Deep neural networks (DNNs) have enabled many emerging artificial intelligence (AI) applications, such as augmented reality (AR), virtual reality (VR), Google Translate, and so on, and there is tremendous demand for running these applications on mobile devices. However, DNNs are computationally intensive, which creates many technical challenges for running DNNs on battery-powered mobile devices.

To address these challenges, many companies, such as Qualcomm, Huawei, and Samsung, have developed AI accelerators called neural processing units (NPUs). Different from traditional CPUs, which are good for general computations, NPUs are designed to accelerate AI and deep learning tasks, such as DNNs, which involve millions and even billions floating-point computations. As a result, NPUs can substantially reduce the processing time of deep learning on mobile devices. For example, on the Mate 10 Pro, the running time of the Visual Geometry Group (VGG) model can be reduced from 2,600 to 120 ms by using an NPU instead of a CPU. Moreover, the power consumption of the NPU is only 500 mW, which is about 83% lower than that of the CPU (3 W).

However, NPUs have some fundamental limitations. First, differing from CPUs, which use 32-bit floating-point numbers for computation and storage, NPUs use 16- and 8-bit floating-point numbers. Second, only a limited number of operations are supported by NPUs, and hence, some operations required in deep learning models have to be approximated. Finally, NPUs have their own memory space, and DNNs must be loaded into NPUs to be executed. Since the memory space of NPUs is small (about 200 MB on the Mate 10 Pro), most DNNs have to be compressed to

Although deep neural networks (DNNs) can require less processing time when running on neural processing units (NPUs) instead of CPUs, their accuracy can degrade. We point out three research directions to improve DNN performance on mobile devices with NPUs.
run on NPUs. Due to these limitations, the accuracy of running DNNs on NPUs may be reduced.

Both processing time and accuracy are critical in many mobile applications. For example, a flying drone needs to accurately detect nearby obstacles in real time to avoid crashing, and many VR/AR applications need to interact with users through gestures and body posture recognition. NPUs can accelerate deep learning, but they incur accuracy loss. In this article, to improve the performance of running DNNs on mobile devices with NPUs, we identify the challenges, propose solutions, and discuss future work along three research directions, that is, model retraining, model partitioning, and computation offloading.

UNDERSTANDING NPUs

To have a better understanding of NPUs, we conducted experiments with three kinds of phones: Pixel 6, OnePlus 7, and Mate 10 Pro. Similar to Tan and Cao, the experiments were based on four DNNs and their public data sets: VGG on the Labeled Faces in the Wild data set, Visual Object Classification (VOC)-Net on the VOC data set, ResNet-50 on the VOC data set, and You Only Look Once (YOLO) on the Microsoft Common Objects in Context (COCO) data set.

Figure 1 compares the processing time of running DNNs on CPUs and NPUs on different mobile devices. As shown in the figure, NPUs are about 20 times faster than CPUs on these three mobile devices. For example, to run YOLO on the Pixel 6, an NPU takes about 110 ms, while a CPU takes about 4,100 ms. However, as demonstrated in Figure 2, NPUs suffer from accuracy loss, and the loss varies based on the deep learning model. For instance, compared to a CPU, using an NPU on the OnePlus 7 has similar accuracy when running VGG and ResNet, a 30% accuracy loss when running VOC-Net, and a 79% accuracy loss when running YOLO.

The accuracy loss is due to the limitations of NPUs, and it is also related to the complexity of the deep learning model. More specifically, VGG compares only the similarity between two feature vectors extracted from the face images. The vectors are classified to the same person if the similarity is above a predefined threshold. Although NPUs may introduce some small error that changes values in the feature vector, the relationship between the similarity and the threshold does not change too much, thus maintaining a similar level of accuracy as the CPU. However, YOLO is much more complex than VGG, and it uses more information in the feature vectors to identify and locate multiple objects in the images. Each value in the feature vector represents the category, the location, or the size of an object. A small error introduced by an NPU can change the prediction completely and, hence, significantly affect the accuracy. Another observation is that the accuracy loss is different for different mobile devices. The accuracy of running VGG on an NPU with the OnePlus 7 is about 40% higher than the Pixel 6. This is because manufacturers design their NPUs differently and use different toolkits to optimize DNNs. As a result, the accuracy loss varies for different mobile devices.

