CARSS: Leveraging Online Arrival Forecasting and Co-Simulation for Adaptive Scheduling

Yichong Chen, Manuel Roveri, Shreshth Tuli, *Member, IEEE,* and Giuliano Casale

***Abstract*—Coupled simulation, also known as co-simulation, has been proposed to support task schedulers by simulating, at runtime, the Quality of Service (QoS) resulting from scheduling actions. However, existing co-simulation methods typically assume a static arrival time series. This assumption limits the diversity of traffic scenarios. To address this, we propose an online adaptive arrival forecasting framework that integrates a change-point detection module and a probabilistic transformer model to couple co-simulators with arrival series forecasting. This framework also updates the prediction model in response to detected changes. Additionally, we introduce the Co-simulated Adaptive Recurrent Surrogate Scheduler (CARSS), which uses simulated QoS metrics from the co-simulator to make scheduling decisions that enhance system performance. Experiments show that our online adaptive forecasting framework has lower forecasting errors than traditional models and reduces co-simulator prediction error by 11% in average response time and 22% in average service-level agreement (SLA) violation on real-world traces. Furthermore, CARSS improves QoS metrics, achieving 2.5% reduction in average energy consumption and reductions of 8.2% and 82.7% in average response time and average SLA violation, respectively, compared to the best baseline scheduler.**

***Index Terms*—fog computing, time series forecasting, change point detection, co-simulation**

I. Introduction

Fog computing is a recent paradigm that combines the benefits of Edge and Cloud Computing. It places latency-sensitive applications on edge devices to reduce latency and enhance privacy, while offloading resource-intensive tasks to the Cloud to lower infrastructure costs. However, the performance of fog nodes is limited by their computing power, storage, and battery, highlighting the need for efficient resource allocation and task scheduling. Existing studies propose various scheduling strategies, such as heuristic, artificial intelligence/machine learning (AI/ML), and deep learning (DL) based algorithms, to optimize Quality-of-Service (QoS), defined as the level of performance, availability and reliability provided by the hosts [1]. Most of these studies assume that the topology, arrival process, bandwidth, etc., remain unchanged. However, real-world fog systems often face dynamic changes, such as host additions/removals or concept drift in task arrivals. We use the term *concept drift* to refer to any change in the distribution of arrivals. In particular, a location shift denotes a change in the mean arrival rate, whereas a scale shift denotes a change in the variability of the arrivals [2], [3].

Manuscript received Month DD, YYYY

Y. Chen and G. Casale are with the Department of Computing, Imperial College London, London, UK

M. Roveri is with DEIB, Politecnico di Milano, Milano, Italy

S. Tuli is with Happening Technology Ltd. London, UK

Co-simulation, which simulates the response of a system to real-time changes, has been used as a digital-twin of a Fog system to help scheduling algorithms to improve QoS [4], [5]. Prior work has examined its benefits in static settings, where arrival process parameters are fixed. However, such assumptions fail to capture concept drift in real-world systems, which impedes the co-simulator from providing valuable estimations. This paper introduces a mechanism to dynamically update the arrival traces used by the co-simulator, enabling it to better reflect current system behavior. These updated traces are then used to obtain accurate QoS estimates and guide the scheduling algorithm to improve QoS.

To ensure that the co-simulator provides a better estimation of QoS parameters, we dynamically generate an arrival series, represented as a time series of counts for each job type, as the input trace to the co-simulator using a probabilistic transformer. We also introduce an online adaptive framework to adapt the arrival generation model to the dynamic environment. This framework monitors the arrival series with a change point detection module and fine-tunes the transformer model when changes are detected.

We propose a hierarchical change detection structure based on the theory in [6] as the detection module. While the original framework effectively detects mean changes (location shifts), it overlooks changes in variability (scale shifts), which are common in real-world arrival traces, such as the Alibaba trace [7]. To address this limitation, we introduce an extended data-depth plus CUSUM (DD^{+}-CUSUM) algorithm to detect both increases and decreases in scale and form a hierarchical change point detection algorithm (HCPD).

We also introduce CARSS: **C**o-simulated **A**daptive **R**ecurrent **S**urrogate **S**cheduler, an AI-based scheduling algorithm that predicts both immediate and future QoS metrics based on scheduling decisions. It leverages arrival series forecasts from the proposed adaptive framework within a fog system co-simulator, enabling the surrogate model to predict QoS metrics for future steps accurately. The proposed scheduler optimizes both short-term and long-term QoS metrics using gradient descent to determine effective scheduling decisions.

Experiments on a real-world dataset show that our forecasting framework enables more accurate co-simulation results than offline artificial parameterization. HCPD detects and estimates change points more precisely and with less delay than Bayesian methods. The probabilistic transformer model further improves arrival series forecasting, thereby enhancing QoS estimation accuracy. Additionally, the online adaptive mechanism refines the transformer model to produce lower MAE in QoS estimation. Meanwhile, CARSS improves QoS

by reducing energy consumption and accelerating task processing on real-world workloads. The adaptive forecasting framework also reduces system operational costs compared to existing arrival prediction methods.

In summary, the key contributions of this paper are:

- We propose a non-parametric online change point detection algorithm that detects location and scale shifts and estimates the change point.
- We design a probabilistic transformer model to forecast the arrival series with a single forward pass.
- We develop an online adaptive arrival forecasting framework that improves arrival prediction for co-simulators, enabling more accurate QoS estimation in dynamic environments.
- We introduce CARSS, a scheduler integrated with a co-simulation and adaptive arrival forecasting framework, to generate scheduling decisions that improve QoS in a fog computing system.

This paper is structured as follows. We begin by discussing related work in Section II. Section III defines the adaptive prediction problem, and Section IV outlines the techniques used in the proposed model. The proposed online adaptive arrival series prediction model is detailed in Section V, followed by the CARSS scheduler in Section VI. Section VII presents the experiment settings and results, and we conclude in Section VIII. Compared to a preliminary version of this work [8], this article utilizes the online adaptive arrival forecasting framework and introduces the novel co-simulated adaptive recurrent surrogate scheduler in Section VI. Experimental results in Section VII show that the proposed CARSS scheduler outperforms the baseline methods.

II. Related Work

A. Co-simulation & Scheduling

The resource allocation and task scheduling problem aims to identify optimal task-node pairs that satisfy constraints, such as fulfilling an SLA, and optimize the considered QoS parameters, such as response time. However, [9] shows that finding the optimal scheduling policy for a set of tasks is usually NP-hard. Consequently, most studies adopt heuristic approaches to obtain near-optimal solutions that meet constraints and maintain high QoS. For example, [10], [11] present task scheduling algorithms that minimize system operational costs, such as energy consumption and device usage fees, without violating SLAs.

Resource allocation and task scheduling have also been studied as joint optimization problems. [12] jointly determines computation offloading and transmission scheduling for delay-sensitive applications, [13] makes reliability-aware offloading decisions that balance energy against reliability, and [14] schedules workflow applications across end-edge-cloud tiers. AI/ML methods have also been applied to resource allocation and task scheduling in fog computing. Several studies use deep reinforcement learning to adapt to stochastic workloads by learning a scheduling policy that predicts the reward for each task–node pair [15], [16]. However, these RL models often adapt slowly in real-world application scenarios [4].

Co-simulation supports scheduling by running a virtual model of the system to predict the QoS of candidate decisions before they are applied, as in COSCO and GOBI*, where simulated QoS guides AI schedulers and improves decision robustness [4], [5]. However, these methods do not adapt to concept drift in the arrivals. Other works use virtual or predictive models, often named digital twins, to guide resource allocation and adapt as conditions evolve [17]. For instance, [18] keeps a virtual model of a human synchronized with its physical counterpart through periodic federated retraining gated by a quality-based validation process. [19] re-optimizes resource allocation for reconfigurable intelligent surface-assisted interaction with a prompt-guided decision transformer when the scene changes, improving QoS under uncertain evolution. [20] forecasts future requests at the edge and splits compute between model retraining and inference under concept drift. However, these works adapt to change by retraining on a fixed schedule or reacting only to directly observed or time-based signals. CARSS instead detects concept drift online through hierarchical change-point detection, so it adapts only when arrivals shift and keeps forecasts accurate without unnecessary retraining.

B. Change Detection Test & Change Point Detection

Change detection tests (CDT) are used to identify abrupt changes in time-series properties such as mean, variance, or correlation. The cumulative sum (CUSUM) has been widely used as an online change detection algorithm [21]. Traditional CUSUM assumes prior knowledge of the distribution family and its parameters, which often does not hold in practice.

Change point detection (CPD) is more challenging compared to change detection since it aims both at detecting and locating the occurrence of a change in a finite sequence of data. [22] detects a change point by separating the time series into two sub-series and finding the split point with the maximum difference. Instead of using statistical tests, [23] applies a kernel function, which projects the features to an infinite-dimensional space, to measure the difference between two sub-series. [24] introduces the Bayesian online change point detection (BOCPD) to identify the changes in an online fashion. However, this method requires assumptions on the distribution family of the time series.

C. AI/ML-Based Time Series Forecasting

Forecasting has been widely applied in resource management and scheduling [1]. Autoregressive integrated moving average (ARIMA) is a traditional statistical model widely used for time-series forecasting [25]. It assumes a linear correlation between past and present observations at different time lags and can also capture trends in non-stationary time series. However, real-world time series may not always be captured effectively by the recursive form assumed for time-series data in autoregressive processes.

