Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

# HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments

Sundas Iftikhar [a,b], Mirza Mohammad Mufleh Ahmad [a], Shreshth Tuli [c], Deepraj Chowdhury [d], Minxian Xu [e], Sukhpal Singh Gill [a,*], Steve Uhlig [a]

[a] *School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK*
[b] *University of Kotli Azad Jammu & Kashmir, Kotli, Pakistan*
[c] *Department of Computing, Imperial College London, UK*
[d] *Department of Electronics & Communication Engineering, International Institute of Information Technology (IIIT), Naya Raipur, India*
[e] *Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China*

ARTICLE INFO

*Keywords:*
Artificial Intelligence
Cloud computing
Fog computing
Edge computing
Resource management
Machine learning
Energy efficiency

ABSTRACT

Cloud computing is a mainstay of modern technology, offering cost-effective and scalable solutions to a variety of different problems. The massive shift of organization resource needs from local systems to cloud-based systems has greatly increased the costs incurred by cloud providers in expanding, maintaining, and supplying server, storage, network, and processing hardware. Due to the large scale at which cloud providers operate, even small performance degradation issues can cause energy or resource usage costs to rise dramatically. One way in which cloud providers may improve cost reduction is by reducing energy consumption. The use of intelligent task-scheduling algorithms to allocate user-deployed jobs to servers can reduce the amount of energy consumed. Conventional task scheduling algorithms involve both heuristic and metaheuristic methods. Recently, the application of Artificial Intelligence (AI) to optimize task scheduling has seen significant progress, including the Gated Graph Convolution Network (GGCN). This paper proposes a new approach called **HunterPlus** which examine the effect of extending the GGCN's Gated Recurrent Unit to a Bidirectional Gated Recurrent Unit. The paper also studies the utilization of Convolutional Neural Networks (CNNs) in optimizing cloud–fog task scheduling. Experimental results show that the CNN scheduler outperforms the GGCN-based models in both energy consumption per task and job completion rate metrics by at least 17 and 10.4 percent, respectively.

## 1. Introduction

Cloud–fog computing refers to the utilization of non-local computing resources such as storage and processors that are accessed via a network and supplied by a cloud provider [1]. Users do not have to purchase any hardware and can rent computing resources on-demand as determined by their needs [2]. Cloud–fog computing offers organizations scalability and flexibility as it is simple and convenient to obtain more computational resources from the cloud provider as organizational needs increase. The diminished need for onsite hardware greatly reduces the costs and the need for hardware maintenance [3]. There is also no longer any need to spend time setting up and provisioning local hardware. There has been a massive expansion of cloud–fog computing usage by

* Corresponding author at: School of Electronic Engineering and Computer Science, Queen Mary University of London, London, E1 4NS, UK.
*E-mail addresses:* s.iftikhar@qmul.ac.uk (S. Iftikhar), m.m.m.ahmad@qmul.ac.uk (M.M.M. Ahmad), s.tuli20@imperial.ac.uk (S. Tuli), deepraj19101@iiitnr.edu.in (D. Chowdhury), mx.xu@siat.ac.cn (M. Xu), s.s.gill@qmul.ac.uk (S.S. Gill), steve.uhlig@qmul.ac.uk (S. Uhlig).

https://doi.org/10.1016/j.iot.2022.100667
Received 11 November 2022; Received in revised form 4 December 2022; Accepted 9 December 2022

businesses, meaning cloud providers must invest in greater numbers of hardware resources. Consequently, cloud providers face a higher energy consumption footprint, further increasing their costs. This makes it desirable to minimize the amount of energy used by these resources [4]. In addition, service reliability may also be impacted by high energy consumption, as higher energy consumption correlates to higher operating temperatures that can create hotspots, which have an impact on performance. Traditional techniques of reducing energy consumption focused on leveraging the dynamics of the cooling systems, such as efficient airflow configurations [5] and the use of controlled fan systems [6]. Another approach is intelligent workload scheduling, where jobs are allocated or migrated to servers such that energy consumption is minimized. Migration is the process by which jobs running on one physical node are moved to another physical node. Migration may be necessary if downtime is scheduled on the existing physical node, or if the physical node is displaying signs of performance degradation. Migration can ensure that the process can continue seamlessly from a captured checkpoint on another machine, instead of starting the process again from scratch on the new machine [7]. Migration is therefore an essential part of modern-day cloud–fog task scheduling, saving time, cost, and energy. Noel et al. [8] demonstrated the effectiveness of dynamic load distribution in reducing hotspots, obtaining significantly improved throughput and latency in storage servers. Furthermore, Li et al. [9] explained that an optimal energy solution may not always be an optimal reliability solution. They attempt to overcome this dilemma using the Ella-W and Ella-B scheduling algorithms. These algorithms work using a novel metric that unifies computing energy, cooling energy, and server reliability. While these methods fail to significantly improve reliability performance, they decrease energy consumption by up to nearly 30%. Young et al. [10] developed a heuristic-based method to minimize energy consumption via task consolidation. Their heuristics are used to allocate servers such that performance remains robust while minimizing energy usage. This method is effective, although the performance improvement is only 18% compared to the random server allocation. There has been also research into evolutionary algorithms, with Gill et al. [11] utilizing the Cuckoo optimization algorithm to maximize resource utilization in the CRUZE cloud management system. Resources are assigned using fitness values calculated using a predefined objective function that considers both the resources' energy consumption and reliability. In addition to conventional algorithmic and heuristic methods, there has been an increasing interest in applying artificial intelligence models that can optimize resource allocation. Ran et al. [12] developed a deep reinforcement learning framework that optimizes both task scheduling and system airflow settings, using feedforward neural networks to model the system rewards and airflow. Chen et al. [13] trained a Recurrent Neural Network (RNN) model on the Google cluster dataset and achieved resource savings of up to 10% by predicting and terminating tasks that were highly likely to fail. Tuli et al. [14] extended the use of RNNs with their HUNTER resource allocation system by integrating graph neural networks into their model, based on the Graph Recurrent Neural Network [15]. HUNTER is a Gated Graph Convolution Network (GGCN) that acts as a surrogate model that estimates Quality of Service (QoS) parameters based on usage metrics of the hosts and tasks. It then schedules its decisions based by determining which combination of hosts and tasks together maximize the given QoS parameter. Our work continues in the direction of applying neural networks to cloud–fog task scheduling to improve energy efficiency, and the main contributions of this paper are:

- We propose a new approach called **HunterPlus**, by extending HUNTER [14] to include a Bidirectional Gated Recurrent Unit (GRU) to evaluate the graph inputs in both forward and backward direction.
- We implement a novel CNN model that takes the host and task metrics in a preprocessed 2D array form and maps it to the energy consumption metric.
- Our evaluation demonstrate that HunterPlus is able to improve energy consumption and job completion rates by at least 17% and 10.4%, respectively, compared to the state-of-the-art baselines.

The rest of the paper is structured as follows: In Section 2, we briefly review algorithms from various domains used for scheduling purposes to reduce energy consumption. In Section 3, we describe the core of our work, where we discuss and present details of our solution models, the dataset used, and the testing framework. In Section 4, we present and discuss the results of our experiment. Finally, Section 5 concludes and provides future directions.

## 2. Background and related work

This section provides a background of the various heuristics, meta-heuristics and machine learning based approaches common in prior work.

### 2.1. Heuristic methods

This section gives some context for the many heuristics based techniques that have been used in previous research.

#### 2.1.1. First Come First Serve (FCFS)

This algorithm schedules tasks based on the order in which they arrive. Maharan et al. [16] demonstrate this method has the highest energy consumption per task when compared with other evolutionary methods.

#### 2.1.2. Max–min

The max–min algorithm works by computing the execution times required for workloads and assigning tasks that require the most time to hosts that can complete them in the minimum amount of time. The algorithm is not as effective in minimizing the time taken per task and the energy consumed per task when compared with genetic methods [17].

### 2.1.3. *Most Efficient Server First (MESF)*

This algorithm determines the most efficient servers available in a heterogeneous host configuration and allocates tasks to the most efficient server first. Subsequent tasks are allocated to the next most efficient server, and so on. When server specifications are similar, tasks are allocated until a noticeable degradation of performance is observed. MESF can be up to 70 times more energy efficient than random task schedulers [18].

## 2.2. *Metaheuristic methods*

Heuristic methods are problem specific, while metaheuristic methods are general methods that can be applied to any given optimization problem. Metaheuristic algorithms consist of 3 operators: transition, evaluation, and determination [19]. Transition modifies the current solution in different ways, and evaluation is done using an objective function that gives the fitness of the new solutions. Determination then picks those solutions which demonstrate optimal results. Examples include the hill climbing algorithm and the simulated annealing algorithm. Evolutionary algorithms also fall into this category and have seen significant research interest for cloud task scheduling.

### 2.2.1. *Genetic Algorithms (GA)*

The genetic algorithm is a population-based metaheuristic approach that has seen widespread use thanks to its good results and adaptability to different domains [19]. The technique takes inspiration from biological evolution, and works through the selection, mutation, crossover, and reproduction operators. Chang-tian et al. [20] use dynamic voltage staging in conjunction with a genetic algorithm that optimizes server makespan and energy consumption. The task scheduling is represented directly using integers and vectors, with samples possessing lower makespan and energy consumption values being assigned higher fitness values. The ETU-GA and ETDF-GA algorithms can show a good balance between maintaining optimal makespan while minimizing energy consumption. Liu et al. [21] used a multi-objective genetic algorithm to find the optimal balance between energy consumption and the cloud providers' profits. Scheduling is represented using matrices. Chen et al. integrated a greedy algorithm with a GA to speed up the scheduling process by eliminating early obvious bad candidates [13]. The chromosomes are represented as 1D arrays that map physical machines to virtual machines (VM). The algorithm minimizes the number of physical machines being used saving a significant amount of energy.

