Dynamic Edge-Twin Computing for Vehicle Tracking

Yuanda Wang, Shigang Chen, Ye Xia, Dimitrios Melissourgos, Haibo Wang
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, Florida
yuandawang@ufl.edu, sgchen@cise.ufl.edu, yx1@cise.ufl.edu, dmelissourgos@ufl.edu, wanghaibo@ufl.edu

***Abstract*—Internet-connected devices have been surging rapidly during the past years. Many important applications based upon such devices have emerged, such as vehicle tracking systems. These applications often require real-time execution of a large number of computation tasks. Edge computing has shown great potential in processing frequent but less-demanding tasks. Additionally, cloud computing allows for great scalability when substantial computing resources are needed. Edge-cloud computing is a paradigm that combines edge computing and cloud computing. A key problem in edge-cloud computing is how to determine the execution location for each computation task. We propose a dynamic edge-twin computing model in the context of edge-cloud computing. It uses an evaluation mechanism to predict the completion times of the task at both an edge device and a cloud server. The completion time includes data transfer time and computing time and it is determined based on real-time information about the task and the computing environment. The task will be executed by the device with a shorter completion time. We have implemented a vehicle tracking system under the edge-twin model. The experimental results show that the edge-twin model outperforms edge-alone computing and cloud-alone computing.**

I. Introduction

Edge-cloud computing is a distributed computing paradigm that utilizes computing resources of both edge devices and cloud servers [1]. Comparing with cloud-based computing, it tries to keep the execution of computation tasks at the edge devices where the tasks are produced. By doing so, it utilizes the computing resources at the edge and save network bandwidth through reduction of long-haul data transfers. The proximity to the data sources and/or to the data consumers also means that real-time data analysis and more timely responses are made possible [2]. When the edge resources are inadequate, the edge-cloud computing model allows offloading some computation tasks to the cloud servers for their powerful processing capability and huge storage space [3].

Edge-cloud computing has remarkable applications in autonomous cars, smart cities, home automation systems, etc [4] and therefore has been gaining attention in the research community. Miao et al. designed a computation offloading strategy for an edge-cloud computing environment [5]. They used artificial intelligence (AI) techniques to schedule computation tasks based on the data size and performance features of edge-cloud computing devices. Villari et al. proposed osmotic computing, which decomposes applications into microservices and deploy these microservices on both edge devices and cloud servers [6]. The above studies use historic workload data to *pre-determine* the execution locations of computation tasks. Similar strategy has been adopted by other work including [7]–[10]. We argue that at the real time when the tasks arise, if the conditions in their computing/networking/data environment are spontaneous and unpredictable, pre-determining the execution location based on historic data will not provide optimal performance because the real-time conditions may deviate from the most likely conditions suggested by the historic executions. For a couple of examples, as computation tasks arrive, the processing capacity of the edge device may vary widely depending on the background workload (e.g., with or without simultaneous arrival of other user requests), the processing capacity of the cloud may vary spontaneously (e.g., whether the purchased resources are currently used to process videos from other cameras), the available network bandwidth may vary significantly from time to time (e.g., a flush crowd of background wireless usage or fluctuation in channel conditions can greatly decrease bandwidth for a period of time), and the data size to be processed may also vary greatly (e.g., video sizes from cameras will be very different between rush hours and light traffic times for the vehicle tracking application). These conditions are unpredictable beforehand. This paper considers how to optimize edge-cloud computing by real-time dynamic determination of task execution location based on current networking/data conditions.

We propose a *dynamic edge-twin computing* model, which adopts an evaluation mechanism to determine the execution location of each computation task based on real-time information about the task and the computation environment. Before executing a computation task, the evaluation mechanism estimates the completion time of the task executed at its edge device and the time executed at its cloud-based twin with designated resources. The completion time at the twin includes data transfer time and computing time, and it is determined based on the current network bandwidth, the data size, and other factors. The task will be executed either by the edge device or by its cloud-based twin, depending on which has a shorter completion time. This paper applies the edge-twin computing model to a vehicle tracking application, which provides a platform for us to evaluate the effectiveness of this model.

