

# CILP: Co-simulation based Imitation Learner for Dynamic Resource Provisioning in Cloud Computing Environments

Shreshth Tuli, Giuliano Casale and Nicholas R. Jennings

**Abstract**—Intelligent Virtual Machine (VM) provisioning is central to cost and resource efficient computation in cloud computing environments. As bootstrapping VMs is time-consuming, a key challenge for latency-critical tasks is to predict future workload demands to provision VMs proactively. However, existing AI-based solutions tend to not holistically consider all crucial aspects such as provisioning overheads, heterogeneous VM costs and Quality of Service (QoS) of the cloud system. To address this, we propose a novel method, called CILP, that formulates the VM provisioning problem as two sub-problems of prediction and optimization, where the provisioning plan is optimized based on predicted workload demands. CILP leverages a neural network as a surrogate model to predict future workload demands with a co-simulated digital-twin of the infrastructure to compute QoS scores. We extend the neural network to also act as an imitation learner that dynamically decides the optimal VM provisioning plan. A transformer based neural model reduces training and inference overheads while our novel two-phase decision making loop facilitates in making informed provisioning decisions. Crucially, we address limitations of prior work by including resource utilization, deployment costs and provisioning overheads to inform the provisioning decisions in our imitation learning framework. Experiments with three public benchmarks demonstrate that CILP gives up to 22% higher resource utilization, 14% higher QoS scores and 44% lower execution costs compared to the current online and offline optimization based state-of-the-art methods.

**Index Terms**—Resource Provisioning, Cloud Computing, Co-Simulation, Imitation Learning.

## I. INTRODUCTION

THE past years have seen widespread adoption of the cloud computing paradigm due to its flexibility, low maintenance and security. The success of the cloud is attributed to the use of virtualization that allows the deployment of independent virtual machines (VMs) on physical machines (PMs) [1], [2]. VMs allow us to efficiently manage the computational resources and reduce operational costs [3]. However, this underpinning technology gives rise to the challenge of efficiently managing resources to ensure optimal service delivery. This becomes crucial for public cloud providers that serve a large number of customers and even companies with private cloud infrastructures to curtail the operational costs of the cloud machines. Resource management is also paramount for users that execute workloads on cloud infrastructures

with limited cost budgets [4], [5], [6], [7]. A key aspect of resource management is VM provisioning, which instantiates and deallocates VMs based on dynamic workload demands. Most prior work aims at the automation of VM provisioning to optimize various performance measures such as energy consumption, operational cost and task response time as we also consider in this work [8], [9], [10].

**Challenges.** As users shift to workloads that rely on the integration with Internet of Things (IoT) sensors and actuators, contemporary applications have become latency-critical [11], [12]. In order to make sure that resources are available as soon as possible, proactive VM provisioning is required to avoid degradation of system performance. This entails predicting the resource demands of workloads in a future state and allocating new or deallocating existing VMs for QoS optimization. This broad-level formulation has been widely used in prior work and is commonly referred to as *predictive VM provisioning* (*PreVMP*) [13], [14], which we consider in this work. However, this problem is challenging due to the non-stationary utilization characteristics of most workloads [15], requiring methods to dynamically adapt provisioning policies to the changes in the environment.

**Existing solutions.** Several proactive VM provisioning methods have been proposed in the past. Since the optimization objectives, such as cost and response time, depend on *a-priori* unknown future utilization characteristics of workloads, the problem is challenging to solve using conventional optimization strategies. Thus, most state-of-the-art methods decompose the provisioning problem into first predicting the demands of running workloads and then optimizing the VM provisioning plan [13], [17], [20]. However, existing solutions tend to ignore crucial aspects such as provisioning overheads and heterogeneous VM costs in a cloud system. As VM provisioning entails the time-consuming creation of VMs, this can affect the response time of the new workloads that are placed on new VMs by the underlying scheduler. Moreover, deallocating VMs, in an effort to possibly reduce operation costs, requires the workloads being executed in that VM to be preemptively migrated to other active VMs in the system, further increasing the response times. Higher response times could increase the fraction of the user-defined Service Level Agreement (SLA) violations. Further, cloud providers typically offer a diverse range of VM types with disparate costs and computational capacities. Choosing the optimal set of VMs is crucial to provide the best QoS in the cloud [3], [21].

**Our contributions.** We propose a novel imitation learning

S. Tuli, G. Casale are with the Department of Computing, Imperial College London, United Kingdom.

N. R. Jennings is also with Loughborough University, United Kingdom.  
E-mails: {s.tuli20, g.casale}@imperial.ac.uk, n.r.jennings@lboro.ac.uk.

Manuscript received —; revised —.

TABLE I  
COMPARISON OF RELATED WORKS ALONG DIFFERENT PARAMETERS (✓ MEANS THAT THE CORRESPONDING FEATURE IS PRESENT).

| Work                | Method                                                  | Coupled Simulation | Heterogeneous Environment | Consider Overheads | Consider Migrations | Low Decision Time |
|---------------------|---------------------------------------------------------|--------------------|---------------------------|--------------------|---------------------|-------------------|
| ARIMA [16]+ACO [17] | Autoregressive Prediction + Meta-heuristic Optimization |                    |                           |                    |                     |                   |
| LSTM [18]+ACO [17]  | Recurrent NN + Meta-heuristic Optimization              |                    |                           |                    |                     |                   |
| Decision-NN [19]    | Deep NN prediction and Gradient based Optimization      |                    | ✓                         |                    |                     |                   |
| Semi-Direct [20]    | Dynamic Programming                                     |                    | ✓                         |                    | ✓                   |                   |
| UAHS [13]           | Gaussian Process Regression + Bayesian Optimization     |                    |                           |                    |                     |                   |
| Narya [8]           | Deep NN prediction + Multi-Armed Bandits                |                    | ✓                         |                    |                     | ✓                 |
| CAHS [14]           | Gaussian Process Regression + Bayesian Optimization     |                    |                           |                    |                     |                   |
| <b>CILP</b>         | Imitation Learning + Co-Simulated Oracle                | ✓                  | ✓                         | ✓                  | ✓                   | ✓                 |

framework that leverages a co-simulated digital-twin, *i.e.*, an approximate system model of the cloud infrastructure, to obviate the lack of overhead and cost metrics in decision optimization [22], [23]. We term this framework as Co-simulation based Imitation Learner (CILP). Our imitation learner is a composite neural network that predicts future workload demands and provisioning decisions by imitating a co-simulator based decision making oracle. This allows *CILP* to take optimal provisioning decisions, ameliorating the need for running costly simulations at test time and enabling *CILP* to efficiently scale with the size of the cloud infrastructure. Experimental evaluation with three public benchmarks demonstrates that *CILP* gives up to 22% higher resource utilization, 14% higher QoS scores, and 44% lower execution costs than the state-of-the-art techniques.

The rest of the paper is organized as follows. Section II overviews related work. Section III provides the system model assumptions, the co-simulator, underpinning scheduler and the optimization objectives. Section IV presents the CILP provisioner. A performance evaluation of the proposed method on using three public benchmark traces is shown in Section VI. Finally, Section VII concludes and presents future directions.

## II. RELATED WORK

Dynamic resource provisioning is a long-studied problem in cloud computing [24]. When workload demands are static or known, the provisioning task reduces to the classic VM provisioning problem [14]. This has been well studied in the past [25], [26]. Several approaches have been proposed that utilize threshold based algorithms [5], [27], [28], Integer Linear Programming (ILP) [29] and other estimation based approaches [30]. However, in scenarios with unknown or fluctuating demands, these methods are known to perform poorly [14]. A summary of related work is given in Table I.

As previously described, most dynamic resource provisioning methods decouple the provisioning problem into two stages: demand prediction and decision optimization [13]. This is commonly referred to as the *predict+optimize* framework in literature. For the former, a number of methods have been proposed that leverage forecasting models such as AutoARIMA [16] or LSTM neural networks [18]. For the latter, conventional methods often use Ant Colony Optimization (ACO) [17], which has been shown to exhibit state-of-the-art QoS scores in recent work [14]. Other methods, such as *Decision-NN*, combine the prediction and optimization steps

by modifying the loss function to train neural networks in conjunction with the optimization algorithm [19]. This method uses a neural network as a surrogate model to directly predict optimization objectives and uses the concept of neural network inversion, wherein the method evaluates gradients of the objective function with respect to inputs and runs optimization in the input space. The Decision-NN based approach has been shown to be better than gradient-free optimization methods, such as genetic algorithms [31]. However, continuous relaxation of the discrete optimization problem used in this work has been shown to adversely impact performance [13]. A similar method, *Semi-Direct*, utilizes dynamic programming to find the optimal provisioning decision, but offers limited scalability with workload size [20].

Recently some provisioning methods have been proposed that utilize deep reinforcement learning to provision VMs in cloud environments [8], [32], [33]. *DRL-Cloud* uses deep-Q networks to make provisioning decisions in cloud setups [32]. *DERP* uses two deep-Q networks to do the same, where having two such networks allows them to reduce bias and improve overall performance of the approach [33]. Another class of methods is referred to as *predictive autoscaling* that scale resources based on demand predictions [34], [35]. For instance, the *Autopilot* approach uses sliding windows to identify the CPU/memory limits of individual tasks [36] and others use repacking to improve system performance [37]. Similarly, other methods use predictive strategies to take more informed resource allocation and task scheduling decision [38].

