GPU and VPU Enabled Virtual Mobile Infrastructure for 3-D Image Rendering and Its Application in Telemedicine

Zhipeng Fu, Jun Zhou, Wanpeng Xu, Changguo Guo, and Qingbo Wu

***Abstract*—Telemedicine for 3-D images on mobile devices presents promising development opportunities. Being constrained by computing power and storage capacity on mobile devices, the processing performance of 3-D medical images is insufficient for more demanding tasks. Using virtual mobile infrastructure technology to utilize cloud resources is a common solution. But it encounters the challenge of poor performance in data transmission, image rendering and image coding. This article presents a graphics processing unit (GPU) and video process unit (VPU) enabled open virtual mobile infrastructure (OpenVMI) for 3-D image rendering to solve the challenge. It makes two improvements. First, a bespoke GPU driver is developed in the Android Docker, optimizing the transmission workflow for data transmission and image rendering. Second, a VPU is added to the hardware layer to code rendered results in H.264 format, replacing CPU coding which consumes a large amount of CPU resources. By adopting the OpenVMI, the telemedicine training system proposed in this article presents an easy-to-set up, cheap and low-latency solution that is particularly helpful for telemedicine training in remote and underdeveloped areas. Performance experiments suggest that the OpenVMI delivers better performance than existing state-of-the-art systems, even in mobile devices with weaker hardware capabilities. Concurrency experiment suggests that a single host server can support up to 24 concurrent training sessions, which makes the OpenVMI very helpful for telemedicine training that demands high concurrency. The OpenVMI-based solution proposed in this article is not restricted to the use of telemedicine training, but also suitable for other application areas, such as virtual reality and augmented reality in mobile environments.**

***Index Terms*—3-D image, graphic rendering, graphics processing unit (GPU), mobile device, telemedicine, video process unit (VPU), virtual mobile infrastructure (VMI), virtual reality(VR). video process unit (VPU).**

Manuscript received 24 January 2023; revised 16 May 2023 and 23 August 2023; accepted 11 September 2023. Date of publication 18 September 2023; date of current version 21 February 2024. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166001; in part by the Major Program of Guangdong Basic and Applied Research under Grant 2019B030302002; and in part by the Major Research and Development Program of PCL, China, under Grant PCL2021A09. *(Corresponding authors: Jun Zhou; Wanpeng Xu.)*

Zhipeng Fu, Wanpeng Xu, and Qingbo Wu are with Industrial Internet of Things Research Institute, Department of New Pattern Network, Peng Cheng Laboratory, Shenzhen 518055, China (e-mail: zhipengfu518@gmail.com; Xuwanpengg@gmail.com; qingbo.wu@pcl.ac.cn).

Jun Zhou is with Industrial Internet of Things Research Institute, Department of New Pattern Network, Peng Cheng Laboratory, Shenzhen 518055, China, and also with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China (e-mail: izhoujun@163.com).

Changguo Guo is with Yuzhou Big Data Laboratory, Chongqing 400050, China, and also with the Advanced Institute of Big Data, Beijing 100195, China (e-mail: guochangguo@yzbdl.ac.cn).

Digital Object Identifier 10.1109/JIOT.2023.3316698

I. Introduction

MEDICAL resources in China are unequally distributed and skills of medical professionals in remote areas often lag behind their urban peers. Medical staff from renowned hospitals often participate in exchange programs and go on secondments in less developed areas. These solutions are temporary and they require physical traveling, which may not be feasible during difficult times, such as pandemic outbreaks.

Telemedicine solutions, such as remote consultation, are commonly used to overcome these geographical constraints [1], [2]. Facilitated by advancements in technologies, such as artificial intelligence (AI), the fifth generation of wireless networks (5G) [3] and the Internet of Things (IoT), medical staff can also engage in more advanced implementations, such as telesurgery and teleimaging [2]. Using telemedicine training as an example, the scope of training is no longer restricted to static data and images. Analysis of data of various dimensions, ranging from treatment records to complex time-varying 3-D image results also becomes feasible thanks to breakthroughs in information technology [4], [5], [6], [8] that offers large storage capacity and high-performance computing power required by advanced telemedicine solutions [9].

Additionally, doctors increasingly rely on the use of smartphones and tablets in their work for the flexibility and portability of mobile devices [10], [11], [12], [13]. As a result, there is also a growing demand for telemedicine on mobile devices [5], [6]. Despite significant improvements in computing power in recent years, the processing performance of complex data on mobile devices is still not sufficient for more demanding tasks [12], [14], [15], [16], such as 3-D scans of human organs and blood vessels in real time. The seamless display of medical 3-D images in mobile environments therefore becomes a key challenge in the development of telemedicine.

A possible workaround is the use of virtual mobile infrastructure (VMI) technology, which refers to a client-server framework with a Virtual Mobile Operating System running on a cloud-based server [17]. Users can access the virtual system remotely from their local mobile devices. The telemedicine application will be initiated in the cloud and displayed on various mobile devices via wire/wireless transmission. In this way, all the computation of the medical 3-D images will be implemented in the cloud and the mobile device is used for display and interactions only. This workaround makes use of the high-performance computing capacity and large storage

capability of the cloud servers. It provides an easy, low-cost and convenient solution for telemedicine systems that work with 3-D images in mobile environments.

But existing VMI solutions have three problems impeding the system performance.

1) There are high-transmission delays that slow down image rendering.
2) There is a lack of commercially developed graphics processing unit (GPU) drivers adapted for mobile environments to invoke cloud GPU resources directly.
3) The coding capacity of open-source drivers is not sufficient for implementing image rendering and data coding at the same time.

This article proposes the open virtual mobile infrastructure (OpenVMI), a VMI-based solution specifically designed for the display of interactive 3-D images in mobile environments. To reduce transmission delays and the consumption of CPU resources, the OpenVMI solution makes two major improvements upon typical VMI systems. This includes developing a bespoke GPUs driver to invoke GPU resources directly for rendering, eliminating multiple stages of instruction translation between OpenGL ES and OpenGL. Also, a video process unit (VPU) is added to the hardware layer to code rendered results in H.264 format, replacing CPU coding which consumes a large amount of CPU resources.

The contributions of this article are the followings.

1) An improved VMI solution specifically designed for the display of 3-D images in mobile environments, the OpenVMI, is proposed. The OpenVMI achieves better performance than existing VMI solutions by integrating the CPU, GPU, and VPU.
2) In order to reduce transmission delays, a bespoke GPU driver is developed for Android Docker to invoke GPU directly.
3) A VPU is added to replace CPU to code rendered results.
4) The OpenVMI is used in a real-life telemedicine training application.

The rest of this article is organized as follows. Section II reviews the current literature. Section III introduces the structure and workflow of the OpenVMI. The improvements made in the OpenVMI are detailed in Section IV. An implementation of the OpenVMI, the Telemedicine Training System, is introduced in Section V, followed by experiments of performance comparison and device concurrency in Section VI. Section VII discusses system features, limitations and development prospects. Conclusion is made in Section VIII.

II. Literature Review

A. Telemedicine Applications on Mobile Devices

Telemedicine application on mobile devices has been growing in popularity in recent years [18], [20], [21], [22], [23], [24], [25], [40]. They are advantageous since the mobile devices act as a portable and widely accessible health data collector to assist point-of-care (POC) diagnostics, offering an alternative to laboratory-based medical experiments [23].

Current POC applications on mobile devices cover a wide range of medical specialties. It is particularly useful during the Covid-19 pandemic for contact tracing and remote healthcare monitoring [1], [2], [26], [27], [28]. For example, Vedaei et al. [26] used an IoT health tracking node that notifies users to maintain a safe physical distance during the pandemic.

To obtain more comprehensive health data, one commonly adopted method is to wear a tracker on the human body that keeps tracking human activities and sending data to the mobile application. For example, Nornaim et al. [27] proposed an IoT-based electrocardiograph (ECG) monitoring system, enabling users to monitor their ECG signals and share data with their caretaker and physician from the mobile application. Latha et al. [20] presented the wireless body area network (WBAN), which monitors blood viscosity, blood pressure and blood sugar level in real time, enabling doctors to respond to emergencies promptly. Angelucci et al. [3] presented a continuous home telemonitoring system, which features a wearable respiratory and activity monitor, an environmental sensor and a pulse oximeter. The monitoring system sends tracked data through a fifth generation of wireless network (5G) smartphone to a multiedge computing server. Guo [29] used the smartphone to power a medical dongle that analyzes blood glucose or uric acid from a test strip.

Apart from wearing an external tracker, there are also attempts to utilize the built-in sensors and hardware in a mobile device. This approach often relies on machine learning to assist diagnosis. Lauraitis et al. [30] presented a smartphone application to examine central nervous system motor disorders in patients suffering from Huntington's, Alzheimer's, and Parkinson's diseases. A patient will be asked to touch designated positions on the screen and the trajectory data is evaluated by a back-propagation neural network classifier. Results will be used as a support for the patient's medical evaluation. Qi et al. [31] utilized the inertial sensors in a smartphone to monitor human activities. The collected data will be subsequently analyzed by AI.

The camera of a mobile device can be used to acquire medical image data. Askarian et al. [24] presented a cataract detecting approach that uses a smartphone to capture the patients' eye images. Gong et al. [32] used a smartphone to catch retinal images for teleophthalmology. Zhang et al. [33] used the smartphone to recapture the scoliosis radiograph images.

Mobile device can also be used as a voice acquirer. Hoyos-Barceló et al. [34] presented a smartphone-based cough detector that uses a smartphone as a voice catcher to acquire audio signal. Cheffena [35] developed an automated fall detection system based on audio features.

Apart from being a data acquirer, mobile devices are used as a display device. For example, the MobileHeart application supports patients with ischemic heart disease by displaying a patient's prescribed exercise programs and helping to track the patient's medication adherence [36]. Estai et al. [18], [19] developed a cloud-based store-and-forward telemedicine platform called "Remote-I," allowing the access of dental images remotely on an Android application. Similarly, Liu et al. [37]

proposed a smart dental health-IoT system that supports AI analysis of dental images in the cloud.

