IEEE Open Journal of the
Computer Society

Received 10 April 2020; accepted 10 May 2020. Date of publication 21 May 2020;
date of current version 16 June 2020. The review of this paper was arranged by Guest Editors Peng Li, Amiya Nayak, and Milos Stojmenovic.

Digital Object Identifier 10.1109/OJCS.2020.2996184

FedVision: Federated Video Analytics With Edge Computing

YANG DENG [1] (Student Member, IEEE), TAO HAN [1] (Member, IEEE), AND NIRWAN ANSARI [2] (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, The University of North Carolina at Charlotte, Charlotte, NC 28105 USA
[2]Helen and John C. Hartmann Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA

CORRESPONDING AUTHOR: YANG DENG (e-mail: ydeng3@uncc.edu)

This work was supported in part by the US National Science Foundation under Grants 1731675 and 1910844.

ABSTRACT Widely deployed smart cameras are generating a large amount of video data and capable of processing frames on devices. Empowered by edge computing, the video data can also be offloaded to edge servers for processing. By leveraging the on-device processing and computation offloading, we propose a federated video analytics system named *FedVision* to efficiently provision video analytics across devices and servers. The challenge of designing *FedVision* is to optimally use the computing and networking resources for video analytics. Since there is no closed-form expression of the system performance, black-box optimization is employed to optimize the system performance. However, using black-box optimization directly incurs excessive system queries that lead to very poor system performance. To solve this problem, we design a new optimization method that integrates black-box optimization with Neural Processes (NPs) as a system performance approximator. This method allows black-box optimizer to query NPs instead of the real system. We validate the performance of *FedVision* and the new optimization method using both numerical results and experiments with a testbed.

INDEX TERMS Edge computing, black-box optimization, neural process, machine learning, video analytics.

I. INTRODUCTION

Edge computing coupled with advances in 5G is enabling a plethora of emerging applications, where various smart devices can be connected with each other via the internet and empowered with data analytics [1], [2]. In particular, a large number of smart cameras can be connected to the network and produce a huge volume of video data. To analyze these data efficiently poses pressing challenges to current networking and computing architectures. For a video analytics service, video frames can be either processed on devices or in remote servers. For frames processed on devices, the analytics accuracy is limited because the computation models, e.g., deep neural networks (DNNs), are tailored for resource-constrained devices. Offloading the frame analytics to servers can improve the accuracy, but the service latency may increase because of transmission delays in the network. Some works have discussed dynamic resource allocation in cloud computing for video analytics and optimization methods for processing video frames on devices [3], [4]. However, to our best knowledge, none of them discusses how to optimize a federated video analytics system that enable a cooperative processing of video frames.

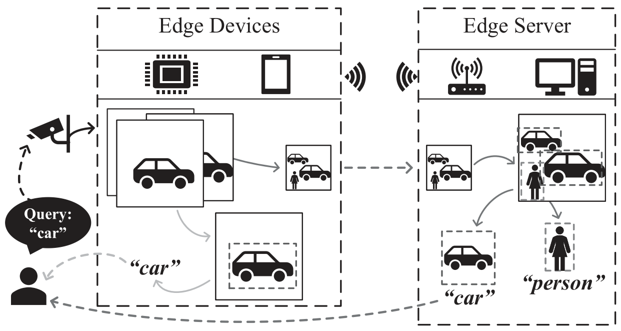

FIGURE 1. An example of video analytics in the *FedVision* System.

Fig. 1 shows an example of the federated video analytics system named *FedVision*. The user queries cars in live video streams. Live video frames are streamed from the cameras to the edge devices. Frames are sampled in the edge devices. A selection of them are processed on devices, e.g., frames containing cars are detected and returned to the user from the edge devices. The other part of sampled frames are transmitted to the edge server for object detection, i.e., detecting a car. In the life cycle of processing the video analytics query, the edge devices and the edge server are federated to provide the detection results to the user.

A series of configurations on edge devices and remote servers can be selected to improve the end-to-end performance of *FedVision*. Owing to the dynamic network conditions and computing workloads, it is almost impossible to derive a closed-form expression of the system performance for different configurations. Black-box optimization is usually adopted to solve the system optimization problem in which the system performance is considered as a black-box function [5]. The black-box optimizer (BBO) interacts with the system to find the optimal configurations. In each iteration of optimization, the optimizer generates a set of configurations within the predefined constraints as a query to the system. The system applies the configurations from the BBO and feeds the performance metrics back to the BBO. Based on the (query, feedback), the BBO approaches the optimal point gradually. However, for optimizing the end-to-end performance of *FedVision*, changing the configurations and obtaining performance metrics from the system are time consuming and expensive. Besides, some of the queries from BBO may lead to very poor system performance. Hence, it is impractical for the black-box optimizer (BBO) to query the *FedVision* system hundreds of times for the optimal configuration.

In this paper, we design the *FedVision* system and solve the end-to-end system optimization problem by a novel method that integrates black-box optimization and Neural Processes (NPs). In order to decrease the number of querying the *FedVision* system, we use NPs to approximate the system and enable BBO to interact with NPs during the system optimization. NPs are designed by using a neural networks to approximate distributions over functions [6] and it combines the benefits of neural networks and Gaussian Process (GP). NPs can learn prior knowledge of the data like GP and be trained by using the gradient descent method like neural networks. Using NPs as the system approximator, we can approximate the distribution of the system performance functions that reflect the system performance under various networking and computing conditions. Leveraging NPs as an approximator, we can eliminate the interactions between BBO and the *FedVision* system.

The contributions are summarized below.

1) We design a federated video analytics system named *FedVision* that optimally uses the computing resources on edge devices and servers to achieve efficient video analytics.
2) We develop a new end-to-end system optimization method that combines the merits of both black-box optimization and Neural Processes for efficient and safe system optimization.

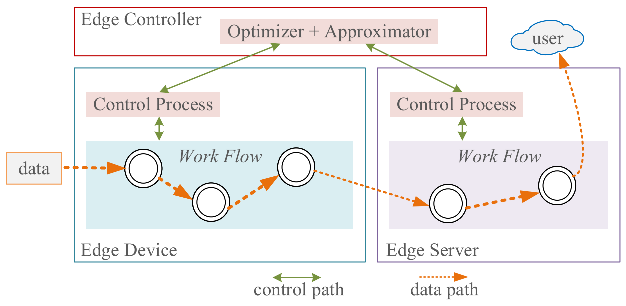

FIGURE 2. ***FedVision* system architecture.**

3) We validate the performance of *FedVision* with the proposed end-to-end system optimization method through both numerical results and experiments.

The rest of the paper is organized as follows. Section II explores the architecture of the system. Section III studies the analytical model for *FedVision* and formulates the optimization problem. Section IV provides an overview of our method. Section V explores using Neural Processes in system performance approximation. Section VI shows the optimization pipeline. Section VII discusses details of implementation of key components in the *FedVision* system. Section VIII includes numerical results and system experimental results based on our testbed. Section IX provides a brief review of related work. Section X concludes the paper.