DL models are also used in time series analysis. Recurrent neural network (RNN) architectures, such as vanilla RNN, long short-term memory (LSTM) [26], and gated recurrent unit (GRU) [27], are commonly used for time series prediction.

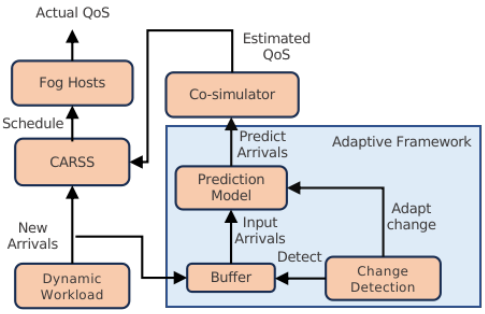


Fig. 1: A dynamic Fog system with an adaptive digital-twin co-simulator

They recurrently process the time series in order to capture correlation features. DeepAR [28] uses an RNN structure to model probabilistic features of the series and predict parameters of the distribution. Although RNN models are effective for modeling time-series data, RNN architectures are slow to train, and their forecasting performance degrades as input length increases.

The transformer model [29] is another popular architecture for time-series forecasting. The attention mechanism in transformers captures correlations across time instances. Moreover, a transformer is faster to train than an RNN model. Some transformer models, such as Longformer [30], SparseTransformer [31], and Transformer-XL [32], are designed for long sequence forecasting. However, standard transformer models are slow during inference because they predict one future instance at a time and use the newly predicted value to make the next prediction.

III. Problem Formulation

We consider a dynamic Fog system that includes a digital-twin co-simulator used to estimate system QoS, as shown in Figure 1. We study a discrete-time scheduling problem in which the Fog system receives tasks from a gateway that collects tasks and sends them to the broker periodically. The scheduling decisions are made by our proposed CARSS scheduler using QoS estimates from the co-simulator. The tasks are executed on the assigned hosts, and the actual QoS is computed.

The distribution of the arrival process can change over time. We wish to design a framework that predicts the number of arrivals in L step ahead with the last t_0 observation of arrivals and adapts the prediction model when concept drift occurs on the distribution of the arrivals. The prediction and adaptation mechanism form our proposed framework.

We define the number of arrivals at time t as $\mathbf{x}_t = (x_t^1, \ldots, x_t^c) \in (\mathbb{Z}^*)^c$, where x_t^i is the number of class ith tasks arriving at time t and c is the number of classes of jobs. Here $\mathbb{Z}^*$ denotes the set of non-negative integers. Thus, we try to predict the future arrival series $\mathbf{X}_{t_0+1:t_0+L} = \{\mathbf{x}_t, t = t_0+1, t_0+2, \ldots, t_0+L\}$ denoting as prediction series, with the latest observations series $\mathbf{X}_{0:t_0} = \{\mathbf{x_t}, t = 0, 1, \ldots t_0\}$ denoting as condition series. Instead of forecasting the exact value of the prediction series, we model the distribution of the prediction series $P(\mathbf{X}_{t_0+1:t_0+L}|\mathbf{h})$ with $\mathbf{h}$ as the parameters of the distribution. We predict $\mathbf{h}$ with a neural network f_θ from the conditioning series $\mathbf{X}_{0:t_0}$ so that $\mathbf{h} = f_\theta(\mathbf{X}_{0:t_0})$, where θ refers to the parameters in the neural network. The goal of our time series forecasting is to find θ that maximizes the likelihood of the prediction series given the condition series.

The prediction model is trained on the historical arrival trace. However, the model may fail to provide accurate predictions when the arrival distribution shifts. Therefore, we introduce a change point detection mechanism within the framework to detect changes in the arrival series. When a change is detected, the prediction model is fine-tuned using the most recent arrivals.

Considering an arrival series $\mathbf{X}_\mathcal{T}$, we assume that it is possible to separate the time series into two non-overlapping partitions such that the distribution of each partition is different. The time instances that divide the time series are the change points. Under this scenario, the change point detection problem can be formulated as follows

$$\mathbf{x}_t \sim \begin{cases} \Phi_0 & t < t^* \\ \Phi_1 & t \geq t^* \end{cases} \tag{1}$$

where Φ_0 is the distribution of the arrivals before the change occurs, Φ_1 is the distribution after the location or scale shift, and t^* is the change point we want to identify.

IV. Background

A. Hierarchical Change Detection Test

CUSUM-type change detection algorithms are effective and have long been used in online change detection [21]. The standard CUSUM-type algorithm cumulatively sums the log-likelihood ratio at each time point. The paradigm detects changes when the cumulative sum exceeds the preset control limits. However, these CUSUM-type change detection algorithms cannot estimate the exact change point in the series.

Hypothesis tests can be used to estimate the location of the change point and are named change point methods. These change point methods split the time series into two sub-series and compute the statistical test value of the difference between the sub-series. Hypothesis testing is performed on the splitting using the maximum statistical test value. The splitting point is estimated as the change point if the hypothesis test rejects the null hypothesis. However, these methods cannot be applied in online detection because performing the hypothesis test in an online setting increases the probability of making a false positive detection, known as sequential testing issues [33].

To address this, [6] proposes a hierarchical change detection test (HCDT), that combines CUSUM change detection algorithms and hypothesis change point methods. HCDT monitors the time series with the CUSUM algorithm. If the new data triggers the CUSUM detection, HCDT performs a hypothesis test using the change point method to validate the detection. The HCDT framework does not perform hypothesis testing at every step to prevent sequential testing issues. Meanwhile, validating the CUSUM detection using the hypothesis change-point method can further reduce false-positive detection.

B. ProbSparse Attention

The transformer model has been widely used for processing time-series data and has achieved significant success across many domains. The attention mechanism introduced in [29] plays a vital role in the transformer model. The attention module takes query $\mathbf{Q} \in \mathbb{R}^{L_Q \times d_m}$, key $\mathbf{K} \in \mathbb{R}^{L_K \times d_m}$ and value $\mathbf{V} \in \mathbb{R}^{L_V \times d_m}$ as inputs, where L_Q, L_K and L_V represent the length of the corresponding series and d_m is the dimension of an instance in the input series. The attention function is formulated as

$$\mathcal{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_m}}\right)\mathbf{V} \tag{2}$$

The output of the attention function $\mathcal{A}$ is the weighted sum of $\mathbf{V}$, where the weights are calculated from the dot product between $\mathbf{Q}$ and $\mathbf{K}$, followed by the softmax function applied row-wise.

Some of the previous works [30], [31] have noted that the attention weights of query q_i have potential sparsity, which means only a few dot-product pairs (q_i, k_j) with large weight contribute to the attention, where q_i and k_j are the ith row and jth row of query $\mathbf{Q}$ and key $\mathbf{K}$ respectively. In this scenario, the attention function should extract key features from the value $\mathbf{V}$ with dominant weights. On the contrary, the attention function reduces to computing the mean of the vector $\mathbf{V}$ when the attention weights are identical. Calculating attention becomes trivial in this context, reducing processing speed and increasing memory usage when processing long sequences.

The *ProbSparse* attention defines a query sparsity measurement to evaluate the potential sparsity of a query q_i [34]. Following the formulation in [35], the attention weight of a dot-product pair (q_i, k_j) can be written as a conditional probability $p(k_j|q_i)$. If sparsity does not exist in the attention weights, the attention weights would be close to a uniform distribution, i.e., $q(k_j|q_i) = 1/L_K$. The Kullback-Leibler divergence $KL(q||p)$ can be used to measure the distance between the attention probability $p(k_j|q_i)$ and the uniform distribution $q(k_j|q_i)$. KL divergence increases when sparsity exists in the attention weights, and it decreases when the attention weights are identical. Therefore, KL divergence can represent the sparsity of the attention weights for q_i. The sparsity measurement of q_i is defined as

$$M(q_i, \mathbf{K}) = \max_j\left(\frac{q_i k_j^T}{\sqrt{d_m}}\right) - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_i k_j^T}{\sqrt{d_m}} \tag{3}$$

To reduce the calculation on dot-product pairs, *ProbSparse* attention randomly selects $\ln L_K$ keys to calculate the sparsity measurement for a query q_i. The queries with the top $\rho \ln L_Q$ measurement are selected and form the new query matrix $\bar{\mathbf{Q}}$ and replace the original query $\mathbf{Q}$ in Eq. 2, where ρ is a tunable sampling factor.

V. Adaptive Arrival Prediction Model

In this section, we present a hierarchical framework to predict a long-term arrival series for the co-simulator to estimate performance metrics of new arrival tasks under the default scheduling policy.

A. Non-parametric Hierarchical Change Point Detection

HCDT is an effective framework for detecting changes in an online scenario. However, HCDT still requires assumptions on distribution parameters and can only detect changes in location. Furthermore, our framework requires knowledge of when the change happens to adapt the prediction to the latest valid data. Therefore, we develop a nonparametric change-point detection method within the HCDT paradigm that estimates change points by detecting both changes in location and scale. We refer to the proposed method as hierarchical change point detection (HCPD).

