# Towards Edge-enabled Distributed Computing Framework for Heterogeneous Android-based Devices

Yongtao Yao*, Bin Liu†, Yiwei Zhao† and Weisong Shi‡
* Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
† OPPO Research USA, 2479 E Bayshore Rd #110, Palo Alto, CA 94303
‡ Department of Computer & Information Sciences, University of Delaware, Newark, DE 19716, USA
yongtaoyao@wayne.edu, {b.liu, Yiwei.Zhao}@oppo.com, weisong@udel.edu

***Abstract*—In this paper, we propose an Android-based distributed computing framework for accelerating DNN inference on Android edge devices. We experimentally demonstrate that the proposed distributed framework can reduce CPU utilization by 24% (making the the CPU utilization close to that of idle status), reduce power consumption by 59.8% to 71.8%, without leading to high-bandwidth througput. The proposed framework can be applied to various Android devices to enable cooperation among edge devices in a distributed computing manner, accelerate DNN inference, and enrich the functionality of Android devices to enhance user experience.**

***Index Terms*—edge computing, distributed computing, Android devices, smartphone, smartwatches, tablet**

## I. Introduction

With the vast improvements in computing technologies (e.g., edge computing and deep learning) and the wide deployment of communication mechanisms (e.g., 4G and 5G) over the past decades, smart home edge devices (e.g., smartphones, smartwatches, and tablets) have been intertwined with our day-to-day activities, bringing more sophisticated and attractive cutting-edge mobile applications (e.g., prediction, personalization, and recommendation) to users in small and portable devices. According to the Digital Market Outlook, the number of smart homes around the world is expected to be around 478 million in 2025 [11]. Gartner predicts that 80% of smartphones shipped by 2022 will surely have on-device AI capabilities [13], and in the meanwhile, the smartwatch takeover has begun, with the clan expanding at the rate of 20% every year [6]. In addition, over 158 million tablet units were shipped in 2021, and 51.9% of the tablet devices on the market run on Android [17]. Furthermore, Android is forecast to remain the most widely-used operating system for these smart home edge devices [2].

On the other hand, successful information technology companies have opened a path to win over and lock in customers by continuously providing the applications and experience they desire, which makes customers feel that they are always at the forefront of technical innovation. In this context, we use smart home edge devices (such as smartphones, smartwatches, and tablets) to browse the Internet, play games, attend virtual meetings, watch or edit videos, and enjoy personalized health monitoring services and many novel applications which were impossible before.

However, old-fashion devices are often not equipped with the latest hardware and sensors, resulting in limited computation and communication capabilities, which makes the novel applications difficult or even impossible to run [12]. Besides, regarding the latest edge devices, although their computing power has increased rapidly in recent years, users still complain that their mobile applications are slow to respond, especially for those applications that are compute-intensive [5]. In addition, the development of battery technology has not kept pace with the rapid iteration of smart home IoT devices and services, making energy consumption a bottleneck for AI to run on individual edge devices. These open problems call for collaborative edge inference (i.e., distributed parallel computing) among a myriad of edge devices, which is also able to effectively utilize the heterogeneous resources at each edge device for parallel model inference and therefore greatly improve user experience.

To keep up with users' demands and applications' needs for ever-increasing computational resources, we propose an Android distributed computing framework for edge deep neural network (DNN) inference acceleration. Specifically, the major contributions of this paper can be summarized as follows.

- To the best of our knowledge, this is the first work that proposes an Android distributed computing framework between smartphones, smartwatches, and tablets for smart home edge applications, especially for computation-intensive DNN models.
- We consider a myriad of metrics, including latency, CPU and memory utilization, network throughput, battery capacity, and CPU temperature, as indicators for load-balanced distributed computing.
- We introduce TensorFlow Lite into our proposed framework to support the execution of complex and sophisticated edge applications (i.e., DNN models) in a parallel computing manner. Moreover, in terms of communication mechanism, our proposed framework employs HTTP communication technology to support the transfer of high-resolution images between a set of edge devices (e.g., five smartphones and smartwatches), which is scale-friendly and satisfies the different requirements of diverse DNN models in terms of input size.
- We perform different experiments in the stand-alone setting and compare them with multiple groups of distributed computing experiments. Our experiment results validate the effectiveness of our proposed Android distributed computing framework, which is able to decrease

2022 IEEE/ACM 7th Symposium on Edge Computing (SEC) | 978-1-6654-8611-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEC54971.2022.00082

the utilization of CPU and memory, keep a reasonable CPU temperature range, and improve energy efficiency for local edge devices.

