MetaNetwork: A Task-agnostic Network Parameters Generation Framework for Improving Device Model Generalization

Zheqi Lv†, Feng Wang‡, Kun Kuang†, Yongwei Wang†, Zhengyu Chen†, Tao Shen†, Hongxia Yang†, Fei Wu†

†Zhejiang University, Hang Zhou, China
‡DAMO Academy, Alibaba Group, Hang Zhou, China

Abstract

Deploying machine learning models on mobile devices has gained increasing attention. To tackle the model generalization problem with the limitations of hardware resources on the device, the device model needs to be lightweight by techniques such as model compression from the cloud model. However, the major obstacle to improve the device model generalization is the distribution shift between the data of cloud and device models, since the data distribution on device model often changes over time (e.g., users might have different preferences in recommendation system). Although real-time fine-tuning and distillation method take this situation into account, these methods require on-device training, which are practically infeasible due to the low computational power and a lack of real-time labeled samples on the device.

In this paper, we propose a novel task-agnostic framework, named MetaNetwork, for generating adaptive device model parameters from cloud without on-device training. Specifically, our MetaNetwork is deployed on cloud and consists of MetaGenerator and MetaStabilizer modules. The MetaGenerator is designed to learn a mapping function from samples to model parameters, and it can generate and deliver the adaptive parameters to the device based on samples uploaded from the device to the cloud. The MetaStabilizer aims to reduce the oscillation of the MetaGenerator, accelerate the convergence and improve the model performance during both training and inference. We evaluate our method on two tasks with three datasets. Extensive experiments show that MetaNetwork can achieve competitive performances in different modalities.

1 Introduction

Conventional studies on deep neural networks mainly focus on improving the model performance with large models (LeCun et al. 1998; Simonyan and Zisserman 2015; He et al. 2016). With the development of hardware and deep learning, more and more smart devices have undertaken the task of intelligent computing in various environments. Smart devices can upload real-time data to the cloud at every moment and use powerful cloud models for inference. However, the load threshold of the cloud server does not allow the device to upload data all the time (Yao et al. 2022; Yan et al. 2022). Also, sometimes the device cannot access the Internet. Therefore, it is necessary to deploy deep learning models on devices. Considering the computing power and storage space on devices, these models must be lightweight.

With the advancement of smart devices and lightweight neural networks, the deployment of deep neural networks on smart devices (e.g., mobile phones, smart security cameras) has become a reality. Commonly used lightweight models deployed on devices include MobileNet (Howard et al. 2017), Sandler et al. 2018), Howard et al. 2019) for image classification, DIN (Zhou et al. 2018), SASRec (Kang and McAuley 2018), GRU4Rec (Hidasi et al. 2016), etc. for recommendation. We term these lightweight models as primary models which are deployed on the device for sequential recommendation or image classification tasks. In general, the data on all devices during this period is collected at regular intervals (days or weeks, named update cycles), and then labeled on the cloud uniformly. These lightweight models are compressed by larger models (Teerapittayanon, McDanel, and Kung 2017; Osia et al. 2020; Wang et al. 2018) or trained directly with these data and deployed on devices (Yao et al. 2022).

However, the parameters of the primary model remain fixed between update cycles, while the distribution of the data input to the device is dynamic and sometimes it may change drastically. For instance, in computer vision tasks, the light and angle of an image input to a smartphone or smart security camera are constantly changing, and in recommendation system tasks, users may have different preferences in a short time. The primary model is the optimal model trained with data from multiple distributions, but it may not be optimal for data from a specific distribution (Yan et al. 2022; Marfoq et al. 2021; Mills, Hu, and Min 2021). To eliminate the distribution shift between the data of cloud and device models and find the optimal parameters of the primary model on data from each distribution, real-time on-device fine-tuning or model compression or other knowledge transfer approaches with local samples turn to be reasonable methods. Yet it can be vulnerable to overfitting due to the limited amount of labeled data on each device. Besides, in many scenarios like some computer vision applications, each device cannot even obtain labeled data in a short time. Moreover, on-device training introduces backward propagation, which consumes more computing resources, thus leading to considerable power consumption of smart devices, and imposing extremely stringent...
requirements on the chips deployed on the device. Therefore, these on-device training methods are not feasible in practical application.