In literature, the current state-of-the-art approaches are *Narya* [8], *UAHS* [13] and *CAHS* [14]. *Narya* is a popular offline approach that is built for mitigating VM interruptions in cloud machines, but can be straightforwardly extended to resource provisioning. It casts the provisioning problem into a Multi-Armed Bandit (MAB) problem, where the objective is to minimize the impact on the QoS of the running workloads. We use an adapted version of *Narya* as a baseline that uses a neural network as a surrogate model with a MAB model to decide provisioning actions. *UAHS* is an online optimization approach that leverages a Gaussian Process Regression model for estimating future workload demands and an uncertainty-based heuristic to run Bayesian Optimization over the provisioning decisions. A similar method, *CAHS*, is an extension of *UAHS* that also includes demand correlations while estimating the *utilization ratio* of cloud machines. However, the formulations developed in these works do not consider pragmatic deploy-

TABLE II  
TABLE OF MAIN NOTATION

| Notation          | Description                                                      |
|-------------------|------------------------------------------------------------------|
| $I_t$             | $t$ -th scheduling interval                                      |
| $\mathcal{H}_t$   | Set of hosts in interval $I_t$                                   |
| $\mathcal{W}_t$   | Set of workloads in $I_t$                                        |
| $c_t^j$           | CPU utilization of workload $w_t^j \in \mathcal{W}_t$            |
| $r_t^j$           | RAM utilization of workload $w_t^j \in \mathcal{W}_t$            |
| $s_t^j$           | Disk utilization of workload $w_t^j \in \mathcal{W}_t$           |
| $W_t^j$           | Feature vectors of workload $w_t^j \in \mathcal{W}_t$            |
| $\mathcal{A}_t^i$ | Set of workloads allocated to $h_t^i \in \mathcal{H}_t$          |
| $H_t^i$           | Feature vectors of host $h_t^i \in \mathcal{H}_t$                |
| $\mathcal{V}$     | Set of VM types                                                  |
| $V_t^i$           | VM type of host $h_t^i \in \mathcal{H}_t$                        |
| $\mu^i$           | Cost per unit time of VM type $V_t^i$                            |
| $P_t$             | Provisioning decision for interval $I_t$                         |
| $D_t$             | Scheduling decision for interval $I_t$ when $P_t = \emptyset$    |
| $\hat{D}_t$       | Scheduling decision for interval $I_t$ when $P_t \neq \emptyset$ |
| $\xi^i$           | Distribution of provisioning time of VM type $V_t^i$             |

ment aspects such as provisioning overheads or execution costs while optimizing the provisioning decisions. Further, for large-scale deployments (100+ VMs), these methods tend to get stuck in local optima [13]. As we empirically demonstrate later, these limitations incur performance penalties in cloud environments. We compare *CILP* against *ARIMA+ACO*, *LSTM+ACO*, *Decision-NN*, *Semi-Direct*, *UAHS*, *CAHS* and *Narya* in Section VI.

### III. PROBLEM FORMULATION

#### A. System Model

We assume a distributed cloud computing environment with multiple VMs that process a set of independent workloads. We consider a discrete-time control problem, *i.e.*, we divide the timeline into fixed size execution intervals (of  $\Delta$  time duration) and denote the  $t$ -th interval by  $I_t$ . We consider a bounded execution with  $T$  intervals; thus,  $t \in \{1, \dots, T\}$ . At  $I_t$ , the set of VM hosts is denoted by  $\mathcal{H}_t$  and the set of workloads by  $\mathcal{W}_t$ . Each workload  $w_t^j \in \mathcal{W}_t$  is characterized by its CPU utilization in terms of the number of instructions per second (IPS), denoted by  $c_t^j$ ; RAM utilization in GBs, denoted by  $r_t^j$ ; and disk storage utilization in GBs, denoted by  $s_t^j$ . Here,  $j \in \{1, \dots, |\mathcal{W}_t|\}$ . The feature vector for workload  $w_t^j$  is denoted by  $W_t^j = [c_t^j, r_t^j, s_t^j]$ . The collection of feature vectors of all workloads in  $I_t$  is denoted by  $W_t$ . In  $I_t$ , we consider a workload allocation, also referred to as a schedule, for each host. The set of workloads allocated to host  $h_t^i \in \mathcal{H}_t$  is denoted by  $\mathcal{A}_t^i \subseteq \mathcal{W}_t$ , where  $i \in \{1, \dots, |\mathcal{H}_t|\}$ .

Similar to workloads, for each host  $h_t^i \in \mathcal{H}_t$ , the feature vector includes cumulative utilization and maximum capacity of resources (CPU, RAM and disk) and is denoted by  $H_t^i = [\sum_{w_t^j \in \mathcal{A}_t^i} W_t^j, \bar{c}^i, \bar{r}^i, \bar{s}^i]$ , where  $\sum$  denotes vector sum and  $\bar{c}^i, \bar{r}^i, \bar{s}^i$  denote IPS, RAM and disk storage capacities of the host. Each host has a VM type that corresponds to a distinct set of utilization capacities and execution costs in a public

cloud deployment. We consider a static set of VM types  $\mathcal{V}$ , where the type of host  $h_t^i$  is denoted by  $V_t^i = (\bar{c}^i, \bar{r}^i, \bar{s}^i, \mu^i, \xi^i)$ . Here,  $\mu^i$  is the cost per unit time and  $\xi^i$  is the distribution of the provisioning time of a VM instance for a VM type  $V_t^i$ . We assume  $\mu^i$  and  $\xi^i$  to be stationary with time  $\forall i$ . This is common with cloud service providers; for instance, Microsoft Azure charges a constant 0.09 USD per hour for a dual-core B2s machine in its East-US datacenter. Similarly, the provisioning time for a VM type  $\xi^i$  remains constant with time as shown by prior work [39]. As elements of  $V_t^i$  are independent of time, these symbols are not sub-scripted with  $t$ .  $V_t$  denotes VM types for all hosts in  $I_t$ . A summary of the symbols is given in Table II.

#### B. VM provisioning

VM provisioning is performed at the start of each interval where we allocate new or deallocate active VMs. We denote the set of VM provisioning actions at the start of  $I_t$  by  $P_t$ , which is a collection of VM types to be provisioned with the number of instances ( $n_t \subseteq \mathcal{V} \times \mathbb{Z}$ ) and hosts to be deallocated from the system ( $d_t \subseteq \mathcal{H}_{t-1}$ ). The new set  $\mathcal{H}_t$  is the set of hosts in the previous interval  $\mathcal{H}_{t-1}$  union the provisioned hosts in  $n_t$  minus the deallocated hosts  $d_t$ . Deallocation of hosts in the system entails migrating workloads executing in those hosts to other preexisting active or newly provisioned hosts in the system. VM provisioning also entails prediction of the workload utilization characteristics for  $I_t$ , *i.e.*,  $W_t$ . We denote this prediction by  $\hat{W}_t$ , for which the provisioner may use historic data  $\{W_k | 1 \leq k < t\}$ . We denote the provisioner by  $f_{\theta}^{prov}$  that uses a scheduling decision  $\hat{D}_{t-1}$  and workload utilization metrics of the previous interval  $W_{t-1}$  to predict  $\hat{W}_t$  and  $P_t$ . As we use a neural network in our model, we use  $\theta$  to denote the weights of such a network. Thus,  $\hat{W}_t, P_t = f_{\theta}^{prov}(\hat{D}_{t-1}, W_{t-1})$ . We now formulate the underlying scheduler that generates  $D_t$ .

#### C. Underlying Scheduler

We consider the presence of a scheduler  $f^{sched}$  that predicts a schedule  $D_t$ . A scheduling decision is the placement of incoming tasks on the set of active VMs in the cloud system. The scheduler uses the feature vectors of workloads and hosts and a provisioning decision  $P_t$ . However, the set of workloads  $\mathcal{W}_{t-1}$  and hosts  $\mathcal{H}_{t-1}$  in  $I_{t-1}$  might change in the next interval  $I_t$ . Thus, the scheduler utilizes  $\hat{W}_t$  for existing workloads,  $\vec{0}$  for new workloads,  $[\vec{0}, \bar{c}^i, \bar{r}^i, \bar{s}^i]$  for new hosts and drops the feature vectors of workloads that complete execution or hosts that are deallocated at the end of  $I_{t-1}$  by the provisioner. If  $P_t$  is empty, *i.e.*, it has no provisions or deallocations, then we can use  $D_t$  directly to schedule tasks. However, for a non-empty decision  $P_t$ , the scheduler also needs to decide where to migrate tasks running in hosts that need to be deallocated. In such cases, the tasks running in the hosts that need to be deallocated need to be migrated to other active hosts before deallocating the host. The schedule that also includes these migration decisions is denoted by  $\hat{D}_t$ . Thus, with the described modification, the schedule for  $I_t$  is evaluated as  $\hat{D}_t = f^{sched}(\mathcal{H}_{t-1}, \hat{W}_t, P_t)$ . The schedule  $D_t$  is

a bipartite graph with edges from  $\mathcal{W}_t$  to  $\mathcal{H}_t$  corresponding to the placement of workloads on hosts as in prior work [40]. The set  $\mathcal{A}_t^i$ , described previously, is inferred from  $D_t$ . The graph nodes are initialized with embeddings corresponding to the feature vectors of workloads and hosts in the system. This graphical modeling of the schedule enables us to scale our neural models with the number of workloads and hosts in the cloud environment.

#### D. Co-Simulated Digital-Twin

A co-simulated digital twin, referred to as a co-simulator in the rest of the discussion, is a software that models the behavior of a physical system, which in our case is a cloud computing platform. Several methods in the past have leveraged digital twins of distributed computing environments, such as public clouds, to obviate the need for testing resource management decisions in physical platforms [41], [42]. Such simulators mimic the behaviors of the physical infrastructure and have been used to generate signals or insights to inform decision making systems [43], [44], [45]. The co-simulator stores the time-series utilization characteristics  $W_t$  and  $H_t$ . It executes a simulation of the cloud model for a given workload features  $W_{t-1}$ , VM types  $V_t$ , provisioning decision  $P_t$  and scheduling decision  $\hat{D}_t$  to generate QoS metrics for interval  $I_t$ . These metrics include energy consumption, the response time of completed workloads, SLA violation rates, execution cost and utilization ratio. It also provisions new VMs and deallocates existing ones as per  $P_t$ . We denote the co-simulator by  $f^{sim}$  and the set of QoS metrics by  $\mathcal{Q}_t = f^{sim}(W_{t-1}, V_t, P_t, \hat{D}_t)$ .