Our contributions are: 1) We propose an edge-twin computing model, which selects either an edge device or a cloud-based virtual machine server (known as the twin of the edge device) to execute a computation task; 2) We design an evaluation mechanism to estimate the completion times at both the edge server and its cloud-based twin. The task will be executed at one of the two locations with a shorter completion time. The evaluation mechanism takes into account the parameters of the task and up-to-date information about the computing environment; 3) We develop a vehicle tracking system to evaluate the edge-twin computing model. We identify the key computing functions of video processing, and transform their code into separate computation tasks that can be either executed at the edge or in the cloud, depending on the result from the evaluation mechanism; 4) We conduct experiments with the vehicle tracking system and compare the performance of the edge-twin computing model with the edge-alone computing and cloud-alone computing models. We show that the edge-twin model performs much better than the traditional computing models.

II. Dynamic Edge-Twin Computing Model

A. Overview

Consider an application of tracking vehicles based on the videos captured from a set of edge devices, each equipped with a camera and local computing/storage resources. Suppose that the cameras are turned on by motion sensors and that the user may issue different requests for real-time monitoring or non-real-time vehicle recordings. Due to dynamic traffic conditions or changes in user requirements, the processing load may vary significantly over time.

Instead of designing edge devices for the maximum possible workload, which may not be even known due to uncertain future traffic conditions or user requirements, this paper adopts an edge-twin model for resource scaling and cost control at the edge. For each edge device, we allocate a VM server (also called the twin of the edge device) in the cloud. An edge device may perform some computation itself, while offloading other computation to its twin in the cloud. Whether to perform a computation task locally or in the cloud depends on many factors, including the time it takes to compute the task locally, the time it takes to compute the task in the cloud, and the time it takes to transfer the data (e.g., video) to the cloud and retrieve the results back. These factors in turn depend on the data size, the nature of the computational aspect of the task, network bandwidth, and communication delay, which may all vary over time.

The problem that we study in this paper is called *dynamic edge-twin computing*, which determines the execution locations (i.e., the edge or its cloud-based twin) of computation tasks based on real-time conditions (such as the data size and network bandwidth) in order to minimize the task completion time. We will study this problem using a vehicle tracking system that we have implemented.

B. Description of the Edge-Twin Computing Model

The edge-twin computing model determines the execution location (i.e., the edge or its cloud-based twin) of computation tasks to minimize the task completion time. An evaluation mechanism running at the edge device is used to predict the completion time of executing a task at the edge device and the completion time of executing at the corresponding cloud-based twin. More specifically, before executing a computation task, we invoke the evaluation mechanism, where the information about the computing and networking environment has been kept up-to-date, including the network bandwidth, the processing capability of the edge device, as well as the capacity of the cloud-based twin. The evaluation mechanism extracts additional relevant parameters from the task data, such as the data size. After the evaluation, the system dispatches the computation task either to the local edge device or to the cloud-based twin, depending on which one has a shorter predicted completion time. Such flexibility in execution location provides room for improving the overall system performance.

To study the edge-twin computing paradigm, we have developed a vehicle tracking system, which will be explained in detail in Section III. Here, we give a brief introduction. The purpose is to describe the edge-twin computing model in an application context. Fig. 1 illustrates the vehicle tracking system built on the edge-twin computing model. The edge devices are equipped with limited local computing/storage resources. Each edge device is connected with a camera wirelessly or by wire. The cameras capture the videos of vehicular traffic. The evaluation mechanism decides where to execute each task, and subsequently, the task is executed. The vehicle tracking system, which is executed under the edge-twin computing model, will extract the vehicle information and records it in a database that supports individual vehicle tracking.