To address these challenges, we are motivated to leverage hypernetwork \cite{Ha, Dai, and Le 2017} and dynamic convolution filter \cite{Jia et al. 2016, Zhou et al. 2021}. Concretely, we develop a device-cloud collaboration framework named MetaNetwork which consists of MetaGenerator and MetaStabilizer modules. First of all, our method collects data from all devices over a period of time and labels them, then trains a MetaGenerator with these labeled data. The MetaGenerator on the cloud learns a mapping from samples to model parameters, and it can generate and deliver model parameters to the device based on the trigger samples which are uploaded to the cloud. The trigger samples uploaded are different in different tasks. In computer vision tasks such as facial expression recognition, the trigger sample is the first frame of images captured by the camera in a session. In recommendation system tasks such as sequential recommendation, the trigger sample is a click sequence consisting of dozens of items recently clicked by the user. After receiving the model parameters generated by MetaGenerator, the smart devices update the primary model. This primary model will do inference on the device for a short time in the future. Please note that the parameters of the primary models are generated by the MetaGenerator through feedforward computation, so it only causes an extremely low time delay.

Supervised learning algorithms perform the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions for a particular problem \cite{Amari et al. 2003, Svetnik et al. 2003}. Combining multiple hypotheses is always a good approach to form a better hypothesis \cite{Lison et al. 2020, Zheng et al. 2019, Tekin, Yoon, and van der Schaar 2016}. Inspired by this philosophy, we propose MetaStabilizer, where multiple parameters generators rather than a single generator are trained in the same way. And the multiple parameters generators are weighted in a self-similar manner to obtain a self-corrected generator. The self-corrected generator will be used to predict parameters of the primary model on the device. MetaStabilizer module can improve the stability of MetaGenerator.

We divide the primary model into static layers and dynamic layers. The parameters of the static layers are fixed and the parameters of the dynamic layers are dynamic and generated by MetaNetwork. In summary, our main contributions can be summarized as:

- We propose a novel device-cloud collaboration mechanism to solve the defects of the poor generalization ability of the primary model on the device.
- We design a universal MetaGenerator that can convert samples of different modalities into model parameters. We also optimize the loss function to make it more suitable in the practical device-cloud collaborative system.
- We design a MetaStabilizer module to reduce the oscillation of MetaGenerator and improve the performance during both training and inference phases.
- We evaluate the proposed MetaGenerator and MetaStabilizer on three datasets and tasks with different modalities including images and text. We also verify the effectiveness and generalizability of our method.

2 Related Work

2.1 Lightweight Neural Network

The performance of traditional convolutional neural networks is already strong enough, but when the model is deployed on the device, the storage space and computing power of the device need to be considered. Therefore, many lightweight CNN models \cite{Iandola et al. 2016, Howard et al. 2017, Sandler et al. 2018, Howard et al. 2019, Zhang et al. 2018, Ma et al. 2018, Tan and Le 2019, Han et al. 2020, Li et al. 2020} have been proposed in recent years. SqueezeNet \cite{Iandola et al. 2016} reduces the number of parameters by making extensive use of fire modules with $1 \times 1$ convolutions. MobileNetV1 \cite{Howard et al. 2017} decomposes traditional convolution kernels into depth-wise convolution kernels and point convolution kernels. MobileNetV2 \cite{Sandler et al. 2018} introduces inverted residuals and linear bottlenecks. MobileNetV3 \cite{Howard et al. 2019} builds the network based on AutoML and manually fine-tunes the optimization to obtain the best network structure, and also improves the performance and efficiency of the activation function. EfficientNet \cite{Tan and Le 2019} uses a neural network architecture method with a hybrid scaling method. GhostNet \cite{Han et al. 2020} applies a linear transformation layer with fewer parameters to generate ghost feature maps. These models all achieve good performance with a small number of parameters and FLOPs, but the performance and generalization ability are still limited by the number of parameters.