#### E. Formulation

At the start of the interval  $I_t$ , given a set of workloads  $\mathcal{W}_t$ , VM types  $\mathcal{V}$ , active hosts  $\mathcal{H}_{t-1}$  we define our problem as to find a feasible provisioning decision  $P_t$  that maximizes the CPU core utilization of hosts to avoid system under-utilization or resource wastage in private clouds. However, this may not capture the heterogeneous pricing policy adopted by cloud providers; thus, we also need to minimize the execution cost for the end user for public cloud deployments. A combination of these two may be required for hybrid cloud environments. We denote these QoS metrics for  $I_t$  in our formulation by  $r_t$  (utilization ratio) and  $\phi_t$  (cost). We define these two metrics as,

$$r_t = \frac{\sum_{w_t^j \in \mathcal{W}_t} c_t^j}{\sum_{h_t^i \in \mathcal{H}_t} \bar{c}_t^i} \text{ and } \phi_t = \sum_{h_t^i \in \mathcal{H}_t} \mu^i \cdot \Delta. \quad (1)$$

The utilization ratio is an important metric that translates to the effective usage efficiency of a cloud system and is a standard metric in prior work [13], [14]. A higher utilization ratio typically corresponds to lower energy consumption amortized over the number of completed tasks, and thus directly reflects the QoS of the system. Similarly, cost is a crucial metric for both cloud providers and users to reduce the financial footprint of workload execution [46]. These metrics also appear in prior work [4], [14] and are easy to compute in our formulation,

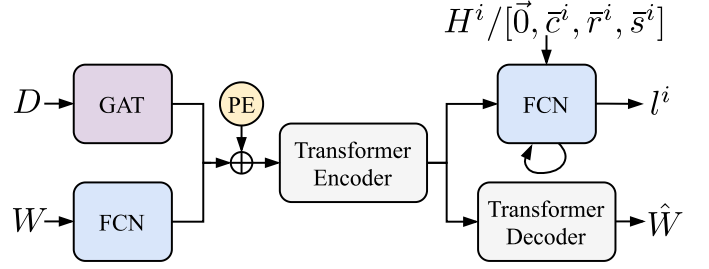


Fig. 1. *CILP* neural model encodes the schedule using a GAT and leverages a Transformer to predict utilization characteristics of the next interval and likelihood scores for each provisioning action.

assuming our VM provisioner predicts  $c_t^j$ s in equation (1). The formal optimization program is described as follows.

$$\begin{aligned} & \underset{\theta}{\text{maximize}} && \sum_{t=1}^T r_t - \gamma \cdot \phi_t \\ & \text{subject to} && \forall t, \forall w_t^j \in \mathcal{W}_t, \sum_{w_t^j \in \mathcal{A}_t^i} W_t^j \leq [\bar{c}^i, \bar{r}^i, \bar{s}^i] \\ & && \forall t, \hat{W}_t, P_t = f_{\theta}^{prov}(\hat{D}_{t-1}, W_{t-1}) \\ & && \forall t, \hat{D}_t = f^{sched}(H_{t-1}, \hat{W}_t, P_t) \\ & && \forall t, r_t, \phi_t \in f^{sim}(W_{t-1}, V_t, P_t, \hat{D}_t) \end{aligned} \quad (2)$$

for a given co-simulator  $f^{sim}$  and scheduler  $f^{sched}$  and  $t = \{1, \dots, T\}$ . In our implementation, we normalize the costs  $\mu^i$  by  $\max_i \mu^i \cdot \Delta$ . Thus, both parts of the convex combination in (2) (i.e., utilization ratio and normalized cost) are in the range  $[0, 1]$  and are unit-less quantities. Further,  $\gamma$  is a weighting parameter that can be set by a user based on the deployment scenario and as per the relative importance of the two metrics for the user. Weighting schemes are common to reduce multi-objective optimization to more efficiently solvable single-objective problems [47], [48], [49]. In particular a convex combination allows us to efficiently combine both metrics.

Even with known workload characteristics  $W_t$  at the start of  $I_t$ , the problem is known to be NP-hard [25].

### IV. TECHNICAL APPROACH

*CILP* learns to predict *a-priori* unknown workload characteristics of the next interval and provisioning decisions and is realized as a neural model  $f_{\theta}^{prov}$ . We first describe the neural network based imitation model (Section IV-A), how we infer actions (Section IV-B), train the model (Section IV-C) and how we translate actions to VM provisions (Section IV-D).

#### A. Neural Model

For interval  $I_t$ , the inputs of the neural model are schedules  $\hat{D}_{t-1}$  and workload utilizations of the previous interval  $W_{t-1}$ . To predict  $\hat{W}_t$  and  $P_t$ , we use a composite neural model by inferring the decision using a graph attention network (GAT), the utilization characteristics using a feed-forward network (also referred to as a fully-connected network or FCN) and a Transformer to capture the temporal trends in the data (see

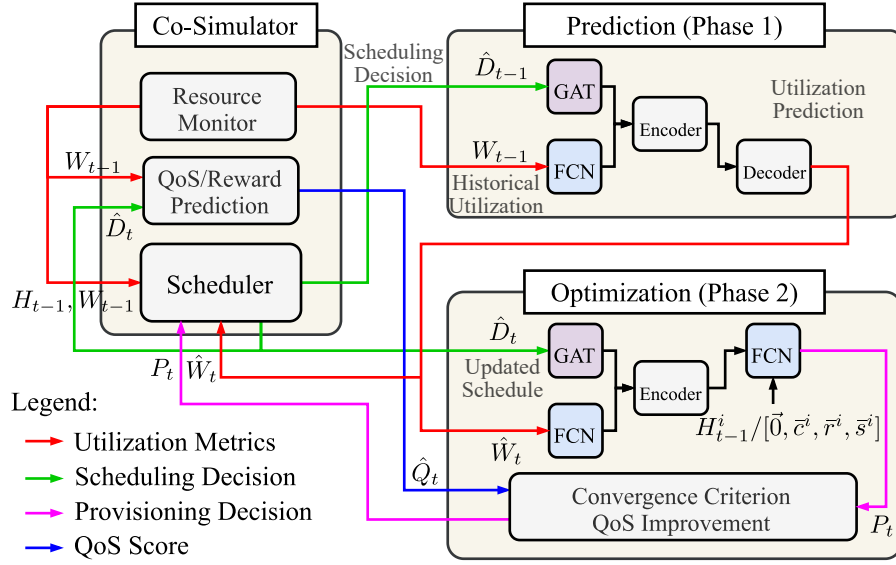


Fig. 2. Top level design of the interleaved co-simulation and prediction in *CILP*. For interval  $I_t$ , with inputs  $\hat{D}_{t-1}$  and  $W_{t-1}$ , we predict  $\hat{W}_t$  and  $P_t$  in two phases. In the first phase, the model uses  $\hat{D}_{t-1}$  and  $W_{t-1}$  to predict  $W_t$ . In the second phase the model predicts  $l_t^i$ 's auto-regressively until the co-simulated QoS score  $\hat{Q}_t$  is non-decreasing.  $\hat{D}_t$  gets updated after each provisioning action using the scheduler.

Figure 1 for an overview). An FCN, also referred to as a feed-forward network, is the most basic form of neural network layer that takes an input vector  $x$  and uses parameters: weight  $W$  and bias  $b$  to generate an output

$$y = W \cdot x + b.$$

We stack multiple such layers to create a *deep* neural network. As a result of stacking several of these layers deep neural networks of sufficient capacity are able to approximate functions of arbitrary complexity [50]. In order to achieve this, however, one needs to include non-linear activation functions between the linear layers. We use the LeakyReLU activation which is shown as

$$\text{LeakyReLU}(x) = \mathbb{1}(x \geq 0) \cdot x + \mathbb{1}(x < 0) \cdot \epsilon \cdot x,$$

for a small constant  $\epsilon$  and  $\mathbb{1}$  denoting the indicator function. Unlike the ReLU activation function that outputs a zero value and has zero gradient for negative inputs, LeakyReLU gives a non-zero gradient. This allows us to circumvent the dead-neuron problem where the model does not converge to a good optimum [51]. Transformers are sequence to sequence models that were initially proposed for NLP. However, in recent literature, they have been shown to outperform recurrent models such as Long-Short-Term-Memory (LSTM) based neural networks particularly because they do not require iterative prediction and allow time-series data to be batched, enabling us to save on training and inference time. Some recent works for resource management in edge and cloud computing environments have shown the promise of Transformers in modeling time-series patterns of workloads and hosts [45], [52], [53].

For notational convenience, while discussing the neural model, we drop the subscript  $t$  without loss in generality. The schedules are encoded as bipartite graph with nodes  $\mathcal{N} = \mathcal{W} \cup \mathcal{H}$  and relations  $(w^j, h^i) \in D$ . Each workload node  $w^j$

and host node  $h^i$  gets the embeddings  $W^j$  and  $H^i$  respectively. For a generic node  $n \in \mathcal{N}$ , we denote its embedding by  $e_n$  and its set of neighbors by  $\mathcal{S}_n$ . Following [54], we perform graph attention convolution as

$$\alpha_n = \text{softmax}_{k \in \mathcal{S}_n} (\text{LeakyReLU}(W_g^0 e_k + b_g^0)),$$

$$e_n = \sigma \left( \sum_{k \in \mathcal{S}_n} \alpha_n \cdot (W_g^1 e_k + b_g^1) \right), \quad (3)$$

where  $\alpha_n$  are the attention weights for node  $n$ . GAT enables the method to efficiently scale with the number of workloads and hosts in the system [55]. When we replace GAT with a feed-forward network to infer over the feature vectors of the nodes, the prediction mean-square-error increases by at least 50% and the inference time by 7% for the test setups described in Section VI. The stacked representation for all nodes ( $e_n$ ) is represented as  $E^G$ . We also pass the normalized input  $W$  of the previous interval through a feed-forward network with sigmoid activation such that

$$E^W = \sigma(\text{FeedForward}(W)). \quad (4)$$

We then pass the concatenated vector  $E = [E^G, E^W]$  through a Transformer encoder after adding positional encodings (PE) [56] using multi-head self-attention, giving an encoded representation,

$$E^0 = \text{TransformerEncoder}(E). \quad (5)$$

Early fusion of  $E^W$  and  $E^G$  enables the downstream predictors to exploit them together. Using  $E^0$ , we then predict an estimate of utilization characteristics of the next interval, *i.e.*  $\hat{W}$ , we use a Transformer decoder as

$$\hat{W} = \text{TransformerDecoder}(E^0). \quad (6)$$

To generate a provisioning action, we predict the likelihood score for each action independently using a feed-forward network. To allow the model to scale with  $|\mathcal{H}|$ , for each active

host in  $\mathcal{H}$ , we use the feature vectors from the previous interval  $H^i$  and for each new VM type we use the vector  $[\vec{0}, \vec{c}^i, \vec{r}^i, \vec{s}^i]$  to infer on which host should be deallocated or provisioned. The likelihood score for each such provisioning action, denoted by  $p^i$  with feature vector  $F^i$  such that  $i \in \{1, \dots, |\mathcal{H}| + |\mathcal{V}|\}$ , is represented as  $l^i$  and calculated as

$$l^i = \sigma(\text{FeedForward}(E^0, F)). \quad (7)$$

This factored-style prediction of the likelihood score for each provisioning action enables our model to be agnostic to the number of hosts in the setup. The final provisioning action, denoted as  $p$ , becomes the action corresponding to the highest likelihood score. These likelihood scores give us a single provisioning action. However, a provisioning decision is a collection of multiple such actions. Moreover, we want to generate likelihood scores using the predicted workload utilization metrics  $\hat{W}$ . To do this, we run inference in two phases.