II. SYSTEM DESCRIPTION

In this section, the architecture of *FedVision* and the specification for video analytics services are described.

A. *FedVision* ARCHITECTURE

FedVision consists of an edge controller and a series of edge devices and servers as illustrated in Fig. 2. There are two paths of data flows: data path and control path. A query from a user will be executed in the workflow across the edge device and server. A workflow consists a chain of functions such as *encode*, *decode*, *compress*, *sample* and *detect*. In the data path, video frames are executed from the first function to the last function in the workflow in sequence. After data are processed through the workflow, the output results are returned to the user.

In Fig. 2, video frames are sampled in the *sample* function. Then, the sampled frames are resized by the *compress* function and sent to the edge server. Before being sent to the edge server, the frames are encoded in *encode* in the edge device. In the edge server, after frames are received and decoded in the function *decode*, the *detect* function will detect the objects in the frames.

The control path in *FedVision* is independent of the data path. Each variable of the functions is configured by the edge controller. In the control path, every edge device and server run a control process to communicate with the edge controller.

The edge controller sends updates of the configuration variables to the edge device and the edge server via the control process. The edge controller receives the information of network status from the control process in the edge server.

B. VIDEO ANALYTICS SERVICE SPECIFICATION

For video analytics services in *FedVision*, several configuration variables can be controlled by the edge controller.

The edge controller controls the detection accuracy and latency by choosing different neural network (NN) models running in the edge device. The edge controller controls the resize rate in the *compress* function to control the size of frames transmitted in the network. The controlled variable in the *sample* function is the number of frames that are skipped from sending to the edge server after one frame is sent, e.g., if the variable equals to two, the edge device skips two frames for every frame sent to the edge server. The edge controller updates these configuration variables in the optimization pipeline.

III. PROBLEM FORMULATION

In *FedVision*, video frames are processed across the edge device and server. *FedVision* optimizes the end-to-end performance for video analytics services. In this section, we formulate the system performance in terms of the end-to-end latency and object detection accuracy. Both the latency and accuracy are viewed as functions of the configuration variables $\mathbf{x}$ and the network status l, where $\mathbf{x}$ is the vector of all available configuration variables. The system performance is evaluated in the unit of *session* which includes N video frames.

The end-to-end latency T is defined as the average latency of N frames for each session:

$$T(\mathbf{x}, l) = \frac{1}{N}\sum_{i=1}^{N}\tau_i = \frac{1}{N}\sum_{i=1}^{N}(\zeta_i + \eta_i + \theta_i) \tag{1}$$

In the equation, ζ is the processing time in the edge device; η is the network transmission latency; θ is the processing time in the edge server. For the case that the i^{th} frame is processed in the edge device, $\eta_i = 0$ and $\theta_i = 0$. Otherwise, $\zeta_i = 0$.

The object detection accuracy A is defined as the average detection accuracy of N frames for each session:

$$A(\mathbf{x}, l) = \frac{1}{N}\sum_{i=1}^{N}\alpha_i = \frac{1}{N}\sum_{i=1}^{N}(\gamma_i + \sigma_i) \tag{2}$$

γ and σ are the detection accuracy in the edge device and server, respectively. For each frame, it is processed in either the edge device or the edge server.

In *FedVision*, we aim to ensure that the system performance meets the requirements given by users. Hence, the optimization problem can be formulated as to minimize the difference between the system performance and the users' requirements.

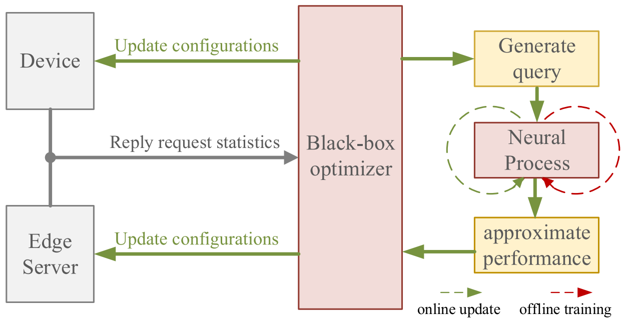

FIGURE 3. ***FedVision*** **controller architecture.**

$$\begin{aligned} \min_{\mathbf{x}} \quad & (T_{req} - T)^2 + (A_{req} - A)^2 \\ \text{s.t.} \quad & \forall i : \tau_i \geq 0, 0 \leq \alpha_i \leq 1, \\ & \gamma_i\sigma_i = 0, \zeta_i(\eta_i + \theta_i) = 0; \\ & \mathbf{x} \in \Omega, l \geq 0 \end{aligned} \tag{3}$$

IV. SOLUTION OVERVIEW

It is challenging to solve the optimization problem. First, the space of the network status l is unlimited, which makes the space of the dependent variables $(\mathbf{x}, l)$ infinite. Second, there are no analytical models to estimate the end-to-end latency and accuracy under different configurations and network status. Thus, we have no closed-form functions of $\zeta, \eta, \theta, \gamma, \sigma$ to design optimization algorithms. Third, it is intractable to query the system performance with different values of configuration variables. For most video analytics systems, it takes a long time to reconfigure the system. Besides, the new configuration may deteriorate the system performance if the values of the configuration variables are not properly chosen.

As shown in Fig. 3, *FedVision* controller consists of an approximator and optimizer running in two phases: offline training described in Section V and online updating in Section VI. In the offline training stage, we apply different configuration variables under different levels of network status to attain the system performances T and A. These configuration and performance pairs are used to train the approximator. In the online updating stage, the optimizer takes the requirements T_{req} and A_{req} in the user's query as the target for optimization. It obtains the corresponding approximate system performances $\hat{T}$ and $\hat{A}$ from the pretrained approximator and then applies $\hat{T}$ and $\hat{A}$ to evaluate the value of the objective function in Section III. *FedVision* controller can adjust the configuration variables across the edge device and edge server.

We apply black-box optimization algorithms in the optimizer. Without knowing either the closed-form expression of the system performance function, the video analytics system is viewed as a black box. We choose NPs as the approximator. NPs are trained on data under a limited number of network bandwidth status to learn the distributions of the system performance T and A. BBO can avoid querying the edge device and server by interacting with NPs.

V. SYSTEM PERFORMANCE APPROXIMATION USING NP

In this section, we discussed the challenges in approximating the system performance of the *FedVision*. We detail the design of the system performance approximator based on NPs.

A. NEURAL PROCESSES BASED APPROXIMATOR

In *FedVision*, the edge device is connected with the edge server via a wireless network. To estimate the latency T and accuracy A of *FedVision*, we use NPs to approximate the distributions of the system latency and accuracy. In designing an approximator for mapping the configurations variables $\mathbf{x}$ to the latency T and accuracy A, the challenge is the dynamic change in the network between the edge device and the edge server. When we take the network status l into consideration, the approximation of $T(\mathbf{x}, l)$ and $A(\mathbf{x}, l)$ requires the approximators to predict values of T and A under different network status because the predicted T and A for the same values of the configuration variables are different as the l changes.

