# A First Look at Mobile Intelligence: Architecture, Experimentation and Challenges

Ziyi Wang, Yong Cui, and Zeqi Lai


## Abstract

Artificial intelligence (AI) technology makes mobile devices become intelligent objects that can learn and act automatically. Although AI will bring great opportunities for mobile applications, little work has focused on the architecture and the interaction with the cloud. In this article, we present three existing architectures of mobile intelligence in detail and introduce its broad application prospects. Furthermore, we conduct a series of experiments to evaluate the performance of the prevalent commercial applications and intelligent frameworks. Our results show that there is a big gap between Quality of Experience (QoE) requirements and the status quo. So far, we have seen only the tip of the iceberg. We pose issues and challenges to advance the area of mobile intelligence and hope to pave the way for future advancements.


## Introduction

AI has recently attracted significant attention from both industry and academia, as it gives the machine the ability to perceive its environment and take actions. Specifically, it can extract high-level features from image, audio, or other signals automatically, leading to a wide range of applications including computer vision, speech and natural language processing. In the meantime, mobile devices have become both ubiquitous and increasingly powerful. A large volume of multimedia data is being produced and released into mobile cellular networks [1]. Therefore, there is an increasing interest in applying AI to mobile environments. Among existing mobile intelligent applications, Machine Learning (ML) is the most commonly-used technology. Thus, in this article we focus on the intelligent applications based on ML.

Previous works on mobile intelligence have only focused on the hardware platforms or the software models. Specifically, some teams are optimizing mobile hardware chips to support the operation of the ML model, and others try their best to build lightweight models without loss of learning performance. However, there is scant research on the architecture choice of mobile intelligent applications. It is important to understand the existing architectures and optimize it from a more global perspective. To fill this gap, in this article we present the first study on the architecture, experimentation and challenges of mobile intelligence.

First, we divide existing intelligent applications into three different architectures, namely cloud-based, local-based and partial offloading. We provide a technical overview including the introduction of the system architecture, major components and detailed functionalities. This architecture is applicable to all the mainstream ML models. Some researchers have developed intelligent applications using local-based [2, 3] and others have adopted cloud-based [4, 5]. There is also research work concerning combined models, such as making dynamic decisions on local-based or cloud-based [6]. Moreover, some researchers are exploring a new architecture: partial offloading [7, 8]. On this basis, we propose three important QoE metrics to evaluate the performance of these mobile intelligent applications.

Around these metrics, we conduct measurements on prevalent commercial applications and intelligent frameworks. In the process of measuring Google Translate, we have selected two functions, namely Word Lens and Speech-to-speech translation, which represent the local-based and cloud-based architectures, respectively. In the process of measuring TensorFlow's application programming interfaces (APIs), we have developed two applications, namely TF-local-based and TF-cloud-based, which represent the local-based and cloud-based architectures, respectively. Using both black-box testing and white-box testing, we get important metrics such as latency, CPU/RAM utilization and discharge rate. For the data obtained, we sort them out and find the mean and standard deviation. We conclude all experiment results and give some analysis. We find that there is indeed a big gap between QoE requirements and the status quo. Furthermore, we conduct a measurement study on partial offloading architecture using the Inception-v3 model [9]. We find that the best partition point for latency is closely related to network bandwidth rate and the computational capability of the mobile device.

Since there are many difficulties and challenges on the way to mobile intelligence, we propose the key challenges that are most likely to appear and give some insights for future improvement. Specifically, we consider unstable network conditions, considerable energy consumption, privacy disclosure, increasing model complexity and coarse-grained partition of the inference process. To the best of our knowledge, this is the first article that provides a wide overview and experimental evaluation for the existing architectures of mobile intelligent applications.

## Architecture

ML models are particularly well suited for performing perceptual tasks, which can sense, learn from and respond to their environment. Depend-

This project is supported by NSFC (No. 61872211), and the National Key R&D Program of China (No. 2017YFB1010002).

