Taming Serverless Cold Start of Cloud Model Inference with Edge Computing

Kongyange Zhao, Zhi Zhou, *Member, IEEE,* Lei Jiao, *Member, IEEE,* Shen Cai, Fei Xu, *Member, IEEE,* Xu Chen, *Senior Member, IEEE*

***Abstract*—Serverless computing is envisioned as the de-facto standard for next-generation cloud computing. However, the cold start dilemma has impeded its adoption by delay-sensitive and burst applications. In this paper, we propose to tame serverless cold start in a cloud inference system with edge computing. Specifically, the proposed solution smooths the serverless cloud workload with user-owned edge computing, reducing the number of cold starts. Leveraging the configurability of requests and serverless functions, the proposed solution further reduces the transmission latency and serverless cost by adapting request configuration (e.g., image resolution) and function configuration (e.g., memory). To alleviate the potential inference accuracy degradation incurred by configuration adaption, we aim to strike a nice balance between inference latency, cost, and accuracy. However, achieving this goal is non-trivial since the underlying optimization is non-convex and involves future uncertain information. To simultaneously address dual challenges, the presented cold-start-aware online algorithms apply the regularization technique to decompose the problem into separate convex subproblems. Then, it applies lazy switching to smooth the number of provisioned functions and thus reduces the cold start. Through rigorous theoretical analysis, realistic prototype evaluations on AWS Lambda, and trace-driven simulations, we comprehensively validate the theoretical and empirical performance of our proposed solution.**

***Index Terms*—serverless computing, cold start, model inference, edge computing, online optimization**

I. Introduction

SERVERLESS computing, also known as function computing or Function-as-a-Service (FaaS), is emerging as the de-facto standard for next-generation cloud computing [1]. Serverless computing enables developers to build applications faster by writing and deploying individual program codes, without the hassle of managing dedicated virtual machines or servers. With serverless computing, an application is decomposed into a suite of small pieces of concise functions that are loosely coupled, allowing the developers to develop, manage and scale the applications in an agile and flexible manner.

Due to its unique merits in application development and cost efficiency, serverless has witnessed successful adoption in a wide variety of applications such as video processing [2], machining learning [3], and Internet-of-Things (IoT) [4]. For emerging IoT applications that typically invoke computational-intensive machine learning inference tasks, completely relying on the resource-limited edge node would deteriorate the application performance. This dilemma poses an urgent need to seamlessly integrate the edge and the cloud. Nowadays, many valuable applications based on cloud-edge collaboration such as video analytics [5] and industrial IoT [6]. To meet this need, representative IoT cloud platforms — as exemplified by AWS IoT Greengrass, Azure IoT Edge, and Google Cloud IoT Edge — all adopt serverless computing to develop and deploy IoT applications across the edge and cloud.

Kongyange Zhao, Zhi Zhou, Shen Cai, Xu Chen are with the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), Guangzhou 510006, China. E-mail: {zhaokyg, caish27}@mail2.sysu.edu.cn, {zhouzhi9, chenxu35}@mail.sysu.edu.cn
Lei Jiao is with the Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, USA. E-mail: jiao@cs.uoregon.edu
Fei Xu is with the Shanghai Key Laboratory of Multidimensional Information Processing, School of Computer Science and Technology, East China Normal University, Shanghai 200062, China. E-mail: fxu@cs.ecnu.edu.cn
Corresponding author: Zhi Zhou.

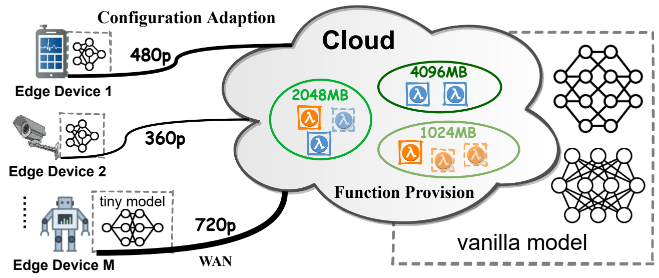


Fig. 1. An illustration of cloud-edge collaborative inference system based on serverless computing, where the edge device deploys a tiny model with fewer parameters for local inference, and the cloud deploys a vanilla model with huge parameters. Inference requests can be offloaded with different resolution configurations to the function with different memory size in a serverless cloud.

While recognizing the benefits of serverless computing, its downsides must not be overlooked. Among these, the most fundamental one is the cold start problem that may severely deteriorate the application performance [7], [8]. In particular, function cold start can be defined as the set-up time required to provision the function's runtime environment when it is invoked for the first time within a predefined survival period (e.g., 5-7 minutes for AWS Lambda [9]). As revealed by the empirical measurements, the cold start latency on mainstream commercial serverless platforms including AWS Lambda, Google Cloud Functions, and Azure Functions typically varies from a few hundred milliseconds to a few seconds [10]. In sharp contrast, for many applications such as web serving, machine learning inference, and IoT data processing, the function execution latency is typically less than one second [11], [12]. The cold start of the functions makes the total runtime significantly more than the actual time required for function executions [13], which should be carefully addressed to achieve predictable performance.

In taming the serverless cold start, existing wisdom can be classified into two categories. The first one is to reduce the cold start time by means such as lightweight function virtualization [14]. The second one is to reduce the number of cold starts via function instance pre-warming [12] or re-using [15]. While these efforts effectively mitigate the cold start, the concerns such as increased security risk and uncontrollable factors incurred by serverless operator-managed solutions may prevent application providers from realistic adoption. Along a different route, for emerging serverless-based cloud-edge collaborative inference as shown in Fig. 1, we advocate a solution managed by application providers to harness the serverless cold start via edge computing. Inference requests generated by edge devices can be processed by the local tiny model, or offloaded to the remote cloud by a vanilla model with larger parameters. The basic idea is to curb the burstiness by adaptively shaping the inference workload offloaded to the cloud with edge computing, governing the number of newly triggered functions by requests to reduce the number of cold starts. Since the edge computing capability is assumed to be owned by users, the above solution does not require the intervention of the serverless operator. Beyond alleviating the cold start latency, our solution further exploits the configurability of inference requests and serverless functions to reduce the network transmission latency and function execution latency. Specifically, for requests such as image recognition, their resolution (360p, 720p, etc.) can be adapted to reduce the data size. While for the function, the allocated resource (i.e., memory size such as 2048MB, 4096MB) can also be adapted to balance the execution latency and cost.

By applying edge computing, request and function configuration adaption, our proposed solution simultaneously reduces data transmission latency, function cold start latency and inference latency, promising predictable performance. However, this improvement comes at the cost of deteriorated inference accuracy, due to the limited edge resource capacity and potentially degraded inference configuration. To address this issue, when jointly optimizing those control knobs, we aim to strike a nice balance among the service latency, resource cost and inference accuracy. However, achieving this goal is highly non-trivial, due to the following two reasons. First, the function cold start latency temporally couples the function provision decisions over time, making the long-term optimization problem involving future uncertain information such as request arrivals that typically fluctuate over time. Thus, it is highly desirable to minimize the long-term latency in an online manner, without future information as a priori. Second, even with an offline setting where all the future information is given, the cold start latency couples the function provision decisions in an intractable non-convex manner. This rules out the direct applications of existing online optimization techniques such as regularization-based optimization and optimal policy based on Markov decision processing.

To address the above dual challenges, we first design a fast algorithm based on the one-step regularization method, which substitutes the intractable non-convex time-coupling term with a convex relative entropy function. By decoupling the long-term optimization problem into a series of single-slot convex programs, our fast algorithm solves the regularization problem in polynomial time based on the previous time slot's historical solution. We further extend the one-step regularization method by constructing a multi-step regularization problem to utilize historical information during the survival window of functions. Through the independence of the cold-start time and functions with different memory, we combine the idea of lazy-switching to design an extended algorithm, which judiciously decides whether to accept the cold-start solution obtained from the multi-step regularization problem. Our extended algorithm fully leverages the warm functions to effectively reduces the occurrence of cold starts in the survival period.

Our main contributions are highlighted as follows.

- We advocate harnessing the serverless cold start via edge computing in the cloud-edge collaborative inference paradigm. We formulate the joint optimization problem of total latency and monetary cost based on a practical function cold-start model, which precisely captures the cold-start latency in a fine-grained manner.
- We propose a fast algorithm as well as an extended algorithm to address the challenges of future uncertain information and non-convex optimization. We rigorously prove that our fast algorithm leads to a parameterized-constant competitive ratio against the offline optimum which assumes all the inputs are given as a prior.
- By implementing a system prototype on AWS Lambda, we evaluate the performance of the proposed algorithms on real-world serverless cloud testbed. Evaluation results show that our fast and extended algorithms significantly improve the utilization of function containers, reducing function cold starts by 27.6% and 51.1% respectively compared with the baseline.
- We further conduct extensive simulation experiments based on real-world traces to verify the effectiveness of our proposed algorithms. Our fast algorithm and extended algorithm reduce the long-term total latency and cost by up to 32.6% and 39.0% compared with the benchmarks, respectively.

The rest parts of the paper are organized as follows. Section II reviews the literature on serverless edge computing, summarizes the related work on mitigating serverless cold-start, and highlights the motivation of our research. Section III introduces the system model and problem formulation for cloud-edge collaborative inference system based on serverless computing. Section IV proposes a fast online algorithm and an extended algorithm for the window switching problem, and we analyze the performance of the proposed algorithm through a parameterized competitive ratio in Section V. Section VI evaluates our proposed algorithms on the real-world testbed based on a system prototype implemented on AWS Lambda. Section VII conducts extensive trace-driven simulations to empirically assess the performance of the proposed online algorithms. Finally, Section VIII discusses the future work and Section IX concludes this paper.

II. Related Work and Motivation

Serverless Edge Computing: With the emerging of edge computing, serverless computing is envisioned as a promising approach to seamlessly integrate the locality benefit of edge computing and the capability advantage of cloud computing [1]. As a pilot effort, a video analytic system named LAVEA [16] collaborates with nearby client, edge and remote cloud nodes via serverless computing, with the goal of achieving low latency. For serverless computing across the edge and cloud, Elgamal *et al.* optimize both performance and cost via joint function fusion and placement [17]. For the emerging paradigm of edge intelligence (i.e., hosting AI applications with edge computing), Rausch *et al.* present a scheduling framework [18] to place serverless AI functions across the edge and cloud, by jointly considering device capabilities, network latency and data locality. Xu *et al.* study the stateful serverless application placement problem with the dependency of functions in edge computing platform, they develop an effective heuristic algorithm and an online learning-driven algorithm with a bounded regret [19]. While these efforts target performance optimization, they do not explicitly address the cold-start issue of serverless computing.

Mitigating Serverless Cold-start: To address the cold-start issue of serverless computing, Chiang *et al.* model the problem of container warming control as a Markov decision process (MDP) [20]. Leveraging the partial submodularity, they derive a hysteretic optimal control policy to reduce the cold-start latency. Vahidinia *et al.* proposed a two-layer adaptive appraoch that utilizes reinforcement learning and Long Short-Term Memory (LSTM) to determine the best time to keep containers warm and the required pre-warmed containers, respectively [21]. To mitigate the cold-start issue in a distributed serverless edge computing environment, Tang *et al.* present a multi-agent deep reinforcement learning method to jointly optimize the task scheduling and computing resource allocation [22]. Also targeting the cold-start of serverless edge computing, Pan *et al.* propose a container caching and cross-edge request dispatching approach [23]. To address the time-coupling challenge of the optimization problem, they present an efficient online algorithm by mapping the problem to the classical ski-rental problem. Among the above researches, methods based on reinforcement learning or LSTM prediction [21], [22] rely on learning from a large amount of available historical data and lack theoretical guarantees. The works [20], [23] model the cold-start latency as a simple convex function across two consecutive time slots, which is a compromise of provable performance in an imprecise model.

Motivation: Given the significance of related works above, our study differs in the following aspects. First, we advocate a solution managed by application providers to tame serverless function cold start for cloud model inference, which facilitates the reduction of security risks and uncontrollable factors incurred by serverless operator-managed solutions. Secondly, according to the cold-start characteristics of existing FaaS products (e.g., AWS Lambda and Google Cloud Functions) in practice [12], [21], we precisely model the cold start latency of serverless functions to a multi-time-slot switching problem based on the survival window. Finally, different from directly applying tools from AI/ML in a black-box manner, we aim to provide online algorithms with formal guarantees to reduce the occurrence of serverless function cold-start. It is non-trivial due to the non-convexity for calculating cold start latency.