B. Image or Video Processing in Mobile Environments

Existing mobile telemedicine applications use mobile devices simply as a data collector and an information display because of their constraints in computing and storage capabilities. When more complicated data types, such as streaming data, are acquired via mobile devices, the task of further data processing is often delegated to desktop computers with more powerful CPUs and GPUs or cloud servers instead. For example, Guo et al. [38] attempted to improve 3-D face reconstruction by utilizing an iPhone X to capture RGB-D images. The data processing task is completed on a desktop PC with the help of GPU computing. Schwartz et al. [14] hoped to use deep learning to provide an alternative to existing image signal processor (ISP) in mobile devices. The camera image processing pipeline they proposed handles tasks, such as demosaicing, denoising, and color correction. However, their solution is desktop-based and relies on the TITAN X graphics card. In terms of video data, mobile devices may struggle even with low-level tasks. Nie et al. [39] aimed to improve the quality of videos captured by hand-held mobile devices, but the video-stitching task is not handled on mobile devices.

There is also a problem of transmitting a large amount of data. One of the solutions is to optimize the data selection process. For better data quality assessment, Korhonen [41] proposed a two-level approach that preselects videos based on low-complexity features in the first level, reducing the amount of data processing in the subsequent level. Wu et al. [12] attempted to improve the data transmission process by setting up a set of criteria for the metadata of smartphones, enabling the cloud servers to select photographs that are most useful to upload.

Apart from improving the data selection process, Minseok et al. [42] adopted the mobile ad hoc cloud technology, which connects multiple mobile devices together to create a virtual supercomputing node. An individual mobile device can thus have access to the high-processing power and large storage space on the cloud.

C. VMI Solutions

VMI technology provides another promising solution to overcome problems of limited computing power in mobile devices. There are many attempts to improve performance in VMI-based solutions.

Liu et al. [43] presented a lightweight VMI platform named cMobiDesk which employs Linux Container to build multiple Android containers by leveraging a noninvasive method to avoid modifying the source code of the mobile OS.

In order to improve the energy-efficiency ratio of VMI system. Anastasopoulos et al. [44] presented a stochastic-programming-based problem formulation that minimizes the VMI energy consumption and satisfies QOS specifications.

For communication problems between identical applications on the local device and the remote VMI server after the same apps are being installed separately, Wang et al. [45] proposed a unified application model named FUSION which classifies inter process communication (IPC) events into two types: 1) the IPC events without accessing local resources and 2) the IPC events accessing local resources.

For problems of large-scale services producing more socket system calls and greater network bridge CPU loads in the VMI system, Choi and Hong [46] proposed an improved Linux kernel-based virtual machine (KVM) hypercall scheme, which reduces the host machine's workload on data exchange, allowing the operation of more guest machines.

In order to improve VMI performance, Su et al. [10] designed a VMI-based solution named vMobiDesk, which optimizes the network transfer mechanisms for the display of virtualized data. The solution redirects users' input events and supports remote audio and camera function with low-virtualization overhead.

Existing VMI-based solutions mainly focus on improving the transmission performance of the VMI [10], [45], [46]. Studies on the rendering and processing of 3-D images in mobile environments have been scarce mainly because of the difficulties in utilizing GPU directly in mobile devices. First, GPU manufacturers have yet to provide commercial drivers for mobile environments. Therefore, most image rendering tasks are still finished on a server or PC workstation. Second, existing open-source drivers are not sufficient for implementing image rendering and image coding at the same time.

Among open-source VMI software, the popular ones include Anbox,[1] Waydroid,[2] and Robox.[3] Anbox meaning "Android in a box," runs an Android under the GNU/Linux by using the container technology. The first version of anbox was released in April 2017 and the last version in February 2023. Anbox is no longer actively developed. The limitation of anbox is that, as a desktop application, only one anbox can run under a single GNU/Linux system. It works almost like an Android emulator, and it does not support the use of the GPU on the host computer.

Waydroid, first released in September 2021, is another container-based Android emulator-like desktop VMI software under GNU/Linux. Waydroid is superior to anbox in terms of system performance and hardware compatibility. Nonetheless, Waydroid does not support the Nvidia GPU and a large number of the AMD GPUs, such as AMD RX6800.

Robox, first released in April 2018, is built upon anbox and co-developed by Huawei and Linaro,[4] the latter being an international organization that develops Arm-based software and aims to foster the Arm software ecosystem. Robox improves upon anbox by introducing extra features like Arm-supporting function and multi-instance virtualization function. Similar to anbox and waydroid, the use of the host server GPU is not supported by robox. Its commercial version, monbox, released in February 2020 by Huawei,[5] supports the use of the host GPU, but since it is proprietary, its access and testing are unavailable publicly.

[1] https://github.com/anbox/anbox
[2] https://github.com/waydroid/waydroid
[3] https://github.com/lag-linaro/robox
[4] https://www.linaro.org/
[5] https://www.huaweicloud.com/special/free-yunshouji-xsms.html

D. VDI Solutions Using GPUs

In contrast to existing VMI-based solutions that seldomly use GPU acceleration, virtual desktop infrastructure (VDI) solutions have been relying on cloud-based GPU to handle complicated rendering tasks, providing valuable insights into the use of GPU acceleration in VMI-based implementations. For example, Bentele et al. [47] summarized four approaches of virtualizing GPUs for virtual machines and presents a solution of GPU-accelerated open source VDI for OpenStack. Wan et al. [48] presented a VDI framework that invokes GPU-accelerator in the graphics hardware abstraction layer. Fornito et al. [49] proposed an infrastructure-as-code method that treats the GPU resource as software and presents a GPU-enabled VDI to provide media service in the cloud. In order to provide cheap GPU service for virtual reality (VR) and augmented reality (AR), Wu et al. [50] presented a VDI-based render farm platform that uses the VMware Horizon Client to provide 32 core vCPU, 8 GB of vGPU and 50 GB of RAM for each virtual desktop. The CMA Meteorological Observation Center [51] provides VDI system that contains NVIDIA vGPU to meet demand for 3-D modeling and CUDA computing.

Empirical studies show that GPU acceleration leads to better system performance. Li et al. [52] presented a GPU-accelerated VDI-based platform for better teaching experience on a virtual desktop. In their comparison of three virtualization technologies with or without GPU for graphics computing acceleration, the cloud service performance improved significantly by using GPU accelerator. Empirical results of another study conducted by Chang et al. [53] further supported this finding. Dong et al. [54] compared VDI capabilities on graphics processing for video playback tasks with or without GPU-virtualization. The result shows that when GPU-virtualization is enabled, VDI even with the lowest specification can deliver videos of excellent quality to end-users.

III. OpenVMI System

Inspired by the development of VDI, mobile developers are also trying to develop applications on the cloud, which has prompted the rise of VMI. The key feature of VMI is that multiple virtualized mobile operating systems, such as Android, are created in the cloud using virtualization technology. After signing in to the cloud-based virtual operating system, a mobile client device can perform the normal functions expected in a smartphone. The key difference between a virtualized cloud-based smartphone and a localized system is that the local device is used only for display and interaction in VMI-based implementation. Applications are stored and run in the cloud. Using a VMI-based solution in mobile environments has the advantages of high security, high convenience and high portability, which makes it increasingly popular in recent years [55], [56]. Inspired by this and built upon the current anbox system, we developed our own VMI software, called the OpenVMI (the OpenVMI), to solve the 3-D medical image rendering problem in telemedicine.

A. Structure of the OpenVMI

Until now, existing VMI schemes use KVM [10], [45], [46], VirtualBox [10], Xen [10], or Linux Container [43], few use Docker. This article adds multi-instance binder (the service process used for different Android processes to communicate with each other) and Ashmem (anonymous shared memory, which is used for Android system to share memory) to the Linux kernel in the cloud operating system to support the Android operating system inside the Docker container.

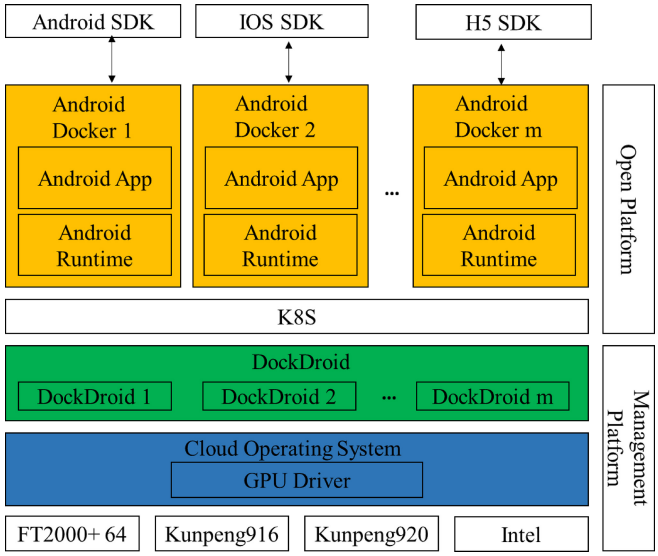


Fig. 1. Structure of the OpenVMI.

The structure of the OpenVMI system consists of six layers as illustrated in Fig. 1. From top to bottom, these are the client SDK layer, the Android Docker layer, the K8S cloud layer, the DockDroid layer, the Cloud Operating System layer and the hardware server layer. The detailed functions of the Client SDK layer, the Android Docker layer, the DockDroid layer and the Cloud Operating System layer are as follows.

1) Client SDK Layer: The OpenVMI can be accessed under Android, the iPhone IOS and smart display with HTML5(H5) system. Therefore the OpenVMI mainly provides three types of client SDK to connect to the server, including the Android SDK, IOS SDK, and H5 SDK as shown in Fig. 1. If a user uses an Android smartphone to access the OpenVMI, then the client application of the OpenVMI of Android SDK will be installed into the customer's Android smartphone.

2) Android Docker Layer: The Android Docker layer, shaded in orange in Figs. 1–3 and 5, consists of multiple Android Dockers. The main function of Android Docker is to provide an Android-like running environment, so that Android application can run in this environment. In addition, Android Docker also provides services to process different tasks like rendering, streaming, coding and displaying. Each Android Docker runs with four modules as illustrated in Fig. 2. These are the Android App module, the Basic Service module, the OpenGL ES module, and the Streaming and Coding module.

The OpenGL ES [57] module implements a subset of Open GL [58] specifically pruned for embedded/mobile system. The OpenGL ES API is a standard allowing individual and organizations to implement and import packages in the Android operating system. The OpenVMI system implements it into dynamic libGL_**.so DLLs.