Digital Object Identifier: 10.1109/MNET.2019.1700470

*The authors are with Tsinghua University. Yong Cui is the corresponding author.*


0890-8044/19/$25.00 © 2019 IEEE

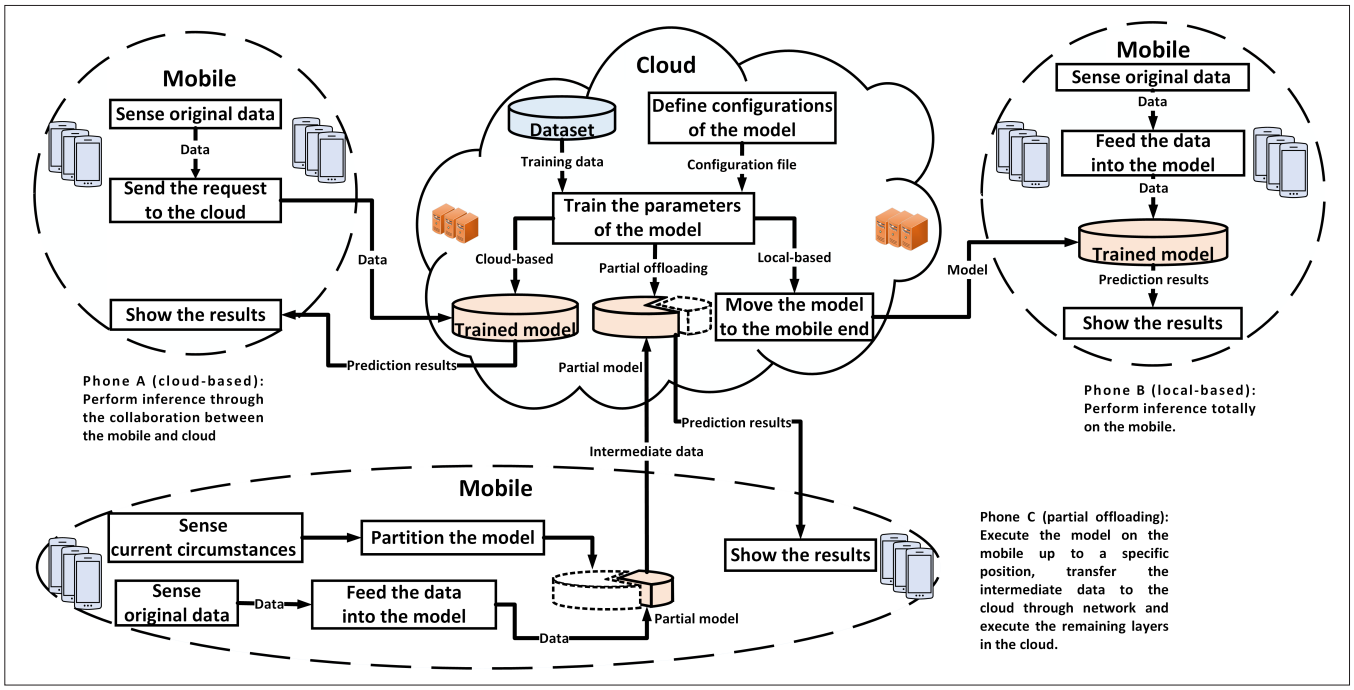


FIGURE 1. Mobile intelligence architecture: cloud-based, local-based, and partial offloading.

ing on the location of these trained models, we divide the existing applications into three different architectures, namely cloud-based, local-based and partial offloading, which are illustrated in Fig. 1. Two major components can be identified in this figure: the mobile client and the cloud server. We first introduce the detailed functionalities of these two components.

**Mobile Client:** The mobile client receives input signals and preprocesses them locally. Then the mobile sends them either to the cloud's ML model, or to the local model. After processing, the mobile client obtains the prediction results and presents the information to the user.