TABLE I
MAIN NOTATIONS

Notation	Description
$\mathcal{M}, \mathcal{N}, \mathcal{K}, \mathcal{T}$	set of devices, function configurations, request configurations, and time slots
i, f, k, t	index of devices, function configurations, request configurations, and time slots
$A_i(t), B_i(t)$	workload and bandwidth of edge device
a_{ik}, b_{ik}	inference accuracy and transmission data amount
C_i	local processing capacity of edge device
E_f	concurrency capacity of serverless function
T_i^e	inference time per request for edge devices
T_f^s	inference time per request for serverless functions
T^c	cold start time of serverless function
I	survival period of serverless function
Q_i	average accuracy requirement of edge device
P_f	running price per unit time for functions

III. System Model and Problem Formulation

In this section, we first present the system model of a cloud-edge collaborative inference system, consisting of edge devices and a public cloud with serverless computing capability. By defining two types of control decisions, we further characterize the latency, accuracy, and cost models in the collaborative inference system. We formulate the long-term optimization problem that is defined as the *Window Switching* problem, and finally summarize the challenges compared with the traditional *Switching* problem. The main notations used in this paper are summarized in Table I.

A. Overview of Cloud-edge Collaborative Inference System

As shown in Fig. 1, we consider an application provider running an IoT service (e.g., video analytics for a smart factory) in the cloud-edge collaborative inference system based on serverless computing. This IoT service continuously senses data (e.g., pictures or video frames) from a set of edge devices [24], which generate inference requests to be processed by deep neural network (DNN) models (e.g., VGG and ResNet [25] for object recognition) deployed locally or in the remote public cloud. For the edge devices, we assume that they are owned by the application provider, and they can be readily managed by a unified central controller such as AWS Management Console [26]. Due to the resource scarcity of the devices, compressed tiny DNN models (e.g., ResNet-18 in the ResNet family [25]) are deployed locally to reduce the resource footprint. While for the remote public cloud, it can be accessed by the devices via the wide-area-network (WAN) whose bandwidth is scarce, volatile, and expensive. To maintain inference accuracy, vanilla DNN models (e.g., ResNet-152 in the ResNet family [25]) with high accuracy are deployed in the cloud. Moreover, we also assume that the cloud serves the inference requests in the mode of serverless computing. This coincides with the recent trend that leading IoT cloud platforms such as AWS IoT Greengrass.

In this paper, we use $\mathcal{M} = \{1, 2, \ldots, M\}$ to denote the set of edge devices managed by the application provider. For each device $i \in \mathcal{M}$, the computing capacity (i.e., the maximum number of inference requests that can be locally processed by the compressed tiny model) is denoted by C_i. To capture the system dynamics such as time-varying request arrivals and bandwidths between devices and the cloud, we assume that the system makes decisions in a time-slotted fashion within a large time span of T. Each time slot $t \in \mathcal{T} = \{1, 2, \dots, T\}$ represents a decision interval that matches the change of the system dynamics. At time slot t, we use $A_i(t)$ and $B_i(t)$ to denote the number of inference requests generated by device i and the bandwidth between each device i and the cloud, which typically fluctuate over time.

In a serverless cloud, a function represents a unit of resource instance to process a request in an event-trigger manner, and the only configuration option for users is the memory size allocated to each function (e.g., 128MB to 10,240MB in AWS Lambda [27]). Notably, the memory configuration directly navigates the trade-off between function execution latency and monetary cost [28]. In this paper, we use $\mathcal{N} = \{0, 1, \dots, N\}$ to denote the set of valid memory configurations for each function to run the vanilla model. When offloading the requests from devices to the serverless cloud, the configurability of inference requests can also be leveraged to balance the transmission latency and the cloud inference accuracy. Specifically, for computer vision tasks such as video analytics, the configuration (e.g., resolution and frame rate) can be degraded to reduce the transmission data size and thus latency [29], at the cost of reduced inference accuracy. We use $\mathcal{K} = \{1, 2, \dots, K\}$ to denote the set of configurations that can be chosen when offloading requests from devices to the cloud.

B. Decision Variables

To jointly optimize the performance, cost and accuracy of the serverless-based cloud-edge collaborative inference system, we concentrate on two types of control decisions in this paper. Note that the edge devices and serverless functions are managed by the application provider through a central console, which perceives dynamic information in the system (e.g., the request arrivals and bandwidth of each device). Therefore, the application provider (i.e., decision maker) specifically makes the following request dispatching decision and function provisioning decision: (i) $x_{ikf}(t)$, for each edge device i, the number of inference requests offloaded to the cloud with configuration k and served by function with configuration f at each time slot t, and (ii) $y_f(t)$, the total number of inference requests served by function with configuration f at each time slot t. The physical meaning of the above two variables is the number of inference requests, so the value is a positive integer.

Considering that the scale of commercial model inference serving systems has continued to grow exponentially [30], [31], a definite trend is that the infrastructure for AI applications is rapidly shifting from the cloud to the edge [32], [33]. Given the potentially huge amount of inference request arrival $A_i(t)$, it is reasonable to relax the integer variables $x_{ikf}(t)$ and $y_f(t)$ into continuous variables by applying the linear programming relaxation, and thus to reduce the problem complexity with negligible optimality loss. Note that the total number of invoked function with configuration f is no smaller than the actual amount of real requests offloaded to the cloud (i.e., $y_f(t) \geq \sum_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}} x_{ikf}(t), \forall t \in \mathcal{T}, f \in \mathcal{N}$), because the central console (controlled by the application provider) may generate "fake requests" for the serverless cloud to warm up functions [34] or extend the survival period for function re-using [23]. Similar to some function warm-up tools such as Dashbird.io [35] and Lambda Warmer [36], the "fake requests" generated by the central console can invoke functions and keep them warm and thus to avoid the cold start of functions when the request arrival surges in the future.

C. Latency, Accuracy, and Cost Model

Based on the above decision variables, we characterize the latency, inference accuracy, and cost models in the system. The overall latency includes inference latency on both the edge and serverless cloud, transmission latency of requests offloaded to the cloud and cold start latency of serverless functions. Note that our latency model does not consider the queuing latency of requests between adjacent slots because the length of the decision slot is much larger than the millisecond latency requirement of the inference request [11]. Therefore, we reasonably assume that all inference requests are processed in the current time slot, which is verified in our real-world evaluation of Section VI.

Edge Inference Latency. We use T_i^e to represent the inference time for device i to process a request with the compressed DNN model locally. Given a total amount $A_i(t) - \sum_{k \in \mathcal{K}} \sum_{f \in \mathcal{N}} x_{ikf}(t)$ of inference requests locally processed at each edge device i, the total edge inference latency can be computed by:

$$L_{EI}(t) = \sum_{i \in \mathcal{M}} \big[A_i(t) - \sum_{k \in \mathcal{K}} \sum_{f \in \mathcal{N}} x_{ikf}(t)\big] T_i^e.$$

Transmission Latency. When choosing configuration k to offload the inference request from device i to the cloud, we use b_{ik} to denote the amount of data to be transferred for a single request. Considering the time-varying bandwidth $B_i(t)$ between device i and the cloud, the total transmission latency in time slot t can be computed by:

$$L_{TR}(t) = \sum_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}} \sum_{f \in \mathcal{N}} x_{ikf}(t) \tfrac{b_{ik}}{B_i(t)}.$$

Serverless Inference Latency. We use T_f^s to represent the inference time of serving a request by function with memory configuration f in the serverless cloud. Note that in practice, the execution time of functions is typically agnostic to the request (input) configuration k. Then, the total serverless inference latency at time slot t can be computed by:

$$L_{SI}(t) = \sum_{f \in \mathcal{N}} \sum_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}} x_{ikf}(t) T_f^s.$$

Serverless Cold Start Latency. As a unique feature of serverless cloud computing, cold start refers to the set-up time required to get a serverless application's environment up when it is invoked for the first time within a defined period [37]. Specifically, after being invoked, the function containers usually expire after a few minutes of unuse [38]. This survival time is typically platform-dependent but function-agnostic [12], and it is assumed to be I time slots in this paper, where I is an integer. The number of cold start functions in each time slot is related to the number of function provided by serverless in the past I time slots, because the functions invoked in these time slots are still in their survival periods. When the same type of function is invoked again during the survival period, reusing the function instance can avoid the cold start of the function that launches the new container [12]. For each serverless function with configuration f, the number of cold starts in the current time slot is the number newly added based on the maximum number of calls in the past I time slots. We use T^c to represent the cold start time of each serverless function, therefore, the serverless cold start latency in time slot t can be computed by:

$$L_{CS}(t) = \sum_{f\in\mathcal{N}} T^c\big[y_f(t) - \max_{\tau=1}^{I} y_f(t-\tau)\big]^+,$$

where $[a-b]^+ = \max\{a-b, 0\}$. For ease of presentation, we define $L^{nCS}\big(\boldsymbol{x}(t)\big) = L_{EI}(t) + L_{TR}(t) + L_{SI}(t)$. Then, the total latency in time slot t can be expressed as : $\mathbb{L}\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) = L^{nCS}\big(\boldsymbol{x}(t)\big) + L_{CS}(t)$.

Inference Accuracy. As we mentioned in Sec. III-A, inference request with different configurations results in varying inference accuracy. Therefore, we use a_{i0} and a_{ik} to denote the inference accuracy of a request from device i processed locally by device i and by the serverless function with configuration k, respectively. To deliver predictable inference accuracy, we enforce that the average inference accuracy perceived by the requests generated by each device i is no lower than a predefined threshold Q_i:

$$\frac{\Big(A_i(t) - \sum_k \sum_f x_{ikf}(t)\Big)a_{i0} + \sum_k \sum_f x_{ikf}(t)a_{ik}}{A_i(t)} \geq Q_i.$$

By defining constants $d_{ik} = a_{ik} - a_{i0}$ and $q_i = Q_i - a_{i0}$, we can simplify the above expression with a concise form of $\sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{N}} x_{ikf}(t)d_{ik} \geq A_i(t)q_i$.

Serverless Cost. For application providers, the monetary cost is incurred by the usage of the serverless cloud resource, which depends on the number of functions invoked [39], [40], the configuration and the execution time (i.e., inference time T_f^s) of the invoked functions. Note that the number of invoked functions includes real inference requests sent from devices as well as fake requests generated by the central console. Because once a function is triggered, regardless of whether it executes the real inference request from the device, the corresponding resource occupancy is generated and billed by the function. A large number of fake requests can improve the chance of reducing function cold start, and at the same time occupy more cloud resources, which reflects the trade-off between cold start time and resource consumption [41]. By denoting P_f as the price of invoking a function with configuration f per unit time period, the total serverlss cost at each time slot t can be formulated as:

$$Cost\big(\boldsymbol{y}(t)\big) = \sum_{f\in\mathcal{N}} y_f(t)P_f T_f^s.$$

D. Problem Formulation

Based on the latency, accuracy, and cost models above, we minimize the weighted sum of (i) the long-term total inference latency and (ii) the long-term monetary cost incurred by the serverless cloud usage over time, and meanwhile maintain the inference accuracy as follows:

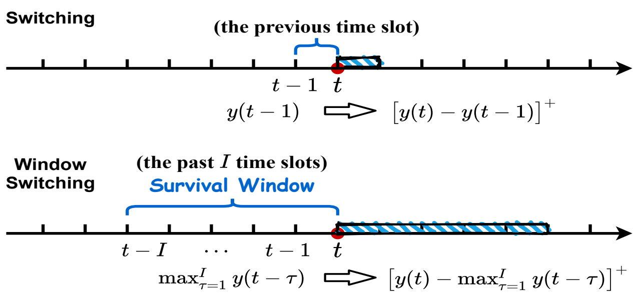


Fig. 2. Comparison of the window switching problem with traditional switching problem. The cold-start model in problem P^{WS} precisely captures the cold-start latency in a fine-grained manner, which is different from switching between adjacent time slots.

