scheduling decision (migrations) to rectify the faults and improve QoS. The discriminator model with co-simulations allow us to decide between the original and modified scheduling decisions. Moreover, the discriminator aids generator training using adversarial loss and does not require it to run co-simulations at test time. PreGAN is a static model which we extend to PreGAN+ that uses a Transformer based encoder and semi-supervised model fine-tuning to dynamically adapt in non-stationary mobile edge environments. Experiments demonstrate that PreGAN and PreGAN+ outperform baselines in terms of energy consumption, response times and SLO violations respectively. They also able to correctly identify faults, giving an average F1 score of 0.9063, higher than the state-of-the-art models.

The current model assumes a master-slave design which may not be the case in peer-to-peer models where each broker node also acts as a worker. As part of future work, we plan to explore extensions to enable deploying PreGAN in serverless platforms with streaming tasks and distributed computing environments different from master-slave. Further, we plan to explore architectural variants of the PreGAN neural network where latent space embeddings of FPE are fed directly to the generator for more informed prediction of task migration decisions.

Software Availability

All code is at https://github.com/imperial-qore/PreGAN and https://github.com/imperial-qore/PreGANPlus.

Acknowledgments

S.T. is grateful to the Imperial College London for funding his Ph.D. through the President's Ph.D. Scholarship scheme.

References

[1] S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain *et al.*, "Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet of Things*, vol. 8, pp. 100–118, 2019.

[2] S. Tuli, F. Mirhakimi, S. Pallewatta, S. Zawad, G. Casale, B. Javadi, F. Yan, R. Buyya, and N. R. Jennings, "AI Augmented Edge and Fog Computing: Trends and Challenges," *arXiv preprint arXiv:2208.00761*, 2022.

[3] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments using A3C learning and Residual Recurrent Neural Networks," *IEEE Transactions on Mobile Computing*, 2020.

[4] J. Liu, S. Wang, A. Zhou, S. A. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2016.

[5] D. Park, S. Kim, Y. An, and J.-Y. Jung, "LiReD: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks," *Sensors*, vol. 18, no. 7, p. 2110, 2018.

[6] S. Ristov, T. Fahringer, D. Peer, T.-P. Pham, M. Gusev, and C. Mas-Machuca, "Resilient techniques against disruptions of volatile cloud resources," in *Guide to Disaster-Resilient Communication Networks*. Springer, 2020, pp. 379–400.

[7] A. Sharif, M. Nickray, and A. Shahidinejad, "Fault-tolerant with load balancing scheduling in a fog-based IoT application," *IET Communications*, vol. 14, no. 16, pp. 2646–2657, 2020.

[8] S. Negi, M. M. S. Rauthan, K. S. Vaisla, and N. Panwar, "CMODLB: an efficient load balancing approach in cloud computing environment," *The Journal of Supercomputing*, pp. 1–53, 2021.

[9] V. Sivagami and K. Easwarakumar, "An improved dynamic fault tolerant management algorithm during vm migration in cloud data center," *Future Generation Computer Systems*, vol. 98, pp. 35–43, 2019.

[10] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and privacy in cloud computing: A survey," in *2010 Sixth International Conference on Semantics, Knowledge and Grids*. IEEE, 2010, pp. 105–112.

[11] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2011.

[12] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[13] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, "Proactive fault tolerance using preemptive migration," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2009, pp. 252–257.

[14] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 864–872.

[15] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4080–4090.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[18] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 703–716.

[19] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, 2022.

[20] S. Tuli, G. Casale, and N. R. Jennings, "PreGAN: Preemptive Migration Prediction Network for Proactive Fault-Tolerant Edge Computing," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2022.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6000–6010.

[22] A. Rawat, R. Sushil, A. Agarwal, A. Sikander, and R. S. Bhadoria, "A new adaptive fault tolerant framework in the cloud," *IETE Journal of Research*, pp. 1–13, 2021.

[23] B. Ray, A. Saha, S. Khatua, and S. Roy, "Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment," *IEEE Transactions on Cloud Computing*, 2020.

[24] B. Mohammed, M. Kiran, K. M. Maiyama, M. M. Kamala, and I.-U. Awan, "Failover strategy for fault tolerance in cloud computing environment," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1243–1274, 2017.