**Cloud Server:** The cloud server has abundant computing resources such as CPU, GPU and TPU, by which the cloud server can complete the training of the ML model. In order to train it, we need to provide the cloud server with the training data and configuration files of the related models. The cloud can also continue to carry well-trained models and provide web APIs to help inference processing.

As shown in Fig. 1, Phones A, B and C represent three typical architectures respectively. Here we briefly describe their workflow and their advantages and disadvantages.

Phone A is the cloud-based phone, which means the mobile client and cloud server work together to make predictions including a training process and an inference process. When training is done on the server, the cloud server obtains the learned parameters for the model. Then we can put the trained model on the server and publish web APIs that mobile devices can use. Since the model is on the server, it is easy to port the application to different platforms. However, inference depends on the network and cannot be done locally on the device.

Phone B is the local-based phone, which means only the mobile makes predictions. We put the trained model into mobile devices and inference locally. We do not need to ask the server over the network during the inference process. It can be faster and more reliable. However, it requires large amounts of CPU and RAM resources on the mobile.

Phone C represents the partial offloading architecture, which is a more flexible and dynamic one. The model is composed of many abstract layers. On one hand, the mobile client partitions the model according to the current circumstances, including network condition, mobile capability and server load. On the other hand, it executes the model up to a specific layer and transfers the intermediate data to the cloud through the network. Then the cloud server executes the remaining layers and sends the prediction results back to the mobile client. This architecture would be more appealing when mobile applications are becoming more and more intelligent.

The architecture above is universal to which the mainstream ML models are all applicable, such as Deep Neural Network (DNN), Reinforcement Learning (RL) models and Generative Adversarial Network (GAN). The only thing we need to do is to make the corresponding replacement for the specific model.

Based on these three architectures, we have seen diverse mobile intelligent assistants such as Google Home, Apple Siri and Microsoft Cortana. All of them use accurate and complex ML technologies to process voice signals. In order to better depict the user experience of these mobile intelligent applications, we introduce three QoE metrics.

**Latency:** Latency refers to the time that elapses between the user's request and the prediction results, including pre-processing, model operation and post-processing. For some real-time interactive intelligent applications, such as mobile Virtual Reality (VR), they require 14ms latency and 60FPS (the phone display refresh rate) [10]. For cloud gaming providers, interaction latency must

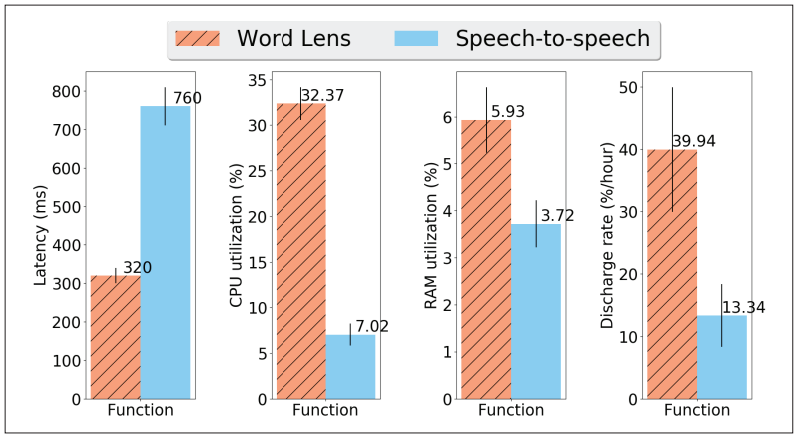


FIGURE 2. Latency, CPU/RAM utilization and discharge rate of Word Lens and speech-to-speech translation.

be kept as short as possible in order to provide a rich experience to cloud gaming players [11].

**Accuracy:** Accuracy refers to the ratio of the number of samples that get the correct results to the total number of samples, which can be used to measure the performance of the model. For some applications requiring a high level of security, such as autonomous driving and road navigation, they require ultrahigh accuracy. Inaccuracy of any prediction result will be life-threatening. Some researchers have proposed that a well-trained DNN can predict the steering angle with an accuracy close to that of a human driver [12].