To support the edge-twin model, we need to identify the key functions (i.e., the key methods in the code), which could be computationally demanding and thus benefit from the cloud-based twin. They are referred to as *computation tasks* under our model. We must then transform the code for each computation task such that it can be executed either at the local device or the cloud-based twin. Suppose an original computation task of the application is denoted as method E. We add new code at the beginning of E to call the evaluation mechanism which makes prediction the computation times of both local execution and remote twin execution. We also copy the code of E to the cloud-based twin as a remote procedure call. The evaluation mechanism is implemented at the edge device. Since both the edge device and its cloud-based twin has the code to implement the code E, we can select either one of them to execute the computation task. The evaluation mechanism will select the location with shorter predicted completion time so as to reduce the completion time of the computation task. If the edge device is selected for execution, the execution of method E continues to its original code for the task and the cloud-based twin is not involved.

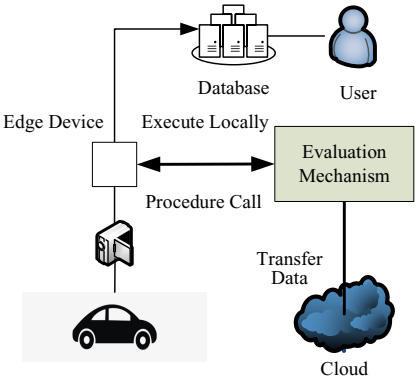

Fig. 1: Vehicle tracking system based on the edge-twin computing model.

If the cloud-based twin is selected for execution, the edge execution of method E will skip its original code, but instead establishing a TCP connection to its cloud-based twin, sending the task data (e.g., video) to the twin, and then waiting for the results to come back before proceeding (e.g., recording the identified vehicles in the local database). When the cloud-based twin receives the task request with input data (video), it will call its local copy of E to process the data and send the results back to the edge device.

III. Vehicle Tracking System

The vehicle tracking system needs to find and archive vehicle data from enormous amount of traffic videos. One traffic video is composed of many frames and the video processing tools need to check each frame to find vehicle data. The workload of processing one traffic video is already very heavy. On top of that, there is usually a large number of traffic videos captured from ubiquitous traffic cameras. As such, the tasks for video processing are regarded as the main computation tasks, which consume a majority of the computing/storage resources.

We have installed cameras at the sides of the streets on our campus for video captures. The vehicle tracking system identifies a target vehicle by the license plate number. The same vehicle can be found from various videos based on its license plate number. We mark the time, date and location of each appearance of the vehicle on a roadmap. Connecting a sequence of locations according to the time gives us a driving path of the vehicle. All the vehicle data are stored in a database connected to the edge. The user has the permission to access the database and look up the vehicle data.

The vehicle tracking system uses the computer-vision tool openCV [11] to process the videos, which reads the whole video as a stream and divides the stream into multiple frames. For each frame of the video, we use the YOLOv2 object detection system to detect vehicles [12]. YOLOv2 has many pre-trained models that help to classify different objects. YOLOv2 can detect different types of vehicles (e.g., cars, trucks, buses). When a vehicle is detected, we track the vehicle all through the video using Kernelized Correlation Filter (KCF), which is a novel tracking framework that utilizes properties of circulant matrix to implement high-speed tracking [13].

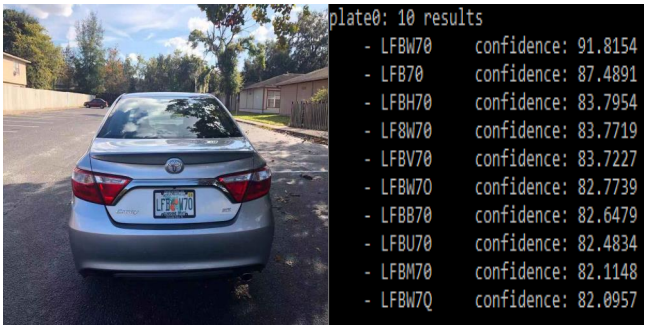

Fig. 2: License plate recognition

The system uses openALPR (automatic license plate number recognition library) to recognize the license plate number of a vehicle [14]. OpenALPR uses a pre-trained neural network model (trained by Tesseract OCR library) to detect the captured letters and numbers in one frame. It supports multiple programming languages, including C++ and Python. As shown in Fig. 2, openALPR returns ten most probable outcomes, each with a confidence level. We select the license plate number with the highest confidence level and use it as the identification of the vehicle.