### 2.2.2. *Ant Colony Optimization (ACO)*

Ant colonies consist of simple rule-based agents (ants). Each agent constructs a separate path during each iteration, and the quality of each path is evaluated and updated to the pheromone matrix, which guides the agents during the subsequent iteration. Ultimately, an optimal path is constructed by all the agents. Liu et al. [22] utilized this algorithm to minimize the number of physical servers required to host a given number of virtual machines and obtained significantly better energy performance (especially for many VMs) compared to the first-fit decreasing algorithm. The Load Balancing Ant Colony Optimization (LBACO) algorithm proposed by Li et al. [23] extended the conventional ACO approach by also considering the prevalent load of a host machine before making an allocation decision. This approach can handle varying loads much better than FCFS and ACO.

## 2.3. *Deep learning-based scheduling*

This approach involves training neural networks over historical cloud trace data and predicting QoS parameters to make scheduling decisions.

### 2.3.1. *Artificial neural networks (ANNs)*

ANNs are graph structures consisting of nodes (known as neurons) and directed edges that exist between neurons in progressing layers of the network, where each neuron gives an output resulting from an activation function applied to the neuron's inputs. ANNs can model complex nonlinear phenomena when network weights are tuned (via learning and 3 backpropagation) and can be branched into either feed-forward or recurrent networks. Outputs in feedforward networks are independent of any previous output, while outputs of recurrent networks depend on the previous network output, due to the presence of looped edges in the graph. Witanto et al. [24] developed and trained a simple ANN model that uses active host, resource usage, and workload data along with current performance metrics to determine the best load consolidation algorithm for scheduling tasks. This model does not predict any QoS parameter; it simply chooses the best algorithm for a given scenario. This technique demonstrates significantly improved performance over no-migration-based scheduling. ANNs have also been combined with Reinforcement Learning models, such as PArameterized action space based Deep Q-Network (PADQN) model developed by Ran et al. [12].

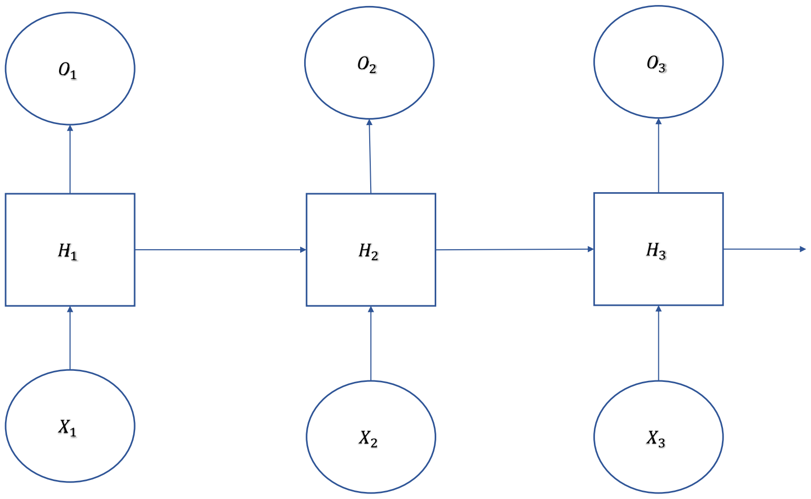


**Fig. 1.** The basic structural unit of an unrolled RNN.

### 2.3.2. *Recurrent neural networks (RNNs)*

RNNs are deep learning models that demonstrate excellent performance with sequential data and are extensively utilized in tasks such as language modeling, text generation, machine translation, and speech recognition [25]. A simple RNN can be mathematically described as an input $X$ and a hidden state H at time $t$, which are transformed into inputs to an activation function using and weight matrices respectively as shown in Fig. 1. A bias may also be added. The structure of a basic RNN is shown in Fig. 1. $X_n$ refers to the input at each timestep. Except for the first timestep, any output $O_n$ depends on the previous hidden state $H_{n-1}$ and the current input $X_n$. RNNs work well with sequential data due to their ability to keep an internal memory of previous inputs. This memory ensures that context is remembered when the RNN processes subsequent input data; previous outputs influence future outputs. Recurrent neural networks are implemented using two different methods. These are long short-term memory (LSTM), and gated recurrent unit (GRU). RNNs can suffer from both vanishing gradients (where backpropagation updates become meaningless) and exploding gradients (where the network becomes unstable). LSTMs can help overcome these problems. They consist of a cell, an input gate, an output gate, and a forget gate [22] . The input gate determines what information from the new inputs should be stored in the internal state. The output gate determines the new hidden state, based on the new input, the new internal state (obtained from the input gate), and the hidden state. The forget gate determines what parts of the internal state should be retained based on the previous hidden state and current output. Together, the three gates control the flow of information in the network. GRUs are simpler versions of LSTMs, having two gates instead of three. These are the reset gates (like the forget gate in LSTM), and the update gates (like the input gate in LSTM). It has the advantage that it is less complex and easier to train. LSTMs and GRUs can also be encapsulated in a Bidirectional layer. This means that training is done by passing inputs through both forward and backward RNN cells. The output is then obtained by concatenating the results of both forward and backward RNN cells at each time step. This provides better context for the model and can improve performance. Tuli et al. [14] used the GGCN model, which is a GRU combined with Graph Neural Network (GNN), to approximate QoS parameters for task scheduling purposes. Karim et al. [26] developed the BHyPreC prediction model, which uses a Bidirectional LSTM stacked on top of similar RNNs to predict future CPU workloads. The model initially takes pre-processed data passed through a 1D Convolution layer, and this is passed into the Bidirectional layer. The model was trained and tested using the Bitbrain dataset and proved to be more stable and more accurate than single-direction LSTM or GRU-based models.

### 2.3.3. *Convolutional Neural Networks (CNNs)*

CNNs are a type of neural network primarily used in computer vision and image processing [27]. Generally, CNNs have convolutional layers, pooling layers, and fully connected layers. Convolutional layers have kernels that slide down and across the input to produce output by multiplying each weight in the kernel with the corresponding matrix element value over which the kernel acts. The result is then summed to get the output. The kernel has weights that are updated through backpropagation during the training process. Pooling layers are used to reduce the dimensionality of the feature maps and for more robust feature detection. CNNs have seen limited use in task scheduling problems compared to other neural networks. Talaat et al. [28] used a CNN to classify appropriate fog servers according to their resource usage metrics.

## 2.4. *Critical analysis*

Table 1 compares HunterPlus with existing works. We analyzed different methods and algorithms for task scheduling with stochastic workloads. Many works used simulations to aid in search methods but none of them used coupled simulation for QoS

**Table 1**
Comparison of proposed work HunterPlus with existing works.

| Work | Cloud–fog | Method type | Workload type | QoS prediction | Coupled simulation for decision making aid |
|---|---|---|---|---|---|
| Mahran et al. [16] | Yes | Heuristic | Stochastic | No | No |
| Liu et al. [22] | Yes | Meta-heuristic | Stochastic | No | No |
| Tuli et al.[14] | Yes | Gated Graph Convolution Network | Stochastic | Yes | Yes |
| Ridwan et al. [8] | Yes | Heuristic | Stochastic | No | No |
| Ying et al. [20] | Yes | Meta-heuristic | Stochastic | No | No |
| Ran et al.[12] | Yes | Deep reinforcement Learning | Stochastic | No | No |
| Witanto et al.[24] | Yes | Deep learning | Stochastic | No | No |
| HunterPlus (this work) | Yes | Convolutional neural network | Stochastic | Yes | Yes |

parameters prediction and to collect additional data for aiding in the decision-making of an AI model except [14]. According to the best of our knowledge, none of the works have used CNN-based coupled simulation for the collection of additional data for the decision-making of an AI model. Furthermore, among the most popular AI algorithms, we selected to utilize Deep Learning (CNN) for scheduling tasks rather than Reinforcement Learning (RL). Many previous works in the literature [29–31] reported that RL approaches are slow to adapt to a highly volatile environment. They are unable to adapt when a host is running at high capacity. This mean they predict an overload host for large number of scheduling intervals. Therefore, task cannot be assigned to same interval and need to wait for either it is assigned to other host or resource released from that hosts. Another issue with RL approaches is their low scalability, which increases waiting time with increase in resources. However, proposed DL (CNN) based approach has capacity to adapt quickly to environment as neural model update in every iteration and do not face scalability issue on even large scale experiments.

## 3. The HunterPlus scheduler

This section describes the system architecture of the HunterPlus scheduler, the sustainability models, and the CNN-based Surrogate model.

### 3.1. System architecture

This section describes the architecture of our system shown in Fig. 2. We considered a standard heterogeneous and distributed fog environment. We assume a single fog datacenter with geographically distributed edge and cloud layers as computational nodes. Tasks are docker container instances that take inputs from sensors and other Internet of Things (IoT) devices and output to the actuators. All management of tasks and hosts is done in the management layer on fog broker. Gateway devices are facilitating communication between end user and fog broker. Compute nodes in the resource layer have diverse computational capacities and are referred as hosts. We divided our system in three layers: IoT layer, Management layer and Resource layer.