### B. Two-phase Inference

An overview of the novel two-phase interleaved co-simulation and prediction in *CILP* is presented in Fig. 2. We initialize an empty provisioning decision  $P_t = \emptyset$ . In the *first phase*, for interval  $I_t$ , using schedule  $D_t$  and workload utilization characteristics  $W_{t-1}$ , we predict  $\hat{W}_t$  that estimates the workload demands in  $I_t$ ; thus  $\hat{W}_t \leftarrow f_{\theta}^{prov}(\hat{D}_{t-1}, W_{t-1})$ . We get  $\hat{W}_t$  as the output of the first phase. These prediction estimates of the workload utilization characteristics using the historical values enable the model to make informed decisions for an estimated future system state. Initially, we start without any provisioning; hence, the scheduling decision is obtained as  $D_t = f^{sched}(H_{t-1}, W_{t-1}, \emptyset)$ . In the *second phase*, we utilize the predicted  $\hat{W}_t$  to leverage estimated workload demands in the next interval. Again, we initially keep  $P_t$  as an empty set. This gives  $\hat{D}_t = f^{sched}(H_{t-1}, \hat{W}_t, \emptyset)$  as the initial schedule. Using  $\hat{D}_t$  and  $\hat{W}_t$ , we evaluate the likelihood scores for each provisioning action  $l_t^i = f^{prov}(\hat{D}_t, \hat{W}_t)$ . The decided action  $p$  is the one with the highest likelihood score. We add this action to  $P_t$  and iteratively run the following, updating  $P_t$  at each step and evaluating a QoS estimate score, denoted by  $\hat{Q}_t$ . Thus,

$$\begin{aligned} \hat{D}_t &\leftarrow f^{sched}(H_{t-1}, \hat{W}_t, P_t), \\ l_t^i &\leftarrow f^{prov}(\hat{D}_t, \hat{W}_t), \\ p &\leftarrow p^{\arg \max_i l^i}, \\ P_t &\leftarrow P_t \cup \{p\}, \\ \hat{Q}_t &\leftarrow f^{sim}(W_{t-1}, V_t, P_t, \hat{D}_t). \end{aligned} \quad (8)$$

Note that the scheduler also decides the preemptive migrations in  $\hat{D}_t$  for every non-empty  $P_t$  that has host deallocation actions. We continue the above until  $\hat{Q}_t$  is non-decreasing. This auto-regressive style of action prediction enables the model to remain parsimonious in terms of the provisioning decisions and avoid excessive overheads. Only those actions are performed that lead to an increase in the expected QoS of the system. Further, the interleaving of action prediction and co-simulation enables us to train an imitation learner, ameliorating

### Algorithm 1: CILP Provisioner

---

```

1 Require: Pretrained model  $f_{\theta}^{prov}$ , scheduler  $f^{sched}$ ,
   co-simulator  $f^{sim}$ ;
2 Initialize  $W_{-1} \leftarrow \vec{0}$ ;
3 for  $t \in \{1, \dots, T\}$  do
4    $\hat{W}_t \leftarrow f_{\theta}^{prov}(D_t, W_{t-1})$ ; /* Predict */
5    $P_t \leftarrow \emptyset$ ; /* Initialize Decision */
6    $\hat{D}_t \leftarrow f^{sched}(H_{t-1}, W_{t-1}, \emptyset)$ ;
7    $l_t^i \leftarrow f^{prov}(\hat{D}_t, \hat{W}_t)$ ;
8    $p \leftarrow p^{\arg \max_i l^i}$ ;
9   while  $\hat{Q}_t$  non-decreasing do
10    Update  $P_t, \hat{Q}_t$  using equation (8);
    /* Decision Update */
11  Execute  $P_t$  and  $\hat{D}_t$ ;

```

---

the need for costly simulations at test time. Parameter sharing between the demand and likelihood prediction reduces the size of the parameter set, enables *CILP* to jointly learn temporal trends and gain training stability. The converged decision  $P_t$  is used for VM provisioning at the start of the interval  $I_t$ .

### C. Model Training

We train the *CILP* model using an imitation learning setup where the teacher is the co-simulator, acting as an oracle that generates ground truth actions [57]. In the second phase, for each input pair  $(\hat{D}_t, \hat{W}_t)$ , the model generates likelihood score  $l_t^i$  for action  $p^i$ . We also co-simulate the provisioning action  $p^i$ , generate scheduling decision  $\hat{D}_t$  and a reward parameter  $\hat{R}_t$  as

$$\begin{aligned} \hat{D}_t &\leftarrow f^{sched}(H_{t-1}, \hat{W}_t, \{p^i\}), \\ r_t^0, \phi_t^0 &\leftarrow f^{sim}(W_{t-1}, V_t, \emptyset, D_t), \\ r_t^1, \phi_t^1 &\leftarrow f^{sim}(W_{t-1}, V_t, \{p^i\}, \hat{D}_t), \\ \hat{R}_t^k &\leftarrow r_t^k - \gamma \cdot \phi_t^k, k \in \{0, 1\}, \end{aligned} \quad (9)$$

where  $r_t$  and  $\phi_t$  are defined as per (1) and superscripts correspond to whether  $p^i$  is executed or not (1 if it is). The reward parameter is the objective function in the equation (2). The ground truth label then becomes  $g_t^i = \arg \max_k R_t^k$ , which signifies whether  $p^i$  improves the reward parameter. Now, for each  $p^i$  we evaluate the *imitation loss* as the binary cross-entropy error

$$L_{BCE}(g_t^i, l_t^i) = -\frac{1}{2}(g^i \cdot \log(p^i) + (1 - g^i) \cdot \log(1 - p^i)). \quad (10)$$

We also utilize the mean-square-error between the predicted demands and those from the dataset in the next interval that we call the *prediction loss* and is evaluated as

$$L_{MSE}(\hat{W}_t, W_t) = \frac{1}{|W_t|} \|\hat{W}_t - W_t\|^2. \quad (11)$$

Thus, the model training loss for a given utilization trace dataset at each interval  $t$  is evaluated as

$$L = L_{MSE}(\hat{W}_t, W_t) + \sum_{h_t^i \in \mathcal{H}_t} L_{BCE}(g_t^i, l_t^i). \quad (12)$$

Unlike prior work, while calculating the utilization ratio  $r_t$ , our co-simulator considers the migration delay and uses an average

resource utilization value of workloads over the execution interval  $I_t$ . This makes our optimization objective resemble more closely to real systems.

#### D. Provisioning in Practice

Using a pre-trained neural model, the *CILP* provisioner is illustrated in Algorithm 1. In each interval, we first predict the workload demands in the next interval (line 4), initialize decision (line 5) and optimize the preemptive migration decision iteratively as described earlier in equation (8) (line 10). The final provisioning decision  $P_t$  and schedule  $\hat{D}_t$  are then executed in the cloud environment. As shown in the algorithm, each iteration in the *CILP* training process includes a loop as in equation (8) that updates the provisioning decision each time. As a decision could include provisioning a new host, that is one of the VM types  $\mathcal{V}$ . A decision could also deallocate a VM from the existing set of active hosts in the system, *i.e.*,  $\mathcal{H}$ . Thus, in the worst case, we run  $|\mathcal{H}| + |\mathcal{V}|$  iterations of the inner loop.

### V. NEURAL ARCHITECTURE DETAILS

We now detail the hyper-parameters for the neural architecture used in *CILP*. We implement all our code using Python-3.8 and PyTorch-1.8.0 [58] library. All hyper-parameter values are obtained by using grid-search and the *raytune* library in PyTorch<sup>1</sup>.

#### A. Graph Attention Network

The attention weights of the graph attention network (GAT) were evaluated using a feed-forward network of 2-hidden layers, each of size 128, with LeakyReLU activation function in all hidden layers. The weights were obtained using a softmax operation. The embeddings for the nodes of the graph were obtained using graph convolution as a convex combination of neighbor embeddings, with attention weights being used in the combination operation (see equation (3)). The final embeddings ( $E^G$ ) were obtained using the sigmoid operation. The Parameterized ReLU activation function with a 0.25 negative input slope was used in the hidden layers for convolution operations.

#### B. Workload encoder

We use a feed-forward network with 4-hidden layers, each of size 256 with LeakyReLU activation function in all hidden layers. More layers improve performance; however, for a fair comparison, we ensure that the total parameter count of the neural model in *CILP* is in the same range as that of the prior work. The final encoded representation ( $E^W$ ) was obtained using the sigmoid operation (see equation (4) in Section IV).