To detect both location and scale shifts in the time series, we apply MCUSUM and DD$^+$-CUSUM change detection algorithms. MCUSUM, presented in [36], is a non-parametric location shift detection algorithm. It replaces the log-likelihood ratio with the difference between the new observation and the sample mean and sets the control chart on the sum of these differences. DD-CUSUM [37] can be used to detect the scale increase. However, the original work does not account for a decrease in scale shift. To address this issue, we propose an extended version of DD-CUSUM named data-depth plus CUSUM (DD$^+$-CUSUM) to detect both increase and decrease shifts in scale.

The data depth *R* statistic measures the distance between an observation and the mean of a distribution. Given observed $\mathbf{x}_t$ and historical data set $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, the sample data depth is defined as

$$DD_{\hat{\Phi}_0}(\mathbf{x}_t) = 1 - \frac{1}{n}\left\|\sum_{i=1, \mathbf{y}_i \neq \mathbf{x}_t}^{n} U(\mathbf{x}_t - \mathbf{y}_i)\right\| \tag{4}$$

where $\hat{\Phi}_0$ represents the empirical distribution of the data, and the operation $U(x) = \frac{x}{||x||}$. Thus, $DD_{\hat{\Phi}_0}(\mathbf{x})$ is close to 0 if the observation $\mathbf{x}$ is far away from the center of the empirical distribution $\hat{\Phi}_0$. On the contrary, $DD_{\hat{\Phi}_0}(\mathbf{x})$ becomes large and attains the maximum value 1 if the observation is near the center of $\hat{\Phi}_0$.

The original DD-CUSUM algorithm only considers the increase of the scale by cumulatively summing the estimated *R* calculated from

$$R^+_{\hat{\Phi}_0}(\mathbf{x}_t) = \frac{\sum_{i=1}^{n} I(DD_{\hat{\Phi}_0}(\mathbf{y}_i) \leq DD_{\hat{\Phi}_0}(\mathbf{x}_t))}{n} \tag{5}$$

which is the count of historical data with a smaller depth than the new observation, where $I(\cdot)$ is the indicator function. If the new observation $\mathbf{x}_t$ is near the center of the historical data set $\mathbf{y}_i$, the data depth of $\mathbf{x}_t$ would be greater than most of the historical data, which leads to a large *R* statistic value. Hence, $1 - R^+_{\hat{\Phi}_0}(\mathbf{x}_t)$ can represent the distance between $\mathbf{x}_t$ and the empirical distribution of the historical data.

To detect a decrease in scale, we first estimate the negative *R* statistic by counting historical data points with larger data depth than the new observation. The negative estimate can be represented as

$$R^-_{\hat{\Phi}_0}(\mathbf{x}_t) = \frac{\sum_{i=1}^{n} I(DD_{\hat{\Phi}_0}(\mathbf{y}_i) \geq DD_{\hat{\Phi}_0}(\mathbf{x}_t))}{n} \tag{6}$$

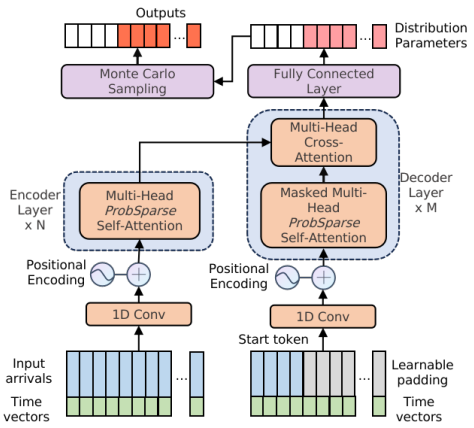


Fig. 2: The architecture of the probabilistic transformer

Then, DD$^+$-CUSUM cumulatively sums both positive and negative R statistic measures, and the control chart is performed on the one with the maximum value. The complete DD$^+$-CUSUM can be formed as follows:

$$S_t^+ = \max(0, S_{t-1}^+ + (1 - R_{\hat{\Phi}_0}^+(\mathbf{x}_t)) - k) \quad (7)$$

$$S_t^- = \max(0, S_{t-1}^- + (1 - R_{\hat{\Phi}_0}^-(\mathbf{x}_t)) - k) \quad (8)$$

$$S_t = \max(S_t^+, S_t^-) \quad (9)$$

where k is the reference allowance that controls detection sensitivity and h is the alarm boundary on the accumulated statistic S_t, where a larger k or h reduces false alarms but lengthens the detection delay [38]. DD$^+$-CUSUM detects a change when $S_t > h$.

The MCUSUM and DD$^+$-CUSUM monitor the arrival simultaneously and store the new observation in a buffer with a fixed size. The buffer drops the oldest observation and adds the newest one when it reaches its size limit. HCPD detects a potential change if either one of these two CUSUM algorithms raises a detection flag.

Once the detection algorithms trigger the alarm, a validation module uses the Lepage-type (LP) hypothesis test [39], which can detect both location and scale shifts, to validate the change and estimate the change point with the stored data in the buffer. We compute the LP test value for every possible split and perform a hypothesis test on the one with the maximum. If the LP test confirms the change, we set the splitting point with the maximum statistical test value as the estimated change point.

B. Arrival Series Forecasting

We design a transformer-based encoder-decoder model with the *ProbSparse* attention as the self-attention module to predict the long-term arrival series. The overall structure of our transformer model is shown in Figure 2.

1) Encoder: The encoder takes the arrival series and time features as input. The time features of the arrival series are represented as a vector and concatenated with the input arrival series as additional features. The input series is first processed with a 1-D convolution layer to project the input series $\mathbf{X} \in \mathbb{R}^{L_{enc} \times c}$ into a high-dimensional space so that $\mathbf{X}_{enc} \in \mathbb{R}^{L_{enc} \times d_{model}}$. The projected series is processed with a positional encoding layer to incorporate temporal order.

The encoder contains multiple self-attention blocks. In each self-attention block, the input is processed with a multi-head self-attention module with *ProbSparse* attention. The multi-head self-attention projects the encoded input $\mathbf{X}_{enc}$ to multiple queries $\mathbf{Q}$ and key-value pairs $\mathbf{K}$-$\mathbf{V}$ so that these projected vectors contain different information that the attention should focus on. The output of the multi-head self-attention module is added to the input of the attention module as a residual block [40] so that the loss can back-propagate through the attention module or directly pass to the input.

2) Decoder: In a standard encoder-decoder transformer model, the decoder takes the encoder output and a start token or the previous predictions as input and produces the next prediction. However, the previous predictions are unavailable at the beginning of the co-simulation scenario. Meanwhile, the model needs to have fast inference to avoid long scheduling times. Unlike [34], which adds zero paddings to the input of the decoder to make predictions simultaneously, we set a sequence of learnable parameters with the length of maximum prediction L_{max} to replace the zero paddings. In the meantime, a subsequence of the encoder input series of length L_{start} serves as the decoder start token. Therefore, the start subsequence and the learnable padding parameters form the input series of the decoder $\mathbf{X}_{dec} \in \mathbb{R}^{(L_{start}+L_{max}) \times d_m}$. Timestamps are also added as time features to the corresponding inputs.

The decoder also has multiple layers. In each layer, X_{dec} is first processed with a multi-head masked *ProbSparse* attention. The masked dot product prevents each position from receiving information from future positions. The self-attention output serves as the query for the cross-attention layer, and the encoder output forms the key-value pairs. The decoder can therefore receive the history series and process it along with the decoder input to predict future arrivals.

3) Distribution Prediction: The arrival series represents the number of tasks that are integers at each timestamp. Commonly, the standard outputs of a DL model are continuous real numbers. To address this issue, one can round the output to the nearest integer, but this rounding can lose information and ignore the distribution feature of count data. Instead of directly predicting the number of future arrivals, we predict the parameters of the distribution of future arrivals. Since the arrival series is a sequence of positive counts whose variance can exceed its mean, our transformer model predicts the parameters of a negative binomial distribution. Unlike a Poisson distribution, which assumes equal mean and variance, the negative binomial captures the bursty arrivals in our traces. The probability of the arrival number of class ith tasks at time t can be represented as

$$p(x|\mu,\kappa) = \frac{\Gamma(x+\frac{1}{\kappa})}{\Gamma(x+1)\Gamma(\frac{1}{\kappa})} \left(\frac{1}{1+\kappa\mu}\right)^{\frac{1}{\kappa}} \left(\frac{\kappa\mu}{1+\kappa\mu}\right)^{x} \quad (10)$$

where $\mu \in \mathbb{R}^+$ is the mean and $\kappa \in \mathbb{R}^+$ is the shape parameter of the corresponding negative binomial distribution. Therefore, the decoder output is passed through two independent fully connected linear layers with a softplus activation function

Algorithm 1 Adaptive Arrival Prediction Framework

Require: History series $\mathbf{X}_{his}$, input length L_{in}, prediction length L_{pred}, fine-tuning length L_{fine}, buffer $\mathbf{B}$ with fixed length, sampling number N_{sample}

1: **Initialization:** HCPD, Probabilistic Transformer f_θ
2: Apply HCPD to detect change point on $\mathbf{X}_{his}$
3: Sample change-point free series from $\mathbf{X}_{his}$
4: Train f_θ with change-point free series
5: Fine-tuning flag $g \leftarrow False$
6: **while** $\mathbf{x}_t$ is available at t **do**
7: Update $\mathbf{B}$ with $\mathbf{x}_t$
8: **if** HCPD detects a change point with $\mathbf{x}_t$ **then**
9: Estimated change point $\hat{t}$
10: $g \leftarrow True$
11: Update $\mathbf{B} \leftarrow \mathbf{B}[\hat{t} : t]$
12: **if** g **then**
13: **if** len($\mathbf{B}$) $> L_{fine}$ **then**
14: Sampling fine-tuning data from $\mathbf{B}$
15: Fine-tuning f_θ
16: $g \leftarrow False$
17: Predict $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\kappa}} \leftarrow f_\theta(\mathbf{X}_{t-L_{in}:t})$
18: Sample N_{sample} arrival series with $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\kappa}}$

to predict $\hat{\mu}$ and $\hat{\kappa}$. The transformer model is trained by minimizing the negative log-likelihood function.