- **IoT Layer:** IoT layer consists of IoT and gateway devices. The workloads are sent by users with inputs from sensors or other IoT devices to the broker. Edge devices such as smartphones and tablets are used as gateway devices.
- **Management Layer:** In the management layer, the broker receives incoming tasks and distributes these to worker nodes in the resource layer. The management layer consists of three main modules: Service manager, Resource manager, and Datacentre manager.
  - **Service Manager:** The service manager module manages heterogeneous services while processing workloads. This module consists of two parts:
  - **SLA Manager:** It contains and manages details of Service Level Agreements (SLA) between the service user and service provider based on Quality of Service.
  - **QoS Manager:** It contains and manages the quality of service requirements for the workloads.
  - **Resource Manager:** The resource manager contains the HUNTERPlus scheduler which takes tasks (realized using containers) as input and orchestrates these tasks using resource metrics from the resource monitor. To schedule resources, the resource manager uses a CNN-based surrogate model that estimates QoS parameters. It performs training and on-the-fly tuning of the CNN model to adapt in non-stationary settings. This manager also runs an exploration strategy that checks the QoS scores for a set of allocations and chooses the best one as the scheduling decision
  - **Datacentre Manager:** Datacentre manager monitors resource utilization of tasks and nodes, QoS parameters, and performs allocation and migration of tasks.
- **Resource Layer:** Resource layer consists of a group of heterogeneous resources. Some of the resources are fog resources which are assumed to be resource constrained but offer lower latency and others are cloud resources which are located in geographically distant locations but are connected via the same virtual network.

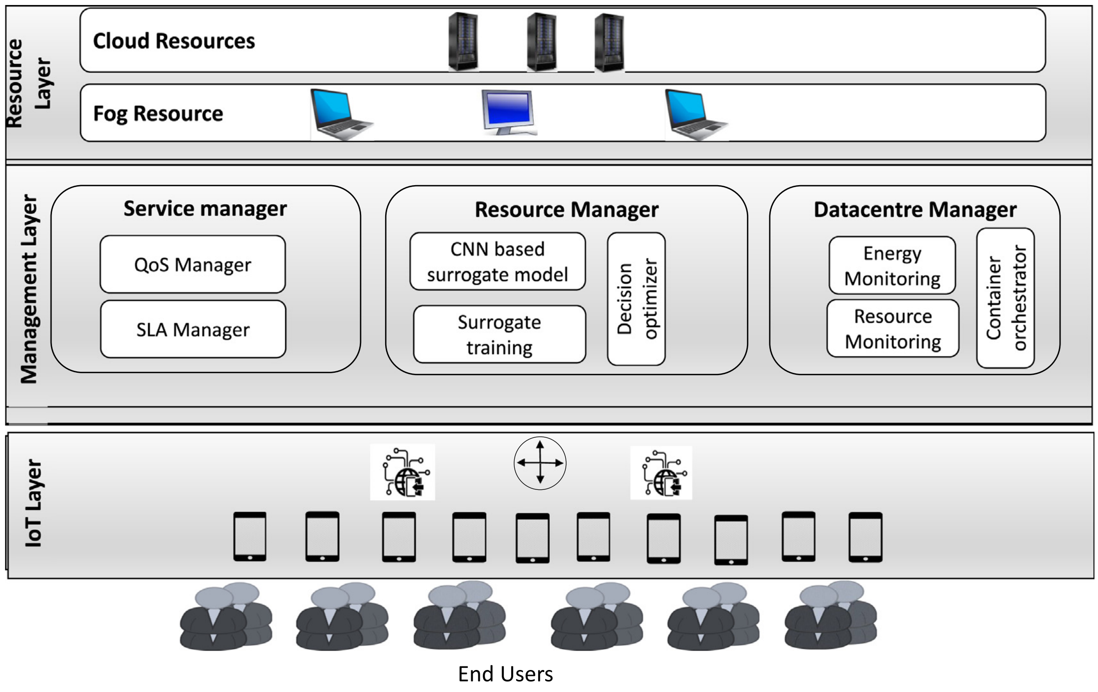


**Fig. 2.** System Architecture of HunterPlus.

### 3.2. *Sustainability models*

We also briefly mention the energy model used in the study. In this work, to decouple the different aspects of sustainable resource management [14], we have designed two different models: energy and cooling. For completeness, we reproduce the formulae from prior work [14], with necessary adaptations for the new formulation.

#### 3.2.1. *Energy model*

The energy model developed by Gill et al. [11] is used. Energy consumption is formulated as the sum of the energy expended by computer hardware and the energy expended on cooling facilities as shown in Eq. (1).

$$E_{\mathrm{T}} = E_{\mathrm{Comp}} + E_{\mathrm{Cool}} \tag{1}$$

Computer hardware includes processors, storage, and networking devices, amongst other items. The cooling energy is obtained as the sum of energy used by air-conditioners, fans, compressors, and pumps. $E_{\mathrm{Comp}}$ can be defined as

$$E_{\mathrm{Comp}} = E_{\mathrm{Pr}} + E_{\mathrm{S}} + E_{\mathrm{Mem}} + E_{\mathrm{Net}} + E_{\mathrm{additional}} \tag{2}$$

Here, $E_{\mathrm{Pr}}$ denotes the energy consumption by processors, $E_{\mathrm{S}}$ represents the storage energy, $E_{\mathrm{mem}}$ represents the memory energy, $E_{\mathrm{Network}}$ represents the network energy, $E_{\mathrm{additional}}$ represents the extra energy.
**Processor.** Here, $\mathrm{E}_{\mathrm{Pr}}$ stands for the total energy used by the Processor, which is determined by adding the energy used when the processor is idle to the power used while it is dynamic processing data. Consequently,

$$E_{\mathrm{Pr}} = \sum_{p=1}^{core} E_{\mathrm{dy}}^{p} + E_{\mathrm{idl}}^{p} \tag{3}$$

where $E_{\mathrm{dy}}^{p}$ and $E_{\mathrm{idle}}^{p}$ are the dynamic and idle energy consumption of the $p$th core respectively. Here, $E_{\mathrm{dy}}$ is calculated using

$$E_{\mathrm{dy}} = \frac{E_{\mathrm{dy}}^{\mathrm{lin}} + E_{\mathrm{dy}}^{\mathrm{non\text{-}lin}}}{2} \tag{4}$$

$E_{\mathrm{dy}}^{\mathrm{lin}}$ is calculated as

$$E_{\mathrm{dy}}^{\mathrm{lin}} = CV^{2}f \tag{5}$$

where $C$ is CPU capacitance, $f$ is CPU clock frequency, and $V$ is CPU voltage. $E_{\mathrm{dy}}^{\mathrm{non\text{-}lin}}$ is calculated using

$$E_{\mathrm{dy}}^{\mathrm{non\text{-}lin}}(h_j) = \mu_1 \cdot u_j + \mu_2 \cdot u_j^2 \tag{6}$$

**Algorithm 1** The HUNTERPlus scheduler

```
Require:
    Pre-trained function approximator $f(x;\theta)$
    Dataset used for training $\Lambda$
    Convergence threshold $\epsilon$
    Iteration limit $\sigma$
    Learning rate $\gamma$
    Initial random decision $\mathcal{D}$
    Performance metric $O$
 1: procedure MINIMIZE($\mathcal{D}$, $f$, $S$)
 2:     Initialize decision matrix $\phi(\mathcal{D})$
 3:     $i = 0$
 4:     do
 5:         $x \leftarrow [S, \phi(\mathcal{D})]$                          ▷ Concatenation
 6:         $\delta \leftarrow \nabla_{\phi(\mathcal{D})} f(x;\theta)$          ▷ Partial gradient
 7:         $\phi(\mathcal{D}) \leftarrow \phi(\mathcal{D}) - \gamma \cdot \delta$        ▷ Decision update
 8:         $i \leftarrow i + 1$
 9:     while $|\delta| > \epsilon$ and $i \leq \sigma$
10:     end do while
11:     Convert matrix $\phi(\mathcal{D})$ to scheduling decision $\mathcal{D}^*$
12:     return $\mathcal{D}^*$
13: end procedure
14: procedure HUNTERPLUS(scheduling interval $I_t$)
15:     if (t == 0)
16:         Initialize random decision $\mathcal{D}$
17:     else
18:         $\mathcal{D} \leftarrow \mathcal{D}^*$                  ▷ Output for the previous interval
19:     Get $S$ as the system state
20:     $\mathcal{D}^* \leftarrow$ MINIMIZE($\mathcal{D}, f, S$)
21:     Fine-tune $f$ with loss =
22:       $MSE(O_t, f([\phi, \mathcal{D}^{t-1}];\theta))$
23:     return $\mathcal{D}^*$
24: end procedure
```

where $\mu_1$ and $\mu_2$ are non-linear model parameters and $u_j$ is CPU utilization of host $h_j$.