2.2 HyperNetwork

HyperNetwork \cite{von Oswald et al. 2020, Shamsian et al. 2021, Zhang, Ren, and Urtasun 2019, Chang, Flokas, and Lipson 2020, Xian et al. 2021, Brock et al. 2018, Navon et al. 2021, Ha, Dai, and Le 2017} is a neural network that generates its parameters for another neural network. When HyperNetwork was first proposed by Ha et al. \cite{Ha, Dai, and Le 2017}, it was to achieve model compression by reducing the amount of parameters that the model needs to train. Subsequently, the research on HyperNetwork gradually increased. Oscar et al. \cite{Chang, Flokas, and Lipson 2020} studied parameter initialization for HyperNetwork. At the same time, the research of HyperNetwork is applied to various tasks, such as one-shot learning \cite{Brock et al. 2018}, continual learning \cite{von Oswald et al. 2020}, graph learning \cite{Zhang, Ren, and Urtasun 2019}, meta-learning \cite{Xian et al. 2021}, federated learning \cite{Shamsian et al. 2021}, etc. In our work, we adapt HyperNetwork to device-cloud system with unique challenges arising from the problem setup.

2.3 Dynamic Filter Network

The Dynamic Filter Network (DFN) \cite{Chen et al. 2020, Yang et al. 2019, Wang et al. 2020, Zhou et al. 2021, He, Deng, and Qiao 2019, Wang et al. 2019, Su et al. 2019, Jia et al. 2016} can generate sample-specific filters for different samples, and then use this specific filter for inference. This is different from the traditional CNN models that use the same
filter for all samples. DDF (Zhou et al. 2021) divides DFN into channel convolution and point convolution, which realizes lightweight, so that DFN can be applied to more network layers. Both CondConv (Yang et al. 2019) and Dynamic-Conv (Chen et al. 2020) use attention to aggregate multiple filters generated for samples. DMNet (He, Deng, and Qiao 2019) and SoloV2 (Wang et al. 2020) apply DFN to semantic segmentation and instance segmentation, respectively.

3 Methodology

3.1 Preliminaries

In this section, we introduce the definition of hypernetworks. The original purpose of HyperNetwork was for model compression. The paper regards the parameter $K^n$ of the $n$ layer of the CNN filter as a matrix of $\mathbb{R}^{C_{in}, f_{in} \times C_{out}, f_{out}}$. Among them, $C_{in}$ and $C_{out}$ represent the number of channels of the input and output feature maps, respectively, and $f_{in}$ and $f_{out}$ represent the width and height of the convolution kernel, respectively. It extracts common information from the parameters of all convolutional layers in a convolutional neural network, and represents it in the form of a two-layer MLP, denoted as $g(\cdot)$. Personality information in an n-layer network is represented by a vector $z \sim \mathbb{R}^n$. Before training starts, $z \sim \mathbb{R}^n$ and $g(\cdot)$ are randomly initialized. During training, like a regular neural network, the network learns the mapping relationship between samples $x$ to $y$.

The difference is that the gradient is returned to $z \sim \mathbb{R}^n$ and $g(\cdot)$ instead of $K^n$, and the stored model file is no longer $K^n$ is instead $z \sim \mathbb{R}^n$ and $g$. During inference, the network parameters $K^n$ of the $n$ layer are generated by the vector $z_n$ and the generator $g$. Because the vector $z_n$ corresponding to the parameters of each layer is much smaller than the original parameter $K^n$ of each layer, storing $z \sim \mathbb{R}^n$ and $g(\cdot)$ saves more space than storing $K^n \sim \mathbb{R}^n$. This looks a lot like matrix factorization.