The rest of this paper is organized as follows: Sec. II reviews related work and building blocks of the proposed distributed computing framework, which are detailed in Sec. III. Extensive experimental results and related discussion are shown in Sec. IV. Sec. V concludes the entire paper.

## II. Background and Related Work

In this section, we review the evolution of distributed computing frameworks and the mainstream distributed computing frameworks for Android mobile devices. Research gaps are identified and inspire our own contributions.

To date, DNN models have been widely deployed across a variety of domains, but it is difficult to deploy DNNs on edge devices with limited computation resources. In this context, model compression technologies (e.g., parameter pruning, low-rank approximation, and knowledge transfer), as well as lightweight machine learning algorithms (e.g., MobileNets, Xception, and Squeezenet), have been proposed to tackle this challenge, but they can not guarantee to solve the problem completely when the data and machine learning models are particularly large. Another popular choice to address these limitations is employing distributed data flow systems such as MapReduce, Spark, Naiad, and XGBoost. They can robustly scale with the increasing dataset size, but when training complex DNNs, the data flow systems fail to scale as they are inefficient at executing iterative workloads [19].

This restriction sparked the development of distributed machine learning (DML) algorithms [4] to aid the implementation of complex iterative neural networks. Later on, federated learning (FL), a novel DML, was initially proposed by Google researchers [9], which is perfect to train models in a collaborative manner while preserving the privacy of sensitive data [3]. However, these frameworks either focus only on distributed computation during the model training phase or are not specifically designed for Android mobile devices.

The Berkeley Open Infrastructure for Network Computing (BOINC) is one of the leading distributed computing platforms used in a variety of fields [1]. BOINC provides clients for various operating systems, including the Android client, to allow volunteers to contribute their unused computing resources. The client provides a management function. After a user joins a computing project on the BOINC platform, the client program will automatically download a new task unit and call the corresponding project's computing program to perform the computation. The basic idea of BONIC is to make full use of computing power, and the platform currently has more than 1 million computers and has been used for voluntary computing in more than 100 research projects. In addition, two functionally similar applications, *HTC Power To Give* and *Samsung: power Sleep*, were developed based on BOINC. These applications can take full advantage of the computing power of idle CPUs while having strict limits on temperature, bandwidth usage, and energy, so the impact on actual usage is almost negligible.

ThinkAir is a popular framework that makes it simple for developers to migrate their smartphone applications to the cloud [10]. ThinkAir exploits the concept of smartphone virtualization in the cloud and provides method-level computation offloading. Advancing on previous work, it focuses on the elasticity and scalability of the cloud and enhances the power of mobile cloud computing by parallelizing method execution using multiple virtual machine (VM) images.

Later, to accelerate Android applications on low-power devices, RAPID [12] was proposed based on ThinkAir, which supports CPU and GPGPU computation offloading on Linux and Android devices. RAPID has extended ThinkAir to also support Java applications running on Linux devices. Moreover, it has added a security layer for encrypting the offloaded data to avoid eavesdropping on sensitive information by malicious users. However, it does not support parallel computing for DNN model inference, which inspires our work to bridge this research gap.

## III. Proposed Framework and Methodology

In this section, we introduce the core ideas of the proposed Android distributed computing framework. Then we describe the hardware configuration information for the experiment setup as well as the dataset and model description.

### A. Hardware Setup

In this paper, we adopt three types of smart home edge devices in total, which are shown in Fig. 1, including one Google Nexus 7 (an old-fashioned tablet), one Samsung Galaxy Watch 4, two Google Pixel 4 (smartphones), and two Huawei Nexus 6 (smartphones). Therefore, we adopt up to six mobile edge devices in this work, and both of them are equipped with Android operation systems.