$$\min \quad P^{WS} = \sum_{t=1}^{T} \Big\{ \mathbb{L}\big(\boldsymbol{x}(t), \boldsymbol{y}(t)\big) + \omega \cdot Cost\big(\boldsymbol{y}(t)\big) \Big\}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{N}} x_{ikf}(t) \leq A_i(t), \qquad \forall t, \forall i, \tag{1a}$$

$$A_i(t) - \sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{N}} x_{ikf}(t) \leq C_i, \quad \forall t, \forall i, \tag{1b}$$

$$\sum_{k\in\mathcal{K}} \sum_{f\in\mathcal{N}} x_{ikf}(t) d_{ik} \geq A_i(t) q_i, \quad \forall t, \forall i, \tag{1c}$$

$$\sum_{i\in\mathcal{M}} \sum_{k\in\mathcal{K}} x_{ikf}(t) \leq y_f(t), \qquad \forall t, \forall f, \tag{1d}$$

$$x_{ikf}(t) \geq 0, y_f(t) \in \big[0, E_f\big], \qquad \forall t, \forall i, \forall k, \forall f. \tag{1e}$$

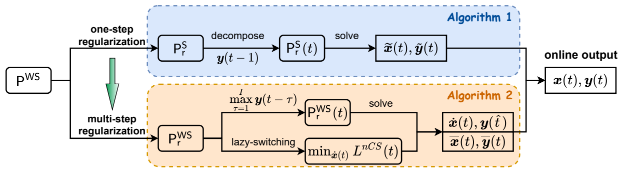
The parameter $\omega > 0$ represents a tunable weight coefficient of the serverless cost according to the measurements which suggests that there is some translational relationship between service latency and cost [42]. Constraint (1a) ensures that the number of requests processed by serverless function does not exceed the total number of requests received by device i in each time slot. Constraint (1b) maintains the resource capacity of each edge device. Constraint (1c) maintains the inference accuracy target Q_i for each edge device i. Constraint (1d) indicates that there may be fake requests to keep the function warm as mentioned in Section III-B. Constraint (1e) is the non-negative constraint for the decision variables, here E_f denotes the maximum number of functions can be invoked concurrently (e.g., the default concurrency limit of AWS Lambda is 1,000).

Based on the survival window of serverless functions across multiple time slots, we define the above optimization problem as a *Window Switching* problem by P^{WS}. Solving the problem P^{WS} is however non-trivial due to the following challenges. Firstly, as we can observe from that the cold start latency temporally couples the control decisions over time, making the long-term optimization problem time-coupling and involves future system information. However, in practice, parameters such as request arrivals and bandwidth typically fluctuate over time and thus cannot be readily predicted. Then, it is challenge to minimize the long-term latency in an online manner, without requiring the future information as a priori. Secondly, the number of cold start functions is related to the historical solutions in the past I time slots (i.e., the maximum number $\max_{\tau=1}^{I} y_f(t-\tau)$ as we mentioned in Section III-C), which is highly different from the traditional *Switching* problem in previous research [23] as shown in Fig. 2. While existing literature such as [22], [23] address serverless cold-start issue by online approaches, they model the cold-start latency based on only two consecutive time slots, rather than the multiple I time slots considered in our problem P^{WS}. Note that our cold-start latency model within I time slots is more practical, as it precisely captures the cold-start latency in a fine-grained manner. However, unlike the 2-time-slot cold-start model which is convex, our cold-start model within I time slot is non-convex even in the offline case.

IV. Cold-Start-Aware Online Algorithms

Due to the non-convex function in the cold-start model, existing methods for convex cold-start models such as [20], [23] cannot solve the proposed *Window Switching* problem. On the one hand, the coupling relationship of decisions in the past I time slots break the theoretical analysis of the algorithm that can effectively handles the decoupling of two consecutive slots. On the other hand, the survival characteristics of serverless function makes the online approaches for traditional *Switching* problem, which does not make full use of historical information, is not satisfactory on measurable performance. Therefore, it motivates us to design new online algorithms for the *Window Switching* problem.

As shown in Fig. 3, we propose a fast one-step algorithm in Section IV-A based on the algorithmic technique of regularization, which utilizes a smooth convex function to replace the intractable non-convex terms. Moreover, in order to fully utilize historical information in the survival window of activated functions, we further propose an extended multi-step algorithm in Section IV-B. Both algorithms can independently solve the *Window Switching* problem and obtain online output of problem P^{WS}. For clear expression, we list the relevant solution notations and inclusion of variables in Table II.



Algorithm 1
one-step regularization
$\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}$
decompose
$\boldsymbol{y}(t-1)$
$\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$
solve
$\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)$
P^{WS}
online output
$\boldsymbol{x}(t), \boldsymbol{y}(t)$
Algorithm 2
$\max_{\tau=1}^{I} \boldsymbol{y}(t-\tau)$
$\mathsf{P}^{\mathsf{WS}}_{\mathsf{r}}(t)$
solve
multi-step regularization
$\mathsf{P}^{\mathsf{WS}}_{\mathsf{r}}$
lazy-switching
$\min_{\dot{\boldsymbol{x}}(t)} L^{nCS}(t)$
$\dot{\boldsymbol{x}}(t), \boldsymbol{y}(\hat{t})$
$\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)$

Fig. 3. The flow diagram of our proposed online algorithms, where Algorithm 1 is based on a one-step regularization method, and Algorithm 2 is extended to multi-step regularization and further incorporates the idea of lazy-switching. Both of them output online solutions to the window switching problem.

A. Regularization-based Online Algorithm

To address the dual challenges of time-coupling and non-convexity incurred by the cold-start term, an intuitive idea is to approximate the non-convex term with a credible convex function. In order to simultaneously prevent the potential cold start of the function caused by the drastic shifts between time slot t and $t-1$, we exploit the algorithmic technique

TABLE II
Solution Notations with Variables

Notation	Description
$\boldsymbol{x}(t), \boldsymbol{y}(t)$	output of online algorithms for solving P^{WS}
$\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)$	optimal solution of one-step regularization problem $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$ in Algorithm 1 (line 4)
$\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)$	optimal solution of multi-step regularization problem $\mathsf{P}^{\mathsf{WS}}_{\mathsf{r}}(t)$ in Algorithm 2 (line 4)
$\dot{\boldsymbol{x}}(t), \boldsymbol{y}(\hat{t})$	optimal solution of $\min L^{nCS}(t)$ with $\boldsymbol{y}(\hat{t})$ in Algorithm 2 (line 6-7)

of regularization from the online learning literature [43]. The basic idea is to substitute the intractable time-coupling and non-convex term with a well-designed function. In this paper, we employ the widely adopted convex relative entropy function [44], [45] to substitute $[y_f(t) - \max_{\tau=1}^{I} y_f(t-\tau)]^+$ as follow:

$$\Delta\big(y_f(t)||y_f(t-1)\big) = y_f(t) \ln \frac{y_f(t)}{y_f(t-1)} + y_f(t-1) - y_f(t).$$

This smooth convex function is the sum of the relative entropy term $y_f(t) \ln \frac{y_f(t)}{y_f(t-1)}$ and a linear term denoting the movement cost $y_f(t-1) - y_f(t)$. To ensure that the fraction is still valid when no inference request processed by function f in time slot $t-1$ (i.e., $y_f(t-1) = 0$), we add a positive constant term ϵ to both $y_f(t)$ and $y_f(t-1)$ in the above convex function. Moreover, we define an approximation weight factor $\eta_f = \ln(1+\frac{E_f}{\epsilon})$ and multiply the improved relative entropy function with $\frac{1}{\eta_f}$ to normalize the cold start latency by regularization.

Let $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}$ represents the regularized switching problem by using the above enhanced regularizer $\Delta(y_f(t)||y_f(t-1))$ to approximate the time-coupling term in the cold start latency $L_{CS}(t)$. The problem $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}$ is still time-coupling, thus the optimal solution to the decoupled problem $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$ is not equivalent to the original problem. Nonetheless, the optimality conditions of the regularized problem yield a lower bound on the performance of the online algorithm solving a series of single-shot problems. So we temporally decompose $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}$ into a series of single-shot convex programs $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$, which can be solved in each individual time slot t based the solution obtained from the previous time slot $t-1$. Specifically, the decomposed subproblem $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$ for each time slot $t \in \mathcal{T}$ can be denoted as follow:

$$\begin{aligned} \min \;\; & P^S_r(t) = L^{nCS}\big(\boldsymbol{x}(t)\big) + \omega \cdot Cost\big(\boldsymbol{y}(t)\big) \\ & + \sum_{f \in \mathcal{N}} \frac{T^c}{\eta_f} \bigg(\big(y_f(t)+\epsilon\big) \ln \frac{y_f(t)+\epsilon}{y_f(t-1)+\epsilon} - y_f(t) \bigg), \\ \text{s.t.} \;\; & \text{Constraint (1a) to (1e).} \end{aligned}$$

Based on the one-step regularization method, we propose a *Once Forward Regularization* (OFR) algorithm as shown in Algorithm 1. At each time slot t, OFR first observes $\boldsymbol{A}(t), \boldsymbol{B}(t)$ and looks forward to get historical information $\widetilde{\boldsymbol{y}}(t-1)$, which has been obtained when solving $\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t-1)$ at time slot $t-1$. Then, OFR generates the optimal regularization solution $\big(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)\big)$ by solving the single-slot regularized switching problem $\mathsf{P}_\mathsf{r}^\mathsf{S}(t)$. Since the problem $\mathsf{P}_\mathsf{r}^\mathsf{S}(t)$ is a standard convex optimization with linear constraints, it can be optimally solved in polynomial time by taking existing convex optimization techniques such as interior-point method [46].

Algorithm 1: Once Forward Regularization — OFR

```
   Input: $\mathcal{M}, \mathcal{N}, \mathcal{K}, \boldsymbol{C}, \boldsymbol{E}, \boldsymbol{P}, \boldsymbol{T}^e, \boldsymbol{T}^s, T^c, \boldsymbol{b}, \boldsymbol{d}, \boldsymbol{q}, \boldsymbol{\eta}, \epsilon, \omega$.
1  Initialize $\widetilde{\boldsymbol{y}}(0) = \boldsymbol{0}$;
2  for time slot $t \in \mathcal{T}$ do
3      Observe $\boldsymbol{A}(t), \boldsymbol{B}(t), \widetilde{\boldsymbol{y}}(t-1)$;
4      Solve $\mathsf{P}_\mathsf{r}^\mathsf{S}(t)$ to obtain the solution $\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)$;
5      $\boldsymbol{x}(t), \boldsymbol{y}(t) = \widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)$;
   Output: $\boldsymbol{x}(t), \boldsymbol{y}(t)$.
```

In practice, the application provider can access the managed devices to observe information through a central console such as AWS Management Console [26]. Based on the recorded historical decisions (i.e., $\widetilde{\boldsymbol{y}}(t-1)$), the central console can quickly solve the convex problem via invoking commercial solvers such as Gurobi [47] to generate control decisions for each time slot t. Although the window switching problem is non-convex and lacks theoretical tools to analyze performance in mathematics, we still use a series of transformations to construct a convex problem and derive a competition ratio based on the primal-dual framework in Section V.

B. Lazy-switching-based Online Algorithm

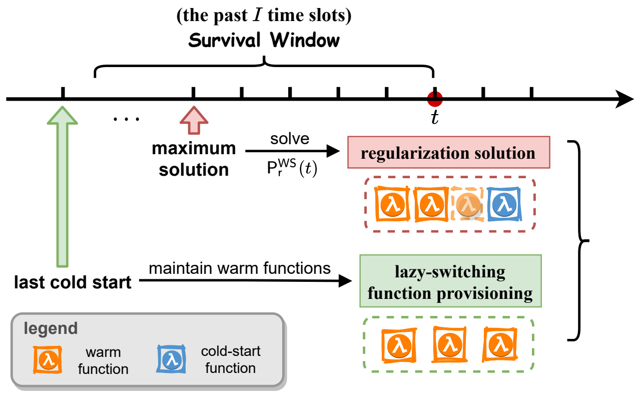


Fig. 4. The extended algorithm based on the idea of lazy-switching judiciously arbitrates whether to adopt the regularization solution, which may generate more cold starts in the current time slot, or to maintain warm functions according to the last cold start.