The vehicle data are stored in a hierarchical database, which consists of three levels. The first level is classified according to the type of vehicles. We classify the vehicles into four types: car, bus, truck and motorcycle. The type of a vehicle can be recognized by YOLOv2. Each type of vehicles includes many different models (e.g., Toyota, Honda, Hyundai). In the second level, we classify vehicles based on different models. For that, we search the captured license plate number in other databases about vehicle history (e.g., Carfax) and detect its model. In the third level, we store the data of an individual vehicle.

IV. Vehicle Tracking System based on Edge-twin Computation Model

In our implementation of the vehicle tracking system under the edge-twin computing model, we focus on reducing the completion time of the compute-intensive tasks (computation tasks), which consume more time than other low-complexity tasks. We use the evaluation mechanism to decide where to place each computation task – at the edge device or its cloud-based twin.

A. Bulk Mode and Frame Mode

In the vehicle tracking system, we extract the video processing tasks from the application as the computation tasks. Since the user may issue different requests for real-time monitoring or non-real-time records of a vehicle, we design the vehicle tracking system to work in two modes. One is the bulk mode, which processes the whole video as a bulk and provides non-real-time vehicle data. The other is the frame mode, which selects a subset of the frames from the video to process. The frame mode can shorten the video processing time and satisfy

the requirements of real-time monitoring. Next, we explain the bulk mode, the frame mode and the evaluation mechanism in more detail.

1) Bulk Mode: The bulk mode is responsible for thoroughly processing a large number of videos without neglecting any details. This mode is useful in applications such as criminal investigation and traffic supervision, where the officials need to track suspicious vehicles or illegal vehicles from massive amount of videos. Since all details may be valuable, we should process each video frame by frame. We use the evaluation mechanism to predict the completion times at both the edge device and its cloud-based twin. The video processing task will be executed at the device with shorter predicted completion time.

2) Frame Mode: To satisfy the need for real-time monitoring, we must ensure that the video-processing throughput can keep up with the video generating rate. We employ two methods to increase the throughput. One is sampling, which is to only process one frame for every M frames. Another is a joint execution mechanism. We transfer the frames to be processed into a pipeline. Whenever either the edge device or its cloud-based twin is idle, the frame at the head of the pipeline is sent to the idle device for processing. Such a design can greatly speed up the video processing, but it may neglect some useful information due to frame omissions.

B. Evaluation Mechanism

The evaluation mechanism is used to predict the completion times of a computation task on both the edge device and its cloud-based twin. The completion time of a computation task can be divided into computing time and data transfer time [15]. The computing time is the time consumed to process the video and generate the vehicle data. The data transfer time is the time consumed to transfer the video to the computation device plus the time to retrieve the execution results. The evaluation mechanism works on one video at a time. Let S be the size of a video and γ denote an approximated ratio between the size of the task-execution results and the size of the video. Usually, larger video files contain more vehicle data [16]. One can obtain the value of γ empirically by taking the average of this ratio from past videos captured by the cameras of the tracking system. Let b be the network bandwidth, which can be measured when transferring the previous video and its results or, if the previous video was too long ago, measured periodically with test data between the edge device and its twin during the long idle time. The anticipated transfer time for the current video can be expressed as:

$$T_{tran} = (S + \gamma S)/b. \tag{1}$$

The number of machine instructions to process these data is proportional to the video size [17]. The computing time of a specific computation task is determined by the number of instructions to be executed and the computational capability of the CPU. The instructions processed by a CPU per second (IPS) can be expressed as:

$$\text{IPS} = \frac{\text{Main Frequency} \times \text{Number of Cores}}{\text{CPI}}, \tag{2}$$

where CPI denotes the average clock cycles consumed to operate an instruction. The CPU with higher IPS has more powerful computing capability.

If we have I instructions to operate for one video, the computing time can expressed as:

$$T_{calc} = \text{I}/\text{IPS}. \tag{3}$$

The number of instructions needed to process one video is generally related to the video size. We may empirically estimate a ratio between the number of instructions and the video size based on the past videos captured by the cameras.