**Energy:** Mobile devices are energy-constrained. However, running these complex models can introduce considerable computing and communication overhead. Although mobile intelligent applications are very attractive to users, they will most likely choose not to use them if the energy consumption is huge. Therefore, energy efficiency is a desired goal in these mobile intelligent applications.

## Experimentation

There have been many daily-used commercial mobile intelligent applications, such as Google Translate [13]. In addition, many effective open-source libraries and frameworks have also appeared, such as Tensorflow, which provides convenience for developing intelligent applications on mobile devices. We conduct a measurement study to quantitatively describe their QoE level. Specifically, we measure from two perspectives: commercial mobile intelligent applications and mobile intelligent frameworks. We also measure the QoE on the partial offloading architecture based on the Inception-v3 model [9]. We run the applications on a Nexus 6P smartphone. The data is sent to the cloud over the wireless network.

### Measurement on Commercial Mobile Intelligent Applications

We first measure Google Translate, one of the most commonly used mobile applications. When using its speech-to-speech translation function, we need to connect to the Internet. Hence, it belongs to the cloud-based architecture. However, Google Translate's augmented reality feature, Word Lens, is done through offline language packs. Consequently, it belongs to the local-based architecture. Since the source code for the app is not public, we conduct a black box test by recording video. Specifically, we collect 100 images and 100 sentences in English, which are transmitted to the mobile application (Google Translate) in the form of image and voice, respectively. In the process of translating these sentences from English into Chinese, we record it into videos. Then we analyze the video frame by frame and calculate the latency of processing each image or voice. As for the CPU and RAM utilization, we use Emmagee software, which is a simple and easy-to-use Android performance monitoring tool. Users can configure monitoring frequency and get performance statistics eventually. In addition, we leverage the Google Battery Historian tool to inspect the discharge rate of the Android device over time. For the data obtained, we sort them out and find the mean and standard deviation, as shown in the Fig. 2.

From the measurement results, we observe that the Word Lens function achieves lower latency, higher utilization rate of CPU/RAM and higher discharge rate. Since it computes locally based on offline language packs, it is faster but more resource-consuming. In contrast, the speech-to-speech function has larger latency, lower utilization rate of CPU/RAM and lower discharge rate. Since it sends voice to the cloud for processing, the network round-trip latency is larger while the local CPU/RAM resource utilization and discharge rate of the mobile device is lower. After more in-depth analysis, we find that the latencies of the two functions are in the hundred-millisecond level, which is relatively large. In the measurement of Word Lens, we find if we move the smartphone in real time, it cannot process immediately to give the right results and it seems to be stalling. In addition, this function only provides accurate translation for short and simple sentences. Once complex texts appear, the accuracy rate is greatly reduced. Worse still, some words are translated while others are not, which seriously affects the user experience. What's more, CPU utilization of this function has reached 32.37 percent and discharge rate has reached 39.94 percent per hour, leading to high workload and energy consumption of the smartphone. In the measurement of speech-to-speech, we find that although the CPU/RAM resource utilization and discharge rate is lower, the latency is larger. When we gradually weaken the wireless network, the latency can reach even a few seconds, which is unbearable.

### Measurement on Mobile Intelligent Frameworks

TensorFlow is one of the most prevalent frameworks in the deep learning ecosystem. It provides an inference interface that can be called to complete the entire neural network processing including input, running and output. In order to measure its performance, we develop two applications that can classify camera images based on the two kinds of architectures. We call them TF-local-based and TF-cloud-based, respectively. TF-local-based can classify images and display the top results in an overlay on the camera image. It runs the neural network totally on the mobile device. In contrast, TF-cloud-based