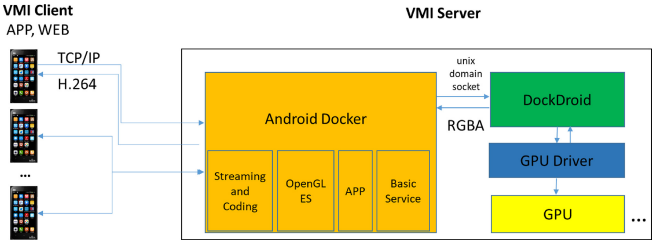
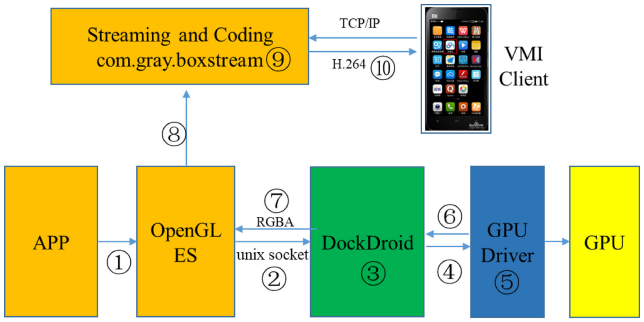


Fig. 2. Module structure of the OpenVMI.



①send the rendering request
②catch the rendering instructions and send to DockDroid
③translate the rendering instructions from OpenGL ES to OpenGL
④send the rendering instructions to GPU Driver
⑤GPU Driver execute rendering instructions into pixels in RGBA format
⑥return rendered result to DockDroid
⑦send the rendered result back to OpenGL ES module
⑧send the rendered result to Streaming and Coding module
⑨encode rendered result from RGBA to H.264 format using CPU
⑩send the encoded data to the VMI client

Fig. 3. Rendering workflow of data transmission in OpenVMI system.

The Streaming and Coding module, whose domain is com.gray.boxstream, is mainly used to capture the rendered result, encode it in H.264 format and send the encoded data to the VMI client device.

3) DockDroid Layer: The DockDroid layer, shaded in green in Figs. 1–3, consists of multiple DockDroid processes that execute in this layer. Each DockDroid process matches with one Android Docker in the Android Docker layer. This module is responsible for receiving OpenGL ES instructions, translating the instructions to OpenGL instructions, transmitting data between Android Docker and the Cloud Operating System, as well as enabling the Android application to invoke hardware resources, such as GPUs for instruction execution.

4) Cloud Operating System Layer: The Cloud Operating System layer, shaded in blue in Figs. 1–3, provides the basic software environment. Typically, a GPU driver is installed in this layer to execute various GPU computing tasks.

B. Workflow of the OpenVMI

For the typical OpenVMI system, the workflow of a rendering task involves multiple layers and multiple modules. Workflow process ①②③④⑤ in Fig. 3 shows the process of image rendering in the typical OpenVMI-based system. When an application requests for 3-D image rendering, the Android Docker will load the render request in OpenGL ES instructions and send the instructions to DockDroid. DockDroid will translate the instructions in OpenGL format and send the instructions to GPU Driver for execution.

The rendered results are pixels in RGBA format. They will be returned first back to DockDroid and subsequently back to the OpenGL ES module in Android Docker. The Streaming and Coding module then captures the rendered results frame by frame at a rate of 60 fps. It encodes the results in H.264 format and sends them to the VMI client. This is shown as workflow process ⑥-⑨ in Fig. 3.

IV. Improvements in the OpenVMI System

GPU acceleration is often used in VDI-based solutions for processing graphical data. However, there are still difficulties in using GPU accelerator directly in VMI, constraining the display of 3-D images in VMI. There are mainly three challenges.

1) Current literature demonstrates that the performance especially the transmission performance of existing VMI is not good enough for image rendering.
2) There is a lack of commercially developed GPU drivers adapted for mobile operating system, such as the Android Operating System. Few GPU manufacturers provide such adaptations. As a result, unlike VDI-based implementation, applications in Android Docker cannot invoke GPU resources directly.
3) For AMD GPU with open-source drivers, their coding capacity is not sufficient to perform image rendering and data coding at the same time.

To overcome the three challenges in the processing and rendering of 3-D images in mobile environments, two important improvements are elaborated. In order to evaluate the effectiveness of each improvement, an experiment for each improvement is conducted.

A. Direct GPU Invocation

The workflow of the typical OpenVMI system in Fig. 3 shows that the data transmission process involves multiple layers and multiple modules, including the DockDroid layer, the OpenGL ES module, and the Streaming and Coding module in the Android Docker layer. Preliminary test data of the unimproved VMI design showed a high-transmission delay, possibly due to the multiple transmission nodes among different modules in different layers. Too much data transmission might also overburden CPU.

A possible improvement could be transmitting the rendered results directly from DockDroid to the Streaming and Coding module in Android Docker, thus reducing transmission nodes involving the OpenGL ES module.

For further investigation, performance experiments of data transmission, including OpenGL ES (INCLUDED) and transmission omitting OpenGL ES (OMITTED), are performed. The parameters of interest are frame per second (fps) of the rendered results on display in a client device and CPU utilization of DockDroid. The upper limit of fps is set at 60 fps. Theoretically, the higher the value of fps and the lower the rate of CPU utilization, the more desirable a scheme is. The Huawei Kunpeng Dual-CPU Server is used, which includes

TABLE I
COMPARISON OF TWO DIFFERENT DATA TRANSMISSION SCHEMES

Experiment No.	CPU Cores [Serial Number] used by Android Docker	CPU Cores [Serial Number] used by DockDroid	Rendered result with INCLUDED scheme (fps)	Rendered result with OMITTED scheme (fps)	CPU Utilization of DockDroid with INCLUDED scheme	CPU Utilization of DockDroid with OMITTED scheme
1	2[2,3]	1[20]	42	55	52%	65%
2	2[2,3]	2[19,20]	45	57.6	56%	78.80%
3	3[3-5]	1[3]	40	50	46%	70%
4	3[3-5]	1[20]	46	59	53%	68.70%
5	3[3-5]	2[20,21]	50	55	62.90%	77%
6	4[4-7]	1[20]	50	58.4	66.80%	73.20%
7	4[4-7]	2[20,21]	51	58.2	77.50%	81.10%
8	4[4-7]	1[4]	51	59	66%	74%

48*2 cores, 512-GB RAM, 480-GB SSD, 4000-GB SATA, AMD Radeon W6800*2 GPU. The 96 CPU cores are serialized from 0 to 95. The performance parameters are tracked by Perfdog,[6] an fps performance test and analysis tool. The performance experiment is repeated separately for eight times with different number of CPU cores assigned to Android Docker and DockDroid.

The results of the experiment are detailed in Table I. The number of CPU cores assigned to Android Docker and DockDroid is detailed in column 2 and column 3, respectively. The serial number of the CPU core used is specified within the square bracket. For example, 2 [19, 20] means that two CPU cores, namely, the Number 19 core and the Number 20 core, are assigned to the process. Key observations from Table I are as follows.

1) For the INCLUDED scheme, the maximum, mean and minimum fps of the eight experiments are 51, 46.9, and 40 fps.
2) If converted to time taken to process a frame, the corresponding time per frame are 19.6, 21.3, and 25 ms for the INCLUDED scheme.
3) For the OMITTED scheme, the maximum, mean and minimum fps are 59, 56.5, 50 fps.
4) If converted to time per frame, they correspond to 16.9, 17.7, 20 ms for the OMITTED scheme.

The fps of the INCLUDED scheme is consistently lower than that of the OMITTED scheme by 8%-23% in the 8 experiments. If converted to time per frame, data transmission with OpenGL ES is slower than without OpenGL ES by 2.5–5 ms for each frame, which means each frame spends an extra 2.5–5 ms on transmission through the OpenGL ES module.

The last two columns in Table I show the CPU utilization of Dockdroid, which is used to infer CPU consumption of the OpenGL ES module, as the two are inversely related. The amount of data processing is the same for both schemes in DockDroid. Assuming the workload processed by DockDroid as 1 unit of workload, then the amount of total workload the CPU is burdened with is (1/CPU utilization of DockDroid). In experiment No.1, this corresponds to 1.92 units of workload ($1/0.52 = 1.92$) for the INCLUDED scheme, and 1.54 units of workload ($1/0.65 = 1.54$) for the OMITTED scheme. This gives a workload difference of 0.38 units. In other words, the CPU is about 25% more loaded in the INCLUDED scheme as more data transmission tasks are involved. The INCLUDED scheme consistently causes a greater amount of workload, ranging between 0.06 to 0.74 more units of workload, over the remaining seven experiments. The mean value of the INCLUDED scheme's extra workload is 0.338 units, corresponding to about 25% more CPU workload of which is consumed by the OpenGL ES module. The experiment suggests that eliminating the OpenGL ES module thus reducing the number of transmission nodes can significantly reduce latency and CPU resource consumption.

Over the eight experiments, experiment No. 3 gives the lowest fps value. This is because in the Huawei Kunpeng server, every four CPU cores are grouped as one CPU cluster. CPU core number 0–3 are grouped as one cluster and CPU core number 4–7 are grouped as another cluster and so on. CPU cores from the same cluster share one level 3 cache, whose cache access is much quicker than cache in other levels. In experiment No.3, three CPU cores, including core number 3, 4, and 5, are used by Android Docker. However, the three CPU cores come from different clusters. Core number 3 comes from one cluster whereas core number 4 and 5 come from another cluster. As a result, the three cores do not share the same level 3 cache so that fps performance is compromised. Furthermore, CPU core number 3 is shared between Android Docker and DockDroid so that it is more loaded, further undermining fps performance.

Further improvements could be directly invoking GPU resources from Android Docker. This will reduce the number of transmission nodes as data no longer passes through the DockDroid layer.

In classical design, render requests are sent to DockDroid for instruction translation from OpenGL ES format into OpenGL format that are recognizable by the GPU. After a rendering task is being finished, rendered results in RGBA format are sent by DockDroid to the Streaming and Coding module in Android Docker.

Despite being simple and easy to set up, this compromised solution not only results in higher transmission delays but also undermines CPU performance, as extra CPU resources are consumed during the translation process.