$$K^n = g(z^n).$$

In the implementation process, $K^n$ is often generated in blocks,

$$K^n = \begin{pmatrix}
K_{n,1,1} & \cdots & K_{n,1,j} \\
\vdots & \ddots & \vdots \\
K_{n,i,1} & \cdots & K_{n,i,j}
\end{pmatrix}
\begin{pmatrix}
z_{n,1,1} & \cdots & z_{n,1,j} \\
\vdots & \ddots & \vdots \\
z_{n,i,1} & \cdots & z_{n,i,j}
\end{pmatrix}
= g(\begin{pmatrix}
z_{n,1,1} & \cdots & z_{n,1,j} \\
\vdots & \ddots & \vdots \\
z_{n,i,1} & \cdots & z_{n,i,j}
\end{pmatrix})
\triangleq g(z^n).$$

3.2 MetaNetwork

As depicted in Fig. 1, we design a framework MetaNetwork which consists of MetaGenerator and MetaStabilizer modules to solve the defects of the poor generalization ability of the primary models on the device. We exploit the advantages of HyperNetwork and Dynamic Filter Network (DFN) and improve them so that they can be applied to the device-cloud collaboration system.

MetaGenerator. To solve the challenges we face in the device-cloud collaboration system, we propose MetaGenerator based on HyperNetwork. MetaGenerator is modified from static to dynamic and sample-specific compared to HyperNetwork. That is, replace the random latent vector $z$ in the HyperNetwork with the embedding of the sample. We use as many encoders as the number of network layers to be generated, because we believe that when generating parameters of different depth network layers, the feature information they need to pay attention to is different.

We use $e_x^n = E^n(I_{x_0})$ to compute the representation $E^n(I_{x_0})$ required to generate the parameters of the $n$-th layer of the primary model. The samples $I_{x_0}$ are used to obtain parameters of the primary model, which is called trigger samples. Note that we only input trigger samples once in each session. Trigger samples are the first frame of images captured by a smartphone or a smart camera over a fixed period of time or the user’s click sequence in the previous session. However, this greatly increases memory consumption and slows down inference. So we make encoders lightweight, we change multiple encoders to share one encoder neck, and use different linear layers to change the trigger features.

$$e_x^n = M^n(E_b(I_{x_0})).$$

$E_b$ here represents the neck of the shared encoder. $M^n$ is a linear layer used to adjust the output of $E_b$ to the $n$-th trigger feature. On the recommendation system tasks, the method we extract the trigger feature is $e_x^n = GRU(s_{x_0})$ Among them, $s_{x_0}$ is a user’s click sequence in the previous session.

When generating parameters for a convolutional layer, we can regard the parameters as a matrix $K^n \in \mathbb{R}^{C_{in}, f_{in} \times C_{out}, f_{out}}$, similar to the HyperNetwork which is mentioned in the Preliminaries section. When generating parameters for a fully connected layer, we treat it as a matrix $K^n \in \mathbb{R}^{N_{in} \times N_{out}}$, where $N_{in}$ and $N_{out}$ represent the number of input neurons and the number of output neurons of the fully connected layer, respectively. Then we use the generator $g(\cdot)$ to convert the trigger feature into parameters of the primary models by $K^n = g(e_x^n)$.

Specifically, we input $e_x^n$ into the following two MLP layers to generate parameters, resize according to the structure of dynamic layers of the primary model. The weights of the two MLP layers are denoted by $W_1$ and $W_2$, respectively, and the biases are denoted by $B_1$ and $B_2$, respectively.

$$w^n_x = (W_1 e_x^n + B_1) W_2 + B_2,$$
$$K^n_x = w^n_x + b^n_x.$$ 

For the fully connected layers, we not only need to generate $w^n_x$, but also need to follow the way of $w^n_x$ to calculate $b^n_x$.

We define the loss function for MetaGenerator.

$$L = \sum_S \sum_t \gamma^t l(y_t, f(x_t|\theta_s, \theta_d) = g(e_x^n|\theta_y)).$$

Among them, $S$ represents all sessions, $y_t$ represents the predictions of the primary model at the moment $t$, $T$ represents the length of a session. $\gamma$ is a hyperparameter used to adjust the training. The closer $\gamma$ is to 1, the more MetaGenerator considers all samples in each session as equally
Therefore, we designed a MetaStabilizer module to solve these problems. However, MetaGenerator often produces large oscillations in real environments. This greatly improves MetaGenerator training and inference efficiency and makes it easier to deploy in real environments.