#### C. Transformer Encoder

The input workload utilization characteristics  $W_t$  is transformed first into a matrix form of size  $|\mathcal{W}_t| \times |W_t^j|$ . We use Transformer encoders and decoders to perform temporal inference over the input workload and host time-series utilization characteristics. We define scaled-dot product attention [56] of three matrices  $Q$  (query),  $K$  (key) and  $V$  (value):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{m}}\right)V. \quad (13)$$

For large values of input size ( $m$ ), the dot product grows large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To circumvent this, we scale the dot-product attention with  $\frac{1}{\sqrt{m}}$ . For input matrices  $Q$ ,  $K$  and  $V$ , we apply Multi-Head Self Attention [56] by first passing it through  $h$  (number of heads) feed-forward layers to get  $Q_i$ ,  $K_i$  and  $V_i$  for  $i \in \{1, \dots, h\}$ , and then applying scaled-dot product attention as

$$\text{MultiHeadAtt}(Q, K, V) = \text{Concat}(H_1, \dots, H_h), \quad (14)$$

where  $H_i = \text{Attention}(Q_i, K_i, V_i)$ .

Multi-Head Attention allows the model to jointly attend to information from different representation sub-spaces at different positions. In addition, we use position encoding of the input matrices as defined in [56]. Our Transformer encoder performs the following operations on the concatenated representation  $E = [E^G, E^W]$  as

$$\begin{aligned} E^1 &= \text{LayerNorm}(E + \text{MultiHeadAtt}(E, E, E)), \\ E^0 &= \text{LayerNorm}(E^1 + \text{FeedForward}(E^1)). \end{aligned} \quad (15)$$

Here LayerNorm is the layer normalization operation described in [59]. We use feed-forward network with 4 layers, each of size 128 and with LeakyReLU activation function in all hidden layers. We use number of heads  $h = 4$  in our multi-head attention operations.

#### D. Decoding Predicted Demands

Our Transformer decoder generates the predicted workload demands  $\hat{W}$  as

$$\begin{aligned} E^2 &= \text{LayerNorm}(W_t + E^0), \\ \hat{W} &= \text{LayerNorm}(E^2 + \text{MultiHeadAtt}(E^2, E^2, E^2)). \end{aligned} \quad (16)$$

Here too, we use a feed-forward network with four layers, each of size 128 and with LeakyReLU activation function in all hidden layers and number of heads  $h = 4$  in our multi-head attention operations.

#### E. Likelihood Prediction

To predict the likelihood scores, we use the host characteristics  $H_{t-1}^i$  of the previous interval or  $[\vec{O}, \vec{c}^i, \vec{r}^i, \vec{s}^i]$  for new hosts in the system. We denote each such action as  $p^i$  with feature vector  $F^i$ . Then,  $l_t^i$  becomes

$$l^i = \sigma(\text{FeedForward}(E^0, F^i)). \quad (17)$$

Our feed-forward network consists of 2-hidden layers, each of size 128, with LeakyReLU activation function in all hidden layers and sigmoid activation to generate  $l^i$ .

<sup>1</sup>[https://pytorch.org/tutorials/beginner/hyperparameter\\_tuning\\_tutorial.html](https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html).

TABLE III  
COMPARING UTILIZATION RATIO  $r_t$ , COST  $\phi_t$  AND QoS SCORE  $\hat{Q}_t$  FOR EACH INTERVAL ACROSS ALL MODELS FOR THREE PUBLIC DATASETS. VALUES REPORTED INCLUDE AVERAGE AND STANDARD DEVIATIONS.

| Model              | Azure2017                           |                                       | Azure2019                           |                                       | Bitbrain                            |                                       |
|--------------------|-------------------------------------|---------------------------------------|-------------------------------------|---------------------------------------|-------------------------------------|---------------------------------------|
|                    | r                                   | cost                                  | r                                   | cost                                  | r                                   | cost                                  |
| <i>ARIMA+ACO</i>   | 0.385 $\pm$ 0.103                   | 0.481 $\pm$ 0.060                     | 0.581 $\pm$ 0.105                   | 0.491 $\pm$ 0.056                     | 0.374 $\pm$ 0.103                   | 0.505 $\pm$ 0.062                     |
| <i>LSTM+ACO</i>    | 0.411 $\pm$ 0.124                   | 0.508 $\pm$ 0.065                     | 0.389 $\pm$ 0.134                   | 0.519 $\pm$ 0.059                     | 0.401 $\pm$ 0.080                   | 0.538 $\pm$ 0.054                     |
| <i>Decision-NN</i> | 0.595 $\pm$ 0.113                   | 0.701 $\pm$ 0.022                     | 0.732 $\pm$ 0.068                   | 0.681 $\pm$ 0.024                     | 0.697 $\pm$ 0.081                   | 0.647 $\pm$ 0.054                     |
| <i>Semi-Direct</i> | 0.655 $\pm$ 0.065                   | 0.679 $\pm$ 0.017                     | 0.868 $\pm$ 0.059                   | 0.746 $\pm$ 0.057                     | 0.733 $\pm$ 0.069                   | 0.736 $\pm$ 0.063                     |
| <i>UAHS</i>        | 0.753 $\pm$ 0.086                   | 0.738 $\pm$ 0.045                     | 0.809 $\pm$ 0.060                   | 0.701 $\pm$ 0.028                     | 0.668 $\pm$ 0.098                   | 0.775 $\pm$ 0.038                     |
| <i>Narya</i>       | 0.773 $\pm$ 0.068                   | 0.607 $\pm$ 0.051                     | 0.740 $\pm$ 0.073                   | 0.624 $\pm$ 0.032                     | 0.696 $\pm$ 0.079                   | 0.579 $\pm$ 0.051                     |
| <i>CAHS</i>        | 0.800 $\pm$ 0.073                   | 0.668 $\pm$ 0.023                     | 0.753 $\pm$ 0.065                   | 0.779 $\pm$ 0.043                     | 0.660 $\pm$ 0.058                   | 0.655 $\pm$ 0.040                     |
| <i>CILP_IL</i>     | 0.608 $\pm$ 0.083                   | 0.495 $\pm$ 0.025                     | 0.736 $\pm$ 0.060                   | 0.506 $\pm$ 0.019                     | 0.617 $\pm$ 0.106                   | 0.492 $\pm$ 0.044                     |
| <i>CILP_Trans</i>  | 0.843 $\pm$ 0.051                   | 0.477 $\pm$ 0.016                     | 0.817 $\pm$ 0.046                   | 0.549 $\pm$ 0.020                     | 0.712 $\pm$ 0.056                   | 0.441 $\pm$ 0.028                     |
| <i>CILP</i>        | <b>0.857 <math>\pm</math> 0.049</b> | <b>0.429 <math>\pm</math> 0.020</b>   | <b>0.893 <math>\pm</math> 0.051</b> | <b>0.438 <math>\pm</math> 0.036</b>   | <b>0.806 <math>\pm</math> 0.051</b> | <b>0.420 <math>\pm</math> 0.029</b>   |
|                    | QoS                                 | Training time                         | QoS                                 | Training time                         | QoS                                 | Training time                         |
|                    |                                     |                                       |                                     |                                       |                                     |                                       |
| <i>ARIMA+ACO</i>   | 0.789 $\pm$ 0.008                   | 185.212 $\pm$ 0.923                   | 0.740 $\pm$ 0.014                   | 202.254 $\pm$ 0.317                   | 0.749 $\pm$ 0.024                   | 361.903 $\pm$ 0.915                   |
| <i>LSTM+ACO</i>    | 0.786 $\pm$ 0.008                   | 345.824 $\pm$ 0.066                   | 0.756 $\pm$ 0.020                   | 414.989 $\pm$ 0.281                   | 0.759 $\pm$ 0.018                   | 657.066 $\pm$ 0.677                   |
| <i>Decision-NN</i> | 0.738 $\pm$ 0.006                   | 295.838 $\pm$ 0.411                   | 0.736 $\pm$ 0.007                   | 355.006 $\pm$ 0.534                   | 0.714 $\pm$ 0.021                   | 562.092 $\pm$ 0.149                   |
| <i>Semi-Direct</i> | 0.736 $\pm$ 0.005                   | 275.894 $\pm$ 0.156                   | 0.701 $\pm$ 0.006                   | 331.073 $\pm$ 0.827                   | 0.711 $\pm$ 0.018                   | 524.199 $\pm$ 0.518                   |
| <i>UAHS</i>        | 0.715 $\pm$ 0.004                   | 315.009 $\pm$ 0.690                   | 0.746 $\pm$ 0.008                   | 378.011 $\pm$ 0.571                   | 0.696 $\pm$ 0.009                   | 598.517 $\pm$ 0.449                   |
| <i>Narya</i>       | 0.771 $\pm$ 0.003                   | 205.196 $\pm$ 0.445                   | 0.727 $\pm$ 0.006                   | 226.235 $\pm$ 0.940                   | 0.727 $\pm$ 0.018                   | 399.872 $\pm$ 0.106                   |
| <i>CAHS</i>        | 0.738 $\pm$ 0.007                   | 281.197 $\pm$ 0.326                   | 0.698 $\pm$ 0.013                   | 337.436 $\pm$ 0.118                   | 0.721 $\pm$ 0.015                   | 534.274 $\pm$ 0.757                   |
| <i>CILP_IL</i>     | 0.780 $\pm$ 0.004                   | <b>133.223 <math>\pm</math> 0.037</b> | 0.763 $\pm$ 0.007                   | <b>159.868 <math>\pm</math> 0.595</b> | 0.736 $\pm$ 0.018                   | <b>253.124 <math>\pm</math> 0.302</b> |
| <i>CILP_Trans</i>  | 0.796 $\pm$ 0.005                   | 291.850 $\pm$ 0.451                   | 0.781 $\pm$ 0.010                   | 412.220 $\pm$ 0.833                   | 0.746 $\pm$ 0.025                   | 693.515 $\pm$ 0.841                   |
| <i>CILP</i>        | <b>0.839 <math>\pm</math> 0.004</b> | 146.544 $\pm$ 0.789                   | <b>0.796 <math>\pm</math> 0.011</b> | 175.853 $\pm$ 0.739                   | <b>0.803 <math>\pm</math> 0.020</b> | 278.434 $\pm$ 0.113                   |

## VI. EXPERIMENTS

### A. Datasets

In order to evaluate the performance of *CILP*, we utilize three public datasets: Azure2017, Azure2019 and Bitbrain. The first two are collected from Microsoft Azure public cloud platform and are representative workload traces across thirty consecutive days [60]. The Azure2017 was generated using 110 cloud VMs, whereas the Azure2019 using 150 VMs, both across 30 consecutive days. The work by Cortez et al. [60] identifies the workloads to be a mix of nearly 30% interactive and 70% delay-insensitive tasks. The final dataset, Bitbrain consists of traces of resource utilization metrics from 1750 VMs running on BitBrain distributed datacenter [61]. The workloads running on these servers are from a variety of industry applications including computational analytical programs used by major banks, credit operators and insurers [61] and are commonly used for benchmarking fog-cloud models [43], [62], [63]. We utilize these traces as utilization characteristics of workloads and generate workloads using the same distribution as done in the traces. As all datasets use an interval duration of five minutes, we set the same value as  $\Delta$  in our experiments *i.e.* the interval duration in our formulation (see Section III).