**Storage.** $E_{\mathrm{S}}$ is the energy used by storage devices for storing data. The data write and read operations account for the energy consumption in such devices. Storage energy is combination of energy consumed by read operation, write operation and idle (when there is no read or write operation):

$$E_{\mathrm{S}} = E_{\mathrm{R}} + E_{\mathrm{W}} + E_{\mathrm{I}} \tag{7}$$

$E_{\mathrm{Mem}}$ is the consumption of energy of the main memory (RAM/DRAM), represented as DR, and cache memory (SRAM) represented as SR, which is calculated using

$$E_{\mathrm{Mem}} = E_{\mathrm{SR}} + E_{\mathrm{DR}} \tag{8}$$

**Network.** $E_{\mathrm{Net}}$ is the energy consumption of network parts such as switches (Switch), routers (Rout), LAN cards (LAN), and gateways (Gate) and is calculated as

$$E_{\mathrm{Net}} = E_{\mathrm{Rout}} + E_{\mathrm{Switch}} + E_{\mathrm{Gate}} + E_{\mathrm{LAN}} \tag{9}$$

**Peripherals.** $E_{\mathrm{additional}}$ represents the energy consumption of other parts, including the current conversion loss and others and is calculated as

$$E_{\mathrm{additional}} = E_{\mathrm{mb}} + \sum_{k \in K} E_{\mathrm{con}}^{k} \tag{10}$$

where $E_{\mathrm{mb}}$ is the energy consumed by motherboard(s) and $\sum_{k \in K} E_{\mathrm{con}}^{k}$ is the energy consumed by a connector (port) running at the frequency *k*, where the set of port frequencies is denoted by $K$.

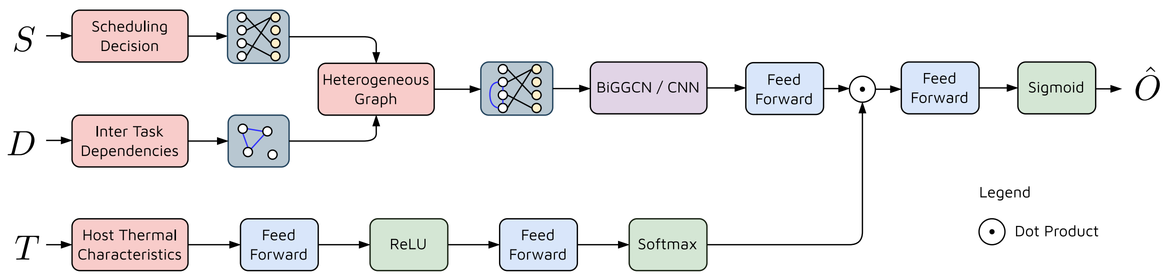


**Fig. 3.** Architecture of HunterPlus neural network.

### 3.2.2. *Cooling model*

: In this model, $E_{\mathrm{Cool}}$ is the energy consumed by cooling devices to maintain the temperature of a cloud datacenter, which is calculated using Eq. (1):

$$E_{\mathrm{Cool}} = E_{\mathrm{Air\ conditioner}} + E_{\mathrm{Compres}} + E_{\mathrm{fan}} + E_{\mathrm{pump}} \tag{11}$$

where $E_{\mathrm{Air\ Conditioner}}$ is the energy consumed by the air-conditioner inside the datacenter, $E_{\mathrm{Compres}}$ is the energy consumed by the compressor, $E_{\mathrm{fan}}$ is energy consumed by the fans and $E_{\mathrm{pump}}$ is that of the pump attached with all-in-one cooling solution made up of water.

## 3.3. *CNN based surrogate model*

For scheduling using CNN-based surrogate model, we utilized both host and task resource usage data to best optimize the allocation of tasks to hosts. The task allocations, host resource usage details, and similar information are represented in a matrix form (a so-called ‘image’ of the present state of the cloud–fog environment). The image is constrained such that only a maximum number of tasks are allowed to exist over a given number of hosts. As we have 4 hosts which can be allocated a maximum of 20 total tasks. It is not necessary that tasks be divided equally among the hosts; instead, this is left to the scheduler to determine. The image consists of 20 rows and 13 columns, where each row represents a task, and the columns represent hosts. Feature vector of tasks include Instructions per Seconds (IPS), RAM, Disk and Bandwidth consumption. The columns also contain a one-hot vector representation of the host to which the task is currently allocated. A task that does not yet exist is represented by all zeros in the columns. During execution, the scheduler extracts all the relevant data about the system, does some preprocessing, and converts it into the image form that is required for input into the CNN. This representation ensures that current tasks are linked together with their hosts. The CNN model itself consists of a series layers with increasing number of channels followed by series of layers with decreasing number of channels. Because the inputs are not images in the classical sense, the max pooling layer that is usually applied after a convolution is discarded.

Due to the nature of the problem, application of max pooling would cause loss of information. Padding is also applied when kernel convolutions are applied, to minimize information loss. Kernel size is defined in tandem with padding to ensure consistent subsequent channel size. The reason for reducing the number of channels afterward is to minimize the number of features once the channel outputs are flattened into the fully connected layer. There is only one fully connected layer ending in the sigmoid activation function. The scheduling decision is taken by obtaining an optimal image via PyTorch, deep learning framework, to optimize features that maximize the QoS values and then extracting the relevant host allocation components that are available in the optimal image matrix.

An algorithmic representation of the working of HunterPlus is shown in Algorithm 1 and Fig. 3. We represent the CNN or GGCN based model using $f(x, \theta)$ where the weights of the model are denoted by $\theta$. We initialize the decision of the scheduler as $\mathcal{D}$. We use the state of the system, denoted by $S$ that consists of the CPU, RAM and Disk utilization characteristics of the host machines in the fog environment. We then run gradient based optimization using the surrogate model as per the Minimize function in Alg. 1. This uses the partial gradient of the surrogate function with respect to the decision vector to execute optimization in the decision space. Finally, once the gradient based search converges as per the threshold based convergence criterion, we enact the final decision $\mathcal{D}^*$ and fine-tune the model using the Mean-Square-Error (MSE) loss to adapt the policy in dynamic settings. The MSE loss is calculated with the performance parameter $O_t$ defined as

$$O_t = 1 - (\alpha \cdot AEC_t + \beta \cdot AT_t + \gamma \cdot SLAV_t), \tag{12}$$

where $AEC_t$, $AT_t$ and $SLAV_t$ denote the average normalized energy consumption, average normalized temperature and SLA violation for the leaving tasks in interval $I_t$. Here, $\alpha, \beta, \gamma$ are convex-combination weights. To minimize the metrics of energy, temperature and SLA violations, we maximize $O$. We use $\alpha = \beta = \gamma = \frac{1}{3}$ in our evaluation, although the performance improvements are seen for other values as well.

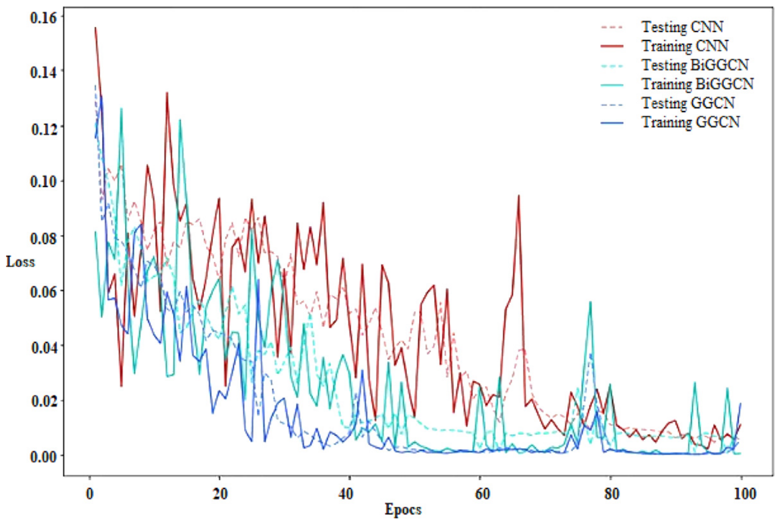


**Fig. 4.** Training and testing losses for each scheduler model.

## 4. Performance evaluation

This section evaluates the proposed model HunterPlus and compares it with baseline models.

### 4.1. Evaluation setup

We rely on a real cloud–fog environment using the COSCO framework [31]. For statistical significance, we run experiments for each model 4 times and average the results. In the physical setup, we initialized 4 Standard Azure B2s virtual machines in the UK, each with 2 cores and 4 GB of RAM. An average of 2 tasks at Poisson distribution (1.5) from the DeFog benchmark were created during each scheduling interval. DeFog consists of diverse and non-stationary AI workloads. We realized these workloads using docker containers to execute on our physical setup. The scheduling interval itself was set to 300 s to allow sufficient time for the deployed tasks to complete execution. Each experiment is run for 20 intervals to collect QoS metrics.

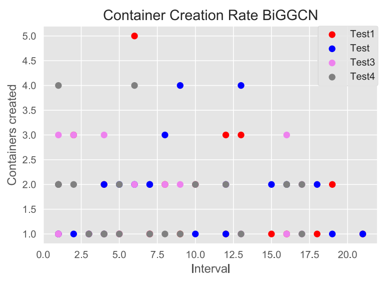
### 4.2. Datasets

The training datasets were generated using a random scheduler for 100 intervals for two different workload settings. The normal workload involved creating a mean of 2 tasks per interval, and the intense workload involved creating a mean of 4 tasks per interval. The initial aim was to have each dataset for over 200 intervals; this was not possible as latency issues significantly slowed down the dataset generation process (due to the random scheduler being used). However, since the number of VMs is 4, 100 scheduling intervals are deemed sufficient to train the models. The datasets include data such as CPU usage, RAM usage, and Disk I/O for both hosts and containers. The host IDs and numbers of containers present in each host is also available in the datasets, along with performance metrics such as response time, energy consumption, and number of SLA violations. The reason for generating 2 datasets is to train models over different workload conditions and quantify the effect the different models would have on performance for a single workload configuration.