Algorithm 1 is fast and efficient, while the one-step regularization method uses the solution of the time slot $t-1$ to approximate the maximum solution of the past I time slots in the time-coupling term. This approximation only utilizes the one-step historical information of the time slot $t-1$. Moreover, in realistic serverless systems, keeping a pool of warm containers is an effective method to alleviate cold starts [48], [49]. Therefore, to further mitigate the cold start of functions, we next design an extended algorithm based on the following two ideas: 1) Fully consider all the historical information of the past I time slots to correct the inaccuracy of the approximation in the one-step regularization method. 2) Use warm functions as much as possible to reduce potential function cold starts during their survival periods.

We first construct a multi-step regularization problem by determining the maximum solution of the previous I time slots in each time slot. Let $\mathsf{P}_\mathsf{r}^\mathsf{WS}$ represents the multi-step regularized *Window Switching* problem, and the decomposed subproblem $\mathsf{P}_\mathsf{r}^\mathsf{WS}(t)$ for each time slot $t \in \mathcal{T}$ can be denoted as follow:

$$\min \; P_r^{WS}(t) = L^{nCS}\big(\boldsymbol{x}(t)\big) + \omega \cdot Cost\big(\boldsymbol{y}(t)\big) + \sum_{f \in \mathcal{N}} \frac{T^c}{\eta_f}\left(\big(y_f(t)+\epsilon\big)\ln \frac{y_f(t)+\epsilon}{\max_{\tau=1}^{I} y(t-\tau)+\epsilon} - y_f(t)\right),$$

$$\text{s.t.} \quad \text{Constraint (1a) to (1e).}$$

In addition, the cold start time between the functions with different memory is similar [9], but their execution time and running cost are quite different, referring to our measurements in Section VII. Therefore, making the most of warm functions (the function within I time slots after execution) can effectively reduce the occurrence of new cold starts, although it may cause a increase in serverless cost or inference latency due to the selection of non-optimal configurations. Furthermore, keeping the function warm can alleviate the massive cold start latency caused by a surge of request arrivals in the near future. The above two insights show that maintaining a number of functions in the survival period (the next I time slots) is beneficial to mitigate the cold start latency. Inspired by the lazy-switching method [50], we apply the relationship between cold-start latency and the sum of other objective terms to judiciously decide to accept the solution from regularied window switching problem $\mathsf{P}_\mathsf{r}^\mathsf{WS}(t)$ or keep a fixed function provisioning decision (i.e. maintain the pool of currently warm functions).

Algorithm 2: Window Forward Lazy-switching — WFL

```
    Input: $\mathcal{M}, \mathcal{N}, \mathcal{K}, \boldsymbol{C}, \boldsymbol{E}, \boldsymbol{P}, \boldsymbol{T}^e, \boldsymbol{T}^s, T^c, \boldsymbol{b}, \boldsymbol{d}, \boldsymbol{q}, \boldsymbol{\eta}, \epsilon, \omega$.
1   Initialize $t = 1, \hat{t} = 1$, and $\boldsymbol{y}(0) = \boldsymbol{0}$;
2   while $t \le T$ do
3       Observe $\boldsymbol{A}(t), \boldsymbol{B}(t), \max_{\tau=1}^{I} \boldsymbol{y}(t-\tau)$;
4       Solve $\mathsf{P}_\mathsf{r}^\mathsf{WS}(t)$ to obtain the solution $\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)$;
5       if $L_{CS}\big(\overline{\boldsymbol{y}}(t)\big) \ge \frac{1}{\rho}\sum_{v=\hat{t}}^{t-1} L^{nCS}\big(\overline{\boldsymbol{x}}(t)\big) + \omega \cdot Cost\big(\overline{\boldsymbol{y}}(t)\big)$ then
6           $\boldsymbol{y}(t) = \boldsymbol{y}(\hat{t})$;
7           $\dot{\boldsymbol{x}}(t) = \min L^{nCS}(t)$ subject to (1a)-(1e);
8           $\boldsymbol{x}(t) = \dot{\boldsymbol{x}}(t)$;
9       if $\boldsymbol{x}(t)$ is not derived then
10          $\boldsymbol{x}(t), \boldsymbol{y}(t) = \overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)$;
11          if $L_{CS}\big(\overline{\boldsymbol{y}}(t)\big) > 0$ then
12              $\hat{t} = t$;
    Output: $\boldsymbol{x}(t), \boldsymbol{y}(t)$.
```

Based on the above ideas as shown in Fig. 4, we further design a *Window Forward Lazy-switching* (WFL) algorithm as shown in Algorithm 2. Specifically, in each time slot, WFL first looks forward for all historical information in the past I time slots (i.e., the survival window) to obtain the maximum solution $\max_{\tau=1}^{I} \boldsymbol{y}(t-\tau)$, and solves a single-slot regularized window switching problem based on the request arrivals $\boldsymbol{A}(t)$ and bandwidth $\boldsymbol{B}(t)$ of the current time slot to obtain the solution $\big(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)\big)$. If the cold start latency caused by solution $\overline{\boldsymbol{y}}(t)$ in the current time slot exceeds $\frac{1}{\rho}$ (a predefined control parameter) times of the cumulative sum of non-cold-start latency and serverless cost since the last cold start occurred (i.e., $t = \hat{t}$), we keep the maximum solution within the time window I and solve the convex optimization problem to obtain a solution $\dot{x}(t)$ after applying $\boldsymbol{y}(t) = \boldsymbol{y}(\hat{t})$. If $\boldsymbol{x}(t)$ is not derived right now, the multi-step regularization solution $\big(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)\big)$ is adopted in the current time slot, and check whether a new cold start occurs to update $\hat{t}$. Note that there are two possible cases if line 8 is executed: (i) the cold start latency achieved by solution $\overline{\boldsymbol{y}}(t)$ does not exceed the threshold (i.e., the opposite of line 5) and (ii) minimizing $L^{nCS}(t)$ subject to (1a)-(1e) by applying $\boldsymbol{y}(t) = \boldsymbol{y}(\hat{t})$ (i.e., line 6-7) is infeasible. The second case is because constraint (1b) and constraint (1d) may not constitute a feasible region when the request arrival $A_i(t)$ is large and $\boldsymbol{y}(t) = \boldsymbol{y}(\hat{t})$ is fixed. At this time, even if the cold start latency exceeds the threshold, we still apply $\boldsymbol{x}(t), \boldsymbol{y}(t) = \overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)$ (i.e., line 9).

The extended algorithm can achieve better performance compared to our fast algorithm, although it increases the time complexity. Because it constructs a multi-step regularization problem to calculate the cold start delay more accurately, and fully leverages warm functions in the survival period to reduce the occurrence of cold starts. However, we cannot derive a competitive ratio for Algorithm 2, because when WFL chooses a fixed function provisioning decision (i.e., line 6), minimizing $L^{nCS}\big(\boldsymbol{x}(t)\big)$ may be unsolvable due to the surge of request arrivals. At this point the algorithm has to choose the regularization solution $\big(\overline{\boldsymbol{x}}(t), \overline{\boldsymbol{y}}(t)\big)$ even if the threshold condition (i.e., line 5) is satisfied. If we consider a pool that always maintain a large number of warm functions, we can avoid the unsolvable situation of minimizing $L^{nCS}\big(\boldsymbol{x}(t)\big)$ when applying $\boldsymbol{y}(t) = \boldsymbol{y}(\hat{t})$ and thus obtain a parameterized competition ratio. Unfortunately, in a realistic system, it is not advisable to keep a large number of warm functions all the time, because it causes much overhead. Although there is no theoretical guarantee for Algorithm 2, our experiments in Section VII demonstrate that our extended algorithm achieves at least a 9.4% improvement with a small increase in execution time compared to the fast algorithm.

V. Performance Analysis

A. Basic Idea

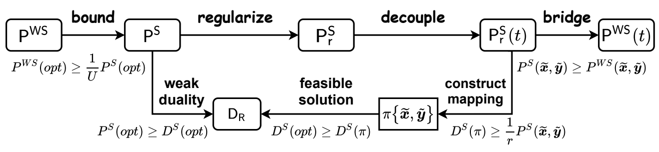

P^{WS}
bound
P^{S}
regularize
$\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}$
decouple
$\mathsf{P}^{\mathsf{S}}_{\mathsf{r}}(t)$
bridge
$\mathsf{P}^{\mathsf{WS}}(t)$
$P^{WS}(opt) \geq \frac{1}{U} P^S(opt)$
$P^S(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \geq P^{WS}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$
weak duality
feasible solution
construct mapping
D_{R}
$\pi\{\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}\}$
$P^S(opt) \geq D^S(opt)$
$D^S(opt) \geq D^S(\pi)$
$D^S(\pi) \geq \frac{1}{r} P^S(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$

Fig. 5. An illustration of the basic idea for performance analysis.

As illustrated in Fig. 5, to formally prove that the objective value of *Window Switching* problem P^{WS} achieved by Algorithm 1 is upper-bounded by a parameterized constant times the offline optimum, we apply the traditional *Switching* problem P^{S} as an important bridge for competitive analysis:

$$\min \ P^S = \sum\nolimits_{t=1}^{T} \Big\{ L^{nCS}\big(\boldsymbol{x}(t)\big) + \omega \cdot Cost\big(\boldsymbol{y}(t)\big) + \sum_{f \in \mathcal{N}} T^c \big[y_f(t) - y_f(t-1)\big]^+ \Big\},$$
$$\text{s.t.} \quad \text{Constraint (1a) to (1e).}$$

The time-coupling term for calculation of the cold start latency in our problem P^{WS} is different from the traditional *Switching* problem, which makes the *Window Switching* problem non-convex and novel. To simplify the expression, we use the definition symbol of each optimization problem to denote the corresponding objective value achieved by different solutions and $P(opt)$ to denote the objective value of problem P achieved by the offline optimal solution. We will establish the following chain of inequalities:

$$P^{WS}\big(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)\big) \tag{2a}$$
$$\leq P^{S}\big(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)\big) \tag{2b}$$
$$\leq r D^{S}\big(\pi(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t))\big) \tag{2c}$$
$$\leq r P^{S}(opt) \tag{2d}$$
$$\leq r U P^{WS}(opt), \tag{2e}$$

where rU is the overall *competitive ratio*. We first derive (2a) $\leq$ (2b) because $\max_{\tau=1}^{I} \widetilde{y}_f(t-\tau) \geq \widetilde{y}_f(t-1), \forall t, f$. Next we obtain the part competitive ratio r by deriving (2b) $\leq$ (2d) through the primal-dual framework summarized in Theorem 1 and the upper bound U by deriving (2d) $\leq$ (2e) in Theorem 2.