One approach is to use Gaussian Processes (GPs) [7] to learn the distribution of latency T and Accuracy A. GPs provide a stochastic framework for approximating distributions by learning distributions over a series of functions. GPs learn the prior knowledge of the data via kernel functions for approximating T and A. The kernel functions are predefined. The choices available for the kernel functions limit the applicability of GPs in various video analytics systems. At the inference stage, the predicted $\hat{T}$ and $\hat{A}$ are sampled randomly over their posterior distributions. The priors are required to calculate the posteriors. However, the computation of the priors become intractable as the amount of data increases.

NPs enhance the efficiency of inference in the neural networks (NNs) without calculating the priors as GPs do. NPs can be viewed as models based on NNs framework to approximate a distribution over functions. NPs combine the benefits of NNs and GPs. In using an NP to learn the distribution, an NN is used to parameterize the stochastic process with the latent variable.

In the NP-based approximator, T and A are approximated by two NPs separately. There are three main computation modules: encoder, aggregator and decoder. In the computing schema of an NP as shown in Fig. 4, the encoder h transforms the input space $(\mathbf{x}, l, T)_i$ into the representation space r_i. The aggregator a generates a single global representation r from multiple r_i to parameterize the distribution of the latent variable z. The decoder g transforms the unknown data point $(\mathbf{x}, l)$ concatenated with the sampled z and r to obtain the prediction for T.

B. TRAINING FOR NEURAL PROCESSES

The approximators for T and A use similar training processes. Here, we take the approximator for T as the example to describe how to train an NP in *FedVision*. To train the NPs-based

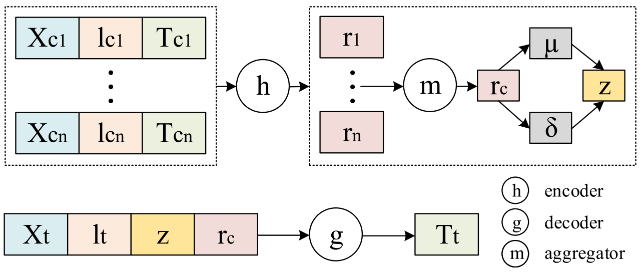

FIGURE 4. Neural Processes Computing Schema.

approximator, we first prepare the dataset. For each configuration variables vector $\mathbf{x}$, we run a session of N video frames to get the corresponding latency performance T. The dataset is collected under different network statuses in terms of low, mid and high bandwidth. In each data pair, besides $\mathbf{x}$ and T, we include l to indicate the network bandwidth status. So, the item in the data set is in the form of $(\mathbf{x}, l, T)$. The network status is dynamically changing in a continuous space and l in the training dataset is a factor that varies in the sampled discrete space. The NP is trained on a limited number of system performance data points from different distributions to infer T from $\mathbf{x}$ in a relatively wide range of the network bandwidth space of l. For prediction, l is useful to measure the distance between data points in different distributions.

The dataset is split into context points and target points: $(\mathbf{x}_c, l_c, T_c)$, $(\mathbf{x}_t, l_t, T_t)$. The latent variable z is modelled by a Gaussian Process which is parameterized via $r_c = r(\mathbf{x}_c, l_c, T_c)$. The NP models the distribution of T below.

$$P(T_t|\mathbf{x}_t, l_t, \mathbf{x}_c, l_c, T_c) = \int P(T_t|\mathbf{x}_t, l_t, z, r_c)Q(z|r_c)dz \quad (4)$$

For the neural network model of the encoder h and decoder g, the parameters are trained via variational approximation, i.e., maximizing the evidence lower bound objective (ELBO) below.

$$\begin{aligned} logP(T_t|\mathbf{x}_t, l_t, \mathbf{x}_c, l_c, T_c) \geq \mathbb{E}_{Q(z|r_c)}[logP(T_t|\mathbf{x}_t, l_t, z, r_c)] \\ - KL(Q(z|r_t)||Q(z|r_c)) \quad (5) \end{aligned}$$

The first term is the expected log-likelihood over the target points. The second is the negative Kullback-Leibler divergence between $Q(z|r_t)$ and $Q(z|r_c)$.

VI. OPTIMIZATION PIPELINE

In this section, we mainly detail the optimization pipeline in *FedVision*. Online updating, consisting of configuration generation and interaction between the optimizer and the approximators, is the main part of the optimization pipeline.

A. CONFIGURATION GENERATION

The space of configurations for the video analytics system can be large, depending on the complexity of the system. In *FedVision*, we have three configuration variables in the vector

$\mathbf{x}$ to adjust to minimize the single-objective function defined in Section III.

Black-box optimization includes derivative-free optimization methods and heuristic optimization methods. Evolutionary Algorithms (EA), or Genetic Algorithms, are popular and effective heuristic methods [8]. In the EA framework, the optimizer generates variables vector $\mathbf{x}$ to query the fitness function f_{bbo} for obtaining the value of $y_{bbo} = f_{bbo}(\mathbf{x})$. In our scenario, the fitness function is the objective function $(T_{req} - T)^2 + (A_{req} - A)^2$.

We choose (1+1) EA [9] which is the most simple evolutionary algorithm. In each iteration of (1+1) EA, it applies the replacement strategy of choosing the best between one parent and one offspring. The mutation is adaptively updated by the $1/5$ rule [10]. If $\mathbf{x}_n$ is the parent, the offspring $\hat{\mathbf{x}}_n = \mathbf{x}_n + \sigma_t N(0, I)$. If $y(\hat{\mathbf{x}}_n)$ is better than $y(\mathbf{x}_n)$, the child becomes the parent for the next iteration, i.e., $\mathbf{x}_{n+1} = \hat{\mathbf{x}}_n$. In the meantime, the mutation step size increases as $\sigma_{n+1} = 1.5\sigma_n$. Otherwise, if the offspring is worse than the parent, the mutation step size σ decreases as $\sigma_{n+1} = 1.5^{-1/4}\sigma_n$. $\mathbf{x}_n$ will continue to be the parent. After a certain number of iterations, the configuration $\mathbf{x}^* = \mathbf{argmax} f_{bbo}(\mathbf{x}, l)$ is obtained. Here, the output of the optimizer is $\mathbf{x}$ and the input to the fitness function is $(\mathbf{x}, l)$.

B. BLACK-BOX OPTIMIZER WITH NEURAL PROCESSES

We notice that, in the black-box optimization, the BBO needs to query the fitness function to search for the optimal point. However, in *FedVision*, we cannot get the exact form of the fitness function f_{bbo} because the function T and the function A are not obtainable. Instead, we use two NPs to approximate T and A respectively. In the online updating stage, we use the NP-based approximator as part of the fitness function to interact with the BBO.

The user initiates the T_{req} and A_{req} for the video query. The network status variable l is updated before the optimization starts. In each optimization iteration, the BBO initiates configuration variables vector $\mathbf{x}$. We query the prediction at the unknown point $(\mathbf{x}, l)$ in NPs for T and A, respectively. Based on the query point, the NPs can predict corresponding values of $\hat{T}$ and $\hat{A}$ with conditioning on the number of randomly chosen context points and the latent variable z. Then, the value of y_{bbo} can be calculated from $\hat{T}$ and $\hat{A}$. Once the limit of the budget is reached, the BBO outputs the current optimal configuration variables vector $\mathbf{x}^*$ and sends it to the edge devices and the edge server.