[6]https://perfdog.wetest.net/

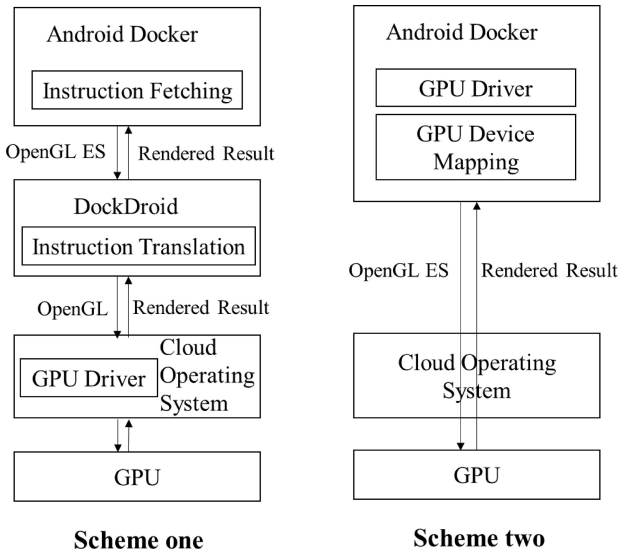


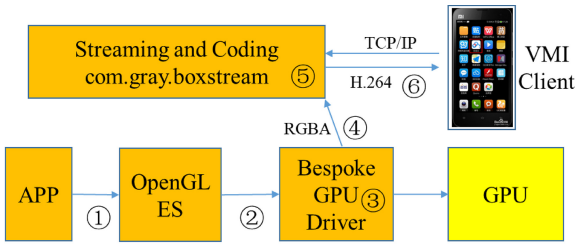
Fig. 4. Two different GPU invocation methods.

The instruction translation process is necessary due to an absence of commercially developed Android GPU drivers, preventing the Android environment from invoking GPU resources directly. As such, a bespoke GPU driver is developed and placed in Android Docker. The function of the bespoke GPU driver is that it can let the GPU recognize the OpenGL ES instruction and execute it directly in Android Docker. This helps to eliminate the need for the DockDroid module to translate the OpenGL ES instruction to OpenGL instruction. Instructions no longer need to pass through DockDroid. The classical GPU invocation method involving DockDroid is denoted as Scheme one where as the improved GPU invocation method is denoted as Scheme two in Fig. 4. The data transmission workflow of the improved OpenVMI scheme is detailed in Fig. 5. A render request goes through workflow process ①②③, and the returns of rendered results are illustrated by workflow process ④⑤⑥.

B. VPU Coding

The hardware layer is capable of executing the rendering and coding of the rendered results. However, the coding capability of the hardware is not utilized because existing open-source GPU drivers are not powerful enough to handle a coding task. Instead, CPUs are often assigned the task of data coding. Under this arrangement, rendered results in RGBA format are sent from GPU to the Streaming and Coding module in Android Docker, which encodes the data into H.264 format. This compromised solution overloads CPU significantly. Preliminary analysis indicated that more than 90% of the CPU capacity is occupied by the stream coding task.

To replace CPU coding, a VPU is added to the hardware layer in the virtual server to code RGBA data into H.264 format, freeing up CPU resources for other tasks thus improving service performance. This improved workflow is shown as workflow process ⑤ in Fig. 5.



①send the rendering request
② catch the rendering instruction and send to Bespoke GPU Driver
③execute rendering instructions into pixels in RGBA format
④ return the rendered result to Streaming and Coding module
⑤ encode rendered result from RGBA to H.264 format using VPU
⑥send the encoded data to the VMI client

Fig. 5. Improved rendering workflow after direct GPU invocation and VPU coding.

TABLE II
COMPARISON OF CPU CODING AND VPU CODING

Server type	GPU type	Coding type	CPU utilization	GPU utilization
Phytium 2000+ (64 core)	Tesla T4	CPU coding	165%	32%
Phytium 2000+ (64 core)	Tesla T4	VPU coding	23%	19%
Kunpeng 920 (48 core* 2)	AMD WX5100	CPU coding	108%	3.3%
Kunpeng 920 (48 core* 2)	AMD WX5100	VPU coding	9%	3.0%

In order to compare the system performance between CPU coding and VPU coding, an experiment of CPU and GPU utilization with respect to different types of coding and different server specifications is conducted. Two types of servers are used, they are the Phytium server which has 64 cores in one CPU and the Huawei Kunpeng dual-CPU server which has 48*2 cores. The CPU utilization is measured in terms of utilization of a single CPU core. In Table II, the CPU utilization reaches 165% for the Phytium 2000+ server and 108% for the Kunpeng 920 server if the coding task is executed by the CPU, meaning that the CPU coding task consumes more than one CPU core. In contrast, VPU coding frees up significant CPU resources so that its CPU utilization is 7–12 times lower than CPU coding. The results of the experiment suggest that adopting VPU coding has reduced CPU consumption and improved system performance significantly.

After the two improvements, the workflow of the improved OpenVMI is illustrated as Fig. 5.

V. OPENVMI-BASED TELEMEDICINE TRAINING SYSTEM

The OpenVMI system is deployed in a real-life implementation, the Telemedicine Training System, which is designed to live-stream telemedicine training for analyzing medical 3-D images on mobile devices. The system supports low-latency rendering of human bones, blood vessels and organs. It also supports interactive functions, such as movements, rotations, and scaling of medical images.

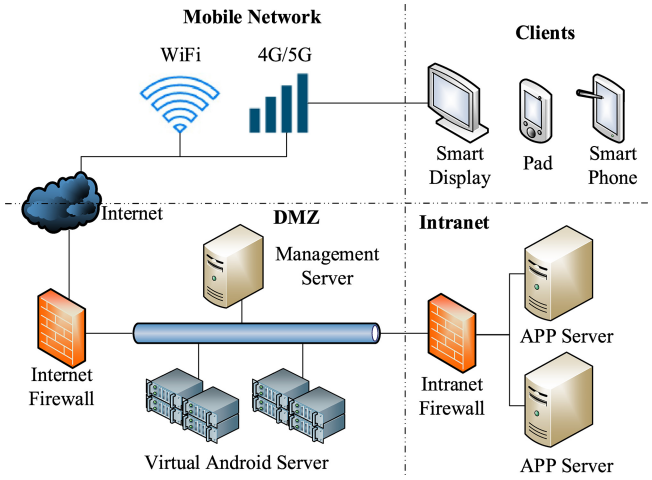


Fig. 6. Topological structure of the telemedicine training system.

A. Topological Structure of the System

The topological structure of the Telemedicine Training System is shown in Fig. 6. It consists of an Intranet zone, a demilitarized zone (DMZ), a mobile network zone and multiple mobile clients. Unlike other telemedicine training applications that connect the mobile devices directly to a cloud server [25], [59], a DMZ is added to the Telemedicine Training System for the virtualization of Android devices in the cloud server. The training system is a layered structure, rather than a mesh-like structure that integrates multiple applications, such as the system in Attila et al. [60] that integrated the interconnection telemedicine systems, hospital information systems, legacy health care systems, smart health devices, and health-related smartphone-apps into a unified service architecture.

The Intranet zone is where the servers of the training application are located. Medical data of different types, such as clinical records, CT/PET-CT imaging results, and MRI results, is stored here.

The DMZ is mainly composed of cloud management servers and virtual Android servers. The module structure of each virtual Android server is as detailed in Fig. 1.

Clients refer to various mobile client devices that have the VMI client application installed to access the training application by connecting to the Mobile Network zone. They can be smartphones, tablets, and smart displays. Each VMI training service can connect to multiple VMI clients simultaneously. For example, if three clients are online at the same time, one will be the trainer client and the other two will be the trainee clients. Demonstrations on the trainer client will be displayed on the trainee clients in real time. Communication between the VMI Client Application and the DMZ requires authentication.

To ensure security, the Intranet Firewall is located between the Intranet zone and the DMZ to protect the servers of the Telemedicine Training System. The Internet Firewall is located between the DMZ and the Mobile Network zone to protect both the DMZ and the Intranet zone. Additionally, the system administrator can grant access only to mobile devices with registered MAC addresses.

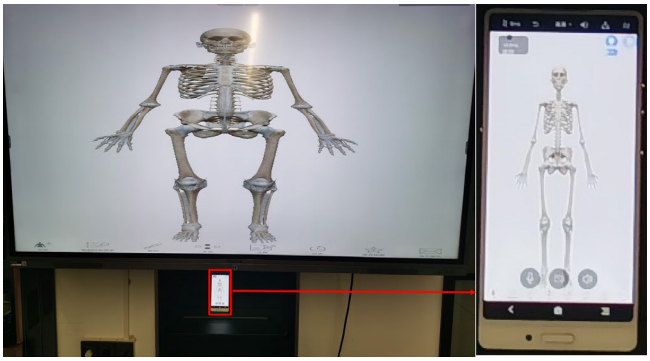


Fig. 7. Telemedicine training system UI can be displayed on multiple clients at the same time, for example a smartphone and a smart display.

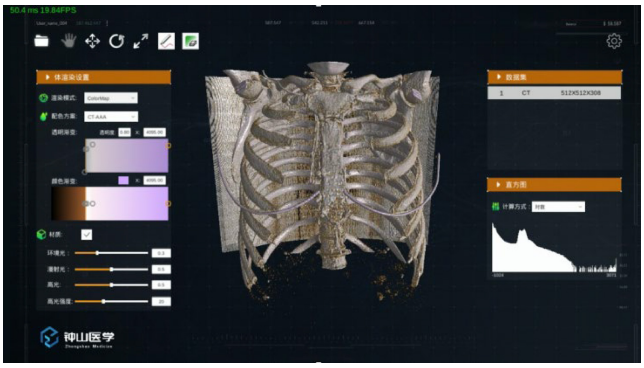


Fig. 8. Interface of the OpenVMI-based telemedicine training system.

The system UI can be displayed on multiple clients at the same time. For example the UIs on a client smartphone and a client smart display are shown in Fig. 7. An interface of the Telemedicine Training System is shown in Fig. 8.

B. Achieved Functionalities That Are Hard to Achieve in Normal Smartphone

By operating in a cloud-based virtual Android and being accessed via the VMI client application installed in a physical mobile device, the OpenVMI-based Telemedicine Training System supports low-latency 3-D image rendering, which is hardly achievable in a local training application. Movements, rotation and scaling of medical images are rendered in real time. The key features supported by the Telemedicine Training System include the following.