B. Competitive Ratio of Regularization

An Equivalent Problem Transformation. We introduce a set of new auxiliary variables $w_f(t)$ which satisfy $w_f(t) \geq y_j(t) - y_j(t-1), \forall t, \forall f$ to replace the time-coupling term $\big[y_f(t) - y_f(t-1)\big]^+$, and a set of *knapsack cover* (KC) constraints $\sum_{f' \in \mathcal{N}} y_{f'}(t) - y_f(t) \geq \sum_i \big(A_i(t) - C_i\big) - E_f, \forall t, \forall f$ to replace the boxing constraints $y_f(t) \in [0, E_f]$. With the above transformations, we rewrite problem P^S in the following equivalent form.

$$\min \ P^S = \sum\nolimits_t \Big\{ L^{nCS}(\boldsymbol{x}(t)) + \omega \cdot Cost(\boldsymbol{y}(t)) + \sum\nolimits_f T^c w_f(t) \Big\}$$
$$\text{s.t.} \quad w_f(t) \geq y_f(t) - y_f(t-1), \ \forall t, \forall f, \tag{3a}$$
$$\sum\nolimits_k \sum\nolimits_f x_{ikf}(t) \leq A_i(t), \ \forall t, \forall i, \tag{3b}$$
$$A_i(t) - \sum\nolimits_k \sum\nolimits_f x_{ikf}(t) \leq C_i, \ \forall t, \forall i, \tag{3c}$$
$$\sum\nolimits_i \sum\nolimits_k x_{ikf}(t) \leq y_f(t), \ \forall t, \forall f, \tag{3d}$$
$$\sum\nolimits_k \sum\nolimits_f x_{ikf}(t) d_{ik} \leq A_i(t) q_i, \ \forall t, \forall i, \tag{3e}$$
$$\sum\nolimits_{f'} y_{f'}(t) - y_f(t) \geq \sum\nolimits_i \big(A_i(t) - C_i\big) - E_f, \ \forall t, \forall f, \tag{3f}$$
$$x_{ikf}(t), y_f(t), w_f(t) \geq 0, \ \forall t, \forall i, \forall k, \forall f. \tag{3g}$$

Formulating the Lagrange Dual Problem of P^{S}. We derive the dual problem D^{S} as follows, where $\nu_f(t), \alpha_i(t), \beta_i(t), \gamma_f(t), \theta_i(t), \lambda_j(t)$ denote the corresponding dual variables for the constraints (3a) to (3f).

$$\max \quad D^S = \sum\nolimits_t \sum\nolimits_i \big(T_i^e - \alpha_i(t) + \theta_i(t) q_i\big) A_i(t)$$
$$+ \sum\nolimits_t \sum\nolimits_f \lambda_f(t) \big[\sum\nolimits_i (A_i(t) - C_i) - E_f\big]$$
$$+ \sum\nolimits_t \sum\nolimits_i \beta_i(t) \big(A_i(t) - C_i\big)$$

$$\text{s.t.}\quad T_f^s - T_i^e + \tfrac{b_{ik}}{B_i(t)} + \alpha_i(t) - \beta_i(t) + \gamma_f(t)$$
$$- \theta_i(t) d_{ik} \geq 0,\ \forall t \in \mathcal{T}, \forall i \in \mathcal{M}, \forall k \in \mathcal{K}, \forall f \in \mathcal{N}, \tag{4a}$$
$$T^c - \nu_f(t) \geq 0,\ \forall t \in \mathcal{T}, \forall f \in \mathcal{N}, \tag{4b}$$
$$P_f T_f^s + \nu_f(t) - \nu_f(t+1) - \gamma_f(t) + \lambda_f(t)$$
$$- \textstyle\sum_{f'} \lambda_{f'}(t) \geq 0,\ \forall t \in \mathcal{T}, \forall f \in \mathcal{N}, \tag{4c}$$

All the dual variables ≥ 0.

Characterizing the Optimality of the Regularized Problem. Algorithm 1 produces the optimal solution $\big(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)\big)$ of the convex problems $\mathsf{P}_\mathsf{r}^\mathsf{S}(t)$, which satisfies the *Karush-Kuhn-Tucker* (KKT) conditions. To simplify the presentation, we write KKT conditions in the disjunctive form as follows:

$$\widetilde{\alpha}_i(t) \perp \Big(A_i(t) - \textstyle\sum_k \sum_f \widetilde{x}_{ikf}(t)\Big), \forall t, i, \tag{5a}$$
$$\widetilde{\beta}_i(t) \perp \Big(\textstyle\sum_k \sum_f \widetilde{x}_{ikf}(t) - \big(A_i(t) - C_i\big)\Big), \forall t, i, \tag{5b}$$
$$\widetilde{\gamma}_f(t) \perp \Big(\widetilde{y}_f(t) - \textstyle\sum_i \sum_k \widetilde{x}_{ikf}(t)\Big), \forall t, f, \tag{5c}$$
$$\widetilde{\theta}_i(t) \perp \Big(\textstyle\sum_k \sum_f \widetilde{x}_{ikf}(t) d_{ik} - A_i(t) q_i\Big), \forall t, i, \tag{5d}$$
$$\widetilde{\lambda}_f(t) \perp \Big(\textstyle\sum_{f'} \widetilde{y}_{f'}(t) + E_f - \sum_i \big(A_i(t) - C_i\big) - \widetilde{y}_f(t)\Big), \forall t, f, \tag{5e}$$
$$T_f^s - T_i^e + \tfrac{b_{ik}}{B_i(t)} + \widetilde{\alpha}_i(t) - \widetilde{\beta}_i(t) + \widetilde{\gamma}_f(t) - \widetilde{\theta}_i(t) d_{ik} = 0, \tag{5f}$$
$$P_f T_f^s + \tfrac{T^c}{\eta_f} \ln \tfrac{\widetilde{y}_f(t) + \epsilon}{\widetilde{y}_f(t-1) + \epsilon} - \widetilde{\gamma}_f(t) + \widetilde{\lambda}_f(t) - \sum_{f'} \widetilde{\lambda}_{f'}(t) = 0, \tag{5g}$$

where $a \perp b$ is equivalent to $a, b \geq 0$ and $ab = 0$.

Constructing the Mapping. We construct a mapping to jointly map $\mathsf{P}_\mathsf{r}^\mathsf{S}$'s optimal primal and dual solutions to a feasible solution of the dual problem D^S as follows:

$$\pi\{\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t)\} = \big(\nu_f(t), \alpha_i(t), \beta_i(t), \gamma_f(t), \theta_i(t), \lambda_f(t)\big),$$

in which we let

$$\nu_f(t) = \frac{T^c}{\eta_f} \ln \frac{E_f + \epsilon}{\widetilde{y}_f(t-1) + \epsilon}, \alpha_i(t) = \widetilde{\alpha}_i(t),$$
$$\beta_i(t) = \widetilde{\beta}_i(t), \gamma_f(t) = \widetilde{\gamma}_f(t), \theta_i(t) = \widetilde{\theta}_i(t), \lambda_f(t) = \widetilde{\lambda}_f(t).$$

Based on the above analysis, we bound non-cold-start latency, serverless cost and cold start latency respectively, also establish the chain of inequalities $D^S\big(\pi(\widetilde{\boldsymbol{x}}(t), \widetilde{\boldsymbol{y}}(t))\big) \leq D^S(opt) \leq P^S(opt)$ according to the *Weak Duality* [51]. Summarizing all derivations, we have the following Theorem 1.

Theorem 1. *The objective value of problem* P^S *achieved by Algorithm 1 is no larger than* r *times of the offline optimum* $P^S(opt)$*, where* r *is given by:*

$$r = \ln\Big(1 + \frac{E_{max}}{\epsilon}\Big) + \frac{E_{max}}{\delta} + 1,$$

and $E_{max} = \max_f E_f$, $\delta = \min_{t,f} \widetilde{y}_f^+(t)$, *where* $\widetilde{y}_f^+(t) \in \big\{\widetilde{y}_f(t) \mid \widetilde{y}_f(t) > 0, \forall t \in \mathcal{T}, f \in \mathcal{N}\big\}$.

Proof. See Appendix A. □

C. Upper Bound of Offline Optimum

Finally, we prove (2d) $\leq$ (2e) and derive the upper bound U of offline optimum between the *Switching* problem P^S and our *Window Switching* problem P^WS in Theorem 2. We define $\mu = T^+/T$, where $T^+ = |\mathcal{T}^+|$ and $\mathcal{T}^+ = \big\{t \mid \sum_i [A_i(t) - C_i] > 0\big\}$, to denote the proportion of the number of time slots in which the total inference requests received by edge devices exceeds their capacity limitation in the total number of time slots over the entire time horizon.

Theorem 2. *The offline optimum of problem* P^S *is no larger than* U *times of the offline optimum of problem* P^WS*, where*

$$U = \frac{1}{\mu} \times \frac{\Big(\sum_i C_i T_i^e + \kappa_2 \sum_f E_f\Big)}{\kappa_1 \min\limits_{t \in \mathcal{T}^+} \sum_i \big(A_i(t) - C_i\big)},$$
$$\kappa_1 = \min_{i,k,t} \frac{b_{ik}}{B_i(t)} + \min_f T_f^s + \min_f T_f^s P_f,$$
$$\kappa_2 = \max_{i,k,t} \frac{b_{ik}}{B_i(t)} + \max_f T_f^s + \max_f T_f^s P_f + T^c.$$

Proof. See Appendix B. □

Remark. According to the definition of the set $\mathcal{T}^+$, $\forall t \in \mathcal{T}^+$ we have $\sum_i \big(A_i(t) - C_i\big) > 0$, which ensures that the parameterized upper bound U is always positive. In addition, the parameter μ contained in the upper bound U may vary with the length of the time horizon, but its value is always equal or close to 1. Because in a realistic inference system, the number of inference requests generated by edge devices can be sufficiently large, exceeding the capacity of devices with limited resources [52]. Therefore, the parameter U does not actually increase with the number of time slots T. This is also verified by measurements based on real-world request traces in the experiment section.

The competitive ratio r decreases with the increase of the parameter ϵ. By increasing ϵ sufficiently, we can obtain the coefficient r that is arbitrarily close to $\frac{E_{max}}{\delta} + 1$, while also increasing the time complexity of Algorithm 1. The coefficient U essentially reflects the distinction between the traditional *Switching* problem and the *Window Switching* problem. When the cold start time T^c of the serverless function increases, the gap between the two offline optimum enlarges, resulting in a reduction in the theoretical performance of Algorithm 1. The overall competitive ratio rU is independent of the function survival time I, because Algorithm 1 factually utilizes the historical information of the previous time slot $t-1$, the best performance is achieved when $I = 1$ as a result. The performance of Algorithm 1 is inferior to that of Algorithm 2 as I increases, because WFL combines the idea of regularization and "lazy switching" [50] to minimize the cold start latency in the current time slot and the future window I, thus achieving better performance than OFR. We will verify the effect of the two algorithms in Section VI and Section VII.

VI. Real-world Evaluation

A. System Prototype

We first evaluate the performance of the proposed algorithm through realistic prototype experiments on the serverless cloud platform of AWS Lambda. The system architecture of the prototype implementation is shown in Fig. 6.

Central Console. We run the central console on a host equipped with AMD EPYC 7532 32-Core Processor, which receives request and bandwidth information from edge devices

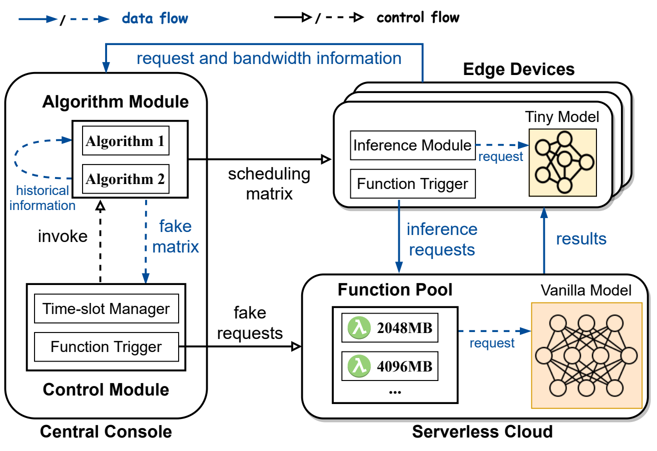


Fig. 6. The architecture diagram of the system prototype consists of a central console running on the host, three edge devices mimicked by containers, and a serverless cloud of AWS. The solid arrows represent the flow between components, and the dotted arrows represent the flow within the components.

and connects to the serverless cloud. The algorithm module can set different scheduling algorithms (including Algorithm 1, Algorithm 2 and other baselines), and the time-slot manager first invoke the algorithm after receiving the device information in each time slot. Next, the output of scheduling matrix is sent to the corresponding edge device, and the fake matrix is sent to the control module. Finally, according to the fake matrix, the function trigger sends fake requests to the serverless cloud for maintaining the function container.

Edge Devices. We mimic edge devices by using three Docker-based containers, which are with the operating system of Debian GNU/Linux 11. The resource allocation of containers reflects the heterogeneity of computing resources of different edge devices. In addition to the CPU, device 3 is also equipped with a GeForce RTX 4090 GPU to achieve local inference acceleration. According to the computing resources of different devices, we deploy different versions of the DNN model YOLOv5 [53] in these containers respectively, and the inference framework is ONNXRuntime [54]. The specific configuration of the containers is in Table III.

TABLE III
CONTAINER PARAMETERS

Device	CPU Cores	Memory	GPU	Model
#1	6	8GB	-	YOLOv5n
#2	8	16GB	-	YOLOv5s
#3	6	8GB	24GB	YOLOv5m

Serverless Cloud. We choose the mainstream AWS Lambda as the serverless computing platform in our system prototype. We deploy the large version of the DNN model YOLOv5 [53] on the cloud, and choose five functions with memory configurations of 2048MB, 3072MB, 4096MB, 6144MB, and 8192MB. The model is stored in AWS's S3 storage service, and the inference framework is also ONNXRuntime [54].