C. Adaptive Arrival Series Prediction Framework

In real-world systems, the distribution of task arrivals may change over time. A DL model trained with historical arrivals may struggle to adapt its predictions to unseen arrival distributions without additional assistance. The prediction of the new arrival distribution might not be as good as the seen one. This can reduce simulation accuracy and negatively affect scheduling, thereby affecting migration and allocation decisions. One solution is to fine-tune and update the DL model when a new observation becomes available.

The procedure of the online adaptive arrival prediction framework is presented in Algorithm 1. Suppose we have a dataset of the historical data $\mathbf{X}_{his}$. Before the online procedure (lines 1-5), HCPD is used to find the change points in $\mathbf{X}_{his}$. The training series is generated from $\mathbf{X}_{his}$, excluding those containing change points. The series with change points contains arrivals from different distributions, which increases the difficulty for the transformer to learn to predict the parameters of future arrivals, as it must recognize the change point position and focus only on data after it. Therefore, we sample the change-free series to enhance the training procedure.

During the online prediction (lines 6-18), the framework observes a new arrival $\mathbf{x}_t$ and saves it in the buffer. Once HCPD detects a change point $\hat{t}$, the framework updates the buffer $\mathbf{B}$ to include arrivals after $\hat{t}$ and raises a flag for the transformer to execute the fine-tuning procedure. In the fine-tuning procedure, the framework first checks the buffer length (line 13). If the buffer contains sufficient data, the framework generates fine-tuning samples from it and fine-tunes the transformer (lines 14-15). Otherwise, the framework defers fine-tuning to gather additional data.

The transformer predicts the parameters $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\kappa}}$ of the future distributions with series $\mathbf{X}_{t-L_{in}:t}$ obtained from $\mathbf{B}$ (line 17). Finally, the framework uses Monte Carlo sampling to generate N_{sample} arrival series with the predicted parameters, each with length L_{pred} (line 18).

To estimate future QoS, the system integrates a co-simulator. This coupled co-simulator runs simulations using the predicted arrival series, schedules them with a predefined scheduler, and computes QoS estimates based on the mean of the simulation results. We denote the co-simulation QoS as $Q^{sim}_{t:t'}$, where t' represents the last interval required to complete the execution of the tasks scheduled at time t.

VI. Co-simulated Adaptive Recurrent Surrogate Scheduler

In this section, we introduce CARSS, a co-simulated adaptive recurrent surrogate scheduler that integrates a deep recurrent neural network surrogate model to predict system QoS metrics and to derive scheduling decisions based on these predictions. Since we consider a discrete-time scheduling problem, the scheduling interval t refers to the period from time t until the start of interval $t+1$.

Before detailing CARSS, we introduce the QoS metrics considered in this study. Specifically, we evaluate three QoS metrics: energy consumption (EC) of the system over the scheduling interval t, average response time (ART), and average service-level agreement violation (ASLAV), all computed up to the end of interval t for tasks scheduled at t.

The EC for interval t is defined as the total energy consumed by the hosts in the system over interval t

$$\mathrm{EC}_t = \sum_{i=1}^{N_{host}} Energy_t^{(i)} \tag{11}$$

where N_{host} is the number of hosts and $Energy_t^{(i)}$ is the energy consumed by the ith host during interval t, measured in kJ. The ART for time t is given as

$$\mathrm{ART}_t = \frac{1}{N_{task}} \sum_{i=1}^{N_{task}} rt_t^{(i)} \tag{12}$$

where N_{task} is the number of tasks that need to be scheduled at the beginning of interval t, and $rt_t^{(i)}$ is the response time of the i-th task, measured up to the end of interval t. The ASLAV is formulated as

$$\mathrm{ASLAV}_t = \frac{1}{N_{task}} \sum_{i=1}^{N_{task}} \max(rt_t^{(i)} - sla^{(i)}, 0) \tag{13}$$

where $sla^{(i)}$ is the maximum response time contractually stipulated in the SLA of the i-th task.

Existing studies, such as [4], [15], mainly focus on optimizing QoS metrics within the current scheduling period and do not account for the long-term effects of scheduling decisions. In contrast, the objective of CARSS is to optimize both immediate and long-term QoS. To achieve this, we incorporate a digital twin co-simulator of the fog system to

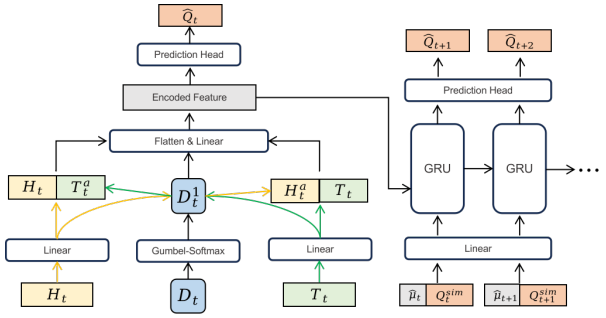


Fig. 3: The architecture of the recurrent surrogate model

estimate long-term QoS metrics. For each scheduling interval at time t, CARSS uses the predicted arrival series as input to the co-simulator to estimate the long-term QoS metrics $Q_{t:t'}^{sim}$. These estimated QoS metrics provide information about future system behavior that guides the scheduler and helps to optimize both short-term and long-term QoS objectives.

A. Recurrent Surrogate Model

Metaheuristics such as Tabu search and genetic algorithms with co-simulation tend to be slow for real-time use, as they require multiple simulation runs. To address this, we introduce a recurrent surrogate model that predicts system QoS given the scheduling decision D_t at step t. Moreover, this surrogate model integrates an RNN, which takes $Q_{t:t'-1}^{sim}$, from the co-simulator and predicts the QoS over the next t' steps, denoted as $\hat{Q}_{t:t'}$, when decision D_t is applied. Here, t' represents the last interval required to complete the execution of the tasks scheduled at time t, which can be determined through co-simulation.

The recurrent surrogate model consists of two components: the left-hand side of Figure 3 encodes features of tasks T_t, hosts H_t and scheduling decision D_t, while the right-hand side is an RNN that takes the encoded features and simulated QoS as input to model temporal dependencies and predict future QoS.

Host features H_t include utilization metrics such as instruction per second (IPS), CPU frequency, RAM, Disk, and Bandwidth consumption. Task features T_t include CPU utilization, RAM and storage requirements, SLA, and the total execution time. The sequence of tasks and hosts is first processed through a linear layer with softplus activation to encode them into a unified feature space.

The scheduling decision D_t is represented as a matrix of one-hot vectors of size $N_{host} \times N_{task}$. An entry $d_t^{(ij)} = 1$ indicates that task j is assigned to host i. Based on this decision matrix, the model aggregates task encodings to form task hosting features for each host and selects the corresponding host encoding as an additional feature for each task.

Therefore, for each host, the aggregated task encoding is computed as $T_t^a = D_t\phi(T_t)$, where T_t^a represents the aggregated task encoding, and $\phi(T_t)$ denotes the task encoding. The aggregated task encoding is then concatenated with the corresponding host encoding to form the combined latent features $\phi(H_t)||T_t^a$, where $||$ is the concatenation operation.

Algorithm 2 CARSS

Require: Iteration limit σ; Convergence Criterion $\mathcal{C}$.
1: Run Algorithm 1 to obtain N_{sample} arrival series
2: Run co-simulation to obtain $Q_{t:t'-1}^{sim}$.
3: Initialize D_t, $i \leftarrow 0$, $\mathcal{O}^* \leftarrow +\infty$
4: **while** $i < \sigma$ and $\neg\mathcal{C}$ **do**
5: $\quad D_t^1 \leftarrow$ Gumbel-softmax(D_t)
6: $\quad$ Compute objective $\mathcal{O}$ with recurrent surrogate model
7: $\quad$ Update D_t with gradient descent
8: $\quad i \leftarrow i + 1$
9: $\quad$ **if** $\mathcal{O} < \mathcal{O}^*$ **then**
10: $\quad\quad D_t^* \leftarrow D_t^1$, $\mathcal{O}^* \leftarrow \mathcal{O}$
return D_t^*

Similarly, for each task, the corresponding host encoding is selected as $H_t^a = D_t^T\phi(H_t)$, where H_t^a is the aggregated host encoding and $\phi(H_t)$ is the host encoding. The selected host encoding is concatenated with the corresponding task encoding to form the combined latent features $\phi(T_t)||H_t^a$.