[25] X. Hu, Y. Li, L. Jia, and M. Qiu, "A novel two-stage unsupervised fault recognition framework combining feature extraction and fuzzy clustering for collaborative AIoT," *IEEE Trans. on Industrial Informatics*, 2021.
[26] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
[27] A. Satpathy, S. K. Addya, A. K. Turuk, B. Majhi, and G. Sahoo, "Crow search based virtual machine placement strategy in cloud data centers with live migration," *Computers & Electrical Engineering*, vol. 69, pp. 334–350, 2018.
[28] L. Wang, W. Mao, J. Zhao, and Y. Xu, "DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, 2021.
[29] S. M. Ataallah, S. M. Nassar, and E. E. Hemayed, "Fault tolerance in cloud computing-survey," in *2015 11th International computer engineering conference (ICENCO)*. IEEE, 2015, pp. 241–245.
[30] C. Li, M. Cerrada, D. Cabrera, R. V. Sanchez, F. Pacheco, G. Ulutagay, and J. Valente de Oliveira, "A comparison of fuzzy clustering algorithms for bearing fault diagnosis," *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 6, pp. 3565–3580, 2018.
[31] T. Long, Y. Ma, L. Wu, Y. Xia, N. Jiang, J. Li, X. Fu, X. You, and B. Zhang, "A novel fault-tolerant scheduling approach for collaborative workflows in an edge-iot environment," *Digital Communications and Networks*, 2022.
[32] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with megh: Efficient live migration of virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1786–1801, 2019.
[33] 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*, 2020, pp. 3395–3404.
[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, 2014, pp. 2672–2680.
[35] S. Tuli, R. Bansal, R. Paul *et al.*, "TANGO: Commonsense Generalization in Predicting Tool Interactions for Mobile Manipulators," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.
[36] S. Tuli, G. Casale, L. Cherkasova, and N. R. Jennings, "Deepft: Fault-tolerant edge computing using a self-supervised deep surrogate model," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.
[37] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data," *Proceedings of VLDB*, vol. 15, no. 6, pp. 1201–1214, 2022.
[38] X. Dong and J. Shen, "Triplet loss in siamese network for object tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 459–474.
[39] A. Shukla, G. S. Cheema, and S. Anand, "Semi-supervised clustering with neural networks," in *2020 IEEE Sixth International Conference on Multimedia Big Data*. IEEE, 2020, pp. 152–161.
[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.
[41] Standard Performance Evaluation Corporation. Spec power consumption models. [Online]. Available: https://www.spec.org/cloud_iaas2018/results/
[42] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "DeFog: fog computing benchmarks," in *The 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 47–58.
[43] K. Gilly, S. Alcaraz, N. Aknin, S. Filiposka, and A. Mishev, "Modelling edge computing in urban mobility simulation scenarios," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 539–543.
[44] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International journal on advances in systems and measurements*, vol. 5, no. 3&4, 2012.
[45] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.
[46] Open Mainframe Project. Anomaly Detection Engine for Linux Logs (ADE). [Online]. Available: https://www.openmainframeproject.org/projects/anomaly-detection-engine-for-linux-logs-ade
[47] N. Saleh and M. Mashaly, "A dynamic simulation environment for container-based cloud data centers using containercloudsim," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*. IEEE, 2019, pp. 332–336.

Shreshth Tuli is a President's Ph.D. Scholar at the Department of Computing, Imperial College London, UK. Prior to this he was an undergraduate student at the Department of Computer Science and Engineering at Indian Institute of Technology - Delhi, India. His research interests include Internet of Things (IoT), Fog Computing and Deep Learning.

Giuliano Casale joined the Department of Computing at Imperial College London in 2010, where he is currently a Reader. Previously, he worked as a research scientist and consultant in the capacity planning industry. His research work has received multiple awards, recently the best paper award at ACM SIGMETRICS. He serves on the editorial boards of IEEE TNSM and ACM TOMPECS and as current chair of ACM SIGMETRICS.

Nicholas R. Jennings is the Vice-Chancellor and President of Loughborough University. Before Loughborough, he was the Vice-Provost for Research and Enterprise and Professor of Artificial Intelligence at Imperial College London, UK's first Regius Professor of Computer Science (a post bestowed by the monarch to recognise exceptionally high quality research) and the UK Government's first Chief Scientific Advisor for National Security.