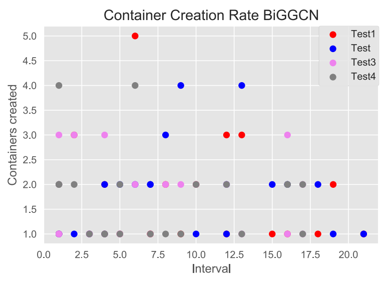
### 4.3. Training

The GGCN, Bidirectional GGCN and CNN models are trained on Datasets 1 and 2. The models take the utilization matrix of the active tasks and the capacity matrix of the target hosts as input. All models are trained using the Adam optimization algorithm, with a learning rate of 0.001, with the loss determined using the mean square error (MSE) loss function. Each model was trained for 100 epochs, and the losses are shown for training and testing in Fig. 4.

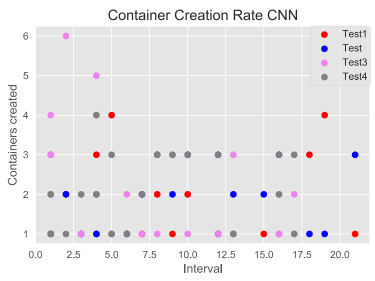
All models display a downward trend, though the GCNN model settles much more quickly than the other two models, which show a lot of fluctuation as the loss decreases. All testing and training losses settle under the 0.02 range, demonstrating good training [32].



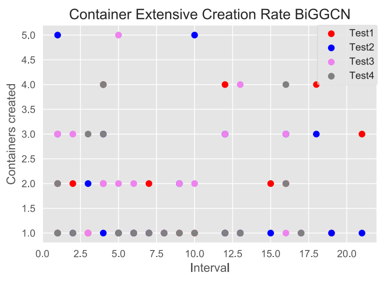
(a) Container Normal creation rate for GGCN



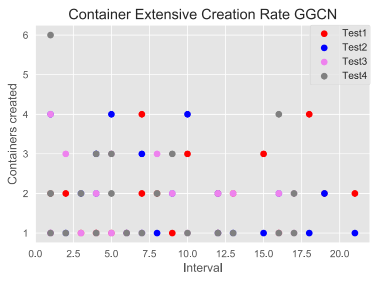
(b) Container Normal creation rate for BiGGCN



(c) Container Normal creation rate for CNN



(d) Container Extensive creation rate for BiGGCN



(e) Container Extensive creation rate for CNN

**Fig. 5.** Container creation rate for all the schedulers.

### 4.4. Testing

Testing was conducted with the aim of observing how the energy consumption QoS parameter varies under different training datasets and models. Each model was tested 4 times to ensure the reliability of the results. The testing was done for 20 intervals per test, with workloads created at a mean of 2 tasks per scheduling interval. Fig. 5 shows the rate of generation of containers for 4 tests of all schedulers.

### 4.5. Evaluation metrics

For comparison of the proposed approach with baseline schedulers (GGCN, Bidirectional GGCN), we considered the following metrics:

- Energy Consumption is calculated using Eq. (1).
- SLA violation rate is the sum of SLA violations divided by the sum of jobs where the value of $SLA_i$ is 1 if SLA of $Job_i$ is violated, o otherwise. It is calculated using Eq. (13).

$$SLA_{\text{violation}} = \frac{\Sigma\ SLAV_i}{\Sigma\ Job_i} \tag{13}$$

- Job Completion Rate is the ratio of completed jobs to the assigned Jobs. It is calculated as given in Eq. (14):

$$Job\ Completion\ rate = \frac{\Sigma\ Job_{completed}}{\Sigma\ Job_{assigned}} \tag{14}$$

- Scheduling time is the average time a scheduler takes to make a decision.
- Wait time is calculated as the average time a job spends in the waiting queue.

We choose Job completion ratio with other metrics as it is important for energy consumption to be minimized, it is equally important that assigned tasks are completed. This metric is different to the SLA violation metric, which looks at whether jobs are completed within a given time. This metric is unique to the given testing circumstances, otherwise SLA violation is the preferred metric when evaluating the reliability of the scheduler. This metric measured the ratio of completed jobs to assigned jobs.

**Table 2**
Energy consumption (Per completed task, KW-h).

| | GGCN 2/Interval | GGCN 4/Interval | BiGGCN 2/Interval | BiGGCN 4/Interval | CNN 2/Interval |
|---|---|---|---|---|---|
| Test1 | 105.5 | 99.0 | 103.1 | 93.8 | 71.4 |
| Test 2 | 122.7 | 103.4 | 103.1 | 68.5 | 74.8 |
| Test 3 | 133.9 | 119.9 | 84.3 | 78.3 | 57.8 |
| Test 4 | 75.3 | 71.5 | 89.8 | 92.8 | 72.4 |
| Average | 109.3 | 98.4 | 95.1 | 83.8 | 69.1 |

**Table 3**
Average job completion rate (Job completion ratio for all tests).

| | | GGCN 2/Interval | GGCN 4/Interval | BiGGCN 2/Interval | BiGGCN 4/Interval | CNN 2/Interval |
|---|---|---|---|---|---|---|
| T1 | Total containers | 37 | 39 | 35 | 38 | 37 |
| | Completed containers | 25 | 25 | 25 | 26 | 32 |
| | Completion ratio | 0.68 | 0.64 | 0.71 | 0.68 | 0.86 |
| T2 | Total containers | 43 | 40 | 38 | 41 | 37 |
| | Completed containers | 22 | 26 | 25 | 36 | 32 |
| | Completion ratio | 0.51 | 0.65 | 0.66 | 0.88 | 0.86 |
| T3 | Total containers | 39 | 36 | 36 | 46 | 48 |
| | Completed containers | 20 | 22 | 29 | 33 | 43 |
| | Completion ratio | 0.51 | 0.61 | 0.81 | 0.72 | 0.90 |
| T4 | Total containers | 38 | 41 | 37 | 33 | 44 |
| | Completed containers | 32 | 35 | 28 | 26 | 34 |
| | Completion ratio | 0.84 | 0.85 | 0.76 | 0.79 | 0.77 |
| Avg | Total containers | 39.25 | 39 | 36.5 | 39.5 | 41.5 |
| | Completed containers | 24.75 | 27 | 26.75 | 30.25 | 35.25 |
| | Completion ratio | 0.64 | 0.69 | 0.73 | 0.77 | 0.85 |

*Abbreviations used are — T1:Test 1, T2:Test 2, T2:Test 3, T4:Test 4, avg: Average.

### 4.6. Results

Fig. 6 shows QoS parameters comparison of Hunterplus with other baseline models. In this section, we will discuss the results of HunterPlus for all the specified evaluation metrics in detail.

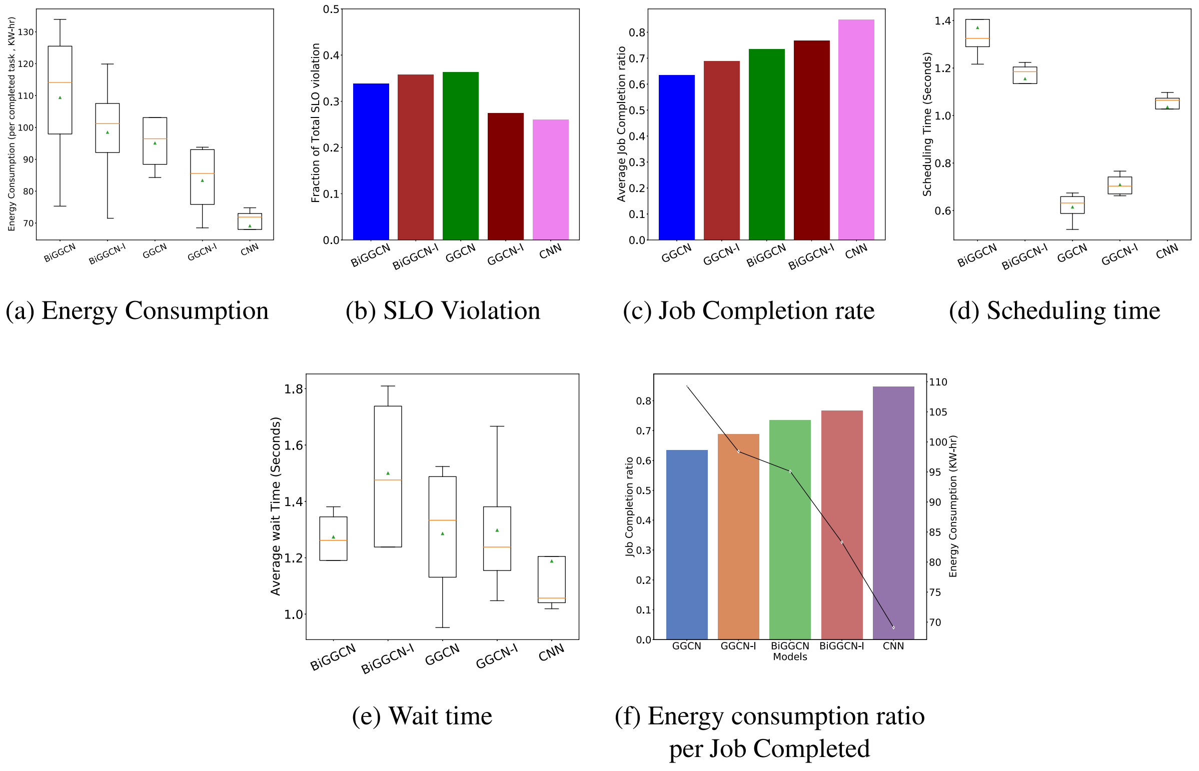
#### 4.6.1. Energy consumption

Table 2 and Fig. 6 summarizes the energy results for all the models.