is a client-server architecture. We first need to start a Flask web server preparing to receive the mobile's request. When the mobile device captures an image, the application will send it to the server through the network. The server receives the image and runs the neural network model to get the final results. The top classification results will be sent back to the mobile edge through the network and presented to the user. We use the Inception-v3 model trained on the ImageNet Large Visual Recognition Challenge dataset for both applications. The model can differentiate between 1,000 different classes. During the measurement, we collect 100 images from the test set and transmit them to these two applications, respectively. Since we have source code for both applications, we measure the latency by inserting timestamps into the code. Latency refers to the time that elapses between the image request and the prediction result. For the CPU and RAM utilization measurement, we still use Emmagee software. For the battery energy measurement, we still use Google Battery Historian. We also compare latency, CPU/RAM utilization and discharge rate between them, which are illustrated in Fig. 3.

From the experimental results, we can find that the latencies of both applications are more than 3000ms, under which condition real-time object classification is not applicable. More seriously, TF-local-based's CPU and RAM utilization reach 49.96 percent and 10.46 percent, respectively, which seriously affects the normal operation of the smartphone. What's more, its discharge rate is about 35.39 percent per hour, which means this application can only last for 2.83 hours.

Combining all the measurement results, we can find that existing cloud-based and local-based solutions do not meet the needs of users. Although ML brings intelligence to mobile applications, there still exist hundreds of milliseconds or even seconds in terms of latency. CPU and RAM utilization is excessively high and the corresponding energy consumption is increasing. In addition, accuracy of the processing results is far from satisfactory. Hence, there is indeed a big gap between QoE requirements and the status quo.

### Measurement on Partial Offloading Architecture

Since both cloud-based and local-based architectures fail to meet the requirements, we make some measurements on a new architecture: partial offloading. We develop an application based on Tensorflow which can classify the images captured by the phone camera. We partition the Inception-v3 model at the layer granularity. Specifically, we set each layer as a partition point. For the given partition point, the mobile-end executes the computation up to it and transfers intermediate data to the cloud. Next, the cloud executes the remaining layers and transfers the prediction results back to the mobile-end. For each partition method, we send 100 test images to the application and compute the average latency. We make experiments under different network bandwidths (0.2, 1 and 5MB/s) and different mobile phones (Pixel and Nexus 6P) which represent various computation capabilities. Since we have source code for both applications, we break down the end-to-end processing latency, including mobile processing, network communication and server processing. The results are shown in Fig. 4. Each bar represents the end-to-end latency for a specific partition method. The leftmost bar represents the cloud-based architecture while the rightmost bar belongs to the local-based architecture.

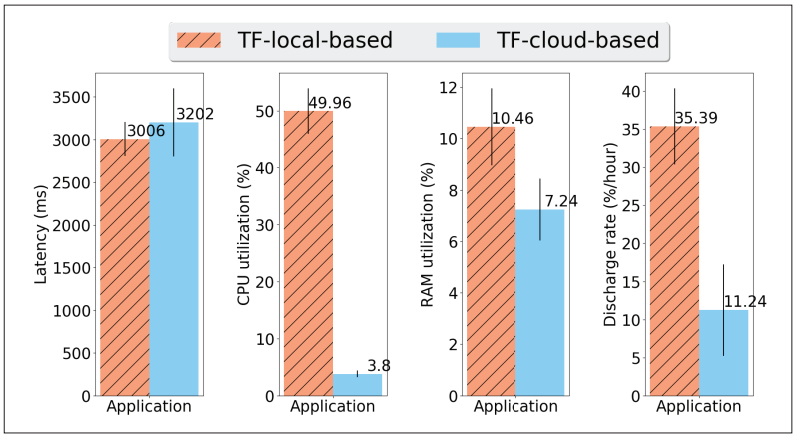


FIGURE 3. Latency, CPU/RAM utilization and discharge rate of TF-local-based and TF-cloud-based.

From the results, we can find that every layer has a totally different computational capacity. The best partition point for latency is different under different circumstances, which is closely related to the network bandwidth rate and the computational capability of the mobile device. We can also find that these existing best results are still high and far from meeting the users' need for latency.