1) *Multiple Rendering Modes:* Multislice and multiplane rendering are often required in a medical imaging training session for the clear demonstration of human structures. The different rendering modes available in the Telemedicine Training System helps to deliver high-quality training.
2) *Customized Textures:* The Telemedicine Training System offers a selection of texture materials for a vivid display and a clear distinction between human organs.
3) *Image Transformation:* A medical image can be transformed flexibly. The instructor is able to perform different functions, including moving, scaling, rotating, and

TABLE III
COMPARISON BETWEEN ANBOX, WAYDROID, ROBOX, AND OPENVMI

	Anbox	Waydroid	Robox	OpenVMI
Time of first released	Apr. 2017	Sep. 2021	Apr. 2018	Sep. 2020
Access mode	Desktop	Desktop	Remote	Remote
Hardware support	Limited	Many	Many	Most
Multi-instance support	No	No	Yes	Yes
Multi-client support	No	No	No	Yes
Direct GPU support	No	No	No	Yes
Host VPU support	No	No	No	Yes

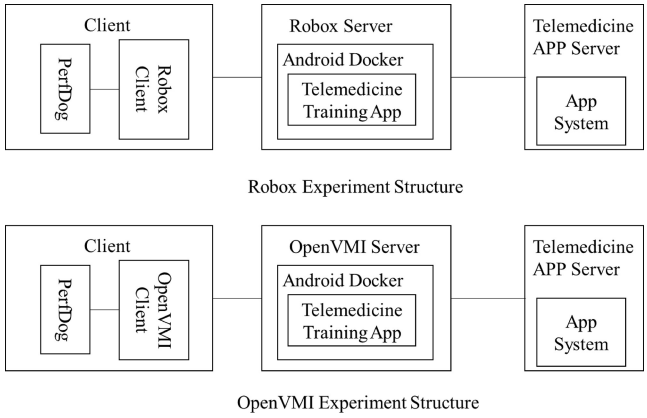


Fig. 9. Structure of the comparison experiment.

resizing a specific selection of an image. Annotation in smartphone is also supported.

VI. EXPERIMENTS

A. Performance Comparison Experiment

We compare the OpenVMI qualitatively with anbox, waydroid and robox, the other state-of-the-art open-source VMI systems mentioned in Section II-C, in Table III. Anbox and waydroid are desktop applications and each host server can start only one instance. In contrast, each host server can start multiple cloud-based OpenVMI instances. In addition, the OpenVMI system also supports many functions not found in anbox and waydroid, such as GPU and VPU usage. As such, the performance of the OpenVMI system is not compared quantitatively with that of anbox and waydroid.

Robox, the cloud-based VMI system co-developed by Huawei and Linaro, shares similar system structure with the OpenVMI. But unlike the OpenVMI, it does not support direct GPU invocation. Monbox, its proprietary commercial version, supports direct GPU invocation. But monbox is publicly inaccessible. As a result, a quantitative comparison is carried out only between the OpenVMI and robox.

The structure of the comparison experiment is shown in Fig. 9. Robox and the OpenVMI are installed separately on a host server of the same hardware and software configuration as detailed in Table IV. The host servers are named as the Robox Server and the OpenVMI Server. The Telemedicine Training Application, which accesses the Telemedicine App server, is installed on the Android Docker built in the Robox Server and the OpenVMI Server. When the telemedicine client application is initiated in the client device, performance of the system is tracked by Perfdog. We mainly focus on four performance parameters: 1) the fps; 2) the CPU utilization of the host server; 3) RAM usage of the host server; and 3) the initiation time of the telemedicine training system (time required between the initiation of the client application and the display of the default UI). The CPU utilization is measured in terms of the CPU used by running processes as a percentage of a single CPU core. The host server is the Huawei TaiShan200-2280V2 Multicore CPU Server, which has 96(48*2) CPU cores, so theoretically the maximum CPU utilization is 9600%. The experiment is repeated for ten times and the means of the parameter, as detailed in Table V, are used for comparison.

In Table V, the mean fps of the robox-based experiment is 7.8, which means the robox-based system takes an average of 128 ms to process and display one frame. This is much higher than that of the OpenVMI-based experiment, which only takes 32.2 ms to process and display one frame, based on a sample mean fps of 31. In contrast to the OpenVMI's direct invocation of GPU resources in Android Docker, the robox-based system does not support direct GPU rendering and VPU coding. Data has to be transmitted to the host operating system for rendering. The rendered results have to be transmitted back to Android Docker and coded from RGBA to H.264 format before being transmitted to the client application. A large amount of CPU resources is consumed on data transmission between Android Docker and the host operating system, resulting in the robox-based system's prolonged frame handling. For the same reason, the mean CPU utilization of the robox-based experiment is 921.8%, which is much higher than the 20% utilization in the OpenVMI-based experiment. Since OpenVMI has more modules to initiate than robox, more RAM space and time are needed for data processing. We therefore expect the OpenVMI-based experiment to underperform

TABLE IV
SYSTEM CONFIGURATION FOR PERFORMANCE COMPARISON

Environment	Configuration
Server	HuaWei TaiShan200-2280v2: CPU: Kunpeng 920, 48Core *2; RAM: 512GB; SSD: 480GB; SATA: 4000GB; Network: 4*GE; GPU: AMD Radeon W6800*2; VPU: Netint T432*2; Ubuntu 20.04
Virtual Android	CPU: 2 Core; RAM: 8GB; Frash Memory: 32GB; Resolution Ratio: 1920 * 1080; Frame Rate: 30 fps; Android 7.1.1, ZhongShan Telemedicine System.
OpenVMI	Version2.0
Robox	Version 2.3

TABLE V
RESULTS OF THE PERFORMANCE COMPARISON EXPERIMENT

Performance parameter	Robox-based	OpenVMI-based
Fps	7.8	31.0
CPU utilization	921.8%	20.0%
RAM usage (GB)	1.92	2.02
Initiation time (s)	15.5	27.3

TABLE VI
CPU UTILIZATION IN TEN DIFFERENT ROBOX-BASED EXPERIMENTS

Test Number	1	2	3	4	5
CPU Utilization	923%	934%	924%	926%	933%
Test Number	6	7	8	9	10
CPU Utilization	911%	920%	915%	907%	925%

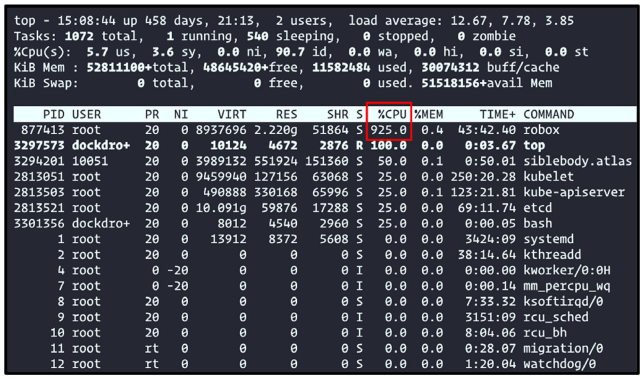


Fig. 10. Snapshot of the host server CPU utilization by using the "top" command for the robox-based experiment.

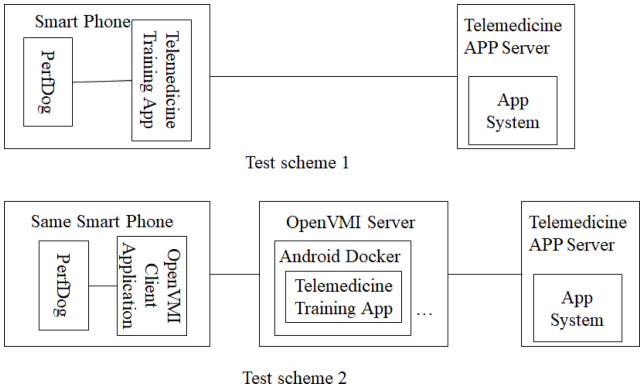


Fig. 11. Structure of the performance experiment.

in RAM usage and system initiation time. Results of the experiment show that the RAM usage of host server is 1.92 GB in the robox-based experiment, which is 5% smaller than the 2.02 GB in the OpenVMI-based experiment. Also, the system initiation time is 15.5 seconds for the robox-based experiment, almost two times faster than the OpenVMI-based experiment.

The mean CPU utilization of the host server reaches 921.8% in the robox-based experiment. This means that in the 96-core-host server, the robox-based system consumes an average of more than nine CPU cores to support the initiation of the Telemedicine Training Application. Table VI shows the CPU utilization of the host server in robox-based experiment for each experiment. Fig. 10 is a snapshot of the real-time CPU utilization of the host server during an experiment of the robox-based system.

B. System Performance Experiment

The Telemedicine Training Application is deployed locally and in cloud for a comparison of application performance. In cloud deployment, illustrated as test scheme 2 in Fig. 11, the OpenVMI Client Application is installed in the client handset to start the OpenVMI-based Telemedicine Training System. Performance parameters, including fps, CPU utilization of the client device, RAM usage of the client device, and the required initiation time are tracked by Perfdog. In local deployment, illustrated as test scheme 1 in Fig. 11, the Telemedicine Training Application is installed locally in the client handset. The same parameters are tracked.

TABLE VII
CONFIGURATION OF THE MOBILE PHONE

Huawei mate30 smart phone	CPU: HuaWei Kirin 990 8 cores, 2.86GHz, 7nm; GPU: 16 cores Mali-G76; RAM: 8 GB Flash Memory: 256GB;
Smartisan smart phone	CPU: Qualcomm Snapdragon 625, 8 cores, 2.0GHz, 14nm; GPU: Adreno 506; RAM: 4GB; Flash Memory: 64GB.
Android Docker	CPU: 2 Core; RAM: 8GB; Frash Memory: 32GB; Resolution Ratio: 1920 * 1080; Frame Rate: 30 fps;

The Telemedicine Training Application is initiated for ten times for each deployment modality and the means of the parameters of interest are used for comparing application performance. The experiment is repeated in two different client devices.

A Huawei smartphone and a Smartisan smartphone are used as the client device. The Smartisan smartphone has weaker hardware configuration so that we can compare application performance across client devices of different capabilities. A dual-core CPU, 8 GB of RAM and 32 GB of Flash Memory are used in Android Docker, as early stage investigation suggested that such configuration is capable to run the Telemedicine Training Application smoothly while minimizing resource usage. The configuration of the devices involved in the experiment is detailed in Table VII.

The results of the experiment are detailed in Table VIII. Unlike the higher RAM usage of the host server as shown in Table V, which arises from the image rendering task and the coding task, RAM usage of the client device is much lower as the RAM of the client device is used only for displaying the rendered results.