As shown in Fig. 6(a), we observe that the GGCN-I and Bidirectional GGCN-I models outperform their normal counterparts, with the GGCN-I model saving an average of 10.9 KW-h worth of energy compared to the GGCN model, while the Bidirectional GGCN-I model saves an average of 12.2 KW-h worth of energy. The nature of the dataset upon which the model is trained has a clear and positive impact on performance. In this case, Dataset, which has a mean workload of 4 tasks every interval, can improve the performance of a task scheduling application where the mean workload is 2 tasks for every interval.

Another interesting observation is the variability in the data for GGCN compared to the other models. The ranges for the normal and intense GGCN models are 58.6 KWh and 48.4 KW-h respectively, whereas the maximum range in any of the remaining models is only 25.3 KW-h. Although extra testing is required for more robust analysis, the initial results indicate that the GGCN model performs unpredictably compared to the other models. The Bidirectional GGCN displays much more consistent performance, suggesting that the addition of the Bidirectional layer to the model and its ability to learn from both past and future data provides better context when scheduling decisions are taken. Parallel to this, the significant fluctuation in performance for the same model may also be in part attributed to the randomness of the jobs' selected. Although task generation is random, there is a possibility that some tests may have been repeatedly given easier tasks to perform which lent themselves to better scheduling. Again, this would influence scheduling performance, but it should not be as significant as the results seen. The performance benefit of adding a Bidirectional layer is clear when GGCN and Bidirectional GGCN are compared. For normal workloads, a Bidirectional GGCN consumes 14.2 KW-h worth of energy less per completed task than a GGCN. For intense workloads, this value increases to 15.1 KWhr. The more consistent performance range of the Bidirectional GGCN and better energy consumption performance clearly demonstrate the effectiveness of the Bidirectional layer.

The CNN model demonstrates significantly improved performance compared to the original GGCN and the Bidirectional GGCN models. It saves 14.2 KW-h more energy than the best performing GGCN-based system, whether trained on any of two datasets. Thus, CNN reduces energy consumption by at least 17% compared to the baseline methods. Furthermore, the CNN model has a range of only 17 KW-h, which is lower than all but 1 of the other models, demonstrating its ability to manage the inherent stochastic behavior in the system due to the dynamism in workload resource requirements and host resource capacities.



(a) Energy Consumption (b) SLO Violation (c) Job Completion rate (d) Scheduling time

(e) Wait time (f) Energy consumption ratio per Job Completed

**Fig. 6.** Comparison of HunterPlus with baseline Schedulers.

### 4.6.2. Job completion rate

While it is important for energy consumption to be minimized, it is equally important that assigned tasks are completed. This metric is different to the SLA violation metric, which looks at whether jobs are completed within a given time. This metric is unique to the given testing circumstances, otherwise SLA violation is the preferred metric when evaluating the reliability of the scheduler. This metric measured the ratio of completed jobs to assigned jobs. Table 3 and Fig. 6(c) summarizes the job completion rate results.

Job completion rates[1] are an important measure of performance for our models. Merely obtaining the energy consumed per task is insufficient as there is a possibility that excessive containers could have been deployed, which naturally means that more will be destroyed, which would drive the energy performance up. To preclude this possibility, from Table 3, we note that the CNN model has an average of 41.5 containers deployed across all tests. Although this is greater than for all other models, the range for average number of containers across all models is just 5, and for most models it is at most greater than half of this range. The mean workload is 2 tasks/interval, deployed over 20 intervals, which gives a total expected workload of 40 tasks. The COSCO framework uses a Python-based statistics library to create the workload using a normal distribution. However, for the sake of a complete analysis, it is worthwhile to show that the generated workloads really do belong to the specified normal distribution. We can use hypothesis testing to determine the likelihood of obtaining 41.5 containers from the workload generator. The mean workload is set at 2 tasks per interval, with a standard deviation (S.D) of 1.5. Therefore, this workload deployment has a normal distribution of $X \sim N(2, 2.25)$, where 2.25 is the variance obtained by squaring the standard deviation. To apply this normal distribution model, we take our sample size as 80.

This approach essentially combines all 4 tests together (T1:Test 1, T2:Test 2, T2:Test 3, T4:Test 4), where each test has 20 samples (intervals). The mean number of tasks deployed is calculated from the data is 2.075. The problem then boils down to determining how likely it is to obtain the above mean value from the $X \sim N(2, 2.25)$ distribution. The approach is to model the sample data using a distribution of the form $\bar{X} \sim N(2, 0.028)$, where 0.028 is obtained by dividing the variance by the sample size. Then, our null hypothesis is that the mean number of tasks is 2, while our alternative hypothesis is that the mean number of tasks is greater than 2.075. The probability $P\left(\bar{X} \geq 2.075\right)$ according to the $\bar{X} \sim N(2, 2.08)$ distribution is calculated as 0.33. As this is greater than 0.05, we can be confident that the results are statistically significant to a level of 5%, and that there is no excessive container deployment that may have skewed the results. Fig. 6(f) shows the relationship between energy consumption and the job completion ratio. As expected, a higher job completion ratio mirrors lower energy consumption per completed task. The CNN model demonstrates the best job completion ratio, completing an average of 85% percent of tasks within the schedulers run. Both

[1] Job completion ratio and Job completion rate are used interchangeably.

**Table 4**
Total containers, completed containers, and job completion rates.

| Interval | CPU(% Usage, Host0, Host1,Host2,Host2) |
|---|---|
| 12 | 100,100,0.2,100 |
| 13 | 100,100,23.1,100 |
| 14 | 100,100,30.5,100 |
| 15 | 100,100,0.1,100 |
| 16 | 100,100,26.5,100 |
| 17 | 100,100,75.9,100 |
| 18 | 75.5100,77.9,100 |

models for GGCN demonstrate variability in performance, like that observed in energy consumption. GGCN trained on dataset1 demonstrates the lowest job completion ratio, finishing just 64% of the created workloads. On the other hand, the GGCN trained on the dataset 2 completes 69% of the workload during its scheduling run. A similar trend is observed for the Bidirectional models, with 73% and 77% of tasks completed for normal and intense dataset-trained models respectively. Compared to the best baseline, CNN achieves an improvement in job completion ratio of 10.4%.

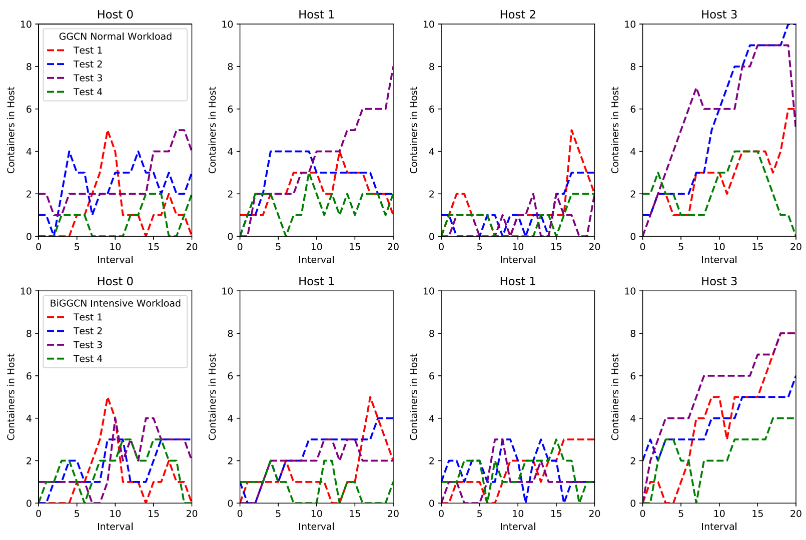
## *4.7. Task allocation analysis*

Artificial neural network schedulers can work well compared to other heuristic and meta-heuristic-based methods, but the reasoning behind scheduling decisions can still be rather opaque, and such networks are often referred to as 'black box' methods. This section aims to analyze some of the scheduling decisions taken by our models. We examine the fluctuations in the number of tasks assigned to host for all tests of any given scheduler. Starting with the GGCN scheduler, we note that the second test for this scheduler has an energy consumption of 122.7 KW-h, and a job completion rate of just 51%. The host allocation data for this shows a preference to minimize the number of tasks assigned to one of the hosts; in particular, before the 16th interval, the model assigns multiple tasks to all hosts except for Host 2. During this period, the model only assigns a new task to host 2 once the existing task has finished. However, when the existing total workload exceeds 15 containers (which happens after the 16th interval), Host 2 is assigned more tasks. This behavior suggests that the model is attempting to minimize energy consumption by utilizing one host as little as possible, while overloading the others. When the other hosts are completely overwhelmed, the scheduler starts assigning tasks to Host 2. This is observed in Table 4, which shows a cross-section of CPU performance for an interval range between intervals 12 and 18. We can observe that since all other hosts are at 100% capacity, the scheduler begins to use host 2. The scheduler uses Host 2 as the host of last resort.