## Challenges

Since there is a huge gap between QoE requirements and the status quo, we should make every effort to bridge it. However, during this process we may face many challenges, as we highlight in this section.

**Network Condition is Unstable, Unsatisfied and Unpredictable:** Network condition is constantly changing and it is difficult to select a fixed formula to characterize it. In addition, for some mobile VR applications, the existing network situation is far away from the QoE requirements. Therefore, it is a challenge to dynamically assign tasks between the mobile and the cloud according to diverse network conditions. A relatively simple method is that we use some regression models to predict the current wireless network conditions based on some real-time probe data.

**Either Local Computing or Communicating with the Cloud will Consume Considerable Energy:** The successful operation of the mobile assistant requires a significant amount of computation and communication overhead. To solve this problem, we need to propose a more efficient mobile energy-saving mechanism. A viable solution is to develop an energy model tool to record data flow and energy flow. It tells us how much energy the model consumes and where the bottleneck exists. Then we can use this information to design new energy-efficient models or to optimize the existing models.

**Cooperation with the Cloud will Inevitably Produce the Issue of Privacy Disclosure:** The data collected by the mobile devices can be very

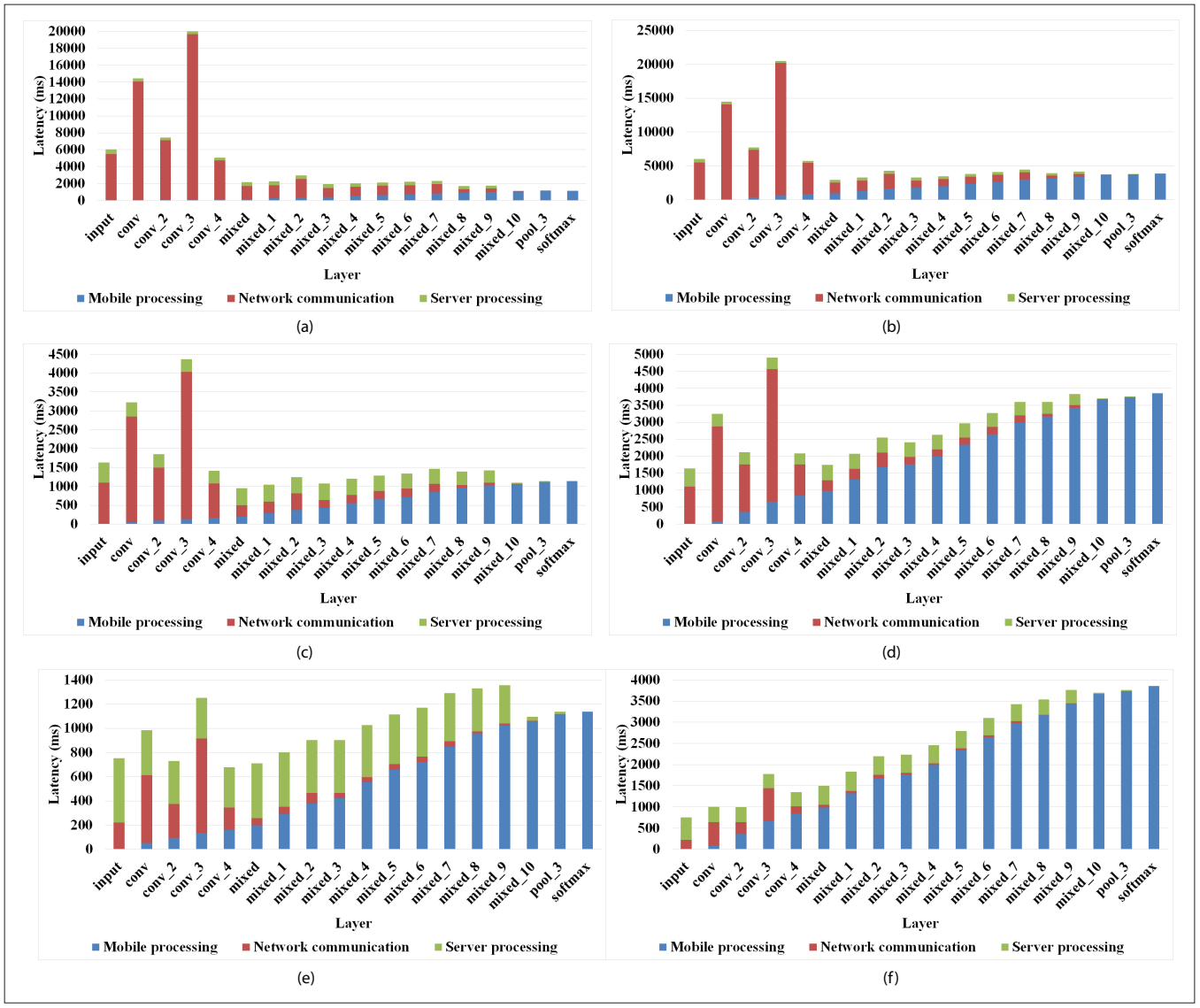


FIGURE4. End-to-end latency when choosing different partition points with different mobile devices and network conditions: a) Pixel, bandwidth = 0.2MB/s; b) Nexus 6P, bandwidth = 0.2MB/s; c) Pixel, bandwidth = 1MB/s; d) Nexus 6P, bandwidth = 1MB/s; e) Pixel, bandwidth = 5MB/s; f) Nexus 6P, bandwidth = 5MB/s.

sensitive and private. Uploading this information onto the cloud without any preprocessing constitutes a great danger to an individual's privacy. In the future, users may have the choice to use a different method to process these data (local-based, cloud-based or partial offloading), depending on which option best suits the situation.

**Model Complexity and Data Size are Increasing:** Take the example of deep learning. The models are becoming more and more complex, with the number of parameters and layers is increasing significantly. Although this change improves the performance of models, it also presents new challenges in adapting resource-constrained mobile to these advanced models. To deal with this challenge, some