MetaStabilizer. In most cases, MetaGenerator can achieve better performance if appropriate hyperparameters are chosen. However, MetaGenerator often produces large oscillations during training, and its performance is sometimes unstable. Therefore, we designed a MetaStabilizer module to solve these problems.

First of all, $m$ generators rather than a single generator are trained in the same way, denoted by $W'_1$. Then splicing multiple $W'_i$ into a matrix, denoted as $W'_1 = \{W'_{1,1}, W'_{1,2}, ..., W'_{1,m}\}$. $W'_{i,j}$ denoted the $i$-th MLP layer of the $j$-th generator. Then we can get the similarity between $W'_1$ and $W'_2$, thus a self-similarity matrix $S$ of dimension $m \times m$ is obtained by $S = W'_1 \ast (W'_1)^T$.

Summing $S$ by row, we can get the weight vector $p'_i = \{p'_{1,1}, p'_{2,1}, ..., p'_m\}$ with dimension $m \times 1$. Among them, $p_i$ can be regarded as the importance of $W'_{1,i}$ in the multiple generators. We also set temperature to adjust the final weight vector $p$,

$$p_i = \text{Softmax}(\frac{p'_i/\tau}{\sum_j p'_j/\tau})$$  \hspace{1cm} (6)

Then we can calculate the final $W_1$ and $W_2$ like,

$$W_1 = \sum_{i=0}^m p_i \ast W'_{1,i}$$ \hspace{1cm} (7)

Finally, we use Eq. 7 to get $W_1$ and $W_2$, and get the model parameters after replacing $W_1$ and $W_2$ in Eq. 4.

4 Experiments

We evaluate the effectiveness of the proposed MetaGenerator and MetaStabilizer on three datasets on tasks with two different modalities including images and text, and conduct extensive analysis experiments.

On the recommendation task, it is assumed that on e-commerce APPs such as eBay, Amazon, and Taobao. In the actual application scenarios, the large model on the cloud completes the recall and fine-rank of goods, and the lightweight primary model on the device completes the re-rank of goods based on the user’s real-time click sequence. The above primary models all need to use data input under different distributions for inference, but the update cycles of these primary models are relatively long. Therefore we
complete experiments on Facial expression recognition task and Sequential Recommendation tasks. Similarly, on smartphones or VR devices, a model need to complete facial expression recognition for different characters at different time periods and in different environments, such as emojis that can dynamically change with faces on iPhones, and digital virtual humans. To get closer to the actual application scenarios, the chosen baselines are all lightweight but powerful models that are easy to deploy on devices and the chosen datasets and tasks can well simulate the real device-cloud collaboration system.

Note that the Parameters and FLOPs shown in our experimental results are the Parameters and FLOPs of the primary model that need to be deployed on the device. Since our MetaGenerator is deployed on the cloud, the parameters and FLOPs of the primary model that needs to be deployed on the device under the MetaNetwork framework are the same as the baselines. We only request the model at the first moment of every video or every session. Therefore, compared with the baseline, our method only causes an extremely small amount of time delay for device-cloud communication and parameters generation at the beginning of a session. Please refer to Appendix for more details and supplementary experiments.

4.1 Experiments on Sequential Recommendation

Datasets. We evaluate MetaNetwork on Movielens-1M and Movielens-100k. Movielens-1M and Movielens-100k are two widely used public benchmark in the field of recommendation system. Following conventional practice, we treat all interacting samples in the dataset as positive samples, and sample negative samples in a 1:4 ratio. In the test set, we refer to (Krichene and Rendle 2020) to eliminate sampling bias, so we take all samples without user interaction as negative samples in the test set.

Baselines. Three sequential recommendation models DIN (Zhou et al. 2018), SASRec (Kang and McAuley 2018) and GRU4Rec (Hidasi et al. 2016) which are widely used in academia and industry, are chosen as baselines.