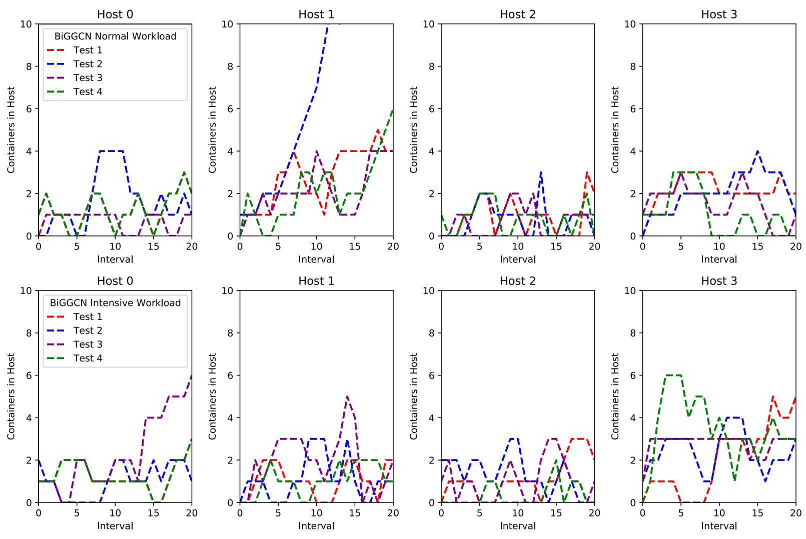
If we compare the results of Test 2 for the normal GGCN model with the other tests, we note that there is a tendency to under-assign Host 2. There is also a tendency to allocate an excessive number of tasks to Host 3. We note that both Test 2 and Test 3 have the highest workloads of the 4 tests, and have the worst energy performance and job completion ratios. This is an indication that the normal GGCN model is unable to handle intense and dynamic workloads. This is the reason for the poor energy performance of the normal GGCN model; the constant overloading of Host 3 means that many tasks are left unfinished. Consequently, the energy performance and the job completion ratio both suffer. Another thing to note is that Test 4 had the lowest workload, and the best energy consumption and job completion ratio results. This is further indication of the scheduler's inability to handle dynamic workloads.

The intense GGCN model displays similar behavior to the normal GGCN model, preferring to allocate as many tasks to Host 3 as possible. Test 4, although having a total of 41 tasks deployed over the 20 intervals, has a consistently low number of tasks deployed to each host. Only Host 3 sees a higher number of tasks towards the end of the scheduling process. There is also the same tendency to assign more tasks to Host 3, although it is relatively less pronounced. The results indicate that several tasks were completed simultaneously; a total of 8 tasks are completed over intervals 6 and 7. The reason for this is that the tasks in intervals 1–5 were distributed very evenly among the hosts, allowing quicker execution of the tasks. Another possible reason is that some of the initially deployed tasks were computationally less expensive than tasks that were deployed later in the scheduling regime.

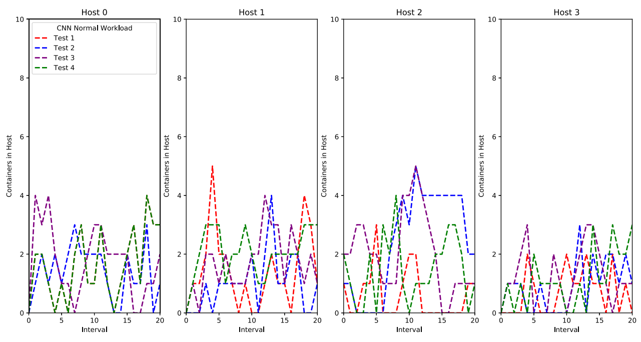
The normal Bidirectional GCNN displays somewhat more stable behavior when allocating and migrating tasks to hosts. It also displays a tendency to minimize the number of tasks allocated to Host 2; the corresponding chart in Fig. 7 clearly shows that there is hardly more than 1 task for any test case assigned to Host 2. Test 2 displays concerning unstable behavior, where the scheduler overloads Host 1 to an astonishing extent (we note that the graph overshoots; the maximum number of tasks in Host 1 was 12 for this test). During the period between intervals 4 and 8, a total of 13 tasks are created, which is excessive, and the Normal Bidirectional GGCN is unable to compromise properly between its attempt to keep Host 2 as empty as possible (to save energy) and allocating the incoming tasks to a host capable of handling them. It is possible the scheduler got stuck due to insufficient training. It is important to note that all models optimize the total energy of the system, rather than the energy consumed per task. This has an impact on performance as the model does not consider the execution time of the tasks. On the other hand, Test 4 shows that if the workload being deployed is not consistently too high over a series of intervals, the Normal Bidirectional GGCN scheduler can allocate tasks to hosts more effectively. The Intense Bidirectional GGCN scheduler also consolidates many tasks into a single host during periods of intense workload creation; however, it ensures that the host is not overwhelmed and does not ignore the other hosts present. The best illustrated by Test 3 and Test 4 for this scheduler.



(a) GGCN Container-Host Allocations with Normal and Intensive workload



(b) BiGGCN Container-Host Allocations with Normal and Intensive workload



(c) CNN Container-Host Allocations with Normal workload

**Fig. 7.** Container-Host Allocations with workload variations.

Finally, we examine the Normal CNN scheduler performance. There is a consistent and even distribution of tasks across all hosts, with an attempt to minimize usage of Host 3. The Normal CNN scheduler demonstrates exceptional ability to handle intense workloads; this can be seen clearly in Test 3. Intervals 8–13 saw a total of 20 tasks being created, yet the scheduler allocates all tasks effectively across all hosts, as shown by the noticeable increase in host task allocation across all hosts, instead of just one. Even though the scheduler generally minimizes task allocation to Host 3, it begins assigning more tasks to this host once excessive workloads are experienced.

## 5. Conclusions and future directions

This work proposes a new CNN-based resource scheduling approach called HunterPlus which extends the existing GGCN scheduler (HUNTER) and develops a new CNN scheduling model, and the results consistently demonstrate the superiority of the CNN model over the GGCN and Bidirectional GGCN schedulers according to the energy consumption per task and job completion ratio metrics. The nature of the dataset has an impact on scheduler performance; datasets generated from higher workload baselines are more effective in training relatively stable schedulers. The addition of the Bidirectional layer also tends to improve performance. GGCN based schedulers tend to consolidate as many tasks as possible in a minimal number of hosts. This strategy proves ineffective when workload deployment is high for consecutive numbers of intervals, as tasks are not executed as quickly. Finally, HunterPlus proves to be the best scheduler among those considered, according to all evaluation metrics. The stable behavior of the CNN model is also shown by the consistent and even task allocation and migration actions.

Many avenues are possible for further research. The current work requires that each host configuration have its own dataset; for example, the datasets used to train the models in this work would be ineffective in a configuration with 5 hosts. This constraint may be solved by preprocessing the data into a normalized form that is agnostic to the number of hosts present in the configuration. The use of a CNN also proved effective in this work, but it does not account for temporal information in the data, i.e., how it changes over time. It only takes a single snapshot of performance at a given time. Creating a combined RNN and CNN model would permit the exploitation of temporal information and possibly improve performance even more. There is also scope to incorporate system reliability metrics when training the model in addition to the energy metric used in this work. For this experiment, we realized task with docker container instance as considered but in the future experiments, we aim to consider other scenarios related to scalability and reliability.

## Software availability

All code, datasets and result reproducibility scripts are publicly available as part of a GitHub repository under CC-BY License. The repository can be accessed at: https://github.com/sun-das/HunterPlus/tree/ggcn.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## Acknowledgments

Sundas Iftikhar would like to thank the Higher Education Commission (HEC) Pakistan, for their support and funding. This work is partially funded by Chinese Academy of Sciences President’s International Fellowship Initiative (Grant No. 2023VTC0006), and Shenzhen Science and Technology Program (Grant No. RCBS20210609104609044). We thank Prof. Fatos Xhafa (Editor-in-Chief), Associate Editor and anonymous reviewers for their constructive suggestions and guidance on improving the content and quality of this paper.

## References

[1] Jagdeep Singh, et al., Fog computing: A taxonomy, systematic review, current trends and research challenges, J. Parallel Distrib. Comput. 157 (2021) 56–85.
[2] Sukhpal Singh Gill, Minxian Xu, Carlo Ottaviani, Panos Patros, Rami Bahsoon, Arash Shaghaghi, Muhammed Golec, Vlado Stankovski, Huaming Wu, Ajith Abraham, et al., AI for next generation computing: Emerging trends and future directions, Int. Things 19 (2022) 100514.
[3] Sundas Iftikhar, et al., FogDLearner: A deep learning-based cardiac health diagnosis framework using fog computing, in: Australasian Computer Science Week 2022, ACM, 2022, pp. 136–144.
[4] Sundas Iftikhar, et al., Fog computing based router-distributor application for sustainable smart home, in: 2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring), IEEE, 2022, pp. 1–5.
[5] Saurabh Shrivastava, Bahgat Sammakia, Roger Schmidt, Madhusudan Iyengar, Comparative analysis of different data center airflow management configurations, in: International Electronic Packaging Technical Conference and Exhibition, Vol. 42002, 2005, pp. 329–336.

[6] Baptiste Durand-Estebe, Cédric Le Bot, Jean Nicolas Mancos, Eric Arquis, Data center optimization using PID regulation in CFD simulations, Energy Build. 66 (2013) 154–164.

[7] Ananya Chakraborty, et al., Journey from cloud of things to fog of things: Survey, new trends, and research directions, Softw. - Pract. Exp. (2022).

[8] Ridwan Rashid Noel, Palden Lama, Taming performance hotspots in cloud storage with dynamic load redistribution, in: 2017 IEEE 10th International Conference on Cloud Computing, CLOUD, IEEE, 2017, pp. 42–49.

[9] Xiang Li, Xiaohong Jiang, Peter Garraghan, Zhaohui Wu, Holistic energy and failure aware workload scheduling in cloud datacenters, Future Gener. Comput. Syst. 78 (2018) 887–900.