Based on these experimental results, we observe that running DNNs on NPUs may not always be the best option, especially when accuracy is more important than processing time. To improve the

![Figure 1](image-url)
performance of running DNNs on mobile devices with NPUs, one research direction is to retrain the model with lower-precision floating-point numbers since they are the most significant difference between NPUs and CPUs. Another research direction is to leverage model partitioning techniques to decompose DNN architectures into different layers running on heterogeneous processors; that is, an NPU is used to reduce the processing time of the computationally intensive layers, while a CPU is used for maintaining higher accuracy. These two research directions focus on accelerating DNNs by exploiting heterogeneous processors on mobile devices. The third research direction is to leverage computation offloading techniques to determine where to run DNNs, based on the network condition, the special characteristics of an NPU, and the optimization goal. In the following sections, we identify challenges and propose solutions along these three research directions.

**MODEL RETRAINING**
The major difference between NPUs and CPUs is the precision of the floating-point numbers. Advanced DNNs are trained and tested using 32-bit floating-point numbers on powerful desktop CPUs and GPUs, and they are not designed for running on NPUs. To address this issue, one solution is to retrain the DNNs, and there is a lot of research focusing on training DNNs with low-precision numbers. For example, Wang et al. proposed to use 8-bit floating-point numbers to train DNN models. Sun et al. proposed gradient scaling and two-phase rounding techniques to minimize quantization errors during training, and they successfully used 4-bit floating-point numbers to train a DNN for classification. Yang et al. modified the learning rate schedule to reduce the low-precision training time. However, these works focus on classification tasks with small images (for example, using the Canadian Institute for Advanced Research-100 data set). In contrast, there are other challenging tasks in the real world, such as face recognition and object detection, which require more advanced DNNs, and these techniques cannot be directly applied to retrain various advanced DNNs for NPUs.

In this article, we retrain VOC-Net by using two different low-precision training strategies in the TensorFlow framework: Float16 and Mixed Float16. Float16 means that 16-bit floating-point numbers are used in both computation and data storage during training. In Mixed Float16, the computation is performed using 16-bit floating-point numbers, but the data and parameters are stored using 32-bit floating-point numbers.

Figure 3 presents the accuracy of the retrained VOC-Net using Float16. The figure does not show the processing time since the retrained model does not change the processing time. As can be seen from the figure, the accuracy of running the retrained model does not change the processing time. As can be seen from the figure, the accuracy of running the retrained VOC-Net using Float16 is about 28% lower than that of running the original model on the CPU. This is because VOC-Net is sensitive to the floating-point number precision. When the model converges, the loss of the retrained model is higher than the original model. Therefore, the accuracy of the retrained model is lower on the CPU. Compared to the original model, the accuracy of the retrained model is higher on an NPU. This is because Float16 emulates the floating-point number precision used on the NPU, and the retrained VOC-Net is better adapted to the NPU than the original model.

Figure 4 gives the accuracy of the retrained VOC-Net using Mixed Float16. Compared to Figure 3, the retrained model can achieve similar accuracy to the original model on a CPU. This is because Mixed Float16 uses 32-bit floating-point numbers for data storage, and
the impact of numerical instability is much lower. As shown in the figure, the accuracy of running the retrained VOC-Net on an NPU is about 7% higher on the Pixel 6 and 5% higher on the OnePlus 7, but it is still much lower than that on the CPU. This is because Mixed Float16 performs computations using only 16-bit floating-point numbers and stores the data with 32-bit floating-point numbers. Differing from the CPU, the NPU uses 16-bit floating-point numbers for both data storage and operations. Such a difference may lead to numerical instability in the NPU, and thus, running the retrained model on the NPU cannot achieve similar accuracy as that on the CPU. From the experiment, we can see that model retraining cannot improve the accuracy too much. Since NPUs support only 16-bit floating-point numbers and 8-bit integers, floating-point numbers may underflow or overflow when running some DNNs on NPUs.

MODEL PARTITIONING
The basic idea of model partitioning is to decompose a DNN architecture into different layers running on heterogeneous processors; that is, an NPU is used to reduce the processing time of computationally intensive layers, while a CPU is used for maintaining higher accuracy. The model partitioning technique has been leveraged to reduce computation time. For example, DeepX\(^7\) divides DNNs into different blocks that can be efficiently run on CPUs and GPUs. Neurosurgeon\(^8\) optimizes energy by running the first few layers of DNNs on local devices to reduce the data size and offloading the remaining layers to a server for processing. Mao et al.\(^9\) proposed to reduce processing time by distributing partitioned DNN models across mobile devices. Teerapittayanon et al.\(^10\) proposed to reduce processing time by adding early exit points to DNNs, where a few layers are inserted into DNNs to estimate the accuracy of the result. When the accuracy is above a certain threshold, DNN execution will be stopped, and the result will be returned. However, none of these works considers the low-accuracy problem introduced by NPUs.