Fig. 1. Three categories of hardware. Subfigure (a)-(d) shows the Google Nexus 7, Samsung Galaxy Watch, Google Pixel 4, and Huawei Nexus 6, respectively.

### B. Dataset Selection

**PDTV dataset**: The public dataset of traffic video (PDTV) [16] provides access to traffic videos of three intersections

with annotations for real transportation applications, such as tracking road users and pedestrian infractions detection (shown in Fig. 2). The video dataset was collected at three sites of Belarus and Canada with a resolution of 640 $\times$ 480 pixels at 20 and 40 frames per second, and the traffic scenes cross diverse traffic, lighting, and weather conditions.

Fig. 2. Examples of PDTV dataset.

### C. Model Selection

**You Only Look Once (YOLO)**: The YOLO series algorithm was firstly proposed in [14], and it is well known for its fast detection speed caused by simple and clear algorithm structure — YOLO formulates the object detection as a regression problem by solving a single CNN that predicts bounding boxes and category probabilities directly from images. The YOLO series algorithms have been continuously improved, and one of the most popular algorithms is YOLOv3 [15], which automatically selects the most suitable initial regression frame by incorporating the $K$-means clustering approach for a specific input dataset [7], [8]. The YOLOv3-Tiny network [18], the fastest algorithm at present, achieves high inference speed at the expense of lowered performance compared with other algorithms.

### D. Framework Description

TensorFlow Lite is adopted for DNN inference in the framework to support the execution of complex and sophisticated edge applications (*i.e.*, DNN models) in distributed computing. Besides, we adopt the HTTP communication method, which is able to solve the above two shortcomings and therefore support the communication between multiple edge devices with high-resolution images. The HTTP-based communication design is suitable for communication between a group of edge devices (e.g., four or five smartphones and smartwatches), and it is friendly for transferring high-resolution images.

Fig. 3 depicts the general description of the proposed Android distributed computing framework. Suppose a smartwatch receives a video clip collected in the vicinity of a smart home for the vehicle detection application. Let $t_k \in \mathbb{R}^{N_x \times N_y}$ denote the $k$-th video frame, $\forall k = 1, \ldots n$. Since the smartphone has limited computation capabilities, it will continuously push video frames to the surrounding mobile edge devices for vehicle detection. For example, it pushes $t_1$, $t_2$, and $t_3$ video frames to the surrounding two smartphones and one tablet respectvely, where the employed vehicle detection DNN (such as YOLOv3-Tiny or YOLOv3) is responsible for analyzing the corresponding video frames. Then, each edge device will immediately push the detection results to the smartphone. The above steps will be repeated as necessary.

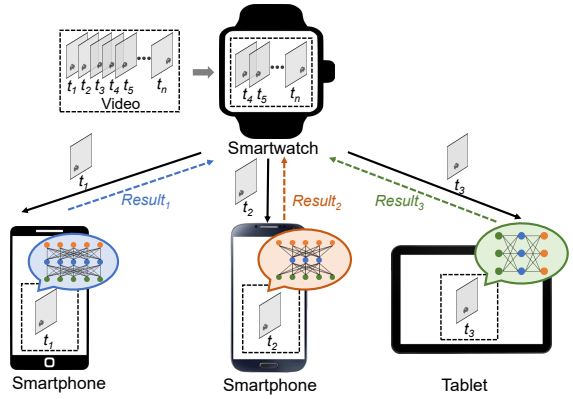


Fig. 3. The framework of the proposed Android distributed computing framework.

## IV. Experimental Design and Results Analysis

### A. Stand-alone Learning

Before employing the proposed Android distributed computing framework, we first perform both YOLOv3 and YOLOv3-Tiny models on a single Google Pixel 4 smartphone, and we term this as the stand-alone learning. The goal is to provide the baseline experiment evaluation results for the proposed Android distributed computing framework. Here, we conduct three groups of experiments to monitor the changes of several important metrics in the idle state, in the execution of the YOLOv3-Tiny model, and in the execution of the YOLOv3 model, respectively.