Note that request configuration (i.e., resolution) is ignored in our system prototype because the bandwidth from the container to the AWS cloud is difficult to measure in real time. Therefore, inference requests offloaded from the device to the serverless cloud are raw data under the default configuration. We supplement the bandwidth and request configuration in the simulation experiment of Section VII.

B. Other Settings

Traces. We use the request traces of three different regions (Back Bay, Beacon Hill and Boston University) on November 27 in Uber and Lyft Dataset Boston [55]. Specifically, we count the number of rideshare requests to generate the normalized request arrivals, and then multiply this value by 1,000 as the inference workload $A_i(t)$ of different devices.

Algorithms. We evaluate three scheduling algorithms in our system prototype as follows:

(i) `OFR`, is Algorithm 1 we proposed in Section IV-A, but ignores the transmission latency and accuracy constraint (1c) in the optimization problem P^{WS}.

(ii) `WFL`, is Algorithm 2 we proposed in Section IV-B, but ignores the transmission latency and accuracy constraint (1c) in the optimization problem P^{WS}.

(iii) `Only`, a serverless-only scheduling algorithm that does not consider edge inference to optimize function cold start. By setting the local capacity $C_i = 0$ and ignoring the transmission latency, cold start latency, and the accuracy constraint (1c) in the optimization problem P^{WS}, the algorithm then solves the single-slot subproblem in each time slot.

Metrics. In order to evaluate the performance of the system prototype under different scheduling algorithms, we record the following five indicators in edge devices and AWS CloudWatch logs:

(i) *local duration*, the duration from the start of local inference with the first request to the completion of processing all requests, is used as the total latency of the edge inference.

(ii) *serverless duration*, the duration from the start of sending the first request to the receipt of the response of the last request, is used as the end-to-end latency of the serverless inference.

(iii) *request duration*, the duration from sending the request to receiving the response, is used as the end-to-end latency of the offloaded request. We use *request_id* to index the requests sent to the serverless cloud and track the inference results of the requests.

(iv) *billed duration*, recorded in AWS CloudWatch logs as the execution time of the function.

(v) *init duration*, recorded in AWS CloudWatch logs as the cold start time of the function.

Note that the bandwidth from the edge device to the serverless cloud is not considered in the experiment of the system prototype, but we can still calculate the transmission time of the request through the end-to-end latency of the request and the execution time of the function.

C. Evaluation Results

Based on the above real-world traces, we apply 3 minutes as the length of a time slot. We run the system prototype with three different scheduling algorithms for 24 hours ($T = 480$), and record the above metrics.

Function Containers, Cost and End-to-end Latency. Through the number of logs captured by AWS CloudWatch and the recorded *request_id*, we count the number of containers used by Lambda over the entire time horizon under different algorithms and the number of requests offloaded to the cloud. Thus, we can calculate the average number of inference requests per function container under different algorithms, as an indicator of function container utilization. In the Cost Explorer of the AWS Management Console [26], we can obtain the cost incurred over the entire time horizon under each algorithm. The end-to-end latency of serverless inference in each time slot depends on the one with the largest *serverless duration* among all devices. We record all the end-to-end latency in $T = 480$ and calculate the average value per time slot. The results of the above multiple dimensions are summarized in Table IV. According to the observations in the table, our proposed `ORF` and `WFL` algorithms significantly improve function container utilization (i.e., the actual number of requests processed in the function survival period) compared to the baseline, where `ORF` and `WFL` each reduces 26% and 49% of function containers compared to `Only`. The higher container utilization is due to the fake request extending the survival period of the function container, so `WFL` pays an additional 6% cost compared to `OFR` with a 33% improvement in utilization. Compared with the cloud-only solution (`Only`), the solutions based on the cloud-edge collaboration (`OFR` and `WFL`) reduce up to 11.6% of serverless cost, because inference by local models does not require additional overhead from the cloud service provider. In addition, the average end-to-end latency of serverless inference under the three algorithms are significantly less than the length of a time slot (180 seconds), which is consistent with our model assumption: all inference requests could be processed in the current time slot.

TABLE IV
MULTIDIMENSIONAL RESULTS UNDER DIFFERENT ALGORITHMS

Alg.	# of Containers	Utilization	Cost	End-to-end Lat.
`Only`	37544	12.1	63.17$	13.4s
`OFR`	27740	15.5	**55.82$**	**11.6s**
`WFL`	**18451**	**20.6**	59.72$	12.8s

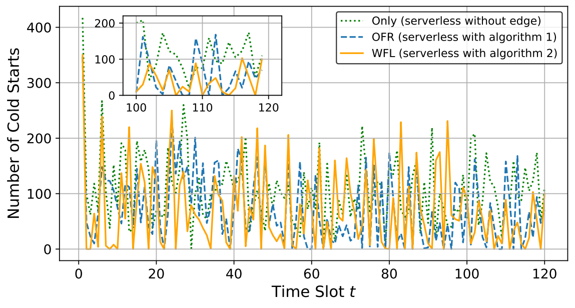


Fig. 7. The number of cold starts in each time slot under different algorithms, and the results from $t = 100$ to $t = 120$ are highlighted.

Cold start. Through the recorded *init duration*, we count the number of functions with cold start in each time slot under different algorithms, as shown in Fig. 7. To simplify the presentation, we only plot the results for the first 120 time slots and highlight the results from $t = 100$ to $t = 120$. We can observe that `WFL` has fewer cold starts compared to `OFR` and `Only`, and the number of cold starts does not exceed 100 in most time slots. In the entire time horizon $T = 480$, the total number of cold starts by `Only`, `OFR` and `WFL` algorithms are 36946, 26739 and 18064 respectively. Compared with the baseline `Only`, `OFR` and `WFL` achieves 27.6% and 51.1% reduction in the number of cold starts respectively.

Latency Composition. Fig. 8 shows the average objective values of various latency under different algorithms, referring to $L_{EI}(t)$, $L_{TR}(t)$, $L_{SI}(t)$ and $L_{CS}(t)$ in $P^{WS}(t)$. Due to the optimization of request configuration is ignored in our system prototype, the transmission latency $L_{TR}(t)$ mainly depends on the number of requests offloaded to the serverless cloud. Therefore, the baseline `Only` that does not consider edge inference has a large transmission latency. The inference latency includes the sum of edge inference $L_{EI}(t)$ and serverless inference $L_{SI}(t)$, and there is no obvious distinction between the three algorithms. Nonetheless, our proposed `OFR` and `WFL` are slightly smaller than the baseline because the local tiny model with less parameters reduce the inference time compared to serverless computing. As for the cold start latency $L_{CS}(t)$, our proposed algorithms has a significant improvement compared to the baseline, and `WFL` further reduces the cold start latency by 32.2% compared to `OFR`.

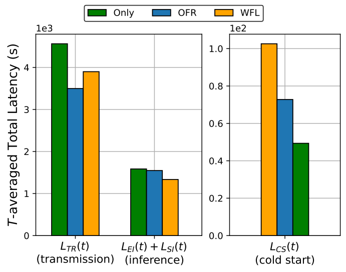


Fig. 8. Various total latency per time slot under different algorithms, i.e. $L_{EI}(t)$, $L_{TR}(t)$, $L_{SI}(t)$ and $L_{CS}(t)$ in objective value of problem P^{WS}. Compared with the baseline, our proposed `OFR` and `WFL` algorithms significantly reduce cold start latency.

Note that the latency in Fig. 8 is the accumulation of the time spent by all requests in each time slot. Since the number of requests with cold start is only a part of the total number, so the value of $L_{CS}(t)$ is much smaller than $L_{TR}(t)$ and $L_{EI}(t) + L_{SI}(t)$. In fact, edge inference, serverless inference and serverless function are all processed in parallel in our system prototype. Therefore, the time duration for processing all requests in each time slot depends on the time when the last inference result is returned, referring to the end-tot-end latency in Table IV.

Average Time on Each Device. In order to show the details of different devices, we select the records in `WFL` algorithm and calculate the various average time per request on each device, as shown in Fig. 9. The function processing (i.e., serverless inference) time and cold start time have no obvious distinction on different devices, because they mainly depend on the configuration of functions and the inference model, which is common with YOLOv5l. Since the local tiny models (YOLOv5n, YOLOv5s, and YOLOv5m) have less parameters, the local processing time (green bars) on the three devices is much smaller than the function processing time (orange bars) on the cloud. Furthermore, benefiting from GPU acceleration, device 3, which deploys the YOLOv5m model with a relatively large number of parameters, has the lowest local processing time. Therefore, edge devices equipped with GPUs can significantly reduce the latency of local inference, so that more complex DNN models can be deployed within limited SLO (Service Level Objective) to improve the accuracy of edge inference in practice.

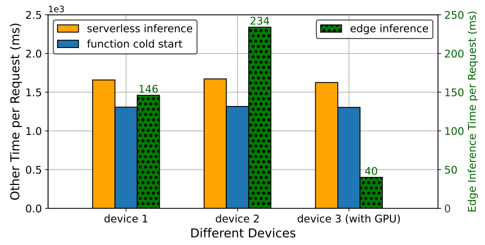


Fig. 9. Various time per request under WFL algorithm on each device, where the local processing time is much smaller than the function inference time and cold start time, refer to the green coordinate scale on the right. Device 3 equipped with GPU has significant acceleration of edge inference.

VII. Simulation Experiment

A. Experimental Setup

We simulate a cloud-edge collaborative inference system consisting of three types of devices, which connect to a remote cloud with serverless computing service. Based on different types of devices and deployed models, we set the number of devices (i.e., M) with different scales to 3, 9, and 15 in our experiment. Taking the object detection task as an example, the inference request contains images with different configuration resolutions of 240p, 360p, 480p, and 720p ($K = 4$). For serverless computing service, serverless functions with different memory sizes of 2048MB, 3072MB, 4096MB, 6144MB, and 8192MB ($N = 5$) are used to provide inference services.

Request Traces and Bandwidth. Similar to real-world evaluation, we also use the request traces of three different regions in Uber and Lyft Dataset Boston [55]. The statistical interval of requests is 3 minutes, which is the length of a single time slot in our experiment. We set the total time horizon $T = 480$, which corresponds to the complete request traces within 24 hours. Given the real-world request traces, the parameter μ in Theorem 2 is always equal to 1 in our experiments, indicating that the parameterized upper bound U is independent of the time span T. We apply the uplink rates of the static device at three different times in the 4G LTE Dataset [56] as the dynamic bandwidth $B_i(t)$ of edge devices. We calculate the data amount of inference requests generated by different edge devices according to the resolution of images with different configurations, and multiply different compression rates (100%, 80%, and 60%) according to the different accuracy requirements and heterogeneous bandwidth as the transmission data size b_{ik} under different configurations of different devices.

Edge Inference. We assume that three mainstream object detection models YOLOv2 [57], SSD [58], and R-FCN [59] are deployed on different edge devices to process inference requests locally. Therefore, the edge inference time T_i^e and

TABLE V
Accuracy Parameters

Deployed Model	Inference Accuracy a_{ik}					Q_i
	240p	360p	480p	720p	edge	
YOLOv2	0.671	0.717	0.758	0.765	0.60	0.75
SSD	0.761	0.799	0.815	0.818	0.62	0.74
R-FCN	0.728	0.771	0.811	0.817	0.65	0.73

TABLE VI
Serverless Function Parameters

Parameter	Memory Size				
	2048MB	3072MB	4096MB	6144MB	8196MB
T_f^s	2261ms	1641ms	1278ms	1050ms	914ms
E_f	1000	800	500	250	150

capacity C_i are directly related to the deployed inference models. Specifically, we set the C_i according to the size of three models after training, and estimate $T_i^e = [250, 500, 170]$ based on the FPS (frames per second) of each model then scale it with constrained edge resources. Taking the inference request of car detection as an example, we calculate the inference accuracy of inference requests with different configurations from different devices through the curve fitting method [60], and set the edge inference accuracy a_{i0} and the service quality Q_i of different devices according to the deployed models. The specific accuracy parameters are shown in Table V.