Results. We directly train the baseline with all the training data, that is, the baseline learns the mapping from samples to labels. When training MetaNetwork, we input the click sequence in the previous session as a trigger sequence to MetaGenerator to get the parameters of the classifier. We then make predictions in this session using the primary model consisting of a static feature extractor and a dynamic classifier. As for baselines, we make predictions in this session using the primary model consisting of a static feature extractor and a static classifier in the conventional way. We choose AUC as the evaluation metric.

We repeat the experiment five times, and compare MetaGenerator with 5 MetaStabilizers (Ours+MS), MetaGenerator (Ours) and several baselines (Base) in Table 1 and Table 2. Table 1 shows that applying MetaGenerator to DIN, SASRec, and GRU4Rec can increase by 1.72%, 0.35%, and 0.34%, respectively, while using MetaStabilizer can increase by 1.69%, 0.49%, and 0.34%, respectively. Such improvement has been considered significant in the field of recommendation systems (Zhou et al. 2018; Kang and McAuley 2018; Hidasi et al. 2016). Table 2 shows that applying MetaGenerator to DIN, SASRec, and GRU4Rec can increase by 2.53%, 0, and 0.32%, respectively, while adding MetaStabilizer can increase by 2.78%, 0.02%, and 0.36%, respectively. Such improvement is considered significant in the recommendation system field except when the baseline is SASRec. While the MetaStabilizer module does not seem to improve performance at times, a drop in the standard deviation of the performance indicates that the model’s stability has been enhanced. When MetaStabilizer is not used, the performance fluctuation of MetaGenerator is usually more than 0.005 (except when DIN in Table 2 is used as the baseline), which has a bad impact on the recommendation system model actually deployed on the device. But this phenomenon is often greatly alleviated after using MetaStabilizer. We also count the Parameters and FLOPs in the Table 1 and Table 2. The parameters of DIN, SASRec and GRU4Rec are 21.83K, 13.63K and 11.62K, respectively. The FLOPs are 0.23B, 0.28B and 0.11B, respectively. Low FLOPs and parameters mean they are easy to deploy on devices.

4.2 Experiments on Facial Expression Recognition

Datasets. We evaluate our method on CK+(Kanade, Cohn, and Tian 2000; Lucey et al. 2010). CK+ is a facial expression recognition dataset, which contains 593 videos of 123 people, of which 327 videos are emotion annotated, and it contains 7 basic emotion categories: Anger, Contempt, Disgust, Fear,

| Method   | AUC(mean) | AUC(std) | Parameters (K) | FLOPs (M) |
|----------|----------|----------|----------------|-----------|
| DIN      |          |          |                |           |
| Base     | 0.8723   | 0.0017   | 21.83          | 0.11      |
| Ours     | 0.8755   | 0.0015   | 0.0004         |           |
| Ours (+MS) | 0.8732 | 0.0017   | 0.0005         |           |

Table 1: Performance comparison of the proposed method and baselines on sequential recommendation. (Movielens-1M dataset)
Happy, Sadness, and Surprise. We treat each video as a session and take the first three frames and the last three frames of each video. In order to simulate the real application environment, we set the label of the first three frames to Natural and the label of the last three frames to the emotion label of this video. We manually checked most of the videos to make sure it was okay to do so.

**Baselines.** We choose MobileNetV3 (Howard et al. 2019), one of the most popular lightweight networks, as the baseline. We then test the performance of MetaGenerator and MetaStabilizer on MobileNetV3-Large and MobileNetV3-Small.

**Results.** We directly train the baseline with all the training data, that is, the baseline learns the mapping from samples to labels. When training MetaNetwork, we input the first frame of each video as a trigger image to MetaGenerator to get the parameters of the dynamic layers including the last convolutional layer and fully connected layers. We then make predictions in this video using the primary model consisting of a static feature extractor and a dynamic classifier. As for baselines, we make predictions in this video using the primary model consisting of a static feature extractor and a static classifier in the conventional way. In the experiments, the number of MetaStabilizers was set to 5. We choose accuracy as the evaluation metric.