teams provide hardware solutions. For example, Huawei's new flagship Kirin 970 is Huawei's first mobile AI computing platform featuring a dedicated neural processing unit (NPU). This chip can perform the same AI computing tasks faster and with less power. In the meantime, some teams are working on extending software frameworks for the mobile. For example, Google has announced Tensorflow Lite, which is a lightweight solution for mobile and embedded devices. It can also support hardware acceleration with the Android Neural Networks API.

**Current Partition of the Inference Process is Still Coarse-Grained:** Actually, many models can be split into different kinds of modules which are respectively responsible for different functions. In addition, distribution of latency varies a lot and is closely related to the corresponding workload. For example, DeepMon [14] indicates that the convolutional layers dominate the execution cycles in the VGG-VeryDeep-16 and YOLO model. DeepEye [15] demonstrates that the loading of fully-connected layers is the most time-consuming task across eight different models. Neurosurgeon [7] indicates that for AlexNet, VGG and DeepFace, convolution layers are the most time-consuming; for MNIST, fully-connected layers are the most time-consuming; for Kaldi and SENNA, layers of the model incur similar latency. Faced

with this situation, we should propose a deep integration architecture between mobile and cloud, which splits the functional modules intelligently according to different workloads, models, network conditions and server loads.

## Conclusion

Since there will be more and more applications implemented with ML technology on the mobile, understanding the existing architectures of the mobile intelligent applications is significant for both industry and academia. In this article, we present a thorough overview of the mobile intelligence by introducing its architectures, components and functionalities, followed by an experimental study that evaluates the prevalent commercial applications and intelligent frameworks. All tested services suffer performance limitations. Our results show that there is a big gap between QoE requirements and the status quo. Finally, we conclude experiment results and propose challenges. To the best of our knowledge, this is the first article that provides a wide overview and experimental evaluation for the existing architectures of the mobile intelligent applications. As for future work, we intend to do more detailed measurements, identify the bottleneck and propose advanced mobile intelligence architectures.