### B. Baselines

We compare *CILP* against 7 baselines. We integrate the *ACO* algorithm with two demand forecasting methods: *AutoARIMA* and *LSTM*, and call these *AutoARIMA+ACO* and *LSTM+ACO*. We also include classical predict+optimize methods *Decision-NN* and *Semi-Direct*. Finally, the state-of-the-art

baselines are *UAHS*, *Narya* and *CAHS* (see Section II). For a fair comparison, we use the same objective function as given in (2) for all baselines. The implementations of the competitors are not public. We have implemented all baselines based on the model and training details mentioned in the respective papers. For hyperparameters, wherever not mentioned, we use the raytune library<sup>2</sup> as done for *CILP* (see Section V). For a fair comparison, we ensure that the number of parameters in the neural models of each method are similar ( $\pm 5\%$  of *CILP*). Further, we tune the neural networks and hyperparameters of the baselines on the same dataset and with the same tuning approaches as *CILP*.

### C. Testbed

We perform our experiments on a Microsoft Azure platform using the Pre-Provisioning Service (PPS) to run our methods, with workloads as *Docker* containers having utilization characteristics like those of our dataset traces (details in Section VI-D). We use diverse VM types in our cloud infrastructure, *i.e.*, *B2s* with a dual-core CPU and 4GB RAM, *B4ms* with a quad-core CPU and 16GB RAM and *B8ms* with an octa-core CPU and 32 GB RAM. All methods may provision up to 200 VMs in our testbed. The  $\xi^i$  variables of these VM types, described in Section III, are set using Gaussian regression based on 100 datapoints corresponding to the provisioning time for each type. The costs  $\mu^i$  are taken from Azure pricing calculator<sup>3</sup> for the East-US Azure datacen-

<sup>2</sup>[https://pytorch.org/tutorials/beginner/hyperparameter\\_tuning\\_tutorial.html](https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html).

<sup>3</sup><https://azure.microsoft.com/en-us/pricing/calculator/>.

ter [64]. The power consumption values of increments of 10% CPU utilization of Azure VM types are taken from Standard Performance Evaluation Corporation benchmark repository<sup>4</sup>.

#### D. Metrics and Implementation

Our QoS score ( $\hat{Q}_t$ ) is a convex combination of three QoS metrics obtained from the co-simulator, *i.e.*, normalized energy consumption ( $q_t^e$ ), average response time ( $q_t^r$ ) of completed workloads and SLA violation fraction ( $q_t^{sla}$ ). For definitions of these metrics, we refer the reader to the COSCO framework [43]. Thus,

$$\hat{Q}_t \leftarrow 1 - (\alpha \cdot q_t^e + \beta \cdot q_t^r + \delta \cdot q_t^{sla}), \quad (18)$$

where  $\alpha, \beta, \delta$  are convex-combination weights set as per the relative importance of these metrics for the end user. For our experiments, we set them to  $1/3$  as per prior work [40] for a fair comparison. The co-simulator  $f^{sim}$  obtains energy estimates using simulated power models; it adds migration time and waiting time for provisioned hosts (using  $\xi^i$ ) to the response times of affected workloads. The final response time values are used to decide the violations of SLA deadlines. To implement *CILP*, we build upon the co-simulation primitives provided by the COSCO framework [43] by modifying and integrating with custom resource provisioning methods. Unlike COSCO, which is for task placement in a *statically provisioned* cloud infrastructure, *CILP* attacks the much harder problem of provisioning a *dynamic* setup where hosts can be created/destroyed on demand. We use the Gradient Optimization using Backpropagation to Input (GOBI) scheduler as our underlying scheduling model and predefined SLA deadlines in COSCO [43]. We use the GOBI task scheduler for all baselines as well for a fair comparison. However, unlike GOBI, we do not use a surrogate based QoS predictor for provisioning, but a model that directly predicts decisions. This saves on decision time and the provisioning overhead, which translates to significant improvements in QoS (see Section VI-G).

We use the Pre-Provisioning Service (PPS) [13] in Azure to run our methods with workloads as *Docker* containers with utilization characteristics as those of traces described in Section VI. *Docker* is a container management platform as a service used to build, ship and run containers on physical or virtual environments. In our implementation, we start to provision new VMs at the start of the interval. As soon as a VM has been provisioned, the workload allocation and execution starts. Our containers ran *sysbench*<sup>5</sup> and *iozone*<sup>6</sup> linux benchmarking tools. The former facilitates matching the IPS of the utilization traces of the datasets and the latter matches the RAM and storage consumption.

We run the *CILP* provisioner (including the imitation model and co-simulated digital-twin) and scheduler on a broker machine that manages the set of Azure workers. The broker node has the following configuration: Intel i7-10700K CPU, 64GB RAM, Nvidia RTX 3080 and Windows 11 OS. The

collection of dataset and training of the *CILP* model was performed on the same machine.

We use HTTP RESTful APIs for communication and seamless integration of a *Flask* based web-environment to deploy and manage containers in our distributed cloud setup [65]. For preemptive migrations, we use the Checkpoint/Restore In Userspace (CRIU) [66] tool for container migration. All sharing of resource utilization characteristics across workers uses the *rsync*<sup>7</sup> utility. For synchronization of outputs and execution of workloads, we utilize the HTTP Notification API.

#### E. Experimental Details

The *CILP* and baseline methods run on a dedicated broker node as described previously. We run for  $T = 200$  intervals. We set the user-defined parameter  $\gamma$  using grid-search over the average QoS score  $\frac{1}{T} \sum_{t=1}^T \hat{Q}_t$  generated from the co-simulator (see sensitivity analysis in Section VI-G). The neural network architecture used for the experiments is detailed in Section V.

#### F. Training Details

Model training used a learning rate of  $5 \times 10^{-4}$ . The Adam optimizer [67] with a weight decay parameter of  $10^{-5}$  and a batch size of 64 was used. As described in Section VI-E, we use data corresponding to 200 intervals from the datasets *Azure2017*, *Azure2019* and *Bitbrain* to test the model. The rest is used to generate the training data to generate the ground-truth utilization characteristics and likelihood scores of provisioning actions using the co-simulated oracle. To train the neural model, we randomly divide the training time series into 80% training data and 20% validation data. We observe that temporal correlations in the training data are often short-range and mid-range in the setting we consider. Hence, we sample 200 points in a minibatch - since the autocorrelation function becomes flat after lag 50. This is a conservative sample length that ensures that correlations with significant magnitude are not damaging the temporal structure of the time series. An early stopping criterion was applied for convergence using the value of the loss function of the validation data.

The loss value  $L$  given by equation 12 for the converged model after the training process was  $2 \times 10^{-3}$ . The prediction performance of the *CILP* neural model directly affects the effective QoS of the system. In case the MSE error of predicting the workload utilization characteristics ( $\hat{W}_t$ ) is high in phase 1, the scheduler would not be able to take well informed scheduling decisions to allocate incoming workloads or running tasks from hosts that need to be deallocated. Further, in case of a high BCE loss in equation 10, the model poorly imitates the oracle decisions from the co-simulator and thereafter takes poor provisioning decisions. Thus, it is critical for both prediction and imitation losses to be low for an effective model.

<sup>4</sup>[https://www.spec.org/cloud\\_iaas2018/results/](https://www.spec.org/cloud_iaas2018/results/)

<sup>5</sup><http://manpages.ubuntu.com/manpages/trusty/man1/sysbench.1.html>.

<sup>6</sup><https://linux.die.net/man/1/iozone>.

<sup>7</sup><https://linux.die.net/man/1/rsync>.

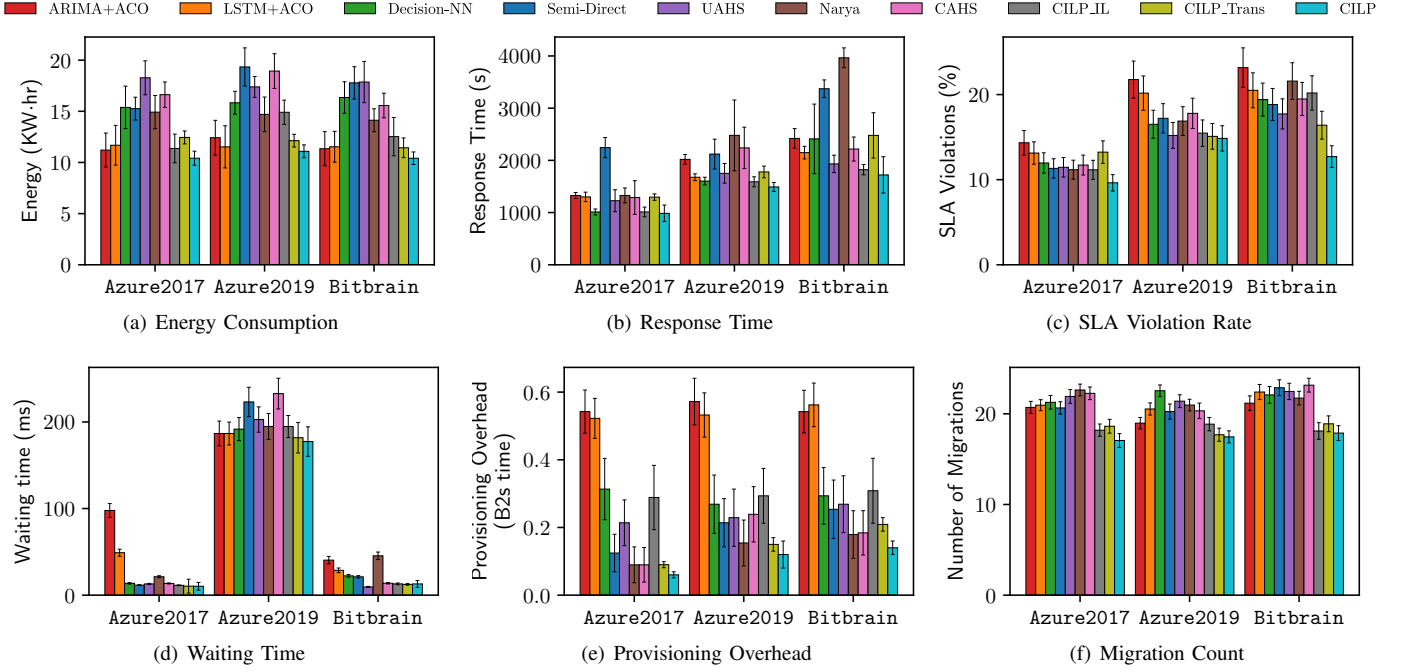


Fig. 3. Comparison of QoS parameters (averaged over intervals) of *CILP* against baselines and ablated models.