There are two important factors in model partitioning: accuracy loss and layer processing time. To measure them, we randomly selected 4,000 images from the COCO data set and ran the YOLO model for object detection using the Mate 10 Pro. We ran one layer of YOLO on an NPU, while the remaining layers were run on a CPU. As detailed in Figure 5, running layer \(P_2\) (a pooling layer) on the NPU while other layers are run on the CPU can reduce the processing time by 6% and incur a 4% accuracy loss. Intuitively, a layer, for example, \(C_6\) (a convolutional layer), should be executed on the NPU if the processing time can be substantially reduced with little or no accuracy loss. A layer
(for instance, $C_{17}$) should be executed on the CPU if executing it on the NPU has a much higher accuracy loss with little processing time reduction.

However, most layers (such as $C_{23}$) are not in these two extreme cases, and hence, it is hard to determine where to run them. When considering the overlapping effects of running multiple layers, the decision is harder. For instance, the accuracy loss of running $C_6$ and $C_{22}$ on the NPU while other layers run on the CPU is 0.08, which is not equivalent to the sum of their accuracy losses (that is, 0.04). The accuracy loss of running the DNN with a layer combination depends on many factors, such as the number of additions and multiplications performed in each layer and the memory space occupied by the input/output data. Due to the complex relationship between the accuracy loss and these factors, it is difficult to derive an equation to estimate the accuracy of running the DNN model with a layer combination.

To address this problem, we propose heuristic-based algorithms to find layer combinations that satisfy application requirements for processing time and accuracy. For example, in a flying drone application, detecting obstacles accurately with a short processing time (in real time) is critical to avoid crashing. For these applications, maximizing the accuracy under some time constraint is more important, and we propose a maximum accuracy algorithm to solve it. The basic idea is to move layers with a higher accuracy loss from an NPU to a CPU, as follows:

1. Initially, all layers are run on the NPU.
2. The layers are sorted in descending order based on their accuracy loss.
3. Starting from the first one, the layers are moved from the NPU to the CPU until the processing time constraint is not satisfied.

For applications such as unlocking a smartphone and making a payment through face recognition, accuracy is more important than processing time. For these applications, we solve the problem of minimizing the processing time while ensuring that the accuracy is above a certain threshold by proposing a minimum time algorithm. The basic idea is to move layers with a longer processing time from the CPU to the NPU, as follows:

1. Initially, all layers are run on the CPU.
2. The layers are sorted in descending order based on their processing time.
3. Starting from the first one, the layers are moved from the CPU to the NPU until the accuracy requirement is not satisfied.

To evaluate the performance of the maximum accuracy and minimum time algorithms, we compare them with an all-CPU approach (that is, the DNN model is always run on a CPU) and an all-NPU method (that is, the DNN model is always run on an NPU). The experiment was conducted on the Mate 10 Pro, which is equipped with an NPU. Its CPU is based on an octa-core processor (four little cores and four big cores). The results are provided in Figure 6. In Figure 6(a), we did not use the all-CPU approach since...
the processing time of running YOLO on the CPU takes about 3.4 s, which is longer than the time requirement. As the time requirement increases, the maximum accuracy algorithm can improve the accuracy by moving more computations to the CPU, although the all-NPU approach has the same accuracy. As shown in Figure 6(b), the minimum time algorithm outperforms the all-NPU and all-CPU techniques. Note that the all-NPU method can reach an accuracy of only 29%, and it is not employed after the accuracy requirement reaches 30%. Our proposed algorithms can significantly improve performance compared to the all-CPU and all-NPU approaches by running computationally intensive layers on an NPU to save time while running precision-sensitive layers on a CPU to maintain high accuracy.

Figure 7 illustrates the processing time of each approach. Since the proposed algorithms run different parts of DNNs on heterogeneous processors, data are moved between the main memory and the NPU memory. The data transmission time cannot be ignored because the NPU processing
time is short and a large amount of data are transmitted between the main memory and NPU. As can be seen in the figure, the data transmission time occupies about 10% of the total processing time in the maximum accuracy and minimum time algorithms. Since this data transmission time is only a small fraction of the total processing time, it does not change the benefit of moving time-consuming layers from a CPU to an NPU. Thus, even considering this data transmission overhead, our proposed algorithms still outperform the all-CPU and all-NPU approaches, as shown in Figure 6.