The fps of the OpenVMI-based application is only 0.91 fps higher than the local application in the Huawei smartphone. The fps of the local application is not compromised thanks to Huawei's powerful hardware configuration. CPU utilization of the OpenVMI-based application in the Huawei smartphone is only 15% as the Huawei smartphone is used only for display and interaction. In contrast, the Huawei smartphone also executes the initiation of the Telemedicine Training Application so that more CPU resources are consumed, explaining the local application's higher CPU utilization than the OpenVMI-based application. The RAM usage is 30% higher in the OpenVMI-based Telemedicine Training System because the OpenVMI Client Application consumes extra RAM space. Finally, the initiation time of the local Telemedicine Training Application is almost twice faster than the cloud-based application. There are three reasons.

TABLE VIII
RESULTS OF THE PERFORMANCE EXPERIMENT ON MOBILE PHONE AND VIRTUAL MOBILE PHONE

	Fps	CPU utilization	RAM usage (MB)	Initiation time(s)
Huawei, Local app	30.12	18%	518	18.8
Huawei, Cloud app	31.03	15%	675	27.7
Smartisan, Local app	1.5	1%	390	69.5
Smartisan, Cloud app	31.0	15%	581	28.0

1) The Huawei smartphone is powerful so it can start a local application quickly.
2) Extra data transmission occurs when the Telemedicine Training Application is initiated in cloud, leading to higher latency in the cloud-based initiation.
3) The OpenVMI client application has to be initiated before it can initiate the Telemedicine Training Application in cloud, further adding to initiation time.

In conclusion, for client devices of powerful hardware configuration, deploying the Telemedicine Training Application in cloud via the OpenVMI produces only slightly better performance compared to local deployment.

In contrast, the Smartisan smartphone, with its weaker hardware capabilities, has 1.5 fps (666.7 ms taken per frame), 1% CPU utilization, 390-MB RAM usage and an initiation time of over 1 min when the Telemedicine Training Application is initiated locally. CPU utilization is low because the local Telemedicine Training Application cannot be initiated normally and it cannot work properly. Since the local initiation performed significantly better in the Huawei smartphone, the low performance in the Smartisan smartphone can be attributed to its weak hardware When the training application is initiated in cloud via the OpenVMI Client application in the Smartisan smartphone, average fps has significantly improved by a factor of 20 times and the average initiation time is reduced by half to under 30 s compared to local initiation. The CPU utilization increases from 1% to 15% and the RAM usage increased from 390 to 581 MB. Furthermore, performance of the OpenVMI-based initiation has been consistent across the Huawei smartphone and the Smartisan smartphone, suggesting that an OpenVMI-based application performs almost independently of the hardware configuration of a client device. The OpenVMI provides a feasible solution for devices of weaker hardware capabilities to access resource-demanding applications.

In addition, once purchased, the hardware configuration of a smartphone is fixed, whereas the configuration of Android Docker can be easily upgraded on demand, giving an OpenVMI-based application a greater degree of flexibility.

C. System Concurrency Experiment

The OpenVMI-based solution proposed by this article supports multiple Android Dockers hence multiple concurrent telemedicine training sessions on a host server. An experiment is conducted to investigate the optimal number of concurrent Dockers on one single host server. First, 8, 12, 16, 24, 32, and 48 concurrent Android Dockers are virtualized on one host server, respectively. Second, for each virtualization, the server utilization performance is monitored. Finally, the server performance with different concurrency is compared for deciding on the optimal number of concurrency.

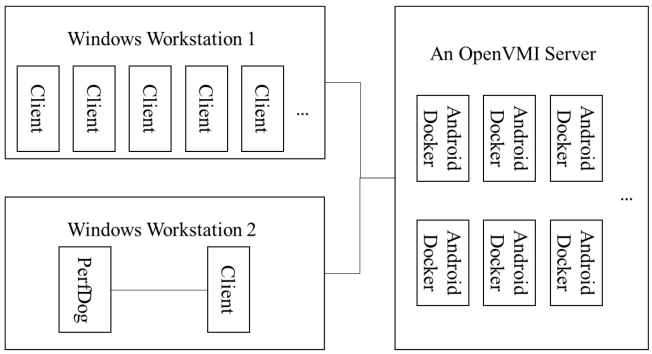


Fig. 12. Structure of concurrency experiment.

TABLE IX
SYSTEM CONFIGURATION ON CLIENT AND VIRTUAL ANDROID

Windows Workstation 1	CPU: Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz 3.60 GHz; RAM: 192 GB; GPU: GTX1080
Windows Workstation 2	CPU: AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz; RAM: 16GB; GPU: AMD Radeon (TM) Graphics
Virtualized Android	CPU: 2 Core; RAM: 8GB; Flash Memory: 32GB; Resolution Ratio: 1920 * 1080; Frame Rate: 30 fps;

In the concurrency experiment, two Windows workstations (Workstation 1 and Workstation 2) are connected to the OpenVMI host server via the Internet as illustrated in Fig. 12. Workstation 1 acts as a client device simulator and Workstation 2 acts as performance tracker. Multiple Android Operating Systems are simulated in Workstation 1 for running multiple OpenVMI client devices at the same time. A single Android Operating System simulator is installed in Workstation 2. PerfDog and the OpenVMI Client application are installed in the Android simulator in Workstation 2 to obtain the test result of the host server performance. The system configuration of the two workstations is detailed in Table IX. The system configuration of the host server is specified in Table X.

To test for the optimal number of concurrent Android Dockers, eight concurrent Android Dockers are created on a host server at first, each of which runs the training application. For test purpose only, each Android Docker is connected to one simulated client device in Workstation 1. Each client device will open the third file in default order, a 3-D human head and neck anatomy, in the com.yysmart.volumerender data

TABLE X
SYSTEM CONFIGURATION OF THE CLOUD SERVER

Server	Operating System	Software in cloud	Software in application layer
HuaWei TaiShan200-2280v2: CPU: Kunpeng 920, 48Core *2; RAM: 512GB; SSD: 480GB; SATA: 4000GB; Network: 4*GE; GPU: AMD Radeon W6800*2; VPU: Netint T432*2.	Ubuntu 20.04	K8S 1.22.3, OpenVMI 2.0	Android 7.1.1, Telemedicine System

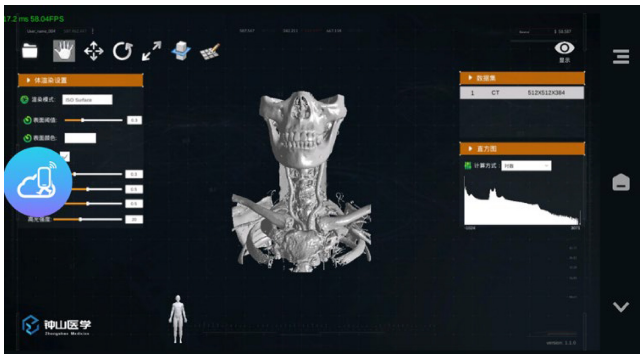


Fig. 13. Main interface using in the concurrency experiment.

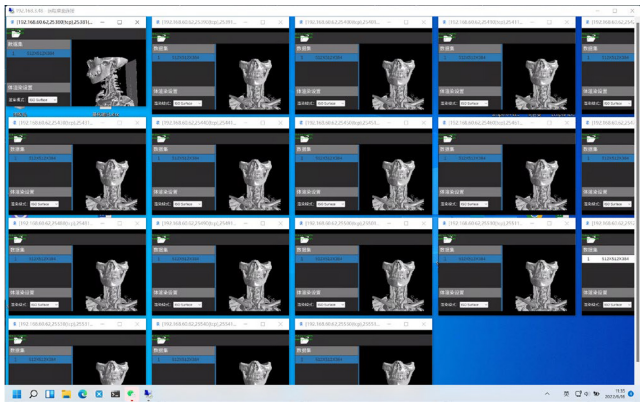


Fig. 14. Screen snapshot of workstation 1, 15 simulated client devices.

package, as shown in Fig. 13. The system runs for 2 h consecutively, with the average value of GPU utilization, VRAM utilization and Docker CPU utilization being documented.

The experiment is repeated for the virtualization of 16, 24, 32, 48 concurrent Android Dockers. A single GPU is used when the number of virtualized smartphones is 32 and below. Dual-core GPUs are used when 48 smartphones are virtualized. Fig. 14 shows a screen snapshot of Workstation 1 when testing for the concurrency of 16 client devices.

The results of the experiment detailed in Table XI show that GPU utilization and VRAM utilization increased with the number of virtualized smartphones. GPU utilization reaches 70% and VRAM utilization reaches 83.6% when the concurrent Docker number is 24. As a comparison, a relatively efficient use of GPU resources without overloading the GPU is at about 80% GPU utilization. When the concurrent Docker number increases to 32, GPU utilization is 76% which is still below 80%, but the average RAM utilization reaches 91.3%, which may undermine system performance at peak usage. When the Docker number further increases to 48, GPU utilization reaches 90% and 85%, respectively, for the dual cores and the VRAM utilization reached 89.3% and 88.6%, respectively. Despite with 16 more concurrent Dockers, the VRAM usage in the dual-core GPU with 48 concurrent Dockers is actually slightly lower than that in the single-core GPU with 32 concurrent Dockers, as the workload is shared between the dual cores. The experiment suggests that the optimal number of concurrent training sessions is 24 for a single host server.

We have also tested for the optimal number of connected clients that one Android Docker supports. The test results show that each Android Docker can connect to five client devices without compromising system performance.

Experiments of concurrency suggest that a single host server supports 24 concurrent Android Dockers, hence 24 concurrent live training sessions for cloud server configuration as detailed in Table X. Each training session allows the connection of five client devices, with one client device being the trainer device, and four others being the trainee device.

It should be noted that the number of optimal concurrent Android Dockers is influenced by the server's CPU, ROM and GPU configuration. As CPU and ROM configuration improve, the optimal number of concurrent devices becomes increasingly driven by GPU configuration. The more powerful GPU is, the more concurrent Android Dockers the cloud server supports. However, there is a tradeoff between GPU configuration and financial cost. For example, the Nvidia Tesla V100 GPU costs 61 000 RMB in China, whereas the less powerful AMD WX5100 GPU accelerator costs only 3100 RMB. Cost effectiveness is also an important factor determining the optimal server configuration hence the number of optimal concurrent Android Dockers.

VII. DISCUSSION AND FUTURE WORK

A. Technical Features of the System

The technical features of the OpenVMI-based Telemedicine Training System for 3-D images are as follows.