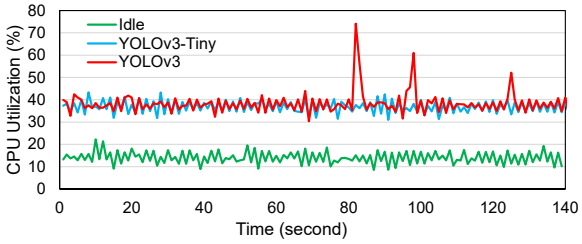


Fig. 4. The changes of CPU utilization during the stand-alone learning.

Fig. 4 shows the changes of the CPU utilization during the stand-alone learning. It can be found that the executions of both YOLOv3-Tiny and YOLOv3 models lead to a significant higher CPU utilization (around 40%) than the idle status (around 15%).

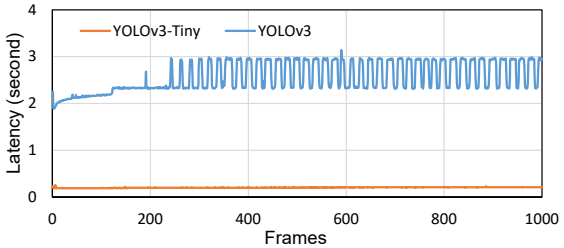


Fig. 5. The changes of inference latency during the stand-alone learning.

Similarly, Fig. 5 shows the changes of the model inference latency for YOLOv3-Tiny and YOLOv3 during the stand-alone learning. We can see that YOLOv3-Tiny takes 0.21

seconds on average to complete an inference, while YOLOv3 takes 2.8 seconds, which is much longer than the time required by YOLOv3-Tiny.

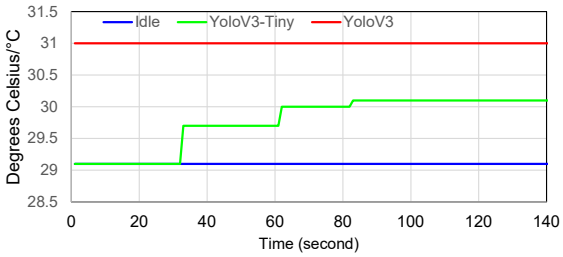


Fig. 6. The changes in CPU temperature during the stand-alone learning.

As shown in Fig. 6, the changing patterns of CPU temperature for the three groups of experiments are different. As to the idle status, the CPU temperature remains constant at around 29°C. When the YOLOv3-Tiny model is executed, the CPU temperature gradually increases, and eventually, the temperature varies slightly between 30 and 31°C.

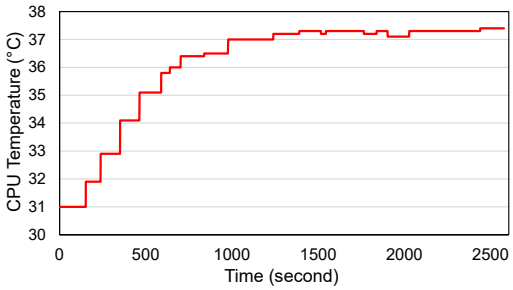


Fig. 7. The changes in CPU temperature when the YOLOv3 model runs for a period of time.

However, when the YOLOv3 model is executed, the CPU temperature initially stabilizes at around 31°C (as shown in Fig. 6) and increases rapidly as time increases (as shown in Fig. 7), with the final temperature varying slightly between 37 and 38°C. Compared to the idle state, the CPU temperature when YOLOv3 is running is increased by 9°C.

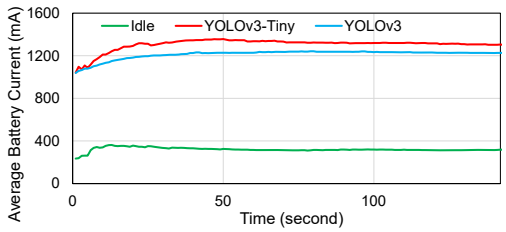


Fig. 8. The changes in the value of the average battery current (mA) during the stand-alone learning.