As the task number varies across intervals, we pad the task set to a fixed maximum size. The concatenated task and host encodings, together with the decision matrix, are then flattened and combined into a system feature tensor of fixed width. This tensor is processed through multiple linear layers with softplus activation to generate an encoded system representation, which serves as input to a prediction head that estimates the QoS metrics $\hat{Q}_t$ (EC, ART, and ASLAV) at step t. We set the dimension of this representation to the GRU hidden state, so it also initializes the hidden state of the GRU and lets the model capture temporal dependencies for future QoS prediction regardless of the number of active tasks.

An RNN is used to predict future QoS rather than a transformer-based model, as it is more efficient and better suited to incorporating system embeddings as hidden states in this scenario. The RNN input consists of the simulated $Q_{t:t'-1}^{sim}$ obtained from the co-simulation, along with the corresponding predicted mean parameter $\hat{\mu}_{t+1:t'}$. The RNN outputs are processed through a prediction head to estimate QoS at the future $\hat{Q}_{t+1:t'}$ when decision D_t is applied. In the proposed model, we employ a GRU as the RNN component because it can control how much information is retained or discarded from the hidden state.

The recurrent surrogate model is trained using mean squared error (MSE) loss between the predicted $\hat{Q}_{t:t'}$ and actual QoS values. To construct the training dataset, fog system simulations are performed using various schedulers, including Random, MAD-MC [10], LR-MMT [41], and GOBI [4]. At each step, host and task features, scheduling decisions, and observed QoS metrics are recorded. These training data will first be used to train the system embedding, as shown on the left-hand side of Figure 3. The training data obtained from the scheduler employed in the co-simulation is further used to train the RNN.

B. Scheduling Optimization

The procedure of CARSS to obtain a scheduling decision is presented in Algorithm 2. For each scheduling step, the

adaptive arrival prediction framework is applied to generate N_{sample} arrival series. The coupled simulator then executes simulations using these sequences, obtaining the estimated $Q^{sim}_{t:t'-1}$ (line 1-2).

The initial scheduling decision D_t is derived from the task assignments. For tasks already running in the system, the one-hot vector encodes their current host assignment. For new tasks, the one-hot vector is randomly initialized. Given the selected task and host features, the scheduling decision D_t, and $Q^{sim}_{t:t'-1}$, the recurrent surrogate model predicts the QoS. The optimization objective function is formulated as

$$\mathcal{O} = \sum_{i=t}^{t'} \alpha \text{EC}_i + \beta \text{ART}_i + \omega \text{ASLAV}_i \tag{14}$$

where α, β and ω are non-negative weights with $\alpha+\beta+\omega = 1$. The scheduler aims to minimize $\mathcal{O}$ by adjusting the scheduling decision D_t. Since D_t is an input to the recurrent surrogate model, back-propagation can be applied to compute the gradient of $\mathcal{O}$ with respect to D_t. Therefore, the decision matrix D_t is updated using gradient descent $D_t = D_t - \gamma \nabla_{D_t}(\mathcal{O})$, where γ is the learning rate (line 7).

However, direct gradient updates result in a decision matrix D_t that no longer consists of one-hot vectors, which can reduce the accuracy of the prediction of the surrogate model. To restore a valid one-hot vector representation, we apply the Gumbel-softmax function [42], defined as

$$y_{ij} = \frac{\exp((\log(d_t^{(ij)}) + b_i)/\tau)}{\sum_{k=1}^{N_{host}} \exp((\log(d_t^{(kj)}) + b_k)/\tau)} \tag{15}$$

where $d_t^{(ij)}$ is the score for assigning task j to host i from D_t and b_i is sampled from a Gumbel(0, 1) distribution. The hyperparameter τ controls the sharpness of the output distribution. The Gumbel-softmax can approximate the argmax function as $\tau \to 0$. During the forward pass, an argmax operation is applied to the Gumbel-softmax output to obtain the one-hot vector, forming a valid decision matrix D_t^1 from D_t for QoS prediction (line 5). In the backward pass, gradients flow through the Gumbel-softmax while bypassing the argmax operation, allowing the scheduling decision D_t to be updated via back-propagation.

This iterative update process continues until D_t converges or reaches the maximum allowed iterations. The one-hot decision matrix D_t^1 corresponding to the smallest loss $\mathcal{O}^*$ is selected as the final scheduling decision.

VII. Experimental Results

A. Dataset

1) Synthetically Generated Dataset: The synthetically generated dataset contains a sequence with 12000 observations and 20 change points, derived from a negative binomial distribution. By adjusting the count number n and the shape factor p, we can control the mean and variance of the observations. The gap between two consecutive changes is at least 400 steps and at most 1000 steps. The n and p are initialized as 50 and 0.5, respectively, and n is increased or decreased by 25 or 50 at each location shift. The shape factor is chosen from 0.1 and 0.5 to simulate the scale shift. Each observation represents arrivals gathered by the gateway over a 60-second period.

2) Alibaba Trace Dataset: We use the 2018 Alibaba arrival trace [7] as a real-world application dataset to evaluate our model since the arrival of tasks exhibits random changes. This dataset records data from over 4000 machines across 8 days. To simplify without losing generality, we focus on arrivals to the first 50 machines by ID. The trace comprises 12 task classes. We count the arrivals for every 60 seconds as one observation, forming a trace of 11219 observations.

For both datasets, the first 7000 steps form the training split and the remaining steps form the evaluation split. Trainable models are fit on the training split, and all schedulers are assessed on the evaluation split.

B. Experiment settings

1) COSCO Simulation Framework: We use the COSCO framework presented in [4] as the simulator because it is explicitly designed for discrete-time Fog system simulation. Tasks are generated from a given arrival series, and their characteristics, including CPU usage, RAM demand and disk requirements, are derived from the BitBrain workload trace [43]. A scheduling broker allocates these tasks based on a specific scheduling algorithm. COSCO executes the assigned tasks in the respective simulated hosts within a specified interval and records the QoS metrics. In our experiments, we simulate a fog environment with 300 heterogeneous machines configured according to the Azure B2s, B4ms, and B8ms instance types. Specifically, we simulate the fog nodes with 60 Azure B2s, 60 Azure B4ms, and 30 Azure B8ms. At the same time, we treat the same Azure machine configuration as cloud nodes located far from the fog layer. We also model the latency and network characteristics between the fog and cloud environments in the simulator, following the network model of COSCO [4]. Cloud nodes have a higher link latency than fog nodes, so placing a task on a cloud node adds a transfer delay that increases response time. We report the machine and network configuration in Table S1 of the online supplement.

2) Baseline Models: Three statistical prediction models: mean prediction, moving average (MA), and auto-regressive integrated moving average (ARIMA) [25] are selected as the baselines in the arrival forecasting experiment. We also consider two widely used deep learning based time series forecasting models, LSTM and DeepAR [28]. For all models, the most recent 50 observations are used as history data, which we refer to as the reference series. Each model is fitted to the reference series to predict the next 25 steps.

We evaluate the performance of CARSS by comparing it against three different scheduling approaches. Two heuristic-based schedulers are considered. The first, *MAD-MC*, calculates the median absolute deviation of CPU utilization with the maximum correlation to other tasks [10]. The second, *LR-MMT*, employs linear regression to predict the overloaded hosts and migrates tasks to the host with the shortest migration time [41].

Additionally, we compare our approach with GOBI, a deep learning-based scheduler [4]. Similar to our method, GOBI

uses a surrogate model to predict system QoS and applies gradient-based optimization to schedule resources. GOBI is trained on the training split of the workload traces. However, unlike our method, it lacks both a long-term perspective and online adaptation, which limits its performance.

3) Adaptive Arrival Forecasting Framework: The HCPD threshold is also selected to achieve an ARL of 500 using the same experiments as for the baseline change point detection models. The probabilistic transformer contains 6 encoder self-attention layers and 6 decoder layers, each with a hidden dimension of 512. The model receives a history sequence of length 50 as the reference series and uses the last 25 steps as the start subsequence to forecast the arrival parameters for the next 25 steps.

4) CARSS: To train the recurrent surrogate model, we generate training data by running the GOBI scheduler for 7000 steps on the simulator. Here, the trained GOBI scheduler serves only as a data-generation policy that produces the QoS traces used to train the recurrent surrogate. The α, β and ω in the loss function (Eq.14) are set to 0.2, 0.4 and 0.4 respectively. To balance scheduling time and convergence, we set the maximum iteration σ to 100, and convergence is reached when the one-hot decision matrix remains unchanged for 50 iterations. During the update procedure, the learning rate γ is set to 0.1, while the hyper-parameter of Gumbel-softmax τ is initialized at 0.5 and decreases by 1% per iteration until it reaches 0.1.

C. Evaluation Metrics

1) Arrival Prediction Evaluation Metrics: When evaluating the performance of the arrival prediction models, we calculate the mean absolute error (MAE) between the QoS of simulations run with predicted traces and those with true arrivals.

2) Scheduler Evaluation Metrics: In addition to the QoS metrics discussed earlier, which are EC, ART, and ASLAV, we also consider the operational cost of the system when comparing the performance of different schedulers. The total cost consists of two components, the energy consumption cost, measured in US dollars, and the compensation paid to customers for SLA violations. The electricity price for the industrial sector in the United States is approximately \$0.0853 per kWh[1]. The energy consumption cost, C_{energy}, is calculated by converting the per-interval energy consumption from kJ to kWh and multiplying the result by the electricity price.