1) *High Efficiency:* Direct GPU invocation leads to less transmission nodes and the elimination of multiple stages of instruction translation between OpenGL ES and OpenGL. In addition, as a VPU is added to the hardware layer to code rendered results in H.264 format, a large amount of CPU resources can be spared for other tasks.
2) *Multidevice Concurrency:* A large number of Android Docker can be started in parallel in a host server. A single virtualized client application in each Android Docker supports live streaming across multiple devices. It also allows real-time interactions between different devices.
3) *High Performance:* Empowered by the powerful host server, the OpenVMI-based Telemedicine Training System has high-performance computing power, high-rendering power and a large storage capacity such that it is able to perform resource-demanding graphic

TABLE XI
RESULTS OF THE PERFORMANCE EXPERIMENT ON DIFFERENT NUMBER OF CONCURRENT DEVICES

Number of concurrent client devices	AVG GPU utilization	AVG VRAM utilization	AVG Docker CPU utilization	AVG Server CPU utilization	AVG Server RAM usage
8	32%	27.9%	23.2%	3.7%	28.3G
12	44%	41.7%	28.5%	5%	33.8G
16	54%	55.6%	28.9%	7.3%	40.8G
24	70%	83.6%	27.83%	11.3%	55.4G
32	76%	91.3%	26.1%	15%	71.5G
48(dual-core GPU)	90%, 85%	89.3%, 88.6%	28.2%	24%	100G

rendering. The physical client devices are used for display and interactions only, making the training system highly deployable.

B. System Advantages

The OpenVMI-based Telemedicine Training System has several advantages as follows.

1) *High Security:* The training system is hosted in a cloud-based server. System functions and system updates are implemented on the server. The physical local mobile device is used only for display and interaction. As a result, any malfunctions of the local device will not impact server functioning and the data stored on the server remains intact. In this way, the system can ensure system stability and data security. Since user data is stored in the cloud, data can be retrieved even when the physical client device is lost or damaged. With security protection at the Management Platform, combined with well established authentication scheme, problems of data breach, data loss and data damage can be contained effectively.
2) *High Flexibility:* By using the OpenVMI technology, the Telemedicine Training Application can be run seamlessly in client smartphones regardless of their hardware configuration. Furthermore, the configuration of Android Docker can be easily upgraded on demand, whereas the hardware configuration of a smartphone is fixed after being purchased.
3) *Easy Promotion:* As the physical local device is used for display and interactions only, the technical requirements for it can be easily met by existing hospital-owned devices and personal devices. This helps to lower capital investment cost.
4) *User Privacy Protection:* Virtualized applications running in the virtual environment only have security access to the server location instead of the location of a physical client device, freeing data breach problem from a client device.

C. Limitations and Future Work

The performance of the local Telemedicine Training Application installed in a powerful Huawei smartphone is comparable to the OpenVMI-based version in terms of fps, CPU utilization and RAM usage. The Huawei smartphone even initiates the application faster than the OpenVMI-based application does. This implies that if the hardware configuration of a mobile device is powerful enough, it can also run resource-demanding tasks like image rendering well. Nonetheless, there exists a tradeoff between hardware capabilities and financial cost, especially in less developed areas.

The OpenVMI is applied in telemedicine training in this article. We hope to explore a greater number of applications of the OpenVMI system as the demand for 3-D image-based applications grow.

The OpenVMI is highly deployable in devices of various hardware capabilities. It provides flexibility and it utilizes the high-computing capabilities of the cloud, making it particularly useful in applications based on VR and AR. The practicability of the OpenVMI in VR and AR will be studied in the future.

The early version of the OpenVMI is hosted open source in the following address: https://github.com/DockDroid/openvmi. The improvements mentioned in this article have been integrated into the commercial version of the OpenVMI. It will also be hosted open source in the GitHub in the near future.

VIII. CONCLUSION

The OpenVMI system proposed by this article presents a low-cost solution for the display of interactive 3-D images in mobile environments. It improves upon a typical VMI in two ways.

1) Direct invocation of GPU resources is achieved by developing a bespoke GPU driver installed in Android Docker.
2) Rendered results are coded in H.264 format by a VPU.

Both improvements result in less transmission delays and a lower consumption of CPU resources, empowering the display of interactive 3-D images via the cloud.

By adopting the OpenVMI, the Telemedicine Training System is an effective way to overcome problems, such as geographical immobility, limited computing power in mobile devices, and capital under investment, that limits the scope of medical training in undeveloped areas. The results of various performance experiments suggest that an OpenVMI-based application is highly deployable across devices of different hardware capabilities and the OpenVMI supports 24 concurrent training sessions, each of which can connect with five client devices at the same time for a single host server.

Finally, since the OpenVMI supports the display of 3-D images across multiple devices concurrently, its application can be extended to other areas that rely on 3-D image rendering heavily, such as VR and AR.

REFERENCES

[1] A. C. Smith et al., "Telehealth for global emergencies: Implications for coronavirus disease 2019 (COVID-19)," *J. Telemed. Telecare*, vol. 26, no. 5, pp. 309–313, 2020.

[2] A. Moglia et al., "5G in healthcare: From COVID-19 to future challenges," *IEEE J. Biomed. Health Inform.*, vol. 26, no. 8, pp. 4187–4196, Aug. 2022.

[3] A. Angelucci, D. Kuller, and A. Aliverti, "A home telemedicine system for continuous respiratory monitoring," *IEEE J. Biomed. Health Inform.*, vol. 25, no. 4, pp. 1247–1256, Apr. 2021.

[4] S. Jiang, *Design and Implementation of Telemedicine System Based on Unity 3D*. Beijing, China: Beijing Jiaotong Univ., 2017.

[5] M. L. E. Jin, M. M. Brown, D. Patwa, A. Nirmalan, and P. A. Edwards, "Telemedicine, telementoring, and telesurgery for surgical practices," *Curr. Problems Surg.*, vol. 58, no. 12, pp. 1–31, 2021.

[6] Y. Li, Y. Li, Z. Deng, and Z. Zhu, "A collaborative telemedicine platform focusing on paranasal sinus segmentation," in *Proc. Int. Conf. Intell. Interact. Multimedia Syst. Services*, 2018 pp. 238–247.

[7] K. Belgacem, M. Kenoui, F. Bouguerra, M. Laidi, A. Semrani, and C. Sellah, "Collaborative visualization and annotations of DICOM images for real-time Web-based telemedicine system," in *Proc. 2021 Int. Conf. Recent Adv. Math. Informat. (ICRAMI)*, 2021, pp. 1–6.

[8] S. Elmoghazy, E. Yaacoub, N. V. Navkar, A. Mohamed, and A. Erbad, "Survey of immersive techniques for surgical care telemedicine applications," in *Proc. 10th Mediterr. Conf. Embedded Comput. (MECO)*, 2021, pp. 1–6.

[9] C. K. Scott, P. Karem, K. Shifflett, L. Vegi, K. Ravi, and M. Brooks, "Evaluating barriers to adopting telemedicine worldwide: A systematic review," *J. Telemed. Telecare*, vol. 24, no. 1, pp. 4–12, 2018.

[10] K. Su, P. Liu, L. Gu, W. Chen, K. Hwang, and Z. Yu, "vMobiDesk: Desktop virtualization for mobile operating systems," *IEEE Access*, vol. 8, pp. 213541–213553, 2020.

[11] M.-M. Moazzami, D. E. Phillips, R. Tan, and G. Xing, "ORBIT: A platform for smartphone-based data-intensive sensing applications," *IEEE Trans. Mobile Comput.*, vol. 16, no. 3, pp. 801–815, Mar. 2017.

[12] Y. Wu, Y. Wang, W. Hu, and G. Cao, "SmartPhoto: A resource-aware crowdsourcing approach for image sensing with smartphones," *IEEE Trans. Mobile Comput.*, vol. 15, no. 5, pp. 1249–1263, May 2016.

[13] H. Cui, D. Tu, F. Tang, P. Xu, H. Liu, and S. Shen, "VidSfM: Robust and accurate structure-from-motion for monocular videos," *IEEE Trans. Image Process.*, vol. 31, pp. 2449–2462, 2022.

[14] E. Schwartz, R. Giryes, and A. M. Bronstein, "DeepISP: Toward learning an end-to-end image processing pipeline," *IEEE Trans. Image Process.*, vol. 28, no. 2, pp. 912–923, Feb. 2019.

[15] S. Katakol, B. Elbarashy, L. Herranz, J. van de Weijer, and A. M. López, "Distributed learning and inference with compressed images," *IEEE Trans. Image Process.*, vol. 30, pp. 3069–3083, 2021.

[16] R. G. Alakbarov and O. R. Alakbarov, "Selection virtual machine in mobile cloud computing," in *Proc. 2018 9th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, 2018, pp. 1–4.

[17] S.-C. Oh, K. Kim, K. Koh, and C. Ahn, "ViMo (virtualization for mobile): A virtual machine monitor supporting full virtualization for ARM mobile systems," in *Proc. Adv. Cogn. Technol. Appl., COGNITIVE*, 2010, pp. 48–53.

[18] M. Estai et al., "End-user acceptance of a cloud-based teledentistry system and Android phone app for remote screening for oral diseases," *J. Telemed. Telecare*, vol. 23, no. 1, pp. 44–52, 2015.

[19] M. Estai et al., "A proof-of-concept evaluation of a cloud-based store-and-forward telemedicine app for screening for oral diseases," *J. Telemed. Telecare*, vol. 22, no. 6, pp. 319–325, 2016.

[20] R. Latha, P. Vetrivelan, and S. Geetha, "Telemedicine setup using wireless body area network over cloud," *Procedia Comput. Sci.*, vol. 165, pp. 285–291, Feb. 2020.

[21] J. Qian, "Analysis of intelligent telemedicine system based on Internet of Things," *Electron. Compon. Inf. Technol.*, vol. 5, no. 7, pp. 9–10, 2021.

[22] W. Luo et al., "Development of telemedicine system for military forces based on WeChat micro-program," *China Med. Dev.*, vol. 34, no. 10, pp. 984–986, 2019.

[23] X. Xu et al., "Advances in smartphone-based Point-of-Care diagnostics," *Proc. IEEE*, vol. 103, no. 2, pp. 236–247, Feb. 2015.

[24] B. Askarian, P. Ho, and J. W. Chong, "Detecting cataract using smartphones," *IEEE J. Transl. Eng. Health Med.*, vol. 9, pp. 1–10, 2021.

[25] R. D. Chand, A. Kumar, A. Kumar, P. Tiwari, R. Rajnish, and S. K. Mishra "Advanced communication technologies for collaborative learning in telemedicine and tele-care," in *Proc. 2019 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, 2019, pp. 601–605.