Another simpler approach that we adopt in our experiments is based on an observation in the experiments that the computing time is generally proportional to the video size. Based on this observation, we define a parameter called the computing factor, denoted as α, which characterizes the video-processing capability of a device. Roughly speaking, it is the average amount of video data that the device can process in one unit of time. This value can be obtained empirically from the past videos. The computing time is expressed as:

$$T_{calc} = S/\alpha. \tag{4}$$

The computing factor is denoted as α_{Edge} and α_{Cloud} for the edge device and its cloud-based twin, respectively.

Combining equations (1) and (4), we have the completion time for the cloud-based twin as follows:

$$T_{\text{COM.,Cloud}} = \frac{(1+\gamma)S}{b} + \frac{S}{\alpha_{\text{Cloud}}}. \tag{5}$$

The completion time for the edge device is

$$T_{\text{COM.,Edge}} = S/\alpha_{\text{Edge}}. \tag{6}$$

The computing factors of the edge device and its cloud-based twin are fixed. The real-time parameters used by the evaluation mechanism are the network bandwidth and the video size.

V. Experiments and Evaluation

A. Experimental Setting

To evaluate the performance of the proposed edge-twin computing model, we implemented the vehicle tracking system as explained in Section III. The system is designed to take videos from one or multiple connected cameras. In our experiments, we feed it with pre-recorded street traffic videos. The edge device is a computer with Intel Core i7-2600 of 4 cores at 3.4 GHz and 32 GB RAM. Its cloud-based twin is emulated by a much more powerful server with Intel Core i7-9700, 12 MB Intel smart cache, 8 cores, and an internal GPU Intel UHD Graphics 630, which can significantly speed up video processing and license plate recognition. The peak network bandwidth between the edge device and its twin is measured at around 90Mbps. To evaluate the edge-twin operations under different network bandwidth values, we implement a software that controls the throughput between the edge and its twin to any specified value. The vehicle tracking system works on both the edge device and its cloud-based twin.

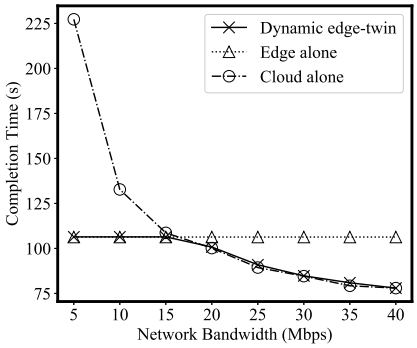

Fig. 3: Completion time for the edge-twin, cloud-alone, edge-alone computing models under different network bandwidth

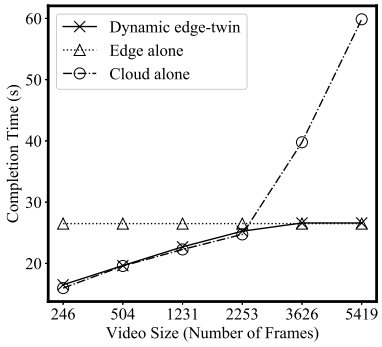

Fig. 4: Completion time for videos of different sizes. The vehicle appears in each video for one second.

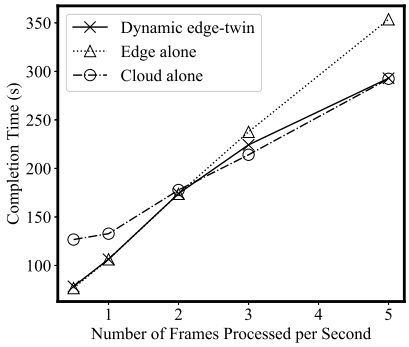

Fig. 5: Completion time for different numbers of frames to be processed

The vehicle data are stored in a database at the edge device. We compare the performance of the proposed edge-twin computing model with the edge-alone model, which performs all computation tasks at the edge device, and the cloud-alone model, which performs all computation tasks at the cloud.