The update mechanism in the edge controller is associated with the dynamic change in the network. We have an interface between the approximator and edge server. Network status l is logged in the edge server. The edge controller fetches the network status l from the edge server periodically when *FedVision* is running. $\Delta\hat{l}$ is in a tolerable range of the disturbance of network status l. When the disturbance $\Delta l > \Delta\hat{l}$, it will trigger the online updating in the edge controller. Besides, the interaction of BBO and NPs can be triggered when the user's requirements for T_{req} and A_{req} change.

VII. IMPLEMENTATION

In this section, we discuss the implementation details of key components in *FedVision*.

Video process pipeline: we execute federated video analytics across the edge device and the edge server. The edge device and the edge server are connected through a wireless link. We use the edge device with the embedded GPU and the edge server with the standard workstation GPU. In the edge device, the processing results of video frames are put into a queue in the form of (*metadata*, *framedata*). The *metadata* contains the processing results such as frame id, frame size, processing time, accuracy and timestamp. The *framedata* is the encoded and compressed image data of each frame. The NN model running in the device is optimized to perform inference at lower precision (FP16 and INT8) by TensorRT [11]. In the edge server, the network transmission latency is calculated from the timestamps recorded in the frame metadata. The edge controller controls three configuration variables in *FedVision:* 1. NN model on the device; 2. ratio of frames in each session processed in the edge server; 3. resize ratio of the frames transmitted.

Neural Processes instances: Two NPs are implemented: one to predict latency T and the other for accuracy A. We notice the trade-off between T and A. We define the objective function to minimize the difference between the real T and the requirement T_{req} along with the difference between and the real A and the requirement A_{req}. We focus on the distribution of $T(\mathbf{x}, l)$ and $A(\mathbf{x}, l)$ separately rather than on the distribution of (T, A). So, we implement two NPs instead of one NP. For preparing training data for NPs, data are collected under different network bandwidths. We use the SQM-QoS tool in the OpenWRT to limit the download and upload speed in order to adjust the bandwidth available in the network.

Interface between NP and BBO: One BBO in the optimizer and two NPs in the approximator are implemented in the edge controller. The query point $(\mathbf{x}, l)$ includes the configuration variables vector $\mathbf{x}$ generated by the BBO and the network status l. To improve the prediction accuracy on the query point, we randomly choose 15 context points with similar network status l as the query points for inferring on the NPs. The BBO is implemented by using Nevergrad [12]. The NPs are implemented in the TensorFlow framework [13].

VIII. EVALUATION

In this section, we first evaluate the performance of the approximator and optimizer in the edge controller with the test functions and the analytical models. This initially proves the feasibility of our methodology for applying the neural process for the black-box optimizer. Then, we illustrate the details of our experiment setup in the testbed for the video analytics application in *FedVision*. Finally, we analyze the experimental results of *FedVision* in two scenarios.

A. APPROXIMATOR AND OPTIMIZER SIMULATION

In this part, to evaluate the integration of approximator and optimizer, we use three common test functions and one simplified analytical model for the edge vision system.

1) TEST FUNCTIONS

We evaluate our proposed method by a collection of commonly used test functions for the BBO. Generally, test functions are used to evaluate characteristics of optimization algorithms such as convergence rate and precision [14]. During optimization, the test function works as the fitness function to interact with the optimizer.

The first test function is Gramacy-Lee Function [15], a continuous unimodal not-convex function defined on 1-dimension space $x \in [0.5, 2.5]$.

$$f(x) = \frac{sin(10\pi x)}{2x} + (x-1)^4 \tag{6}$$

The function has one local minimum $f(x^*) = -0.86901$ where $x^* = 0.54856$.

The second test function is Ackley Function [16], a continuous multimodal not-convex function defined on 2-dimension space $x_1, x_2 \in [-2, 2]$.

$$\begin{aligned} f(x_1, x_2) &\\ &= -20\exp\left[-0.2\sqrt{0.5(x_1^2+x_2^2)}\right] \\ &\quad - \exp\left[0.5(\cos 2\pi x_1 + \cos 2\pi x_2)\right] + e + 20 \end{aligned} \tag{7}$$

The function has one global minimum $f(\mathbf{x}^*) = 0$ where $\mathbf{x}^* = (0, 0)$.

The third test function is Wolfe Function [17], a continuous multimodal differentiable not-convex function defined on 3-dimension space $x_1, x_2, x_3 \in [0, 2]$.

$$f(x_1, x_2, x_3) = \frac{4}{3}(x_1^2 + x_2^2 - x_1x_2)^{0.75} + x_3 \tag{8}$$

The global minima $f(\mathbf{x}^*) = 0$ where $\mathbf{x}^* = (0, 0, 0)$.

For each test function, the optimization problem is defined as $min\ f(\mathbf{x}),\ s.t.\ \mathbf{x} \in \Omega$.

2) ANALYSIS OF TEST FUNCTIONS RESULTS

We mainly focus on two aspects. One is how close the NP can approximate the real test function distribution. The other is comparing the NP with standard fitness functions in the black-box optimization.

Each test function is sampled randomly to build the dataset respectively. The dataset is split into the context and target set for training the neural process. We use the target negative log likelihood (NLL) to evaluate how well the NP is trained. The target NLL is defined as $-(\mathbb{E}_{Q(z|r_c)}[logP(T_t|\mathbf{x}_t, l_t, z, r_c)] - KL(Q(z|r_t)||Q(z|r_c)))$ In the upper half of Fig. 5, we plotted the target NLL of the NPs for the test functions. The orange line is the 1-D test function's target NLL while the blue line is the 2-D's. In the three figures on the lower half, we show the relation between the target NLL and how NP is approximating the test function. In these three figures, we plot the prediction on the 1-dimensional Gramacy-Lee Function as the training iterates from 0 to 80000. Ten points are selected as context points randomly while 100 test points are chosen evenly from 0.5 to 1 for plotting. The orange line in each figure is the line of the mean predicted value of the test points x. At iteration 0 which is the random start point with the error of 1.03, the predicted values of function are far away from the true values. At iteration 40000 with error of -0.47, the approximate values of test points near the context points are pretty close to the true values. When the iteration number reaches 80000, the error goes down to -1.32 and the approximate function can fit the test function very well. After 80000 iterations, the target NLL bounces around -1.3 in a small range. Once the value of the error enters the range around -1.2, the NP is considered well trained.