To the best of our knowledge, this is the first article that provides a wide overview and experimental evaluation for the existing architectures of the mobile intelligent applications. As for future work, we intend to do more detailed measurements and indentify the bottleneck and propose advanced mobile intelligence architectures.

## References

[1] J. Liu *et al.*, "Device-to-Device Communication for Mobile Multimedia in Emerging 5G Networks," *TOMCCAP*, vol. 12, no. 5s, 2016, p. 75.
[2] N. D. Lane, P. Georgiev, and L. Qendro, "Deepear: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning," *Proc. UbiComp*, ACM, 2015, pp. 283–94.
[3] I. McGraw *et al.*, "Personalized Speech Recognition on Mobile Devices," *Proc. IEEE ICASSP*, 2016, pp. 5955–59.
[4] P. Peng *et al.*, "DeepCamera: A Unified Framework for Recognizing Places-of-Interest Based on Deep Convnets," *CIKM*, ACM, 2015, pp. 1891–94.
[5] C. Zhang, X. Ouyang, and P. Patras, "ZipNet-GAN: Inferring Fine-Grained Mobile Traffic Patterns via a Generative Adversarial Neural Network," *Proc. CoNEXT*. ACM, 2017, pp. 363–75.
[6] X. Ran *et al.*, "Delivering Deep Learning to Mobile Devices via Offloading," *Proc. VR/AR Network@SIGCOMM*. ACM, 2017, pp. 42–47.
[7] Y. Kang *et al.*, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," *Proc. ASPLOS*, ACM, 2017, pp. 615–29.
[8] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services," arXiv preprint arXiv:1801.08618, 2018.
[9] C. Szegedy *et al.*, "Rethinking the Inception Architecture for Computer Vision," *CVPR*, 2016, pp. 2818–26.
[10] Z. Lai *et al.*, "Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices," *Proc. MobiCom*. ACM, 2017, pp. 409–21.
[11] R. Shea *et al.*, "Cloud Gaming: Architecture and Performance," *IEEE Network*, vol. 27, no. 4, 2013, pp. 16–21.
[12] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," arXiv preprint arXiv:1604.07316, 2016.
[13] Y. Wu *et al.*, "Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation," arXiv preprint arXiv:1609.08144, 2016.
[14] L. N Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-Based Deep Learning Framework for Continuous Vision Applications," *MobiSys*. ACM, 2017, pp. 82–95.
[15] A. Mathur *et al.*, DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware," *Proc. MobiSys*. ACM, 2017, pp. 68–81.

## Biographies

Ziyi Wang (wangziyi0821@gmail.com) is now a Ph.D. student in the Department of Computer Science and Technology, Tsinghua University. His supervisor is Prof. Yong Cui. His research interests include mobile systems and mobile computing.

Yong Cui (cuiyong@tsinghua.edu.cn) received his B.E. and Ph.D. degrees in computer science and engineering from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a full professor at Tsinghua University and Co-Chair of IETF IPv6 Transition WG Softwire. His major research interests include mobile wireless Internet and computer network architecture. Having published more than 100 papers in refereed journals and conferences, he received the National Award for Technological Invention in 2013, and the Influential Invention Award of the China Information Industry in both 2012 and 2004. Holding more than 40 patents, he has authored three Internet standard documents, including RFC 7040 and RFC 5565, for his proposal on IPv6 transition technologies. He serves on the Editorial Board on both IEEE TPDS and IEEE TCC.

Zeqi Lai (laizq13@mails.tsinghua.edu.cn) is now a Ph.D. student in the Department of Computer Science and Technology, Tsinghua University. His supervisor is Prof. Yong Cui. His research interests include mobile computing and cloud storage.