Fig. 8 shows changes in the value of the average battery current (mA) during the stand-alone learning. In the idle status, the average battery current is stable at 316 mA. After the execution of YOLOv3 and YOLOv3-Tiny models, the average battery current increases considerably to about 1181 mA and 1293 mA, respectively. Note that the battery is discharged in constant voltage mode with a stable voltage of 3869 mV. Hence, Fig. 8 also indirectly reflects the change in the average energy consumption during the discharging process.

### *B. Distributed Computing Experiment*

Regarding distributed computing, we conduct four groups of experiments, considering different numbers of edge devices (such as two, three, four, and five edge devices) for distributed computing, to figure out the effectiveness of the proposed framework. In this section, we only use the YOLOv3-Tiny model.

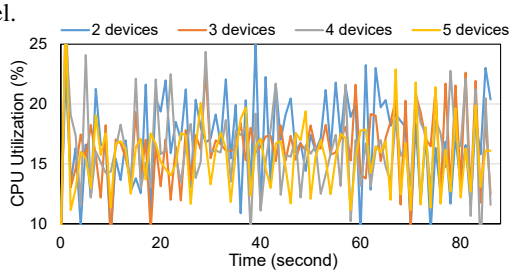


Fig. 9. The changes in the CPU utilization of the local devices during distributed computing.

Fig. 9 presents the changes in the CPU utilization for four groups of experiments. We can find that there is no significant relationship between CPU utilization and the size of the cluster. For each group, the average utilization rate of the local device (i.e., Google Pixel 4 smartphone) is around 16%, which is close to the utilization rate in the idle state and 24% lower than the utilization rate (i.e, 40%) resulting from the inference of YOLOv3-Tiny under stand-alone learning setting (as shown in Fig. 1). This is because most of the computation tasks are distributed to the surrounding devices for execution.

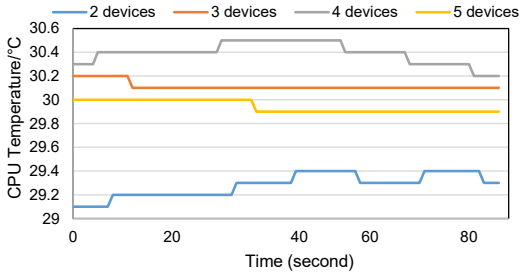


Fig. 10. The changes in the CPU temperature of the local devices during distributed computing.

Fig. 10 shows the changes in the CPU temperature for the four groups of experiments. Compared to Fig. 6, the CPU temperature differences between each of these four groups and the CPU temperature when YOLOv3-Tiny is running under the stand-alone scenario are not very large (the difference is in the range of 29°C to 30.6°C). It is worth noting that the temperature is the lowest among the four groups when there are only two devices in the cluster, while the highest temperature is obtained when the cluster contains four devices instead of five devices, which indicates that the CPU temperature of clusters is not positively correlated with the cluster size. This is because the larger the cluster size, the less computation is

performed by the local devices, but the greater the resulting bandwidth transfer.

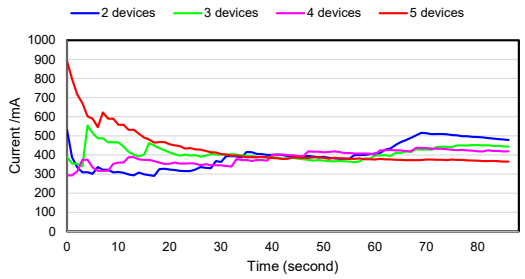


Fig. 11. The changes in the average battery current of the local devices during distributed computing.

Fig. 11 reflects the changes of the average battery current in four distributed computing experiment groups. All groups of current values eventually stabilized between 365mA and 520mA, and we can infer that the change in cluster size did not have a significant impact on the battery discharge. Besides, we can find that the distributed computing framework is able to greatly reduce the power consumption n of the local device by 59.8% to 71.8% (compared to the red curve that finally stabilizes at 1293 mA in Fig. 8).