B. Bulk Mode

The bulk mode processes all the frames of a video. We have empirically determined from 10 videos that the ratio between the execution results and video size, γ, is close to 5%. We have also experimented with 10 videos (which are of different sizes) and measured the computing factors as $\alpha_{\text{Cloud}} = 1.33$ MB/s and $\alpha_{\text{Edge}} = 0.78$ MB/s.

1) Network Bandwidth: The network bandwidth ranges from 1 Mbps (megabits per second) to 40 Mbps. The video size is 82.6 MB (megabytes) with 3626 frames. Fig. 3 compares the completion time of the video processing under the edge-twin computing, cloud-alone computing and edge-alone computing models.

The completion times are equal when the network bandwidth is 15.8 Mbps. Therefore, the video will be sent to the cloud-based twin when the bandwidth is above 15.8 Mbps, and it will be processed at the edge device when the bandwidth is below 15.8 Mbps.

2) Selective Processing of Videos: Sometimes, we know that the target vehicle only appears in a subset of the video frames. We only need to process the part of the video that contains the vehicle. In this section, we describe the experiments with videos of different sizes; but in all the videos the vehicle appearance time is fixed to be one second. Since the video sizes are different, the percentage of the video to be processed, which is denoted by P, varies for different videos.

The two completion times to be compared are:

$$\frac{(1+\gamma)S}{b} + \frac{SP}{\alpha_{\text{Cloud}}}, \quad \frac{SP}{\alpha_{\text{Edge}}}. \tag{7}$$

We set the network bandwidth to be 25Mbps. Fig. 4 shows that if P is high, the cloud-based twin completes the task sooner than the edge device. If P is low, the edge device completes sooner.

3) Dependency on the Number of Frames Processed: Suppose the video is captured at 30 frames per second. For some usage scenarios, there is no need to process every single frame. In the edge-twin computing model, the decision of where to execute the computation tasks may depend on the number of frames to be processed per second.

In Fig. 5, we show a scenario where the network bandwidth is 10 Mbps and the video has 3626 frames. When we need to process more frames of the video, the computing time occupies a larger portion of the completion time. Under such a circumstance, the cloud-based twin completes the computation task faster. Conversely, when we process fewer frames of the video, the transfer time to the cloud becomes significant; in that case, the edge device completes the task faster because there is no need to transfer the video.

C. Frame Mode

The frame mode is more suitable for real-time processing. It does not use the evaluation mechanism. Instead, it fully utilizes the computational resources of both the edge device and the cloud-based twin by sending the computation tasks to either one that is idle. Specifically, we transform the frame into an image. All the images to be processed wait in a pipeline. Whenever the edge device or the cloud-based twin is idle, the image at the head of the pipeline is sent to the idle device for processing. Suppose the video is captured at 30 frames per second.

1) Completion Time: We select a video of 7.8 MB with 246 frames and sample one frame to process for every 60 frames [18]. That is, $N = 0.5$ frames per second. We compare the completion time of the frame mode with that of the edge-alone computing and cloud-alone computing. To increase the performance difference between edge device and its cloud-based twin, we constrain the maximum CPU utilization of the edge device to be 10%. Thus, the edge device is much slower than the cloud-based twin. We vary the bandwidth from 1 Mbps to 10 Mbps.

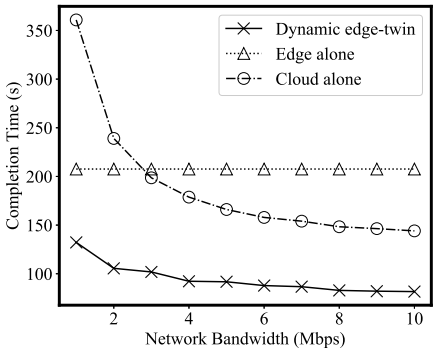

Fig. 6: Completion time under different bandwidth

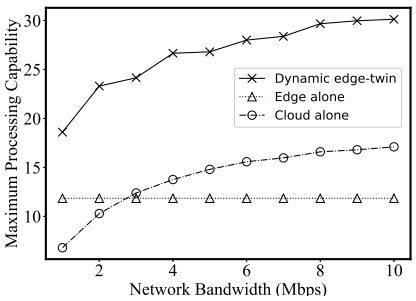

Fig. 7: Maximum number of frames that can be processed per minute

From Fig. 6, we see that the cloud-based twin helps to reduce the completion time as the network bandwidth increases. On the other hand, the edge device is not affected by the network bandwidth. The frame mode performs much better than edge-alone or cloud-alone computing and its performance improves as the network bandwidth increases.