**COMPUTATION OFFLOADING**

The aforementioned two techniques accelerate deep learning by exploiting heterogeneous processors on mobile devices. Such decisions may be due to limited network connections, a lack of server support, and privacy issues. For example, some users prefer processing sensitive data locally, and many smart health applications belong to this category. For many other mobile apps, such as AR and cognitive assistance, users may be willing to achieve better performance by offloading computation to an edge server.\(^{13,14,15}\)

Since the server has more computation capacity, more advanced deep learning models with high accuracy can be executed quickly. However, when the network condition is poor and when the data size is large, which is usually true for video analytics, the offloading approach may take longer because of data transmission delays. On the other hand, the NPU-based approach is faster but with less accuracy. We propose an offloading framework to combine these two approaches for real-time video analytics on mobile devices, with the goal of maximizing accuracy under some time constraint.

In our offloading framework, video frames are first processed on an NPU, which is very fast with negligible delay, and only frames with low classification accuracy are offloaded to the server to improve accuracy. Thus, the key problem is to determine the classification accuracy, and we rely on the confidence score of running DNNs on NPUs, which is computed based on the extracted feature vector of a DNN. If the confidence score is higher than a threshold, the classification result on an NPU is most likely correct and can be directly used; otherwise, the data should be offloaded for further processing to improve accuracy.

The major challenge is that the confidence score of many advanced DNNs cannot accurately predict classification results. To illustrate the problem, we conduct an experiment in which AlexNet is used to classify some video frames randomly selected from the Fudan–Columbia Video Dataset. The frames are divided into 10 bins with a 10% confidence interval. For each bin, we compute the accuracy of running AlexNet on these frames. The result is provided in Figure 8(a).

Ideally, the confidence score and the classification accuracy should follow a similar distribution. Then, the classification with a confidence score of 0.9 is more likely to be correct than that with a score of 0.5. However, as shown in Figure 8(a), the accuracy remains 0.5 for frames with a confidence score

![FIGURE 8. The accuracy versus the confidence score. The (a) original confidence score and (b) calibrated confidence score.](image-url)
much higher than 0.5 (for example, 0.9). To address this problem, we have to calibrate the confidence score. Suppose the DNN model generates n different confidence scores \(x_1, x_2, \ldots, x_n\); \(n\) logistic models will be trained for calibration, and each model \(i\) is used to calibrate a confidence score \(x_i\). More specifically, model \(i\) takes the \(n\) confidence scores as input and outputs a new confidence score \(x_i\). Figure 8(b) explains why the calibrated confidence score is more effective. As the confidence score increases from zero to one, the accuracy increases from 0.11 to one.

**Offloading scheduling**

With the confidence score calibration, our framework determines which frames should be offloaded for further processing. Although there are some existing works\(^{10,16}\) on this topic, they rely on uncalibrated confidence scores, which is not effective for some DNNs, and a fixed confidence score threshold is used to determine which frames should be offloaded. However, the offloading decision depends on many factors, such as the network bandwidth and the data size of the frames, and thus, a fixed confidence score threshold may not work well.

To address this issue, we propose a confidence-based offloading (CBO) algorithm. CBO adaptively adjusts the confidence score threshold based on the network condition, the data size of video frames, and the DNN. In CBO, the frames with a lower confidence score should be offloaded to increase accuracy if there is available bandwidth. CBO can also reduce the resolution of the offloaded frames to upload more frames with a low confidence score. Suppose \(n\) frames have been processed locally on an NPU. If a frame \(i\) with confidence score \(p_i\) is offloaded to the server in resolution \(r_i\), CBO computes the offloading time \(T_i\) and the accuracy improvement \(A(p_i, r_i)\) based on the frame data size, network condition, and DNN accuracy information. Let \(f(i, T)\) denote the maximum accuracy improvement by offloading the first \(i\) frames within time \(T\). Our goal is to maximize \(f(n, T)\). The overlapping subproblems can be represented as \(f(i, T) = \max_{j} f(i-1, T-T_j) + A(p_i, r_i)\). CBO solves the problem by using a dynamic programming algorithm and selects a confidence score threshold to make offloading decisions for these \(n\) frames.