### G. Comparison with Baselines

Table III presents the utilization ratio, cost in USD and QoS score ( $\hat{Q}_t$ ) averaged over the execution intervals, and training times of all models. Figure 3 shows the individual QoS metrics for each method. For all datasets, *CILP* outperforms the baselines in terms of average utilization ratio, execution cost and QoS scores. Compared to state-of-the-art baselines, *i.e.*, *UAHS*, *Narya* and *CAHS*, *CILP* gives up to 13.81%-22.12% higher values of average utilization ratio, 41.87%-45.80% lower average execution cost and 14.04%-17.34% higher average QoS score. We also observe that *CILP* gives lower training times compared to all baselines. *CILP* gives 30.31%-57.68%. This clearly demonstrates the advantage of having a transformer model with positional encoding to push the complete time-series data as an input instead of sequentially inferring over local windows as in auto-regressive (*ARIMA*), recurrent models (*LSTM*) or feed-forward models (*Decision-NN*, *Semi-Direct*, *UAHS*, *Narya*, *CAHS*).

When comparing individual QoS metrics (see Figure 3), *CILP* gives the least average energy consumption of 10.41 KW-hr, 7.78% lower than *ARIMA+ACO* with lowest energy consumption across all baselines. This is due to the cost minimization in *CILP* that ensures the least number of hosts are active in the system, leading to a lower energy footprint. *CILP* also gives the lowest average response time, 7.41% lower than the best baseline *Decision-NN* that leads to the lowest SLA violation rates that are up to 28.32% lower than the best baseline *UAHS*. This is due to the QoS augmented decision strategy in the *CILP* methodology. Additionally, the improvements in response times come directly from proactive provisioning in *CILP*, which minimizes the time tasks spent waiting to be scheduled, as shown in Figure 3(d). Finally, *CILP* also gives the least overheads in terms of provisioning delays

and causes the minimum number of migrations, thanks to its co-simulation driven stopping criterion in Algorithm 1.

### H. Ablation Analysis

To test the efficacy of the co-simulated imitation learning and Transformer based neural models in *CILP*, we modify the approach as follows. First, we consider a model without co-simulated reward scores  $\hat{R}_t$ , but instead evaluate equation (1) without migration and provisioning overheads to train the neural model. We call this the *CILP\_IL* model. Second, we replace the Transformer encoder and decoder with feed-forward networks (with the same number of parameter weights) to test the importance of temporal trends that the Transformer captures. We call this the *CILP\_Trans* model. The results in Table III and Figure 3 show a drop in all performance metrics for these models when compared to *CILP*, demonstrating the effectiveness of the Transformer based neural architecture and model training using co-simulated imitation learning. Even though the training times of the *CILP\_IL* model is the lowest, the lack of overhead information does not allow it to take informed decisions, giving rise to poor QoS scores.

### I. Sensitivity Analysis

Figure 4 shows the performance of the *CILP* method for different values of user-defined hyper-parameter  $\gamma$ . Low  $\gamma$  corresponds to private-cloud deployments that do not have any monetary running costs, but require maximization of utilization ratio to reduce resource wastage. High  $\gamma$  corresponds to workload execution on public clouds with limited cost budgets, where the user aims at optimizing both system utilization and running costs. For comparison with baselines, we use  $\gamma = 0.4, 0.5, 0.4$  for the three datasets, as per grid-search, while maximizing QoS scores. The *CILP* provisioner gives

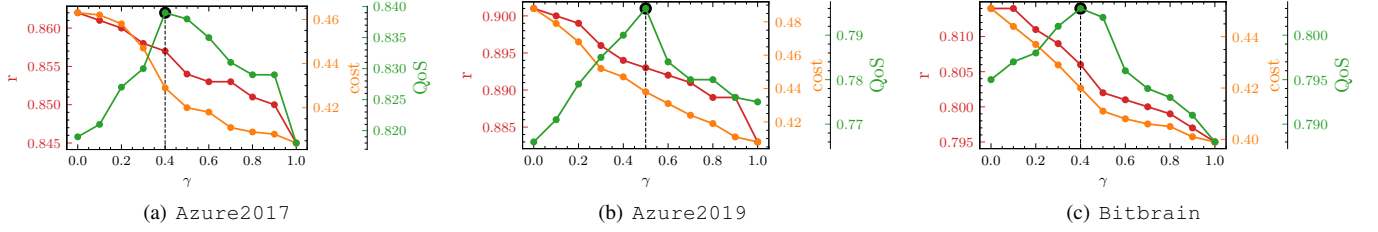


Fig. 4. Sensitivity analysis with the weight of the execution cost in the objective function ( $\gamma$ ). All standard deviations are  $<0.05$ . The  $\gamma$  values that give the highest QoS have been highlighted.

higher QoS scores than each baseline for every value of  $\gamma$ . These results provide clear evidence of the robustness of the *CILP* to different deployment scenarios and hyper-parameter settings.

### J. Discussion

The results demonstrate that *CILP* outperforms baselines not only in terms of QoS parameters, but also gives lower overheads. As provisioning overhead is hard to predict in dynamic settings, it is technically challenging to include it as part of the optimization objective. Unlike prior methods that run optimization using a model (*ARIMA+ACO*, *Decision-NN*, *Semi-Direct*, *Narya*, *UAHS* and *CAHS*), *CILP* directly generates an action, saving search time and, consequently, the provisioning overhead. A higher provisioning overhead leads to increased decision time and slower availability of required resources. *CILP*'s lower provisioning overhead reduces the average response time and SLA violation rates, significantly improving the system QoS.

Akin to a *CILP* style imitation learner, our adapted version of *Narya* is the only baseline that learns to predict provisioning actions directly (using MAB) instead of running online optimization of the provisioning decision at run-time (such as other baselines *LSTM+ACO*, *Decision-NN*, *SemiDirect*, etc.). Unlike online-optimization based baselines, the adapted *Narya* model gives lower provisioning overheads, as seen in Figure 3(e). However, the use of co-simulation based stopping criterion in the two-phase loop (line 10 in Algorithm 1) leads to lower number of VM provisions and consequently reduced provisioning overheads. Our results corroborates that *CILP* outperforms even offline learners such as *Narya* and as well as online optimization based provisioning methods.

## VII. CONCLUSIONS

This work fundamentally focuses on the VM provisioning problem. We formulate it as a predict+optimize problem. We propose *CILP*, a co-simulated digital-twin based imitation learner that dynamically predicts VM provisioning decisions to optimize the QoS of a cloud computing environment. It uses a Transformer based composite neural network to first predict the workload demands in the near future and auto-regressively find the optimal set of provisioning actions. Imitation learning enables us to train the action predictor in order to imitate the QoS effects of each action using a co-simulator.

As we discuss related works, we demonstrate that prior methods optimize primarily the utilization ratio as a metric,

and do not consider the counter-metric of the operational cost of the system. The proposed *CILP* approach uses cost as an optimization metric explicitly in its objective function. To make the final provisioning decisions, we use the QoS estimate from our co-simulation model, which uses a convex combination of the real QoS metrics: energy consumption, response time and SLA violation rate. Using this simulator, we are able to implicitly identify the impact of various provisioning decisions on the real QoS metrics, including the impact of overheads of VM provisioning (in the form of higher response time, for instance) that is modeled in the simulator. Extensive experiments on three public datasets show that *CILP* outperforms the state-of-the-art in QoS metrics, including energy consumption, resource utilization, execution cost and SLA violation rates.

As part of future work, we shall explore how *CILP* can be extended to not only make optimal provisioning decisions, but also workload scheduling decisions using the co-simulator. The predict+optimize formulation is applicable to other problem settings as well, allowing us to leverage *CILP* for decision making wherever there is access to a dedicated infrastructure and its co-simulator.

### SOFTWARE AVAILABILITY

The code has been made available as a public GitHub repository under BSD-3 License at <https://github.com/imperial-gore/CILP>.

### ACKNOWLEDGMENTS

Shreshth Tuli is supported by the President's PhD scholarship at Imperial College London.

### REFERENCES

- [1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [2] D.-N. Le, R. Kumar, G. N. Nguyen, and J. M. Chatterjee, *Cloud computing and virtualization*. John Wiley & Sons, 2018.
- [3] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 433–441.
- [4] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, 2015.
- [5] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware cloud provisioning," in *2015 IEEE 11th International Conference on e-Science*. IEEE, 2015, pp. 136–144.