2) Maximum Processing Capability: The maximum processing capability is the maximum number of frames that can be processed per minute. Fig. 7 compares the maximum processing capability of the three computing models. We see that the maximum processing capability of the frame mode is much higher than that of the edge-alone computing or cloud-alone computing models. The performance of the frame mode increases with the network bandwidth.

VI. Conclusion

In this paper, we describe the design and implementation of the edge-twin computing model, which utilizes the computing resources of both the edge device and its cloud-based twin. We use a vehicle tracking system as an example to demonstrate the proposed computing model. Within the edge-twin model, we have designed two video-processing modes: the bulk mode and the frame mode. The bulk mode is used for processing large amounts of videos. The frame mode is used for real-time video processing. Through experiments, we compare the performance of the edge-twin model with the edge-alone computing and cloud-alone computing models. We analyze the factors that influence the performance of the edge-twin model and we show that the edge-twin model performs much better than the traditional edge-alone computing or cloud-alone computing.

Acknowledgement

This work was supported in part by National Science Foundation of US under grant CNS-1909077 and a grant from Florida Center for Cybersecurity.

References

[1] P. Corcoran and S. K. Datta, "Mobile-edge computing and the internet of things for consumers: Extending cloud computing and services to the edge of the network," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 73–74, 2016.

[2] I. Gur, "21st-century smart cities with 5g and edge computing," 2020. [Online]. Available: https://www.saguna.net/blog/21st-century-smart-cities-with-5g-and-edge-computing/

[3] P. Miller, "What is edge computing?" [Online]. Available: https://www.theverge.com/circuitbreaker/2018/5/7/17327584/edge-computing-cloud-google-microsoft-apple-amazon\#:~:text=Edge\%20computing\%20is\%20computing\%20that's,cloud\%20is\%20coming\%20to\%20you.

[4] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[5] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, and M. S. Hossain, "Intelligent task prediction and computation offloading based on mobile-edge cloud computing," *Future Generation Computer Systems*, vol. 102, pp. 925–931, 2020.

[6] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, 2020.

[7] T. Bai, C. Pan, Y. Deng, M. Elkashlan, A. Nallanathan, and L. Hanzo, "Latency minimization for intelligent reflecting surface aided mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 11, pp. 2666–2682, 2020.

[8] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient uav-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3424–3438, 2020.

[9] Y. He, Y. Wang, C. Qiu, Q. Lin, J. Li, and Z. Ming, "Blockchain-based edge computing resource allocation in iot: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2226–2237, 2021.

[10] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1227–1240, 2020.

[11] Wikipedia contributors, "Opencv — Wikipedia, the free encyclopedia," 2021, [Online; accessed 27-April-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1011992803

[12] ——, "Object detection — Wikipedia, the free encyclopedia," 2021, [Online; accessed 27-April-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Object_detection&oldid=1010314280

[13] openCV, "Trackerkcf," https://docs.opencv.org/3.4/d2/dff/classcv_1_1TrackerKCF.html\#details, online.

[14] WikiPedia, "Openalpr," https://en.wikipedia.org/wiki/OpenALPR.

[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[16] P. Garg and P. Singh, "Video-assisted anal fistula treatment (vaaft) in cryptoglandular fistula-in-ano: A systematic review and proportional meta-analysis," *International Journal of Surgery*, vol. 46, pp. 85–91, 2017.

[17] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," *CoRR*, vol. abs/1609.08675, 2016. [Online]. Available: http://arxiv.org/abs/1609.08675

[18] O. Odegbile, S. Chen, Y. Wang, and D. Melissourgos, "Accurate hierarchical traffic measurement in datacenters through differentiated memory allocation," in *2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, 2020, pp. 122–130.