![Figure 2: Performance comparison of the proposed methods and baselines on facial expression recognition.](image)

We repeat the experiment five times, and compare MetaGenerator with 5 MetaStabilizers (Ours+MS), MetaGenerator (Ours) and several baselines (Base) in Table 4. The experimental results show that applying MetaGenerator to MobileNetV3-Large and MobileNetV3-Small can increase by 3.27% and 5.80%, respectively. MetaGenerator with 5 MetaStabilizers can increase by 7.13% and 10.71%, respectively. We also count the Parameters and FLOPs in the Table 4. The parameters of MobileNetV3-Large and MobileNetV3-Small are 2.69M and 1.24M, respectively. The FLOPs are 0.27B and 0.06B, respectively. Low FLOPs and parameters mean they are easy to deploy on devices.

We further analyze the oscillation of MetaGenerator with 5 MetaStabilizers (Ours+MS), MetaGenerator (Ours) and several baselines (Base) during training and testing in Fig. 2 and Fig. 3. In Fig. 2 the upper and lower bounds of the box plot are represented by black horizontal lines, the standard deviation is represented by blue horizontal lines, and the two ends of the box represent the quartiles, the red horizontal line represents the median, and the blue circle represents the mean. In Fig. 3, the dark-colored line is the mean value of the data trained for multiple times, and the light-colored area is the fluctuation range, that is, the maximum and minimum values of multiple training data. For the sake of beauty, the curves in the figure are obtained by 1:25 sampling, that is, 1 epoch in the figure represents the actual 25 epochs. It shows that MetaGenerator+MetaStabilizer is the best in terms of accuracy and convergence speed, the second is MetaGenerator, and the worst is baseline. Moreover, even with sampling, it can still be found that the MetaGenerator has huge oscillations during training. The MetaStabilizer module greatly alleviates this phenomenon and eliminates the possibility of huge oscillations in the MetaGenerator during training.

Combining the above figures and tables, it is not difficult to find that MetaGenerator improves performance, but it is not stable. When the MetaStabilizer module is used, the performance is more stable and get further improved.

**Generalizability.** In Fig. 4 we compare the performance of MetaGenerator with 5 MetaStabilizers (MetaGenerator+MetaStabilizer), naive MetaGenerator (MetaGenerator), and MobileNetV3-Large (Base) with fixed parameters in 5 randomly selected sessions. Our method has a clear advantage in almost every session. The performance of the generated model parameters seems to get worse over time, but the advantage of the model with fixed model parameters gradually increases. These phenomena show that MetaNetwork can indeed generate model parameters that perform well in this session for different sessions.

**MetaStabilizer.** We set a different number of MetaStabilizers and repeat the experiments for 5 times to observe its effect on accuracy and loss. The results in Table 5 show that as the number of MetaStabilizers increases, MetaNetwork also performs better.

In Fig. 5 the dark-colored line is the mean value of the
Table 3: Time delay caused by device-cloud communication.

| Task                  | Model            | Size  | 4G(5MB/s) | 4G(15MB/s) | 5G (50MB/s) | 5G(100MB/s) |
|-----------------------|------------------|-------|-----------|------------|-------------|-------------|
| Expression Recognition| MobileNetV3-Large| 5.31MB| 1.06s     | 0.35s      | 0.10s       | 0.05s       |
|                       | MobileNetV3-Small| 3.06MB| 0.61s     | 0.20s      | 0.06s       | 0.05s       |
| Sequential Recommendation| DIN          | 8.06kB| 1.60ms    | 0.53ms     | 0.16ms      | 0.08ms      |
|                       | SASRec          |       |           |            |             |             |
|                       | GRU4Rec         |       |           |            |             |             |

Table 4: Performance comparison of the proposed methods and baselines on facial expression recognition.(CK+ dataset)

| Method            | Acc. (%) | Std. (%) | Parameters (M) | FLOPs (B) |
|-------------------|----------|----------|----------------|-----------|
| MobileNetV3-Large | Base     | 76.97    | 2.76           | 2.69      | 0.27       |
| Ours (+MS)        |          |          |                |           |
| MobileNetV3-Small | Base     | 67.73    | 2.34           | 1.24      | 0.06       |
| Ours (+MS)        |          |          |                |           |