The SLA violation penalty is determined by the extent of the delay beyond the SLA. We set the maximum penalty at \$50 per task, with the actual compensation computed as a fraction of this amount based on the violation severity. The penalty for the i-th task exceeding its SLA is formulated as

$$C_{sla}^{(i)} = \frac{\max(rt^{(i)} - sla^{(i)}, 0)}{sla^{(i)}} \cdot C_{max} \tag{16}$$

[1]Source: Statista, U.S. industrial electricity price (2024).

TABLE I: The MAE of ART and ASLAV estimated with forecasting models on Synthetic trace and Alibaba trace

Dataset	Synthetic trace		Alibaba trace	
	ART	ASLAV	ART	ASLAV
Mean	24.97	4.58	40.33	18.59
MA	24.17	4.27	52.84	23.57
ARIMA	26.16	7.45	47.87	22.96
LSTM	25.70	6.61	38.72	16.81
DeepAR	24.70	5.47	35.87	14.31
NonAdapt	24.72	5.38	34.87	12.68
Adaptive	**15.28**	**3.09**	**31.74**	**11.21**

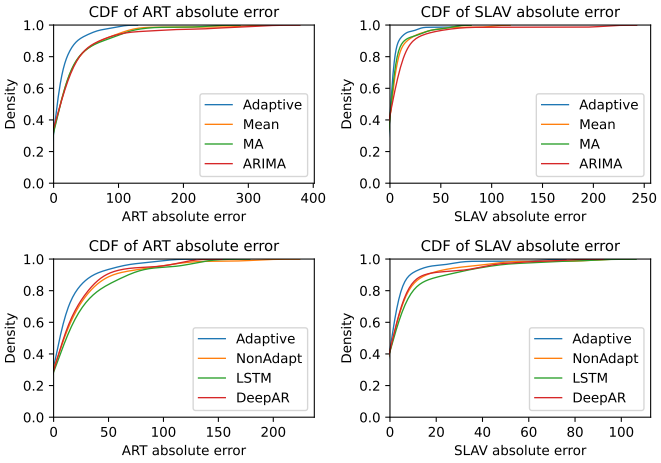


Fig. 4: The CDFs of the absolute error of the QoS metrics on synthetically generated trace

where $C_{sla}^{(i)}$ is the penalty incurred for an SLA violation, and C_{max} is the maximum penalty per task. Therefore, the total operational cost of the system at each step is given by

$$C = C_{energy} + \sum_{i=1}^{N_{task}} C_{sla}^{(i)} \tag{17}$$

Operational cost is calculated at each scheduling interval, and the mean operational cost is considered as one of the QoS metrics in the experiments.

D. Results

1) Framework evaluation: To evaluate the performance of our proposed online adaptive framework, we run simulations using a first-come, first-served (FCFS) with Round-Robin strategy with a synthetically generated trace and the Alibaba arrival trace, starting at the 7000th step, with ART and ASLAV as sample labels at each step. The framework and five baseline models make forecasts at each step. We also evaluate the effect of the adaptive mechanism by removing HCPD and the fine-tuning procedure from the framework, resulting in a model without adaptation. The digital-twin co-simulator runs simulations using predicted arrivals to estimate the ART and ASLAV for the corresponding step. The performance of the model is measured by the mean absolute error between the estimated ART and ASLAV and their true values.

Table I shows the MAE of the ART and ASLAV estimated by the co-simulator with arrivals estimated by the corresponding models. Our model outperforms the baseline models,

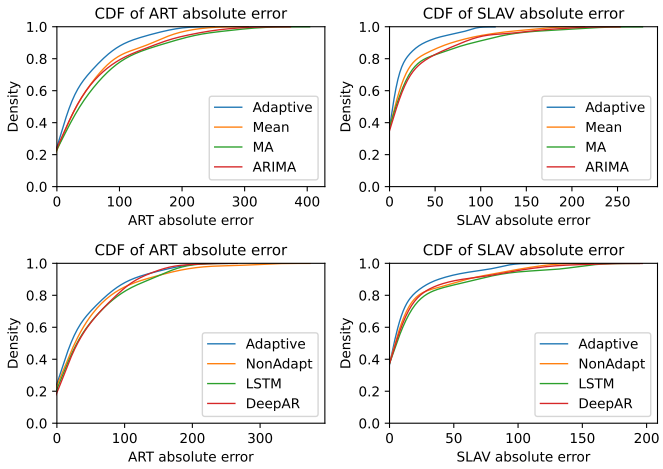


Fig. 5: The CDFs of the absolute error of the QoS metrics on Alibaba trace

TABLE II: QoS metrics of schedulers on Synthetic trace and Alibaba trace

Dataset	Methods	EC (kJ)	ART (s)	ASLAV (s)	Cost ($)
Synthetic	Round-Robin	1839.27	646.31	7.0138	10.244
	MAD-MC	1810.66	527.71	0.2407	0.3931
	LR-MMT	1737.54	524.25	0.1860	0.3118
	GOBI	1703.63	512.78	0.2691	0.4318
	CARSS	**1689.50**	**485.53**	**0.0042**	**0.0461**
Alibaba	Round-Robin	1835.83	765.55	29.5620	52.2986
	MAD-MC	1826.62	541.42	10.2390	15.6006
	LR-MMT	1753.05	543.85	10.4768	15.9615
	GOBI	1731.91	529.75	4.2725	6.5338
	CARSS	**1688.57**	**486.43**	**0.7372**	**1.1604**

TABLE III: Average scheduling time by scheduler type.

	MAD-MC	LR-MMT	GOBI	CARSS
Synthetic	0.03s	2.13s	0.61s	1.09s
Alibaba	0.05s	2.20s	0.63s	1.29s

achieving a 37% lower MAE for ART and a 28% lower MAE for ASLAV on the synthetic trace. In addition, the proposed model achieves the best performance, with MAE 11% and 22% lower on ART and ASLAV, respectively, on the Alibaba trace.

Figures 4 and 5 show the cumulative distribution function (CDF) of the absolute error of ART and ASLAV on the synthetic trace and Alibaba trace. These figures show that most of the traces predicted by the proposed model can well represent future arrivals. Therefore, the QoS estimated by the simulator using the adaptive framework is more accurate than that of the other baseline methods.

These tables and figures also show that the adaptation mechanism enables the arrival forecasting model to adapt to concept drift in the arrival trace. The MAE of ART and ASLAV estimated with the adaptive framework is 38% and 42% lower than that without the adaptation mechanism (NonAdapt) on the synthetic trace, and 9% and 12% lower when tested on the Alibaba trace. The adaptive framework is also more robust than the non-adaptive one.

We evaluate the complexity of the adaptive framework by its average prediction time, inclusive of fine-tuning. The MA and ARIMA models take 0.074 and 0.083 seconds, respectively, on the synthetic trace, and 0.081 and 0.087 seconds on the Alibaba trace. Our adaptive framework takes 0.072 seconds on the synthetic trace and 0.094 seconds on the Alibaba trace. The processing speed of our framework is comparable to MA, 15% faster than ARIMA on the synthetic trace, but 13% and 7% slower than MA and ARIMA on the Alibaba trace due to more frequent change detection and fine-tuning.

2) Scheduler evaluation: Table II shows the EC, ART and ASLAV of CARSS and three baseline schedulers, MAD-MC, LR-MMT and GOBI, after simulating on both the synthetic trace and the Alibaba trace. We also report the performance of the system using the naive Round-Robin scheduler. Regarding EC, CARSS reduces energy consumption compared to MAD-MC and maintains a similar level to LR-MMT and GOBI. Specifically, it reduces EC to 1689.50 kJ and 1688.57 kJ, which are improved by up to 6.6% and 7.5% on the Synthetic and Alibaba traces, respectively.

Additionally, our approach significantly improves ART and ASLAV. Specifically, on the Synthetic trace, our method reduces ART to 485.53 seconds, lower by up to 8.0%, and attains the lowest ASLAV of 0.004 seconds among all schedulers. On the Alibaba trace, our method achieves an ART of 486.43 seconds, a reduction of up to 10.6%, and again the lowest ASLAV at 0.74 seconds, against 4.27 to 10.48 seconds for the baselines.

These results demonstrate the effectiveness of CARSS in minimizing task energy consumption and response time across diverse workload conditions. Furthermore, CARSS attains the lowest mean operational cost on both traces. It costs $0.05 against $0.31 to $0.43 for the baselines on the synthetic trace, and $1.16 against $6.53 to $15.96 on the Alibaba trace.

Table III shows the average scheduling time[2] of each scheduler. CARSS requires 1.09 and 1.29 seconds to obtain a scheduling decision on the Synthetic trace and Alibaba trace, respectively, whereas LR-MMT takes more than two seconds. Although MAD-MC and GOBI make scheduling decisions faster than CARSS, the QoS achieved with CARSS is substantially better and the resulting QoS metric is much lower. The overhead of about one second is small relative to the scheduling interval, during which many tasks are allocated. This overhead is the cost of the additional optimization that lowers ART, ASLAV, and operational cost. We therefore consider it acceptable for online scheduling in a large-scale fog system.

3) CARSS with different arrival forecasting methods: We further evaluate the effectiveness of the proposed adaptive arrival prediction framework by comparing QoS metrics with CARSS integrated with the Mean, MA, ARIMA, LSTM, and DeepAR predictors.