[10] Young Choon Lee, Albert Y. Zomaya, Energy efficient utilization of resources in cloud computing systems, J. Supercomput. 60 (2) (2012) 268–280.

[11] Sukhpal Singh Gill, Peter Garraghan, Vlado Stankovski, Giuliano Casale, Ruppa K. Thulasiram, Soumya K. Ghosh, Kotagiri Ramamohanarao, Rajkumar Buyya, Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge, J. Syst. Softw. 155 (2019) 104–129.

[12] Yongyi Ran, Han Hu, Xin Zhou, Yonggang Wen, Deepee: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019, pp. 645–655.

[13] Shi Chen, Jie Wu, Zhihui Lu, A cloud computing resource scheduling policy based on genetic algorithm with multiple fitness, in: 2012 IEEE 12th International Conference on Computer and Information Technology, IEEE, 2012, pp. 177–184.

[14] Shreshth Tuli, Sukhpal Singh Gill, Minxian Xu, Peter Garraghan, Rami Bahsoon, Schahram Dustdar, Rizos Sakellariou, Omer Rana, Rajkumar Buyya, Giuliano Casale, et al., HUNTER: AI based holistic resource management for sustainable cloud computing, J. Syst. Softw. 184 (2022) 111124.

[15] Luana Ruiz, Fernando Gama, Alejandro Ribeiro, Gated graph recurrent neural networks, IEEE Trans. Signal Process. 68 (2020) 6303–6318.

[16] Mpyana Mwamba Merlec, A Dynamic Resource Management and Task Migration System for Mobile Ad Hoc Computational Cloud, Korea University (KERIS DCollection), 2017.

[17] Ipsita Kar, R.N. Ramakant Parida, Himansu Das, Energy aware scheduling using genetic algorithm in cloud data centers, in: 2016 International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT, IEEE, 2016, pp. 3545–3550.

[18] Ziqian Dong, Ning Liu, Roberto Rojas-Cessa, Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers, J. Cloud Comput. 4 (1) (2015) 1–14.

[19] Chun-Wei Tsai, Joel J.P.C. Rodrigues, Metaheuristic scheduling for cloud: A survey, IEEE Syst. J. 8 (1) (2013) 279–291.

[20] Ying Changtian, Yu Jiong, Energy-aware genetic algorithms for task scheduling in cloud computing, in: 2012 Seventh ChinaGrid Annual Conference, IEEE, 2012, pp. 43–48.

[21] Jing Liu, Xing-Guo Luo, Xing-Ming Zhang, Fan Zhang, Bai-Nan Li, Job scheduling model for cloud computing based on multi-objective genetic algorithm, Int. J. Comput. Sci. Issues (IJCSI) 10 (1) (2013) 134.

[22] Xiao-Fang Liu, Zhi-Hui Zhan, Ke-Jing Du, Wei-Neng Chen, Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 41–48.

[23] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, Dan Wang, Cloud task scheduling based on load balancing ant colony optimization, in: 2011 Sixth Annual ChinaGrid Conference, IEEE, 2011, pp. 3–9.

[24] Joseph Nathanael Witanto, Hyotaek Lim, Mohammed Atiquzzaman, Adaptive selection of dynamic VM consolidation algorithm using neural network for cloud resource management, Future Gener. Comput. Syst. 87 (2018) 35–42.

[25] Alex Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, Physica D 404 (2020) 132306.

[26] Md Ebtidaul Karim, Mirza Mohd Shahriar Maswood, Sunanda Das, Abdullah G. Alharbi, Bhyprec: a novel bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine, IEEE Access 9 (2021) 131476–131495.

[27] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, Laith Farhan, Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions, J. Big Data 8 (1) (2021) 1–74.

[28] Fatma M. Talaat, Hesham A. Ali, Mohamed S. Saraya, Ahmed I. Saleh, Effective scheduling algorithm for load balancing in fog environment using CNN and MPSO, Knowl. Inf. Syst. 64 (3) (2022) 773–797.

[29] Shreshth Tuli, Shashikant Ilager, Kotagiri Ramamohanarao, Rajkumar Buyya, Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks, IEEE Trans. Mob. Comput. (2020).

[30] Charles Findling, Vasilisa Skvortsova, Rémi Dromnelle, Stefano Palminteri, Valentin Wyart, Computational noise in reward-guided learning drives behavioral variability in volatile environments, Nature Neurosci. 22 (12) (2019) 2066–2077.

[31] Shreshth Tuli, Shivananda R. Poojara, Satish N. Srirama, Giuliano Casale, Nicholas R. Jennings, COSCO: Container orchestration using co-simulation and gradient based optimization for fog computing environments, IEEE Trans. Parallel Distrib. Syst. 33 (1) (2021) 101–116.

[32] Jason Brownlee, Loss and loss functions for training deep learning neural networks, Mach. Learn. Mastery 23 (2019).

**Sundas Iftikhar** is a Ph.D. Scholar at the School of Electronic Engineering and Computer Science, Queen Mary University of London. Prior to this, she held positions as Research associate and Lecturer at University of Kotli Azad Jammu and Kashmir, Azad Kashmir Pakistan. She did her Masters in computer software engineering from National University of Science and Technology, Pakistan. Her research interest include Cloud computing, Fog computing, and resource Management in Fog

**Mirza Mohammad Mufleh Ahmad** is a Computer Science M.Sc. student at Queen Mary, University of London, UK. Currently, he is Graduate Software Engineer at NatWest Group. Mirza has a B.Sc. in Computer Science from Middle East Technical University, Turkey. His research interests include IoT, Machine Learning and Cloud Computing.

**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. He has worked as a visiting research fellow at the CLOUDS Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. His research interests include Fog Computing and Deep Learning.

**Deepraj Chowdhury** is with department of electronics and communication, IIIT-Naya Raipur. He has Co-authored 4 research papers in different conferences like ICACCP 2021, INDICON 2021. He also has 3 Indian copyright registered, and applied for 2 Indian Patent. He is also serving as a reviewer in Wiley Transaction on emerging Telecommunication Technologies

**Minxian Xu** is currently an associate professor at Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. He received the B.Sc. degree and the M.Sc. degree, both in software engineering from University of Electronic Science and Technology of China. He obtained his Ph.D. degree from the University of Melbourne in 2019. His research interests include resource scheduling and optimization in cloud computing. He has co-authored 30+ peer-reviewed papers published in prominent international journals and conferences, such as ACM Computing Surveys, IEEE Transactions on Sustainable Computing, IEEE Transactions on Cloud Computing, Journal of Parallel and Distributed Computing, Software: Practice and Experience, International Conference on Service-Oriented Computing. His Ph.D. Thesis was awarded the 2019 IEEE TCSC Outstanding Ph.D. Dissertation Award. He is member of CCF and IEEE. More information can be found at: http://www.minxianxu.info.

**Sukhpal Singh Gill** is a Lecturer (Assistant Professor) in Cloud Computing at the School of Electronic Engineering and Computer Science, Queen Mary University of London, UK. Prior to his present stint, Dr. Gill has held positions as a Research Associate at the School of Computing and Communications, Lancaster University, UK and also as a Postdoctoral Research Fellow at CLOUDS Laboratory, The University of Melbourne, Australia. Dr. Gill is serving as an Associate Editor in Wiley ETT and IET Networks Journal. He has co-authored 70+ peer-reviewed papers (with H-index 30+) and has published in prominent international journals and conferences such as IEEE TCC, IEEE TSC, IEEE TII, IEEE IoT Journal, Elsevier JSS and IEEE CCGRID. He has received several awards, including the Distinguished Reviewer Award from SPE (Wiley), 2018, Best Paper Award AusPDC at ACSW 2021 and has also served as the PC member for venues such as PerCom, UCC, CCGRID, CLOUDS, ICFEC, AusPDC. His research interests include Cloud Computing, Fog Computing, Software Engineering, Internet of Things and Energy Efficiency. For further information, please visit http://www.ssgill.me.

**Steve Uhlig** obtained a Ph.D. degree in Applied Sciences from the University of Louvain, Belgium, in 2004. From 2004 to 2006, he was a Postdoctoral Fellow of the Belgian National Fund for Scientific Research (F.N.R.S.). His thesis won the annual IBM Belgium/F.N.R.S. Computer Science Prize 2005. Between 2004 and 2006, he was a visiting scientist at Intel Research Cambridge, UK, and at the Applied Mathematics Department of University of Adelaide, Australia. Between 2006 and 2008, he was with Delft University of Technology, the Netherlands. Prior to joining Queen Mary University of London, he was a Senior Research Scientist with Technische Universität Berlin/Deutsche Telekom Laboratories, Berlin, Germany. Since January 2012, he has been the Professor of Networks and Head of the Networks Research group at Queen Mary, University of London. Between 2012 and 2016, he was a guest professor at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. With expertise in network monitoring, large-scale network measurements and analysis, and network engineering, during his career he has been published in over 100 peer-reviewed journals, and awarded over £3million in grant funding. Awarded a Turing Fellow, Steve is also the Principal Investigator on a new project funded by the Alan Turing Institute: 'Learning-based reactive Internet Engineering' (LIME). He is currently the Editor in Chief of ACM SIGCOMM Computer Communication Review, the newsletter of the ACM SIGCOMM SIG on data communications. Since December 2020, Steve has also held the position of Head of School of Electronic Engineering and Computer Science. Current Research interests: Internet measurements, software-defined networking, content delivery.