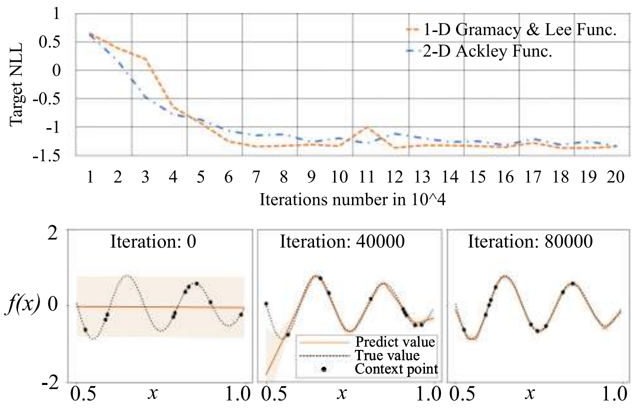

FIGURE 5. Test functions: simulation results.

TABLE 1. Test Function

	1-D Gramacy	2-D Ackley	3-D Wolfe
x^*	0.54856	(0, 0)	(0, 0, 0)
x_{func}	0.74874	(0.0, 0.0)	(0.0, 0.0, 0.0)
x_{np}	0.54592	(0.024, 0.056)	(0.065, 0.048, 0.054)
y^*	-0.86901	0	0
y_{func}	-0.66328	0.0	0.0
y_{np}	-0.86585	0.27207	0.07349

Now we test the optimization results of the BBO with the NP. Once target NLL of the NP do not vary too much, we use the trained NP to predict the value of the test function at the test point $\mathbf{x}$. In each optimization iteration, the BBO queries for a new set of configuration variables and obtains a feedback value from the test environment. $(\mathbf{x}^*, y^*)$ is the optimal point in the domain for each test function. $(\hat{\mathbf{x}}_{np}, \hat{y}_{np})$ is the optimal point obtained after the BBO queries the NP for 100 iterations. In comparison, $(\hat{\mathbf{x}}_{func}, \hat{y}_{func})$ is the optimal point obtained from interacting with the test function for 100 iterations. The results for three test functions are listed in the Table 1. For the one-dimension Gramacy function, $\mathbf{x}^* = 0.54856$ and the optimal point found by our method is $\hat{x}_{np} = 0.54592$. The difference of $\hat{x}_{np}$ and x^* is less than 0.01. For the two-dimension Ackley function, the difference of ($\hat{\mathbf{x}}_{np}$ and $\mathbf{x}^*$ is larger than the difference of $\hat{\mathbf{x}}_{func}$ and $\mathbf{x}^*$. This is partly because the start point of the BBO is the same as the optimal point of the test function. When the dimension of test functions increases to two or three, the differences become larger. This is related to the space of context points for training the NP.

3) ANALYTICAL MODELS

We build a simplified analytical model to test our proposed method. This model consists of the functions for ζ, η, θ in T and γ, σ in A.

a) Computing Performance Model: The computing performance A includes two parts. For the first part, the computing model for the edge device, $\gamma_i = f(x_i^k)$ is the function of the services x^k available in the edge device. x_i^k is the k^{th} service provided to the i^{th} application request. Here, we suppose there are more than one single service available in the edge device, which is closer to a realistic situation.

As for the edge server, the performance is relevant to certain factors. Here, we suppose only one kind of service available for one application request in the edge server. The computing performance in the edge server is defined as $\sigma_i = \lambda(z_i)g(a_i^k)$, where a_i^k is the i^{th} service available in the edge server for the k^{th} application request. We notice that not all application requests are processed in the edge server. If the edge server is involved, higher computing performance is expected. Otherwise, a satisfactory performance with a faster response can be achieved. We use a decision factor $\lambda(z_i)$ to indicate whether a request will be sent to the edge server for further processing or not. $\lambda(z_i)$ is equal to 1 or 0 in terms of z_i. The request will be processed in the edge server if $\lambda(z_i) = 1$ and zero otherwise. In this way, the edge computing system introduces flexibility in the configuration space for end-to-end optimization.

b) Computing Latency Model: Following the computing model defined in the previous part, we define the computing latency model for functions running across the edge device and the edge server. Computing latency includes both the edge device processing time and the edge server processing time. First, the edge device latency model is formulated as $\zeta_i = h(x_i^k)$, where $h(x_i^k)$ is the function of edge processing time of the i^{th} service for the k^{th} application request. Here, we model the edge server processing latency $\theta_i = j(y_i^3)$ as a function of the cube of y_i which is related to both computing performance and latency.

c) Network Transmission Latency Model: As described in the performance model, x_i and z_i have influence on the computing performance. Here in the following equation, $w(y_i^2)$ is a function of the square of y_i to represent a positively correlated function to the network transmission latency. Let $m(l_i)$ be the function of the network bandwidth which is negatively correlated to the network transmission latency. Now, we have the latency model $\eta_i = \lambda(z_i)\frac{w(y_i^2)}{m(l_i)}$ for computing and transmission.

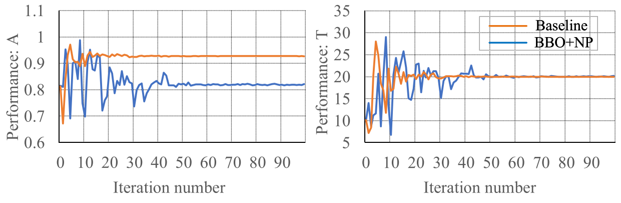

FIGURE 6. **System performance simulation results.**

4) ANALYSIS OF ANALYTICAL MODELS RESULTS

Based on the analytical model, the end-to-end latency τ_i and the computing performance α_i for a single application request are defined below.

$$\alpha_i = (1 - \lambda(z_i))f(x_i) + \lambda(z_i)g(a_i^k) \tag{9}$$

$$\tau_i = (1 - \lambda(z_i))h(x_i) + \lambda(z_i)\left(j(y_i^3) + \frac{w(y_i^2)}{m(l_i)}\right) \tag{10}$$

For this analytical model simulation, we showcase a numerical instance of τ and α. In this instance, we have $f(x) = (x - 0.5)/(4.5 - 0.5)$, $g(a_i^k) = 1$, $h(x) = 1.5x + 2.5$, $j(y) = 24y^3$, $w(y) = 5y^2$, $m(l) = 2^l$. Here, we choose the target $T_{req} = 20$ and $A_{req} = 0.8$. We formulate the problem below.

$$\begin{aligned} \min_{x,y,z} \quad & (20 - T)^2 - (0.8 - A)^2 \\ \text{s.t.} \quad & \forall i : \tau_i \geq 0, 0 \leq \alpha_i \leq 1; \\ & x \in [2.5, 4.0], y \in [0.25, 1.0], \\ & z \in [0, 3] \end{aligned} \tag{11}$$

For each session, $T = (1.5x + 2.5)(1 - 1/(z + 1)) + (24y^3 + 5y^2/2^l)/(z + 1)$, $A = (1 - 1/(z + 1))(x - 0.5)/3 + 1/(z + 1)$. We set the iteration budget to 100. We use one NP for approximating the distribution of T and the other NP for A. In each iteration, we set $l = 1$. The BBO updates the optimal point $(x, y, z)^*$ after querying the NPs for getting $(T, A)_i$ to calculate the feedback in the objective function.

To evaluate how well our method solves the optimization problem in the analytical model, we compare the performance of our method to the baseline method. In the baseline method, The value of T and A are obtained from the numerical model defined above. The value of the fitness function is calculated from the same objective function in our method.