Figure 9 compares the performance of local, server, CBO, and CBO-w/o approaches under different network conditions. In the local approach, the data are processed only on the local NPU, and the performance does not change with the network condition. The local approach has the same accuracy as CBO when the bandwidth is low since most frames have to be processed locally. However, as the bandwidth increases, CBO outperforms the local technique significantly because it can improve the accuracy by offloading misclassified frames to the server for further processing. The server approach offloads all data to the server for processing. When the network condition is poor, its performance is much worse than CBO. This is because the frames have to be offloaded at an extremely low resolution to satisfy the time constraint, and running DNNs with low-resolution frames cannot achieve high accuracy due to information loss. CBO-w/o is the same as CBO except that it uses an uncalibrated confidence score to make offloading decisions. As can be seen, CBO outperforms CBO-w/o since the uncalibrated confidence score cannot accurately estimate the correctness of the classification result on the NPU.

**DISCUSSIONS AND FUTURE WORK**

In this article, we pointed out three research directions to accelerate deep learning on mobile devices with NPUs. There are still many important issues...
for future research along these directions. In model retraining, we leveraged the low-precision retraining method TensorFlow framework to retrain VOC-Net. However, the result shows that the accuracy cannot be improved significantly after retraining. To address this issue, one possible research direction is to normalize the data and restrict the data range. For example, BatchNorm layers can be added before or after the activation layers to normalize the data. Moreover, since NPUs support only a limited set of operations, more research should be conducted to design DNNs and retrain existing DNNs based on these operations.

In computation offloading, we proposed a CBO algorithm to maximize the accuracy by determining which frames should be offloaded to the server for further processing. The current offloading framework considers only the tradeoff between processing time and accuracy. However, running computationally intensive DNNs on mobile devices also consumes a large amount of energy. Based on our experiment conducted on the Mate 10 Pro, the power consumption of the NPU is 500 mW, which is much less than the wireless interface. The offloading-based approach can achieve higher accuracy, but it may cost more energy under poor network conditions. On the other hand, the NPU-based approach is energy efficient but with less accuracy. Therefore, one future research direction is to study the tradeoffs among energy consumption, processing time, and accuracy in computation offloading.

In model partitioning, we discussed how to decompose the DNN architecture into different layers running on a CPU and an NPU so that application requirements for accuracy and processing time could be satisfied. Although our proposed maximum accuracy and minimum time algorithms can outperform all-NPU and all-CPU approaches, they try only a few layer combinations and select the best one based on heuristics. Since there are many layer combinations for a DNN, a better solution may exist. Since measuring the accuracy of a layer combination is time-consuming, it is impractical to measure the accuracy for a large number of layer combinations. To efficiently search more layer combinations, the major challenge is to build a model to estimate the accuracy loss of a layer combination. This model is nonlinear and depends on many factors, such as the accuracy loss of the layers, the number of layers running on the NPU, and the parameter size of the layers. As future research, machine learning techniques can be leveraged to estimate the accuracy loss and design algorithms to search for better layer combinations.

Another research direction is to consider the heterogeneity inside CPUs, such as the current big.LITTLE core architecture, where big cores can provide better performance at the cost of more energy consumption, and little cores are slower with less power consumption. Considering various application requirements for processing time and energy, running all layers on either big cores or little cores may not be the best option. Therefore, instead of using only one type of core, we will study the performance and energy tradeoffs of big and little cores when running DNN models and optimize the model partition algorithms considering the big. LITTLE core architecture.
ACKNOWLEDGMENT
This work was supported, in part, by the National Science Foundation, under grants CCF-2125208 and CCF-2215043.

REFERENCES

ABOUT THE AUTHORS
TIANXIANG TAN is a machine learning engineer at TikTok, Mountain View, CA 94041 USA. His research interests include mobile cloud computing, edge computing, and deep learning. Tan received a Ph.D. in computer science from The Pennsylvania State University. He is a Student Member of IEEE. Contact him at txt51@psu.edu.

GUOHONG CAO is a distinguished professor of computer science and engineering at The Pennsylvania State University, State College, PA 16802 USA. His research interests include mobile computing, wireless networks, machine learning, wireless security and privacy, and the Internet of Things. Cao received a Ph.D. in computer science from The Ohio State University. He is a Fellow of IEEE and the American Association for the Advancement of Science. Contact him at gcao@cse.psu.edu.