Serverless Function. Considering the sufficient resources, we assume that a more accurate model (YOLOv5 model [53]) is allocated to functions on the serverless cloud. We deploy the YOLOv5l model on AWS Lambda with different memory configurations, and measure the inference time and cold start time of each function by recording *Init Duration* and *Billed Duration* in the CloudWatch logs, as shown in Fig. 10. We repeatedly invoke each function 20 times after cold start (i.e., the first call), calculate the average running time as the processing time T_f^s. According to AWS Lambda's default value of 1,000 for function concurrency, we set the values of parameter E_f for different functions according to their memory size as shown in Table VI. Regarding the cold start time, we repeat the test 10 times for functions with different memory, and the time interval between each call is larger than 10 minutes to ensure that the survival window of the function is exceeded. The experimental results show that the cold start time of functions with different memory has no visible difference, because it is closely related to the deployed package size [9]. In our experiment, the default cold start time T^c is set to 2s, and the function running price refers to [27].

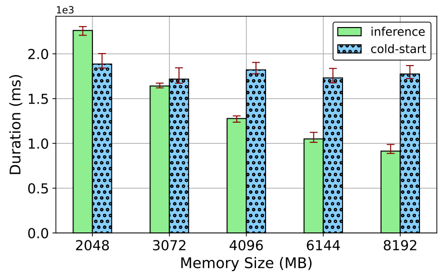


Fig. 10. Function inference time and cold-start time of YOLOv5l model inference with different memory size on AWS Lambda, the average result of each function repeated 20 times.

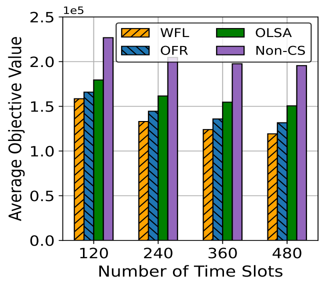


Fig. 11. Average objective value.

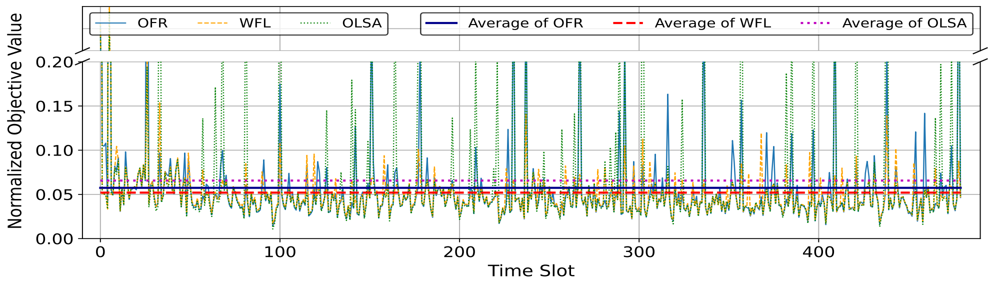


Fig. 12. Normalized objective value in each time slot with different algorithms and the marked mean value.

Benchmarks and Metrics. We compare our algorithms to OLSA [61], which is a state-of-the-art online algorithm applied to tackle problems with the switching term [62], [63]. To demonstrate the efficiency in reducing the cold start of serverless functions, we also compare them with the scheme Non-CS, which directly solves the convex problem without considering the cold start latency. As for metrics, we use the following intuitive quantitative indicators to measure the performance of different algorithms: 1) Objective Value (i.e., the total latency and serverless cost achieved by the algorithm for the original optimization problem P^{WS}), we use the average objective value and normalized objective value. 2) Total Number of Cold Starts (i.e., the total number of functions which have a cold start when they are triggered), we also apply the cold starts of the request arrivals over the entire time horizon to calculate the retention ratio of cold starts by different algorithms. 3) Approximate Competitive Ratio (i.e.,the total objective value achieved by different algorithms divided by the approximate offline optimum), we use the offline optimal value of the problem $P^{NS} = L^{nCS} + Cost$ without considering the cold start latency as an approximation, since the original optimization problem P^{WS} with non-convex terms cannot be directly solved even given priors. Note that the approximate competitive ratio is an upper bound of the true competitive ratio, because the approximate offline optimum must be no larger than the offline optimum of the original problem.

B. Simulation Results

If there is no special instruction, our experimental results are based on the default parameters: $T = 480, M = 3, I = 3, T^c = 2\text{s}, \omega = 10 \times 10^{-6}$.

Objective Value. We first plot the average objective value of different algorithms under varying total time slots T in Fig. 11. Our proposed OFR and WFL significantly outperform benchmarks, and WFL always achieves the lowest average total latency and serverless cost. In all 480 time slots, OFR achieves a maximum improvement of 32.6% and 12.6% compared to Non-CS and OLSA respectively, and WFL achieves a maximum improvement of 39.0% and 20.8% compared to Non-CS and OLSA respectively. Fig. 12 visualizes the normalized objective value of our proposed OFR and WFL as well as the benchmark OLSA per time slot over the entire time horizon. WFL has a more stable performance against uncertain future information (i.e., fluctuating workload and bandwidth), and achieves 9.4% improvement compared to OFR. Note that the average objective value decreases as T increases in Fig. 11, since the system is initialized without pre-warmed serverless functions. All invoked functions in the first time slot experience a cold start, resulting in a large amount of cold-start latency in total objective value. This value is then averaged by the increasing T, thus causing above phenomenon.

Total Number of Cold Starts. In order to demonstrate the efficiency of the proposed algorithm on cold-start reduction, we plot the total number of cold starts with different algorithms on the entire time horizon in Fig. 13, and calculate the cold-start retention ratio of each algorithm on the dataset. Compared with Non-CS and OLSA, OFR achieves 69.3% and 39.8% reduction in the number of cold starts respectively, and WFL achieves 84.3% and 69.3% reduction respectively. The proposed WFL achieves the smallest total number of cold starts and only 14.1% retention of cold starts on the dataset. Compared with the real-world evaluation result in Fig. 7, the algorithms in the simulation experiment have better performance in mitigating the function cold start. This may be because on the real-world testbed, the survival period of the function is within a small range, which is different from being precise to a certain minute in our cold-start latency model. As a result, the cold start reduction of the proposed algorithm in the real-world evaluation is slightly inferior to the performance in the simulation experiment. The function survival period changes with the strategy adjustment of different serverless computing platforms, which inspires us to adjust the hyper-parameter I in practice.

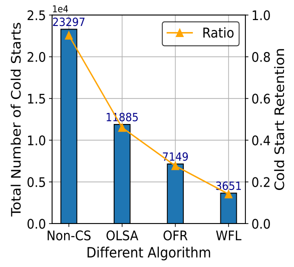


Fig. 13. Total number of cold starts and retention ratio.

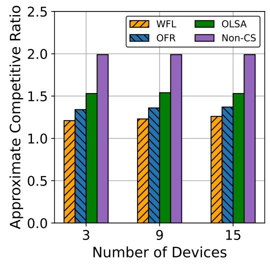


Fig. 14. Approximate competitive ratio at different scales.

Approximate Competitive Ratio. In order to verify the overall competitive ratio given in Section V, we measure the approximate competitive ratio of several algorithms at different scales (i.e., the number of devices $M = 3, 9, 15$) as shown in Fig. 14. The results show that there is no significant difference in the approximate competitive ratio of the algorithms under different M. Our proposed OFR and

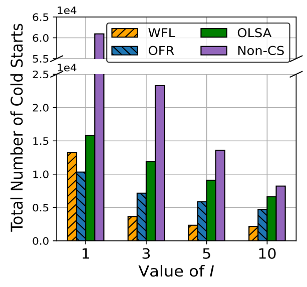


Fig. 15. Impact of survival length I.

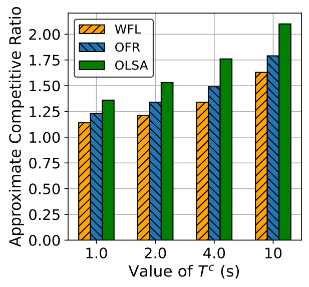


Fig. 16. Impact of cold start T^c.

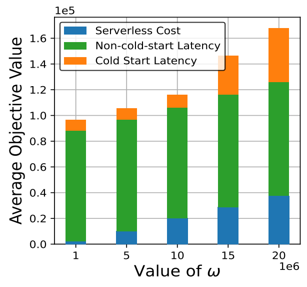


Fig. 17. Impact of coefficient ω.

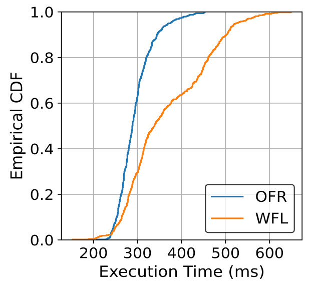


Fig. 18. Algorithm execution time.

WFL algorithm outperform the comparison methods, and WFL always achieves the smallest approximate competitive ratio (closest to the offline optimum). As we mentioned before that the approximate competitive ratio is the upper bound of the true competitive ratio, so the true competitive ratios achieved by OFR and WFL at different scales do not exceed 1.37 and 1.26, respectively.

Impact of Survival Length. Fig. 15 shows the total number of cold starts of different algorithms under different serverless function survival lengths I. We can observe that as the I increases, the total number of cold starts achieved by all algorithms decreases significantly, and our proposed OFR and WFL always have the smallest total number of cold starts. When $I = 1$, the *Window Switching* problem degenerates into the traditional *Switching* problem. At this time, OFR based on the one-step historical information achieves a lower number of cold starts and better performance than WFL, indicating that when the survival period is very short, the regularization method is more effective than the lazy switching method.

Impact of Cold Start Time. Fig. 16 shows the approximate competitive ratio of different algorithms under different cold start time T^c. We omit the results of the Non-CS method because the performance without considering the cold-start latency deteriorates rapidly when T^c increases leading to particularly large approximate competitive ratios. Compared with OLSA, our proposed OFR and WFL consistently achieve lower approximate competitive ratios (not exceeding 1.8). We notice that when T^c increases to 10s, the approximate competitive ratios of different algorithms all increase substantially (OLSA's even exceeds 2.0). Because our cold-start model is time-coupling with future uncertainty, the optimality gap of any online algorithm would enlarge as the cold-start time increases, which is also reflected in κ_2 in our Theorem 2.

Impact of Serverless Cost. Fig. 17 shows the average objective value of WFL over the entire time horizon under different weight coefficients of serverless cost, as well as the proportion of non-cold-start latency, cold-start latency and serverless cost in the objective value. The average objective value increases as the weight coefficient ω increases. When $\omega \leq 10$, the main reason for the increase in objective value is the increased serverless cost. When $\omega > 10$, the cold start latency starts to increase significantly, because the expensive running cost makes it difficult to reduce the cold start occurrence by retaining the warm functions. For the convenience of presentation we only plot the result of WFL, a similar phenomenon is also observed on OFR.

Execution Time. Fig. 18 depicts the cumulative distribution of the execution time of our proposed algorithms. The average execution time of OFR and WFL over all 480 time slots is about 294.9ms and 366.8ms, respectively, which are much less than the commonly adopted length of 3 minutes of a single time slot. Hence, our proposed algorithms are practically efficient.

VIII. Discussion and Future Work

The measurements and evaluations in this paper are mainly based on AWS Lambda. However, our proposed algorithms and system prototype can be generally extended to other major serverless computing platforms. Nevertheless, some hyper-parameters used in the algorithm are platform-dependent, such as survival period I and function price P_f, need to be re-tuned. Note that the parameter I is platform-dependent but fixed in our cold-start latency model. However, our proposed algorithms and theoretical analysis can be naturally extended to the general case when the survival periods vary for different functions. We are also willing to discuss possible future work for this research as follows.

Dynamic survival period. We assume that the survival period of all serverless functions is fixed as the time goes in our cold-start latency model. This assumption may change as strategies adjustment of commercial serverless computing platforms, such as based on the workload of the function. The dynamic survival period of the function will bring novel challenges to solving the window switching problem.

Further exploration of competition ratio. Both in real-world testbed and simulation experiments, our proposed extended algorithm achieve significant performance improvements compared to the fast algorithm. However, there is currently a lack of theoretical analysis for the lower bound of algorithm performance. Intuitively, the competitive ratio that proves this bound is likely to be related to the survival period of the function.