In Fig. 6, after about 50 iterations, our method can optimize T to 20 and A to 0.8. In contrast, the baseline method can meet the requirement T_{req} after around 20 iterations but A is stuck at the point above 0.9 which is far away from the A_{req}. The trade-off is between T and A. It may be because the BBO changes y to make the first part in the objective function but A is not the function of y in the second part.

In Fig. 7, the left part (a) shows the change of the value of the objective function during the optimization. The right part (b) shows the update of the optimal point in the BBO for both methods. We notice that there is a slight increase in our method at iteration 23. This is because the prediction of the

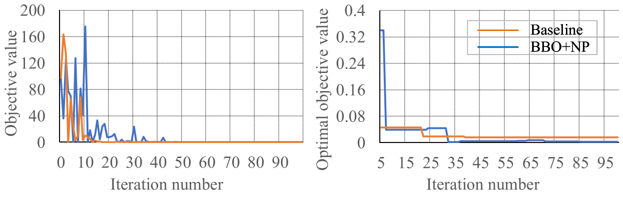

FIGURE 7. Objective function simulation results.

NP on the query point is not always accurate but varying in a small range. If the variation is within a tolerated range, the optimizer works as well as the baseline method.

B. EXPERIMENT SETUP

In the previous two parts, the performance of our method is evaluated in an analytical way. After obtaining some initial results from conducting the analytical simulation, we want to showcase the feasibility of our method in an edge-assisted video analytics system. In this part, we describe the experiment setup in our testbed for federated video analytics.

We utilize Nvidia Jetson TX2 [18] for the edge device in the system. Nvidia Jetson TX2 is available in the market with two Nvidia Pascal GPU, 128 CUDA cores each. Along with the embedded GPU, a quad-core ARM Cortex-A57 contributes multi-threading to the device alongside a dual-core Nvidia Denver 2 for high single-thread performance with dynamic code optimization. This combination of embedded GPU and high-performance CPU enables the edge device to perform at a speed of more than one TFLOP/s. The small size of the device and the capability of running DNN make it suitable in our testbed setup to deploy on-device AI processing and stream videos to the edge server for further processing.

The edge server in our setup is implemented in the Dell XPS 8930 workstation which is equipped with an Intel core i7-8700 12-Core Processor, Nvidia GeForce GTX 1080, and 32GB RAM. For the scalability and flexibility of the deployment of services, we deploy the services for video analytics in dockers. Since we need to access the GPU in the workstation for running DNN at a higher speed, we use the GPU-enabled Docker container, nvidia-docker. The nvidia-docker can limit the numbers of CPU cores and specific network interface bound while the GPU resources are not allowed to be limited in the docker. These features are convenient for running multiple different services in the same edge server separately. We initiate one nvidia-docker with 4 cores and 2.5GB memory assigned.

The edge controller is connected to a router with the edge server. The edge device is connected wirelessly through the router to the edge server and the edge controller. The time of the video frames streaming from the device to the server makes up most of the network transmission latency in *FedVision*. The network transmission latency is relevant to the data size transmitted and the available bandwidth of the link. Fig. 8(b) shows the normalized network latency of transmitting same frames under network bandwidths varying from

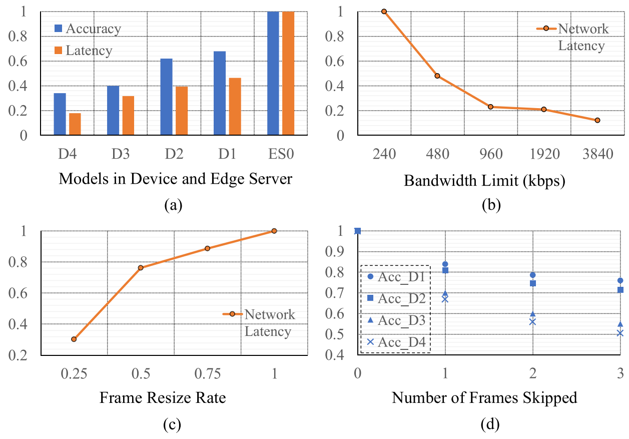

FIGURE 8. Configuration variables and network bandwidth analysis.

240 kbps to 3840 kbps. Fig. 8(c) shows the relation between the resize factor on the frame transmitted and the scaled network latency.

In a video processing system, object detection is an essential function in the video analytics services. As for the algorithms of object detection in video frames, the locations of the objects and the kinds of objects in the frame are the main benchmarks to evaluate the performance of the algorithms. Another factor is the speed of detection. The processing latency is associated with the NN model's size. We investigate four detection models for the edge device and one detection model for the edge server. In Fig. 8(a), we compare these five existing object detection models trained on coco dataset [19]. D4 and D3 are tiny-yolo-v3 model [20] with input frame size of 224×224 and 320×320, respectively. D2 is ssdlite-mobilenet-v2 with input frame size of 224×224. D1 is ssd-inception-v2 with input frame size of 300×300. In the edge server, we use the ssd-resnet-50-fpn model. The NN model running in the edge server has the higher accuracy and larger latency as compared to the models in the edge device. These three pretrained SSD models are available in the TensorFlow model zoo [21]. The accuracy shown in Fig. 8 is the mAP scaled to one. The choice of the detector has a great impact on the system's end-to-end performance. The higher the accuracy is, the bigger the latency is. Besides the native characteristics of the object detectors, several techniques are available for specialized NNs in the edge device. The main outcome of these methods is to decrease the number of frames to be processed and the processing time for each frame. As a result, the gap between accuracy and latency is eliminated. For instance, in Fig. 8(d), without considering the frame contents and other techniques, the accuracy will be deteriorated as the sample rate decreases. These methods can benefit the end-to-end performance. We consider these methods in the experiments but these are not our focus. Instead, we view all these methods as the configuration variables for the system.

In the experiments done in the following, we have three configuration variables x_1, x_2, x_3 to adjust. x_1 is the choice

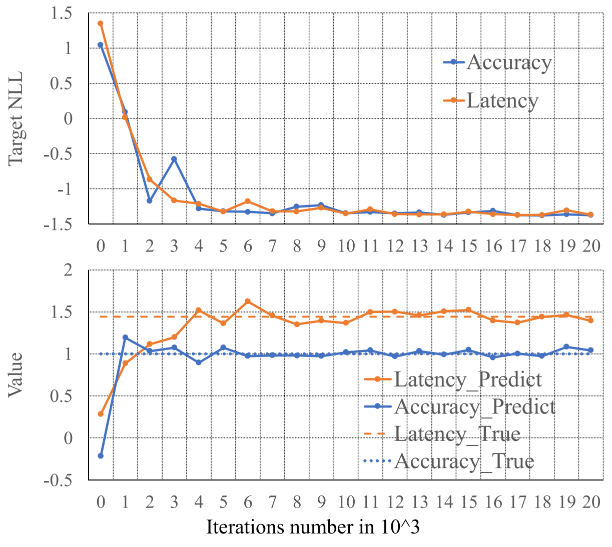

FIGURE 9. Neural Processes for latency and accuracy prediction.

of the detector on the device. x_2 is the resize rate of the frames. x_3 is the number of frames to skip for offloading. For the end-to-end latency T, both computing latency and network transmission latency are considered. For the computing performance A, we use the object detection accuracy as our metrics. The accuracy of one frame would be only determined by the on-device detector if the frame is not offloaded to the server. Otherwise, the accuracy would be determined by the accuracy in the server. We use the video dataset from [4] to detect the object, car, in the video frames.