Table 5: Effects of the number of metastabilizers on performance.

| Method            | MetaStabilizer | Accuracy(%) |
|-------------------|----------------|-------------|
| MobileNetV3-Large | Base           | 76.97       |
|                   | -              | 76.97       |
|                   | 2              | 80.40       |
|                   | 3              | 81.88       |
|                   | 5              | 82.45       |
| MobileNetV3-Small | Base           | 67.88       |
|                   | -              | 71.82       |
|                   | 2              | 73.11       |
|                   | 3              | 73.14       |
|                   | 5              | 75.15       |

data trained for multiple times, and the light-colored area is the fluctuation range, that is, the maximum and minimum values of multiple training data. For the sake of beauty, the curves in the figure are obtained by 1:10 sampling, that is, 1 epoch in the figure represents the actual 10 epochs. The figure shows that as the number of MetaStabilizers increases, the oscillations of the accuracy curve and loss curve are greatly reduced, and the performance is improved.

4.3 Time Delay Analysis

In Table 3, we list models (Model), the size of data needs to be transferred (Size), four network environments, namely ordinary 4G network (5MB/s), good 4G network (15MB/s), ordinary 5G network (50MB/s) and good 5G network (100MB/s). On the Sequential Recommendation task, although the frequency of session switching is high, the bandwidth requirement is extremely low due to the extremely few parameters that need to be delivered. On Facial Expression Recognition task, the frame rate is about $10 \sim 20$, which is real-time under a 5G network.

5 Conclusion

In this paper, we propose a novel task-agnostic framework, named MetaNetwork, for generating adaptive device model parameters from cloud without on-device training. Specifically, our MetaNetwork is deployed on cloud and consists of MetaGenerator and MetaStabilizer modules. The MetaGenerator learns a mapping function from samples to model parameters, and it generates and delivers adaptive parameters to the device based on samples uploaded from the device to the cloud. The MetaStabilizer reduces the oscillation of the MetaGenerator, accelerates the convergence and improves the model performance during both training and inference. Experiments show that MetaNetwork can achieve competitive performances in different modalities.
A Appendix

A.1 Implementation details

We summarize the parameters and training schedules of MetaNetwork on the three datasets in Table 6.

Table 6: Parameters and training schedules of MetaNetwork.

| Dataset          | Parameters | Setting |
|------------------|------------|---------|
| CK+              | GPU        | Tesla V100               |
|                  | Optimizer  | Adam                |
|                  | Learning rate | 0.001            |
|                  | Weight decay   | 0.0005            |
|                  | Batch size  | 128 or 64            |
|                  | Image shape  | 224×224            |
|                  | γ           | 1                   |
|                  | Temperature | 1                   |
| Movielens-1M     | GPU        | Tesla V100               |
|                  | Optimizer  | Adam                |
|                  | Learning rate | 0.001            |
|                  | Batch size  | 512                 |
|                  | Sequence length | 30                |
|                  | γ           | 1                   |
|                  | Temperature | 1                   |
| Movlenes-100k    | GPU        | Tesla V100               |
|                  | Optimizer  | Adam                |
|                  | Learning rate | 0.001            |
|                  | Batch size  | 512                 |
|                  | Sequence length | 30               |
|                  | γ           | 1                   |
|                  | Temperature | 1                   |

A.2 Parameters Visualization

Figure 6: The 3D visualization results of the dynamic layer of MetaNetwork and the fixed layers of the base model using the t-SNE method.

We choose MobilenetV3-Large as the baseline, and use t-SNE to reduce and standardize the dynamic adaptive model parameters generated by MetaNetwork and the fixed model parameters of the baseline. The former is represented by dots in various colors, and the latter is represented by large-sized asterisks. Fig. 6 shows the dimensionality reduction results of the last convolutional layer and the last fully connected layer. The experimental results show that MetaNetwork can generate adaptive parameters for the convolutional and fully connected layers according to the data of different distributions.