[26] S. S. Vedaei et al., "COVID-SAFE: An IoT-based system for automated health monitoring and surveillance in post-pandemic life," *IEEE Access*, vol. 8, pp. 188538–188551, 2020.

[27] M. H. Nornaim, N. A. Abdul-Kadir, F. K. C. Harun, and M. A. A. Razak, "A wireless ECG device with mobile applications for Android," in *Proc. 7th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, 2020, pp. 168–171.

[28] R. Wang, J. Xu, Y. Ma, M. Talha, M. S. Al-Rakhami, and A. Ghoneim, "Auxiliary diagnosis of COVID-19 based on 5G-enabled federated learning," *IEEE Netw.*, vol. 35, no. 3, pp. 14–20, May/Jun. 2021.

[29] J. Guo, "Smartphone-powered electrochemical biosensing dongle for emerging medical IoTs application," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2592–2597, Jun. 2018.

[30] A. Lauraitis, R. Maskeliūnas, R. Damaševičius, D. Polap, and M. Woźniak "A smartphone application for automated decision support in cognitive task based evaluation of central nervous system motor disorders," *IEEE J. Biomed. Health Inform.*, vol. 23, no. 5, pp. 1865–1876, Sep. 2019.

[31] W. Qi, H. Su, and A. Aliverti, "A smartphone-based adaptive recognition and real-time monitoring system for human activities," *IEEE Trans. Human-Mach. Syst.*, vol. 50, no. 5, pp. 414–423, Oct. 2020.

[32] C. Gong et al., "RetinaMatch: Efficient template matching of retina images for teleophthalmology," *IEEE Trans. Med. Imag.*, vol. 38, no. 8, pp. 1993–2004, Aug. 2019.

[33] T. Zhang, Y. Li, J. P. Y. Cheung, S. Dokos, and K.-Y. K. Wong "Learning-based coronal spine alignment prediction using smartphone-acquired scoliosis radiograph images," *IEEE Access*, vol. 9, pp. 38287–38295, 2021.

[34] C. Hoyos-Barceló, J. Monge-Álvarez, M. Zeeshan Shakir, J.-M. Alcaraz-Calero, and P. Casaseca-de-la-Higuera, "Efficient k-NN implementation for real-time detection of cough events in smartphones," *IEEE J. Biomed. Health Inform.*, vol. 22, no. 5, pp. 1662–1671, Sep. 2018.

[35] M. Cheffena, "Fall detection using smartphone audio features," *IEEE J. Biomed. Health Inform.*, vol. 20, no. 4, pp. 1073–1080, Jul. 2016.

[36] I. Frederix, S. Sankaran, K. Coninx, and P. Dendale, "MobileHeart, a mobile smartphone-based application that supports and monitors coronary artery disease patients during rehabilitation," in *Proc. 2016 38th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, 2016, pp. 513–516.

[37] L. Liu, J. Xu, Y. Huan, Z. Zou, S.-C. Yeh, and L.-R. Zheng, "A smart dental health-IoT platform based on intelligent hardware, deep learning, and mobile terminal," *IEEE J. Biomed. Health Inform.*, vol. 24, no. 3, pp. 898–906, Mar. 2020.

[38] Y. Guo, L. Cai, and J. Zhang, "3D face from X: Learning face shape from diverse sources," *IEEE Trans. Image Process.*, vol. 30, pp. 3815–3827, 2021.

[39] Y. Nie, T. Su, Z. Zhang, H. Sun, and G. Li, "Dynamic video stitching via shakiness removing," *IEEE Trans. Image Process.*, vol. 27, pp. 164–178, 2018.

[40] Z. Fu, J. Zhou, and W. Xu, "A GPU-enabled mobile telemedicine training system for graphic rendering," in *Proc. Int. Conf. Mobile Comput. Netw. (MobiCom'22). ACM*, Sydney, NSW, Australia, 2022, pp. 877–879, doi: 10.1145/3495243.3558269.

[41] J. Korhonen, "Two-level approach for no-reference consumer video quality assessment," *IEEE Trans. Image Process.*, vol. 28, pp. 5923–5938, 2019.

[42] J. Minseok, P. Myong-Soon, and S. C. Shah, "A mobile ad hoc cloud for automated video surveillance system," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, 2017, pp. 1001–1005.

[43] P. Liu, Y. Chen, L. Fu, M. Yan, and Z. Wang, "cMobiDesk: A lightweight solution for android desktop virtualization," in *Proc. 7th Int. Conf. Cloud Comput. Big Data Analyt. (ICCCBDA)*, 2022, pp. 234–239.

[44] M. P. Anastasopoulos et al., "Planning of dynamic mobile optical virtual network infrastructures supporting cloud services," in *Proc. 2014 Eur. Conf. Netw. Commun. (EuCNC)*, 2014, pp. 1–5.

[45] C.-M. Wang, Y.-S. Wu, and H.-H. Chung, "FUSION: A unified application model for virtual mobile infrastructure," in *Proc. IEEE Conf. Dependable Secure Comput.*, 2017, pp. 224–231.

[46] E. Choi and J. Hong, "Design and implementation of virtual machine control and streaming scheme using Linux kernel-based virtual machine hypercall for virtual mobile infrastructure," in *Proc. Conf. Res. Adapt. Converg. Syst.*, 2019, pp. 57–60.

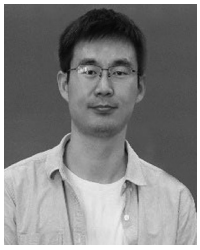
[47] M. Bentele, D. Von Suchodoletz, M. Messner, and S. Rettberg, "Towards a GPU-accelerated open source VDI for Openstack," in *Proc. Int. Conf. Cloud Comput.*, 2022, pp. 149–164.
[48] F. Wan, N. Chang, and J. Zhou, "Design ideas of mobile Internet desktop system based on virtualization technology in cloud computing," in *Proc. Int. Conf. Adv. Ambient Comput. Intell. (ICAACI)*, 2020, pp. 193–196.
[49] K. Fornito, C. Zembower, and S. Sneddon, "Using infrastructure-as-code and the public cloud to power on-air media creation platforms," in *Proc. SMPTE*, 2019, pp. 1–9.
[50] J. Wu, C.-C. Kuo, S.-T. Hsiao, K.-H. Chang, and S.-H. Liu, "A cloud experiment for virtual reality and augmented reality in NCHC render farm," in *Proc. 2020 Nicogr. Int. (NicoInt)*, 2020, pp. 78–81.
[51] Y. Wang, S. Lv, and W. Li, "The meteorological cloud desktop system of CMA meteorological observation center," in *Proc. Int. Conf. Meteorol. Observ. (ICMO)*, 2019, pp. 1–3.
[52] J.-Y. Li et al., "The implementation of a GPU-accelerated virtual desktop infrastructure platform," in *Proc. 2017 Int. Conf. Green Inform. (ICGI)*, 2017, pp. 85–92.
[53] C.-H. Chang, C.-T. Yang, J.-Y. Lee, C.-L. Lai, and C.-C. Kuo, "On construction and performance evaluation of a virtual desktop infrastructure with GPU accelerated," *IEEE Access*, vol. 8, pp. 170162–170173, 2020.
[54] H. Dong et al., "Towards enabling residential virtual-desktop computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 745–762, Jan./Mar. 2023.
[55] T.-D. Nguyen, P. P. Hung, T. H. Dai, N. H. Quoc, C.-T. Huynh, and E.-N. Huh "Prediction-based energy policy for mobile virtual desktop infrastructure in a cloud environment," *Inf. Sci.*, vol. 319, pp. 132–151, Oct. 2015.
[56] T. T. Adeliyi and O. O. Olugbara, "Optimizing remote access using mobile cloud virtual desktop infrastructure," in *Proc. 2021 Conf. Inf. Commun. Technol. Soc. (ICTAS)*, 2021, pp. 1–4.
[57] D. Ginsburg, B. Purnomo, D. Shreiner, and A. Munshi, *OpenGL ES 3.0 Programming Guide*. Reading, MA, USA: Addison-Wesley, 2014.
[58] J. Kessenich, G. Sellers, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 With SPIR-V*. Reading, MA, USA: Addison-Wesley, 2016.
[59] Y. Granot, A. Ivorra, and B. Rubinsky, "A new concept for medical imaging centered on cellular phone technology," *PLoS One*, vol. 3, no. 4, pp. 1–7, 2008.
[60] A. Attila, Á. Garai, and I. Péntek, "Common open telemedicine hub and infrastructure with interface recommendation," in *Proc. IEEE 11th Int. Symp. Appl. Computat. Intell. Inform. (SACI)*, 2016, pp. 385–390.

Zhipeng Fu received the B.S., M.S., and Ph.D. degrees in computer science and technology from the School of Computer, National University of Defense Technology, Changsha, China, in 2003, 2006, and 2014, respectively.

He is currently an Engineer with the Industrial Internet of Things Research Institute, Department of New Pattern Network, Peng Cheng Laboratory, Shenzhen, China. His research interests include mobile operating system, computer vision, artificial intelligence, and Internet of Things.

Jun Zhou is currently pursuing the Ph.D. degree with Sun Yat-sen University, Guangzhou, China.

He is with the Industrial Internet of Things Research Institute, Department of New Pattern Network Peng Cheng Laboratory, Shenzhen, China. His research interests include mobile operating system, cloud computing, graphic computing, and resource scheduling.



Wanpeng Xu received the B.S. and M.S. degrees in remote sensing science and technology from Space Engineering University, Beijing, China, in 2008 and 2013, respectively, and the Ph.D. degree in information and communication engineering from the School of Aerospace Information, Space Engineering University, Beijing, in 2022.

His research interests include computer vision, virtual reality, and automated driving.

Changguo Guo received the B.S., M.S., and Ph.D. degrees in computer science and technology from the School of Computer, National University of Defense Technology, Changsha, China, in 1996, 1998, and 2002, respectively.

He is currently a Professor and the Vice President of the Advanced Institute of Big Data, and the Director of Yuzhou Big Data Laboratory, Chongqing, China. His research interests include big data, Internet of Things, and cloud computing.

Qingbo Wu received the Ph.D. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 2010.

He is currently a Professor and the Director of the Industrial Internet of Things Research Institute, Department of New Pattern Network, Peng Cheng Laboratory, Shenzhen, China. His research interests include operating system, Internet of Things, and cloud computing.