- [6] R. Chard, K. Chard, R. Wolski, R. Madduri, B. Ng, K. Bubendorfer, and I. Foster, "Cost-aware cloud profiling, prediction, and provisioning as a service," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 48–59, 2017.
- [7] E. Radhika and G. S. Sadasivam, "Budget optimized dynamic virtual machine provisioning in hybrid cloud using fuzzy analytic hierarchy process," *Expert Systems with Applications*, vol. 183, p. 115398, 2021.
- [8] S. Levy, R. Yao, Y. Wu, Y. Dang, P. Huang, Z. Mu, P. Zhao, T. Ramani, N. Govindaraju, X. Li *et al.*, "Narya: Predictive and adaptive failure mitigation to avert production cloud vm interruptions," in *Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020.
- [9] M. S. Al-Asaly, M. A. Bencherif, A. Alsanad, and M. M. Hassan, "A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment," *Neural Computing and Applications*, vol. 34, no. 13, pp. 10 211–10 228, 2022.
- [10] W. Ahmad, B. Alam, S. Ahuja, and S. Malik, "A dynamic vm provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for big data workflow applications in a cloud environment," *Cluster Computing*, vol. 24, no. 1, pp. 249–278, 2021.
- [11] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel *et al.*, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, 2017.
- [12] S. Tuli, G. Casale, and N. R. Jennings, "Pregan: Preemptive migration prediction network for proactive fault-tolerant edge computing," in *IEEE INFOCOM*. IEEE, 2022, pp. 670–679.
- [13] C. Luo, B. Qiao, X. Chen, P. Zhao, R. Yao, H. Zhang, W. Wu, A. Zhou, and Q. Lin, "Intelligent virtual machine provisioning in cloud computing," in *International Joint Conference on AI*, 2020, pp. 1495–1502.
- [14] C. Luo, B. Qiao, W. Xing, X. Chen, P. Zhao, C. Du, R. Yao, H. Zhang, W. Wu, S. Cai *et al.*, "Correlation-aware heuristic search for intelligent virtual machine provisioning in cloud systems," in *AAAI*, vol. 35, no. 14, 2021, pp. 12 363–12 372.
- [15] F. Ebadifard and S. M. Babamir, "Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment," *Cluster Computing*, vol. 24, no. 2, pp. 1075–1101, 2021.
- [16] P. Singh, P. Gupta, and K. Jyoti, "TASM: Technocrat ARIMA and SVR model for workload prediction of web applications in cloud," *Cluster Computing*, vol. 22, no. 2, pp. 619–633, 2019.
- [17] M. Aliyu, M. Murali, A. Y. Gital, and S. Boukari, "Efficient meta-heuristic population-based and deterministic algorithm for resource provisioning using ant colony optimization and spanning tree," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 10, no. 2, pp. 1–21, 2020.
- [18] S. Ouham, Y. Hadi, and A. Ullah, "An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model," *Neural Computing and Applications*, pp. 1–13, 2021.
- [19] B. Wilder, B. Dilkina, and M. Tambe, "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization," in *AAAI*, vol. 33, no. 01, 2019, pp. 1658–1665.
- [20] P. J. Stuckey, T. Guns, J. Bailey, C. Leckie, K. Ramamohanarao, J. Chan *et al.*, "Dynamic programming for predict+optimise," in *AAAI*, vol. 34, no. 02, 2020, pp. 1444–1451.
- [21] M. Sohani and S. Jain, "A predictive priority-based dynamic resource provisioning scheme with load balancing in heterogeneous cloud computing," *IEEE Access*, vol. 9, pp. 62 653–62 664, 2021.
- [22] M. J. Kaur, V. P. Mishra, and P. Maheshwari, "The convergence of digital twin, iot, and machine learning: transforming data into action," in *Digital twin technologies and smart cities*. Springer, 2020, pp. 3–17.
- [23] M. S. Kumar, A. Choudhary, I. Gupta, and P. K. Jana, "An efficient resource provisioning algorithm for workflow execution in cloud platform," *Cluster Computing*, vol. 25, no. 6, pp. 4233–4255, 2022.
- [24] Z. Xu, Y. Zhang, H. Li, W. Yang, and Q. Qi, "Dynamic resource provisioning for cyber-physical systems in cloud-fog-edge computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–16, 2020.
- [25] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, "Power-aware and performance-guaranteed virtual machine placement in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1385–1400, 2018.
- [26] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: a hybrid approach," *Cluster Computing*, vol. 24, no. 1, pp. 319–342, 2021.
- [27] J. R. Gunasekaran, P. Thinakaran, M. T. Kandemir, B. Urgaonkar, G. Kesidis, and C. Das, "Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 199–208.
- [28] R. K. Naha and M. Othman, "Cost-aware service brokering and performance sentient load balancing algorithms in the cloud," *Journal of Network and Computer Applications*, vol. 75, pp. 47–57, 2016.
- [29] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "Kingfisher: A system for elastic cost-aware provisioning in the cloud," *Dept. of CS, UMass, Tech. Rep. UM-CS-2010-005*, 2010.
- [30] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, pp. 82–98, 2014.
- [31] P. S. Rawat, P. Dimri, P. Gupta, and G. P. Saroha, "Resource provisioning in scalable cloud using bio-inspired artificial neural network model," *Applied Soft Computing*, vol. 99, p. 106876, 2021.
- [32] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *2018 23rd Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2018, pp. 129–134.
- [33] C. Bitsakos, I. Konstantinou, and N. Koziris, "Derp: A deep reinforcement learning cloud system for elastic resource provisioning," in *2018 IEEE international conference on cloud computing technology and science (CloudCom)*. IEEE, 2018, pp. 21–29.
- [34] S. Verma and A. Bala, "Auto-scaling techniques for iot-based cloud applications: a review," *Cluster Computing*, vol. 24, no. 3, pp. 2425–2459, 2021.
- [35] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modeling for cloud microservice applications," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 25–32.
- [36] K. Rzacca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand *et al.*, "Autopilot: workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.
- [37] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, pp. 1–10.
- [38] N. Huber, F. Brosig, and S. Kounev, "Model-based self-adaptive resource allocation in virtualized environments," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2011, pp. 90–99.
- [39] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.
- [40] S. Tuli, S. S. Gill, M. Xu, P. Garraghan, R. Bahsoon, S. Dustdar, R. Sakellariou, O. Rana, R. Buyya, G. Casale *et al.*, "HUNTER: AI based holistic resource management for sustainable cloud computing," *Journal of Systems and Software*, pp. 111–124, 2021.
- [41] K. Borodulin, G. Radchenko, A. Shestakov, L. Sokolinsky, A. Tchernykh, and R. Prodan, "Towards digital twins cloud platform: Microservices and computational workflows to rule a smart factory," in *Proceedings of the 10th international conference on utility and cloud computing*, 2017, pp. 209–210.
- [42] L. Hu, N.-T. Nguyen, W. Tao, M. C. Leu, X. F. Liu, M. R. Shahriar, and S. N. Al Sunny, "Modeling of cloud-based digital twins for smart manufacturing with mt connect," *Procedia manufacturing*, vol. 26, pp. 1193–1203, 2018.
- [43] 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, 2021.
- [44] K. M. Alam and A. El Saddik, "C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE access*, vol. 5, pp. 2050–2062, 2017.
- [45] S. Tuli, G. Casale, and N. R. Jennings, "Simtune: bridging the simulator reality gap for resource management in edge-cloud computing," *Scientific Reports*, vol. 12, no. 1, p. 19158, 2022.
- [46] A. Rashid and A. Chaturvedi, "Cloud computing characteristics and services: a brief review," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 2, pp. 421–426, 2019.
- [47] S. Tuli, G. Casale, and N. R. Jennings, "Dragon: Decentralized fault tolerance in edge federations," *IEEE Transactions on Network and Service Management*, 2022.
- [48] 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.

- [49] N. Téllez, M. Jimeno, A. Salazar, and E. Nino-Ruiz, "A tabu search method for load balancing in fog computing," *Int. J. Artif. Intell.*, vol. 16, no. 2, pp. 1–30, 2018.
- [50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [51] A. K. Dubey and V. Jain, "Comparative study of convolution neural network's relu and leaky-relu activation functions," in *Applications of Computing, Automation and Wireless Systems in Electrical Engineering: Proceedings of MARC 2018*. Springer, 2019, pp. 873–880.
- [52] Y. Xu and J. Zhao, "Actor-critic with transformer for cloud computing resource three stage job scheduling," in *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2022, pp. 33–37.
- [53] Y.-S. Lee, Y.-S. Lee, H.-R. Jang, S.-B. Oh, Y.-I. Yoon, and T.-W. Um, "Prediction of content success and cloud-resource management in internet-of-media-things environments," *Electronics*, vol. 11, no. 8, p. 1284, 2022.
- [54] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.
- [55] —, "Graph attention networks," in *International Conference on Learning Representations*.
- [56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.
- [57] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggravated: Differentiable imitation learning for sequential prediction," in *International Conference on Machine Learning*, 2017, pp. 3309–3318.
- [58] 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.
- [59] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
- [60] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *SOSP*, 2017, pp. 153–167.
- [61] S. Shen, V. van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *CCGrid*. IEEE, 2015, pp. 465–474.
- [62] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Ugaonkar, and Y. Zhao, "An efficient and fair multi-resource allocation mechanism for heterogeneous servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2686–2699, 2018.
- [63] Z. Li, C. Yan, X. Yu, and N. Yu, "Bayesian network-based virtual machines consolidation method," *Future Generation Computer Systems*, vol. 69, pp. 75–87, 2017.
- [64] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Microsoft azure and cloud computing," in *Microsoft Azure*. Springer, 2015, pp. 3–26.
- [65] M. Grinberg, *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.
- [66] R. S. Venkatesh, T. Smejkal, D. S. Milojicic, and A. Gavrilovska, "Fast in-memory criu for docker containers," in *The International Symposium on Memory Systems*, 2019, pp. 53–65.
- [67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.



in science and innovation. His research interests include Fog Computing and Deep Learning. For further information, visit <https://shreshthtuli.github.io/>.



boards of IEEE TNSM and ACM TOMPECS, as editor in chief of Elsevier PEVA, and as the current chair of ACM SIGMETRICS.



**Nicholas R. Jennings** is the Vice-Chancellor and President of Loughborough University. He is an internationally-recognised authority in the areas of AI, autonomous systems, cyber-security and agent-based computing. He is a member of the UK government's AI Council, the governing body of the Engineering and Physical Sciences Research Council, and chair of the Royal Academy of Engineering's Policy Committee. Before Loughborough, he was the Vice-Provost for Research and Enterprise and Professor of Artificial Intelligence at Imperial College London, the 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.