C. ANALYSIS OF *FedVision* EXPERIMENTAL RESULTS

In our experiments, we evaluate our optimization pipeline in two cases. *(1)* Given the performance requirements by the user, *FedVision* starts with the initial configuration. We study the performance of our method for optimizing end-to-end accuracy and latency with respect to the requirements when the network status changes dramatically. *(2)* After the system starts, we study the case that our method updates the configuration variables when the user changes the performance requirements.

We train two NPs for the approximator in the edge controller. The context data $(\mathbf{X}, l, Y)$ for training NP is in the form of (x_1, x_2, x_3, l, Y). l is the bandwidth factor calculated by averaging network latency over 1800 frames. Y in the NP for approximating system accuracy is A, while the other Y for the latency is T. The dataset for training NPs is collected by processing one-minute video with different configurations in various network bandwidths.

In Fig. 9, each NP is trained for 20,000 iterations. For every 1000 iterations, the target NLL is plotted in the top half of the figure and the query point will be queried in both NPs for the corresponding latency and accuracy. We choose a configuration vector (2,1,0) with a measured bandwidth factor equal to 1.325. From the bottom half of the figure, we can tell that after

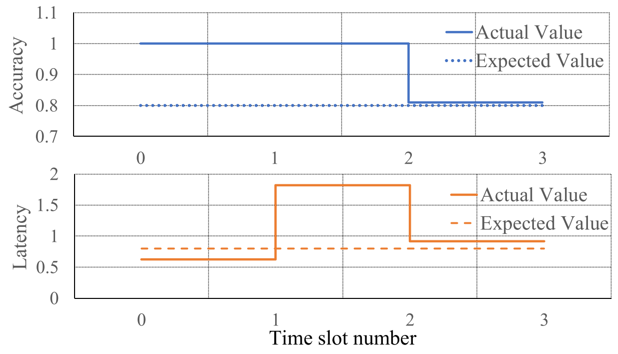

FIGURE 10. Configuration update as network status changes.

about 10,000 iterations, the values of the query become stable. Specifically, the predicted latency is bouncing around the real value at 1.4427. The predicted accuracy is close to the real number of 1.0 based on the accuracy of models implemented in the devices and the server. In this set of configuration, every frame is processed in the edge server. With the well-trained NPs, we intend to solve the problem defined in the Section III for *FedVision*.

In the first case of the our experiment, we plot the performance of accuracy and latency in four stages in Fig. 10. From stage 0, the system starts working with the performance requirement pair (T_{req}, A_{req}) in $(0.8, 0.8)$, i.e., the accuracy is supposed to be around 0.8 and latency around 0.8. At stage 0, the optimizer generates a configuration vector $(2, 0.25, 0)$ with a bandwidth factor measured at 0.5. From 0 to 1, the actual latency is 0.6233 and the accuracy is 1.0. We find the gap between the actual performance and the requirements is big. This is partially attributed to the chosen requirement being relatively relaxed for the trade-off between accuracy and latency. At stage 1, we change the network bandwidth limit for links between the edge device and the edge server. As a result, the measured bandwidth factor changes from 0.5 to 1.7. In our setup, the accuracy is irrelevant to the change of bandwidth and so it remains the same in stage 1 while the end-to-end latency surges from 0.6233 to 1.8226. This triggers the edge controller to start generating a new configuration vector to meet the performance requirement of (0.8,0.8). At stage 2, the new configuration vector, (2,1,1), is applied. Both accuracy and latency are closer to the requirements. The accuracy drops from 1.0 to 0.81 and latency drops from 1.8226 to 0.92. We see the gap becomes smaller as the network status becomes more critical to the end-to-end performance. Although 0.92 of latency is slightly larger than 0.8, it is only about half of the latency without configuration update. *FedVision* can efficiently update the system configurations for optimizing the end-to-end latency and accuracy as the network changes dramatically.

For the second case, as shown in Fig. 11, the requirement is changed from (0.8, 0.8) to (0.2,0.7) at the stage one. This use case shows how FedVision may strive for much lower latency at the slightest expense of accuracy. At stage 2, the system

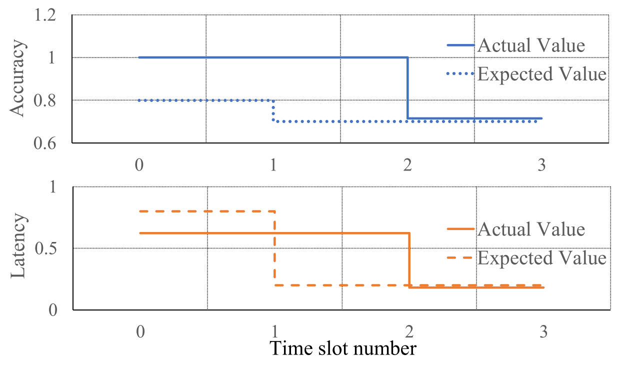

FIGURE 11. **Configuration update as performance requirements change.**

is updated to (2, 0.75, 3) with a bandwidth factor of 0.5. Then, the latency drops to 0.18 below 0.2 and the accuracy turns out to be 0.715 which is closer to the requirement 0.7. Technically, the performance is satisfactory as compared to the performance without configuration update. In this case, we require a more critical latency. We can see the new requirement can still be met by adjusting the configurations within the constraints. In other words, we can test the limit of the system by imposing more extreme requirements. When the requirement is out of the limit, our method would not meet the requirement either. Then, we can have the approximate limit of the system performance.

In this part, we have tested our optimization method in *FedVision* from two perspectives. The first is to show that our method can adapt to the network dynamics in optimizing the configurations. The second is to show that our method can optimize the configurations to meet the user's requirement change. From the results in both cases, we validate the performance of *FedVision* using our proposed optimization method.

IX. RELATED WORK

Distributed multi-camera systems have been deployed in big cities around the world. In [22], a pervasive smart camera prototype based on standard hardware and Linux software architecture was implemented. In the applications of video analytics, object detection is widely implemented as a fundamental function. The combination of the neural network partition, data compression, and differential communication was proposed and evaluated for video analytics [23]. A wide-area visual surveillance system integrated with automated video analytics ability is presented [24]. A scheme was proposed to minimize the data transmission for geo-distributed data analytics [25]. A scheduler was designed to automatically partition Neural Network computation between edge and cloud [26]. A reinforcement learning based method was used to decide when and where to migrate tasks among edge servers [27]. These works investigated the multi-camera video analytics system from system architecture perspective to the computer vision perspective. Techniques for accelerating video frames processing and tailoring neural networks for both edge devices and cloud servers are illustrated. However, they ignored the influence of the dynamic change in the network on the system end-to-end performance. We find that the dynamic changes in network is unpredictable and have direct influence on the video anlytics system performance. In *FedVision*, we utilize some of the techniques from these papers as configuration variables, and we propose a new optimization method to automatically configure the system for desired performance. We address the problem of how the configurations of the system can be adjusted based on network dynamics.