The results in Table IV show that CARSS, combined with the adaptive arrival prediction framework, achieves lower ART on both the synthetic and Alibaba traces, and the lowest

[2]Experiments are performed on a system with configuration: AMD Ryzen 7 5700X, Nvidia RTX 4080

TABLE IV: QoS metrics of CARSS integrated with various arrival prediction methods on the Synthetic and Alibaba traces

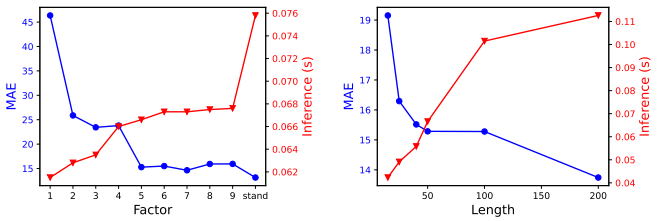
Dataset	Methods	EC (kJ)	ART (s)	ASLAV (s)	Cost ($)
Synthetic	Mean	1703.45	496.09	0.0023	0.0438
	MA	1702.09	495.01	0.0016	0.0427
	ARIMA	1704.22	496.84	0.0026	0.0441
	LSTM	1699.27	489.68	0.0061	0.0491
	DeepAR	1693.62	488.69	0.0067	0.0498
	NonAdapt	1694.39	489.36	0.0029	0.0444
	Adaptive	**1689.50**	**485.53**	0.0042	0.0461
Alibaba	Mean	1710.75	501.44	1.5361	2.3748
	MA	1714.00	506.74	1.9847	3.0567
	ARIMA	1711.70	502.49	1.6970	2.6194
	LSTM	1706.17	500.68	1.7192	2.6537
	DeepAR	1704.42	499.69	1.6851	2.6012
	NonAdapt	1710.86	498.81	1.2579	1.9521
	Adaptive	**1688.57**	**486.43**	**0.7372**	**1.1604**

ASLAV on the Alibaba trace, compared to the baseline arrival prediction methods. On the synthetic trace, the ASLAV of every method is negligible, below 0.01 s. Specifically, on the synthetic trace, the adaptive framework yields reductions of up to 2.3% in ART, while its ASLAV stays at the same negligible level as the baselines. On the Alibaba trace, the ART improvement reaches up to 4.0%, and the adaptive framework attains the lowest ASLAV at 0.74 s, against 1.54 to 1.98 s for the baselines. Meanwhile, the proposed approach reduces energy consumption by up to 0.9% on the synthetic trace and 1.5% on the Alibaba trace. Moreover, the adaptive framework yields the lowest operational cost on the Alibaba trace, with a cost of $1.16 compared with $2.37 to $3.06 for the baselines. On the synthetic trace, all costs are small and comparable, ranging from $0.043 to $0.050, because ASLAV remains close to zero, and the resulting cost differences are marginal.

In addition, the proposed prediction model with an adaptive mechanism improves CARSS performance, reducing ART by 0.8% on the synthetic trace, and reducing ART by 2.5% while lowering ASLAV from 1.26 s to 0.74 s on the Alibaba trace. These results demonstrate that accurate future QoS predictions enable the scheduler to make better scheduling decisions, leading to faster task processing while reducing energy consumption by 0.3% on the synthetic trace and 1.3% on the Alibaba trace. Additionally, the mean operational cost decreases from $1.95 to $1.16 on the Alibaba trace, whereas on the synthetic trace the cost variation is negligible.

The results also show that the arrival forecasting method can help CARSS achieve better QoS metrics than the Mean predictor. This is because the forecasting model can generate more accurate arrival series, which the co-simulator then uses to produce reliable QoS predictions, particularly on complex real-world datasets. In addition, CARSS with the fully adaptive arrival forecasting framework further improves QoS metrics compared to both the Mean and non-adaptive variants on the Alibaba trace. This indicates that the adaptive mechanism is essential to ensure the arrival forecasting model can adapt to changes in the arrival process, thereby helping CARSS produce scheduling decisions that yield better QoS.

4) Sensitivity Analysis: We conduct four sensitivity analyses that examine the factor ρ in the *ProbSparse* attention mechanism, the history length used for arrival prediction, the hyperparameter settings in the CARSS objective function in Eq. 14, and the Gumbel-softmax temperature τ.



(a) Factor ρ (b) History sequence length

Fig. 6: (a) shows the MAE of ART and inference time for different values of ρ. *Stand* represents the standard attention function. (b) presents the MAE of ART and inference time for varying history sequence lengths.

Factor ρ. The factor ρ in the $ProbSparse$ attention determines the number of queries used in attention calculations and directly affects both the fine-tuning performance and inference speed of the prediction model. To assess its effectiveness, we also compare $ProbSparse$ with the standard full attention mechanism.

Figure 6a indicates that as the factor increases, the MAE of ART decreases due to more queries influencing the weighted sum via attention instead of simply averaging the series. A similar trend is observed for the MAE of ASLAV. However, increasing the factor also increases the computational cost during fine-tuning and inference, as more queries are involved in the attention computation. Despite negligible improvements in MAEs with standard attention beyond a factor of 5 and slower speeds, the proposed framework using $ProbSparse$ attention with this factor achieves efficient arrival estimation and fast inference.

History length. To assess the impact of history sequence length, we train the prediction model with varying input sequence lengths and evaluate its effect on forecasting accuracy. Additionally, we compare the inference time of the model across different history lengths to understand the trade-off between sequence length and computational efficiency.

Figure 6b illustrates the impact of varying the history window size on both the MAE of ART and inference time. As the window size increases from 15 to 100, the MAE of ART steadily decreases, indicating that incorporating a longer history can improve the ability of the forecasting model to capture temporal patterns. A similar trend is observed for ASLAV, with a reduction in MAE as the window size increases. As the window size grows, the model requires more computation to process the longer sequences, resulting in slower inference. Notably, when the length of the history sequence is 100, the marginal gains in MAE diminish while the inference time continues to increase.

Parameter settings in the objective function. We conduct a sensitivity analysis on the parameters α, β, and ω in the objective function Eq. 14 to examine their impact on the performance of CARSS. We evaluate extreme weight configurations for each QoS objective, namely EC, ART,

TABLE V: QoS metrics for CARSS changing α, β, and ω.

Dataset	α	β	ω	EC	ART	ASLAV	Cost
Synthetic	0.8	0.1	0.1	1684.88	520.83	2.2062	3.2477
	0.1	0.8	0.1	1732.45	457.35	0.0025	0.0446
	0.1	0.1	0.8	1697.05	481.95	0.0029	0.0444
	0.2	0.4	0.4	1689.50	485.53	0.0042	0.0461
	0.4	0.3	0.3	1685.46	489.75	0.0062	0.0535
Alibaba	0.8	0.1	0.1	1677.21	492.21	1.4119	2.1855
	0.1	0.8	0.1	1696.78	479.42	0.7354	1.1582
	0.1	0.1	0.8	1692.68	482.84	0.7082	1.1165
	0.2	0.4	0.4	1688.57	486.43	0.7372	1.1604
	0.4	0.3	0.3	1680.25	487.57	0.9123	1.4265

TABLE VI: QoS metrics of CARSS on various Gumbel-softmax temperature values τ

Dataset	τ	EC (kJ)	ART (s)	ASLAV (s)	Cost ($)
Synthetic	0.1	1707.91	507.11	0.0025	0.0461
	0.3	1690.28	488.13	0.0008	0.0412
	0.5	1689.50	485.53	0.0042	0.0461
	0.7	1690.44	486.27	0.0004	0.0413
	0.9	1690.83	486.57	0.0002	0.0406
	1.5	1691.23	487.18	0.0005	0.0411
Alibaba	0.1	1705.23	501.90	1.3584	2.1046
	0.3	1689.68	486.81	1.4109	2.1841
	0.5	1688.57	486.43	0.7372	1.1604
	0.7	1689.44	488.00	1.6645	2.5699
	0.9	1689.93	488.90	1.6995	2.6234
	1.5	1690.57	489.07	1.7921	2.7641

and ASLAV, and compare these against two moderate-weight settings, including our default, on both the synthetic and Alibaba datasets.

Table V shows that a high weight on α gives the lowest EC, a high weight on β gives the lowest ART, and a high weight on ω gives the lowest ASLAV on the Alibaba trace, where SLA violations are frequent enough to separate the settings. Concentrating almost all weight on α gives the worst overall performance, since the near-zero weight on the SLA term yields the largest ASLAV and the highest cost, reaching \$3.25 on the synthetic trace and \$2.19 on Alibaba. The SLA-dominant setting $\alpha = \beta = 0.1$, $\omega = 0.8$ achieves the lowest cost on both traces but at a higher EC, whereas our default $\alpha = 0.2$, $\beta = \omega = 0.4$ stays within about 4% of this cost while attaining a lower EC. We therefore consider our default parameter selection a robust setting, as it keeps EC, ART, and ASLAV low together and gives consistent performance across both workloads.

Gumbel-softmax temperature τ. The temperature τ controls the sharpness of the Gumbel-softmax output and thereby governs the exploration-exploitation trade-off when CARSS searches for scheduling decisions. To evaluate the effect of the Gumbel-softmax temperature τ in CARSS, we vary $\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.5\}$ and report the corresponding performance metrics.