Real-world system implementation. The system prototype can be further combined with real edge devices to implement a real-world cloud-edge collaborative inference system. This may help inspire the application and deployment of inference systems based on serverless computing in practice.

IX. Conclusion

In this paper, we tame the serverless cold start in the cloud inference system with edge computing. We define a practical cold-start model which is related to multiple time slots in the past, so the long-term optimization problem is highly non-trivial to solve because of the non-convex time-coupling term. Based on the regularization and lazy-switching method, we propose a fast algorithm (Once Forward Regularization) and an extended algorithm (Window Forward Lazy-switching). We prove the competitive ratio and evaluate the algorithm performance based on a system prototype with AWS Lambda. We further verify the practical efficiency of the proposed algorithms through extensive simulation experiments driven by real-world data sets. Our proposed algorithms significantly reduce the number of cold starts and achieve lower approximate competitive ratio.

Acknowledgments

This work was supported by the National Key Research & Development (R&D) Plan under Grant 2022YFB4500004, in part by the National Science Foundation of China under Grants U20A20159, 61972432, and 62372184, the Guangdong Basic and Applied Basic Research Foundation under Grant 2021B151520008, and the Science and Technology Commission of Shanghai Municipality under Grant 22DZ2229004. It was also supported in part by the U.S. National Science Foundation under Grants CNS-2047719 and CNS-2225949.

References

[1] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[2] M. Zhang, F. Wang, Y. Zhu, J. Liu, and B. Li, "Serverless empowered video analytics for ubiquitous networked cameras," *IEEE Network*, vol. 35, no. 6, pp. 186–193, 2021.

[3] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λdnn: Achieving predictable distributed dnn training with serverless architectures," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 450–463, 2021.

[4] A. Das, A. Leaf, C. A. Varela, and S. Patterson, "Skedulix: Hybrid cloud scheduling for cost-efficient execution of serverless applications," in *Proc. of IEEE CLOUD*, 2020.

[5] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in *Proc. of ACM MMsys*, 2021, pp. 80–93.

[6] Y. Laili, F. Guo, L. Ren, X. Li, Y. Li, and L. Zhang, "Parallel scheduling of large-scale tasks for industrial cloud-edge collaboration," *IEEE Internet of Things Journal*, 2021.

[7] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold start influencing factors in function as a service," in *Proc. of IEEE/ACM UCC*, 2018.

[8] R. B. Roy, T. Patel, and D. Tiwari, "Icebreaker: warming serverless functions better with heterogeneity," in *Proc. of ACM ASPLOS*, 2022.

[9] M. Shilkov. (2021) Cold Starts in AWS Lambda. [Online]. Available: https://mikhail.io/serverless/coldstarts/aws/

[10] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *Proc. of USENIX ATC*, 2018.

[11] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective,slo-aware machine learning inference serving," in *Proc. of USENIX ATC*, 2019.

[12] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. of USENIX ATC*, 2020.

[13] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards High-Performance Serverless Computing," in *Proc. of USENIX ATC*, 2018, pp. 923–935.

[14] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *Proc. of USENIX NSDI*, 2020.

[15] A. Fuerst and P. Sharma, "Faascache: keeping serverless computing alive with greedy-dual caching," in *Proc. of ACM ASPLOS*, 2021.

[16] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proc. of ACM/IEEE SEC*, 2017.

[17] T. Elgamal, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *Proc. of ACM/IEEE SEC*, 2018.

[18] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, "Towards a serverless platform for edge AI," in *Proc. of USENIX HotEdge*, 2019.

[19] Z. Xu, L. Zhou, W. Liang, Q. Xia, W. Xu, W. Ren, H. Ren, and P. Zhou, "Stateful serverless application placement in mec with function and state dependencies," *IEEE Transactions on Computers*, 2023.

[20] Y.-H. Chiang, C. Zhu, H. Lin, and Y. Ji, "Hysteretic optimality of container warming control in serverless computing systems," *IEEE Networking Letters*, vol. 3, no. 3, pp. 138–141, 2021.

[21] P. Vahidinia, B. Farahani, and F. S. Aliee, "Mitigating cold start problem in serverless computing: A reinforcement learning approach," *IEEE Internet of Things Journal*, 2022.

[22] Q. Tang, R. Xie, F. R. Yu, T. Chen, R. Zhang, T. Huang, and Y. Liu, "Distributed task scheduling in serverless edge computing networks for the internet of things: A learning approach," *IEEE Internet of Things Journal*, 2022.

[23] L. Pan, L. Wang, S. Chen, and F. Liu, "Retention-aware container caching for serverless edge computing," *Proc. of IEEE INFOCOM*, 2022.

[24] H. Guo, Y. Wang, J. Liu, and C. Liu, "Multi-uav cooperative task offloading and resource allocation in 5g advanced and beyond," *IEEE Transactions on Wireless Communications*, 2023.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE CVPR*, 2016.

[26] "AWS Management Console," 2023. [Online]. Available: https://aws.amazon.com/console/

[27] "AWS Lambda Pricing," 2023. [Online]. Available: https://aws.amazon.com/lambda/pricing/

[28] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "Cose: Configuring serverless functions using statistical learning," in *Proc. of IEEE INFOCOM*, 2020.

[29] S. Zhang, C. Wang, Y. Jin, J. Wu, Z. Qian, M. Xiao, and S. Lu, "Adaptive configuration selection and bandwidth allocation for edge-based video analytics," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 285–298, 2021.

[30] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *Proc. of IEEE HPCA*, 2018.

[31] Alibaba Clouder. (2019) One Trillion Calls, One Billion People, and One Billion Images All in a Day. [Online]. Available: https://www.alibabacloud.com/blog/one-trillion-calls-one-billion-people-and-one-billion-images-all-in-a-day_595462

[32] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *Proc. of IEEE HPCA*, 2019.

[33] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: a first look at public edge platforms," in *Proc. of ACM IMC*, 2021.

[34] Z. Xu, H. Zhang, X. Geng, Q. Wu, and H. Ma, "Adaptive function launching acceleration in serverless computing platforms," in *Proc. of IEEE ICPADS*, 2019.

[35] (2023) Agentless observability for serverless applications. [Online]. Available: https://dashbird.io/

[36] J. Daly. (2023) A module to optimize AWS Lambda function cold starts. [Online]. Available: https://github.com/jeremydaly/lambda-warmer

[37] P. Vahidinia, B. Farahani, and F. S. Aliee, "Cold start in serverless computing: Current trends and mitigation strategies," in *Proc. of IEEE COINS*, 2020.

[38] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *Proc. of IEEE ICDCSW*, 2017.

[39] Z. Wen, Y. Wang, and F. Liu, "Stepconf: Slo-aware dynamic resource configuration for serverless function workflows," in *Proc. of IEEE INFOCOM*, 2022.

[40] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *Proc. of IEEE/ACM CCGRID*, 2020.

[41] R. Moreno Vozmediano, E. Huedo Cuesta, R. Santiago Montero, and I. Martín Llorente, "Latency and resource consumption analysis for serverless edge analytics," 2022.

[42] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs, "The internet at the speed of light," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, 2014.

[43] N. Buchbinder, S. Chen, and J. Naor, "Competitive analysis via regularization," in *Proc. of ACM/SIAM SODA*, 2014.
[44] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "Moera: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1843–1856, 2018.
[45] W. You, L. Jiao, S. Bhattacharya, and Y. Zhang, "Dynamic distributed edge resource provisioning via online learning across timescales," in *Proc. of IEEE SECON*, 2020.
[46] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.
[47] "Gurobi Optimizer," 2023. [Online]. Available: https://www.gurobi.com/
[48] P.-M. Lin and A. Glikson, "Mitigating cold starts in serverless platforms: A pool-based approach," *arXiv preprint arXiv:1903.12221*, 2019.
[49] (2023) Fast, open source serverless framework for kubernetes. [Online]. Available: https://fission.io/docs/
[50] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. Lau, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2710–2721, 2013.
[51] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[52] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.
[53] (2023) YOLOv5. [Online]. Available: https://github.com/ultralytics/yolov5
[54] ONNX Runtime developers, "ONNX Runtime," https://www.onnxruntime.ai, 2021, version: 1.10.0.
[55] (2018) Uber and Lyft Dataset Boston. [Online]. Available: https://www.kaggle.com/datasets/brllrb/uber-and-lyft-dataset-boston-ma
[56] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: a 4g LTE dataset with channel and context metrics," in *Proc. of ACM MMSys*, 2018.
[57] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proc. of IEEE CVPR*, 2017.
[58] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proc. of ECCV*, 2016.
[59] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.
[60] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. of IEEE INFOCOM*, 2020.
[61] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. of IEEE INFOCOM*, 2019.
[62] X. Qi, H. Xu, Z. Ma, and S. Chen, "Joint network selection and task offloading in mobile edge computing," in *Proc. of IEEE/ACM CCGrid*, 2021.
[63] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware vnf deployment and migration for 5g network slice," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2115–2128, 2021.

Kongyange Zhao received the B.E. degree from the South China University of Technology, China, in 2020, and the M.E. degree from the School of Computer Science and Engineering, Sun Yat-sen University (SYSU), China, in 2022. He is currently pursuing his Ph.D. degree in Sun Yat-sen University, Guangzhou, China. His research interests include edge computing, edge intelligence, and serverless computing.

Zhi Zhou received the B.S., M.E., and Ph.D. degrees in 2012, 2014, and 2017, respectively, all from the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the School of Data and Computer Science at Sun Yat-sen University, Guangzhou, China. In 2016, he was a visiting scholar at University of Göttingen. He was nominated for the 2019 China Computer Federation CCF Outstanding Doctoral Dissertation Award, the sole recipient of the 2018 ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award, and a recipient of the Best Paper Award of IEEE UIC 2018. His research interests include edge computing, cloud computing, and distributed systems.

Lei Jiao received the Ph.D. degree in computer science from the University of Göttingen, Germany. He is currently a faculty member at the Department of Computer Science, University of Oregon, USA. Previously he worked at Nokia Bell Labs in Ireland. He is interested in the mathematics of optimization, control, learning, and mechanism design applied to computer and telecommunication systems, networks, and services. He is a recipient of the NSF CAREER award. He publishes papers in journals such as JSAC, ToN, TPDS, and TMC and in conferences such as INFOCOM, MOBIHOC, ICNP, ICDCS, SECON, and IPDPS. He also received the Best Paper Awards of IEEE LANMAN 2013 and IEEE CNS 2019, and the 2016 Alcatel-Lucent Bell Labs UK and Ireland Recognition Award. He has been on the program committees of INFOCOM, MOBIHOC, ICDCS, TheWebConf, and IWQoS, and served as the program chair of multiple workshops with INFOCOM and ICDCS.

Shen Cai received the B.S. degree in electronic information engineering from the College of Electronics and Information Engineering, Shenzhen University (SZU), Shenzhen, China in 2020. He is currently pursuing the master's degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His research interests include cloud computing, serverless computing, machine learning inference system.

Fei Xu received the BS, ME, and PhD degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2007, 2009, and 2014, respectively. He received Outstanding Doctoral Dissertation Award in Hubei province, China, and ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award in 2015. He is currently an associate professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests include cloud computing and datacenter, virtualization technology, and distributed systems.

Xu Chen is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012, and worked as a Postdoctoral Research Associate at Arizona State University, Tempe, USA, from 2012 to 2014, and a Humboldt Scholar Fellow at the Institute of Computer Science of the University of Goettingen, Germany from 2014 to 2016. He received the prestigious Humboldt research fellowship awarded by the Alexander von Humboldt Foundation of Germany, 2014 Hong Kong Young Scientist Runner-up Award, 2017 IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award, 2017 IEEE ComSoc Young Professional Best Paper Award, Honorable Mention Award of 2010 IEEE international conference on Intelligence and Security Informatics (ISI), Best Paper Runner-up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and Best Paper Award of 2017 IEEE Intranational Conference on Communications (ICC). He is currently an Area Editor of the IEEE OPEN JOURNAL OF THE Communications Society, an Associate Editor of the IEEE TRANSACTIONS WIRELESS COMMUNICATIONS, IEEE INTERNET OF THINGS JOURNAL and IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC) Series on Network Softwarization and Enablers.