X. CONCLUSION

In this paper, we have designed the federated video analytics system named *FedVision* and developed a new method for end-to-end performance optimization in the *FedVision* system. Our method integrates Neural Processes and the Black-box optimization to optimize the latency and accuracy of queried video analytics. Our method allows the black-box optimizer to optimize the system performance via querying the NPs instead of the real edge computing system. The evaluation results show that the proposed method can adapt to network dynamics in optimizing the configuration variables and meeting the latency and accuracy requirements.

REFERENCES

[1] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.
[2] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 756–764.
[3] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. 14th USENIX Conf. Networked Syst. Des. Implementation*, 2017, pp. 377–392.
[4] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586–1597, Aug. 2017.
[5] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, ser. Mathematics and Statistics. Berlin, Germany: Springer, 2017.
[6] M. Garnelo *et al.*, "Neural processes," Jul. 2018, *arXiv:1807.01622*.
[7] C. E. Rasmussen, *Gaussian Processes in Machine Learning*, ser. In Advanced Lectures on Machine Learning. Berlin, Germany: Springer, 2004.
[8] T. Back, U. Hammel, and H. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 3–17, Apr. 1997.
[9] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theor. Comput. Sci.*, vol. 276, no. 1, pp. 51–81, 2002.
[10] M. Schumer and K. Steiglitz, "Adaptive step size random search," *IEEE Trans. Autom. Control*, vol. AC-13, no. 3, pp. 270–276, Jun. 1968.
[11] P. Wojciechowski, P. Mukherjee, and S. Sharma, "How to speed up deep learning inference using TensorRT," Nov. 2018. [Online]. Available: https://devblogs.nvidia.com/speed-up-inference-tensorrt/
[12] J. Rapin and O. Teytaud, "Nevergrad-A gradient-free optimization platform," 2018. [Online]. Available: https://GitHub.com/FacebookResearch/Nevergrad
[13] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, Nov. 2016, pp. 265–283.
[14] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," *Int. J. Math. Modell. Numer. Optim.*, vol. 4, no. 2, Aug. 2013.
[15] R. B. Gramacy and H. K. H. Lee, "Cases for the nugget in modeling computer experiments," *Statist. Comput.*, vol. 22, no. 3, pp. 713–722, 2012.

[16] T. Bck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 1–23, 1993.
[17] H. P. Schwefe, *Numerical Optimization for Computer Models*, Hoboken, NJ, USA: Wiley, 1981.
[18] D. Franklin, "NVIDIA Jetson TX2 delivers twice the intelligence to the edge," Mar. 2017. [Online]. Available: https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/
[19] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, Sep. 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/
[20] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," Apr. 2018, *arXiv:1804.02767*.
[21] TensorFlow, "Tensorflow detection model zoo," 2018. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
[22] W. Schriebl, T. Winkler, A. Starzacher, and B. Rinner, "A pervasive smart camera network architecture applied for multi-camera object classification," in *Proc. 3rd ACM/IEEE Int. Conf. Distrib. Smart Cameras*, Aug. 2009, pp. 1–8.
[23] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 2046–2049, Aug. 2018.
[24] C. Fan, Y. Wang, and C. Huang, "Heterogeneous information fusion and visualization for a large-scale intelligent video surveillance system," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 47, no. 4, pp. 593–604, Apr. 2017.
[25] A. Vulimiri *et al.*, "Wanalytics: Geo-distributed analytics for a data intensive world," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1087–1092.
[26] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 615–629.
[27] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 111–118, 2019.

YANG DENG (Student Member, IEEE) received the B.E. degree in electrical engineering from the South China University of Technology (SCUT), in 2014 and the M.E. degree in electrical engineering from the Harbin Institute of Technology (HIT), in 2016. He is a Ph.D. candidate with the Department of Electrical and Computer Engineering at the University of North Carolina at Charlotte. His research interests include machine learning, edge computing, Internet of Things, software-defined network, and smart grid.

TAO HAN (Member, IEEE) received the B.E. degree in electrical engineering from the Dalian University of Technology (DUT), China, in 2006, the M.E. degree in computer engineering from the Beijing University of Posts and Telecommunications (BUPT), China, in 2009, and the Ph.D. degree in electrical engineering from the New Jersey Institute of Technology (NJIT), in 2015. He is an Assistant Professor with the Department of Electrical and Computer Engineering at the University of North Carolina at Charlotte. Dr. Han was the recipient of IEEE International Conference on Communications (ICC) Best Paper Award 2019, IEEE Communications Society's Transmission, Access, and Optical Systems (TAOS) Best Paper Award 2019, Newark College of Engineering Outstanding Dissertation Award 2016, NJIT Hashimoto Prize 2015, and New Jersey Inventors Hall of Fame Graduate Student Award 2014. He serves as an Associate Editor of *IEEE Communications Letters* and TPC member for numerous IEEE conferences. His research interest includes mobile edge networking, machine learning, mobile X reality, 5G system, Internet of Things, and smart grid.

NIRWAN ANSARI (Fellow, IEEE), received the BSEE degree (summa cum laude with a perfect GPA) from the New Jersey Institute of Technology (NJIT). He received the Ph.D. degree from Purdue University and the MSEE degree from the University of Michigan. He is a Distinguished Professor of electrical and computer engineering at NJIT. He is also a Fellow of the National Academy of Inventors. He authored Green *Mobile Networks: A Networking Perspective* (Wiley-IEEE, 2017) with T. Han, and coauthored two other books. He has also coauthored more than 600 technical publications. He has guest-edited a number of special issues covering various emerging topics in communications and networking. His current research focuses on green communications and networking, cloud computing, drone-assisted networking, and various aspects of broadband networks. He has served on the editorial/advisory board of over ten journals including as Associate Editor-in-Chief of *IEEE Wireless Communications Magazine*. He was elected to serve in the IEEE Communications Society (ComSoc) Board of Governors as a Member-at-Large, and has chaired some ComSoc technical and steering committees, is current Director of ComSoc Educational Services Board. He has been serving on many committees such as the IEEE Fellow Committee, and has been actively organizing numerous IEEE International Conferences/Symposia/Workshops. He is frequently invited to deliver keynote addresses, distinguished lectures, tutorials, and invited talks. Some of his recognitions include several excellence in teaching awards, a few best paper awards, the NCE Excellence in Research Award, several ComSoc TC technical recognition awards, the NJ Inventors Hall of Fame Inventor of the Year Award, the Thomas Alva Edison Patent Award, Purdue University Outstanding Electrical and Computer Engineering Award, the NCE 100 Medal, and designation as a COMSOC Distinguished Lecturer. He has also been granted more than 40 U.S. patents.