As shown in Table VI, the performance metrics remain relatively stable when τ ranges from 0.3 to 1.5. On the Alibaba trace, $\tau = 0.5$ attains the best value on every metric. On the synthetic trace, $\tau = 0.5$ attains the lowest EC and ART, while the ASLAV of every setting remains close to zero, below 0.01s, so the small differences in synthetic ASLAV and cost do not clearly distinguish $\tau = 0.5$ from the other settings. The performance degrades when $\tau = 0.1$ and $\tau = 1.5$ on the Alibaba trace. This is because a higher τ produces smoother outputs, which allows broader exploration of the host–task allocation space. However, if τ is too large (e.g., $\tau > 1$), the model may over-explore and fail to exploit good solutions. In contrast, a smaller τ, such as $\tau < 0.3$, produces sharper outputs that favor the exploitation of known allocations but limit exploration. This can trap the model in a local minimum, leading it to miss better solutions. Therefore, we choose $\tau = 0.5$ to better balance the trade-off between exploration and exploitation when searching for scheduling decisions.

VIII. Conclusion

In this paper, we present an online adaptive arrivals prediction framework that integrates hierarchical change point detection to monitor and detect concept drift and a probabilistic transformer model to forecast arrivals. The adaptive arrival forecasting framework can leverage a co-simulator to adapt to concept drift in arrival traces and improve QoS estimation, which CARSS further utilizes to make scheduling decisions that improve QoS. Experiments on synthetic and real-world traces show that the proposed framework effectively adapts to drift and provides useful arrival forecasts to the co-simulator, thereby improving QoS estimation. Moreover, CARSS, integrated with the adaptive framework, achieves improved QoS metrics, including lower energy consumption and faster task processing, than baseline schedulers.

Future work may also extend the proposed framework to workflow scenarios, where tasks are organized in a directed acyclic graph (DAG), and the execution of each task depends on the completion of its parent tasks.

References

[1] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing," *JISA*, Sept. 2014.

[2] Q. Wen, W. Chen, L. Sun, Z. Zhang, L. Wang, R. Jin, T. Tan, *et al.*, "Onenet: Enhancing time series forecasting models under concept drift by online ensembling," *Advances in Neural Information Processing Systems*, vol. 36, pp. 69949–69980, 2023.

[3] Z. Liu, R. Godahewa, K. Bandara, and C. Bergmeir, "Handling concept drift in global time series forecasting," in *Forecasting with artificial intelligence: theory and applications*, pp. 163–189, Springer, 2023.

[4] 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 TPDS*, Jan. 2022.

[5] S. Tuli, G. Casale, and N. R. Jennings, "GOSH: Task Scheduling Using Deep Surrogate Models in Fog Computing Environments," *IEEE TPDS*, Nov. 2022.

[6] C. Alippi, G. Boracchi, and M. Roveri, "Hierarchical Change-Detection Tests," *IEEE TNNLS*, Feb. 2017.

[7] A. Inc., "Alibaba production cluster data v2018," 2018.

[8] Y. Chen, M. Roveri, S. Tuli, and G. Casale, "Coupling QoS co-simulation with online adaptive arrival forecasting," in *Proc. of CNSM*, pp. 1–9, IEEE, 2023.

[9] J. Ullman, "NP-complete scheduling problems," *JCSS*, June 1975.

[10] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers," *Concurrency and Computation*, 2012.

[11] K. Han, Z. Xie, and X. Lv, "Fog Computing Task Scheduling Strategy Based on Improved Genetic Algorithm," *Computer Science*, Apr. 2018.

[12] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 29–43, 2019.

[13] K. Peng, B. Zhao, M. Bilal, and X. Xu, "Reliability-aware computation offloading for delay-sensitive applications in mec-enabled aerial computing," *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 3, pp. 1511–1519, 2022.

[14] B. Zhao, K. Peng, K. Zhang, H. Sun, Z. Tu, and D. Chu, "Serflow: A multi-stage service-enhanced mechanism for workflow applications in cpss with end-edge-cloud collaboration," *IEEE Internet of Things Journal*, 2025.

[15] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks," *IEEE TMC*, Mar. 2022.

[16] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," *IEEE TSC*, Sept. 2019.

[17] F. Zhu, J. Chen, J. Wen, Y. Yang, C. Yi, Y. Tie, P. Zhang, J. Cai, D. Niyato, and M. Guizani, "From data mirror to smart copilot: A survey on nextg semantic communication for propelling digital twin world into cognitive stage," *IEEE Communications Surveys & Tutorials*, 2026.

[18] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, "Differentially private federated multi-task learning framework for enhancing human-to-virtual connectivity in human digital twin," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3533–3547, 2023.

[19] J. Chen, C. Yi, S. Gong, H. Du, W. Wu, J. Kang, and D. Niyato, "Generative ai-aided qoe-aware resource allocations for rls-assisted digital twin interaction with uncertain evolution," *IEEE Transactions on Mobile Computing*, 2025.

[20] X. Ai, W. Liang, and C. Liu, "Joint optimization of model retraining and inference services in dt-assisted edge computing," *IEEE Transactions on Networking*, vol. 34, pp. 1804–1819, 2025.

[21] S. Kuvattana, S. Sukparungsee, P. Busababodhin, and Y. Areepong, "Performance Comparison of Bivariate Copulas on the CUSUM and EWMA Control Charts," p. 5, 2015.

[22] D. M. Hawkins, P. Qiu, and C. W. Kang, "The Changepoint Model for Statistical Process Control," *JQT*, Oct. 2003.

[23] Z. Harchaoui, E. Moulines, and F. Bach, "Kernel Change-point Analysis," in *Advances in NeurIPS*, vol. 21, 2008.

[24] R. P. Adams and D. J. C. MacKay, "Bayesian Online Changepoint Detection," Oct. 2007.

[25] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] K. Cho, B. Van Merriënboer, Ç. Gulçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

[28] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *IJF*, July 2020.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in NeurIPS*, 2017.

[30] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," Dec. 2020.

[31] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating Long Sequences with Sparse Transformers," Apr. 2019.

[32] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context," in *Proc. of the 57th AM of the ACL*, July 2019.

[33] A. I. Goldman, "Issues in designing sequential stopping rules for monitoring side effects in clinical trials," *Controlled Clinical Trials*.

[34] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proc. of AAAI*, May 2021.

[35] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov, "Transformer Dissection," in *Proc. of the EMNLP-IJCNLP*, Jan. 2019.

[36] R. B. Crosier, "Multivariate Generalizations of Cumulative Sum Quality-Control Schemes," *Technometrics*, Aug. 1988.

[37] J. Li, X. Zhang, and D. R. Jeske, "Nonparametric multivariate CUSUM control charts for location and scale changes," *Journal of Nonparametric Statistics*, Mar. 2013.

[38] D. M. Hawkins and D. H. Olwell, *Cumulative sum charts and charting for quality improvement*. Springer Science & Business Media, 2012.

[39] G. J. Ross, D. K. Tasoulis, and N. M. Adams, "Nonparametric Monitoring of Data Streams for Changes in Location and Scale," *Technometrics*, Nov. 2011.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of the CVPR*, IEEE, June 2016.

[41] H. Nashaat, N. Ashry, and R. Rizk, "Smart elastic scheduling algorithm for virtual machine migration in cloud computing," *The Journal of Supercomputing*, vol. 75, pp. 3842–3865, 2019.

[42] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. of ICLR*, OpenReview.net, 2017.

[43] S. Shen, V. Van Beek, and A. Iosup, "Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters," in *Proc. of the CCGC*, May 2015.

Yichong Chen received his M.S. in Computing from the Department of Computing, Imperial College London, London, UK, in 2020. He is currently pursuing his Ph.D. at the Quality of Service Research Lab at Imperial College London, London, UK. His research interests include distributed computing, deep learning, and Internet of Things.

Manuel Roveri (Senior Member, IEEE) is a Full Professor with DEIB, Politecnico di Milano. He published more than 120 articles in international journals and conference proceedings. His research interests include Tiny Machine Learning and privacy-preserving machine learning. Prof. Roveri is the recipient of multiple recognitions including best paper awards at IEEE TNNLS and IEEE CIM. He served as the chair and a member in several IEEE committees and subcommittees and as an Associate Editor for IEEE TNNLS, IEEE TAI, IEEE TETCI, IEEE CIM and Neural Networks.

Shreshth Tuli is an incoming Director of AI (Model Orchestration) at Lenovo. His primary research areas include: Language models, orchestration, and distributed computing. He has a PhD in AI and Fog Computing from Imperial College London. Prior to that, he was a student at the Indian Institute of Technology Delhi. He is also a member of the Mensa community.

Giuliano Casale joined the Department of Computing at Imperial College London in 2010, where he is currently a Reader. He does research in performance engineering and cloud computing, topics on which he has published more than 150 refereed papers. He has served on the technical program committee of several conferences in the area of performance and dependability. His research work has received multiple recognitions, including best paper awards at ACM SIGMETRICS, IEEE/IFIP DSN, and IEEE INFOCOM. During 2019-2023 he was the chair of ACM SIGMETRICS. He serves on the editorial boards of ACM TOMPECS and ACM TAAS and as Editor-in-Chief of Elsevier Performance Evaluation.