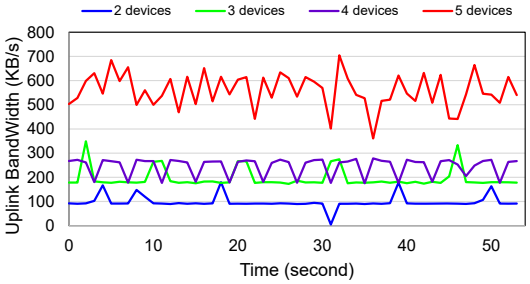


Fig. 12. The changes in the uplink bandwidth of the local devices during distributed computing.

Fig. 12 illustrates the challenges of uplink bandwidth. Clearly, as the cluster size increases, the local devices need to send more data per unit time, and the bandwidth consumption increases accordingly. In this project, the transferred image size is resized to $416 \times 416$ for DNN inference. It can be seen that with five devices, the bandwidth consumption is only 600 KB/s, which shows the scalability of the proposed distributed computing framework.

In conclusion, we validate the effectiveness of the proposed Android distributed computing framework by comparing it to stand-alone learning. The experiment results prove that the proposed framework is able to reduce CPU utilization and energy consumption of local devices without leading to significantly higher bandwidth. However, it does not have a significant impact on CPU temperature.

## V. Concluding Remarks

In this paper, we propose an Android distributed computing framework to accelerate edge DNN inference that can be applied to Android-based phones, TVs, TV boxes, watches, and tablets. In particular, smartphones are almost always working and are one of the ideal platforms for distributed computing. The experimental results demonstrated the proposed distributed computing framework can reduce the CPU utilization by 24% (making the CPU utilization of the smartphone close to the idle state); on the other hand, it can save 59.8% to 71.8% of the energy consumption. It is also demonstrated that the proposed framework will not lead to a high bandwidth footprint.

## References


[1] "Boinc projects," https://boinc.berkeley.edu/projects.php, (Accessed on 03/09/2022).

[2] T. Alsop, "Number of tablet users worldwide from 2013 to 2021 (in billions)," https://www.statista.com/statistics/377977/tablet-users-worldwide-forecast/, 2022.

[3] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[4] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, and M. Viroli, "Modelling and simulation of opportunistic IoT services with aggregate computing," *Future Generation Computer Systems*, vol. 91, pp. 252–262, 2019.

[5] X. Chen, J. Chen, B. Liu, Y. Ma, Y. Zhang, and H. Zhong, "AndroidOff: Offloading android application based on cost estimation," *Journal of Systems and Software*, vol. 158, p. 110418, 2019.

[6] Crossbeats, "What to expect from smartwatches in 2023 and beyond," https://crossbeats.com/blogs/sound-bytes/what-to-expect-from-smartwatches-in-2023-and-beyond, 2022.

[7] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2147–2154.

[8] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[9] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[11] J. Lasquety-Reyes, "Number of smart homes forecast in the world from 2017 to 2025," https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world, 2021.

[12] R. Montella, S. Kosta, D. Oro, J. Vera, C. Fernández, C. Palmieri, D. Di Luccio, G. Giunta, M. Lapegna, and G. Laccetti, "Accelerating Linux and Android applications on low-power devices through remote GPGPU offloading," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, p. e4286, 2017.

[13] Newsroom, "Gartner highlights 10 uses for ai-powered smartphones," https://www.gartner.com/en/newsroom/press-releases/2018-03-20-gartner-highlights-10-uses-for-ai-powered-smartphones, 2018.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[15] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[16] N. Saunier, H. Ardö, J.-P. Jodoin, A. Laureshyn, M. Nilsson, Å. Svensson, L. Miranda-Moreno, G.-A. Bilodeau, and K. Åström, "A public video dataset for road transportation applications," in *Transportation Research Board Annual Meeting Compendium of Papers*, 2014, pp. 14–2379.

[17] Soocial, "23 tablet statistics that are pretty surprising (2022)," https://www.soocial.com/tablet-statistics/, 2022.

[18] D. Xiao, F. Shan, Z. Li, B. T. Le, X. Liu, and X. Li, "A target detection model based on improved tiny-yolov3 under the environment of mining truck," *IEEE Access*, vol. 7, pp. 123 757–123 764, 2019.

[19] K. Zhang, S. Alqahtani, and M. Demirbas, "A comparison of distributed machine learning platforms," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.