Automated Customization of On-Device Inference for Quality-of-Experience Enhancement

Yang Bai, *Member, IEEE*, Lixing Chen, *Member, IEEE*, Shaolei Ren, *Senior Member, IEEE*, and Jie Xu, *Senior Member, IEEE*

Abstract—The rapid uptake of intelligent applications is pushing deep learning (DL) capabilities to mobile devices. However, the heterogeneities in device capacity, DNN performances, and user preferences make it challenging to provide satisfactory Quality of Experience (QoE) to mobile users. This paper studies automated customization for DL inference on mobile devices (termed as on-device inference), and our goal is to enhance user QoE by configuring the on-device inference with an appropriate DNN for users under different usage scenarios. The core of our method is a DNN selection module that learns user QoE patterns on-the-fly and identifies the best-fit DNN for on-device inference with the learned knowledge. It leverages an online learning algorithm, *NeuralUCB*, that has excellent generalization ability for handling various user QoE patterns. We also embed the knowledge transfer technique in NeuralUCB to expedite the learning process. However, NeuralUCB frequently solicits QoE ratings from users, which incurs non-negligible inconvenience. To address this problem, we design feedback solicitation schemes to reduce the number of QoE solicitations while maintaining the learning efficiency of NeuralUCB. A pragmatic problem, *aggregated QoE*, is further investigated to improve the practicality of our framework. We conduct experiments on both synthetic and real-world data. The results indicate that our method efficiently learns the user QoE pattern with few solicitations and provides drastic QoE enhancement for mobile devices.

Index Terms—On-device DNN inference, model selection, multi-armed bandit, quality of experience

1 Introduction

Deep learning (DL) has revolutionized a broad spectrum of domains and achieved remarkable performance comparable to or even exceeding human levels. Such DL intelligence typically runs at cloud-scale data centers [1], but as the mobile industry prospers, there is a growing trend to push the DL intelligence toward mobile devices [2]. More precisely, the inference of Deep Neural Networks (DNN) is brought down to mobile devices, i.e., hand-held devices like smartphones and tablet computers. For example, Apple Siri uses DL for speech synthesis on smartphones, and Facebook keeps improving the experience of DNN inference in its mobile app for more than 2 billion users [3]. Running DNN inferences on mobile devices (hereinafter, referred to as on-device inference), users receive improved service quality with reduced service latency, and the inference process also becomes less dependent on Cloud or other edge computing platforms [4]. Various mechanisms, e.g., model selection [5], DNN compression [6], and hyperparameter optimization [7]) have been studied to improve the performance of on-device inferences in terms of accuracy, inference delay, energy consumption, etc. Although these technical metrics serve as good indications of on-device inference performance, they are unable to precisely capture users' overall satisfaction with received services. In light of this challenge, industry and academia are embracing a more holistic approach, Quality of Experience (QoE), to quantify the relationship between user experience and service performances [8]. The QoE domain encapsulates methods and metrics for evaluating mobile services in a way that aligns with how end-users actually experience those services. Compared to technical metrics, QoE is a fuzzier metric that exhibits complicated interdependencies between service performances, external environments, and human perceptions [9]. Improving user-perceived QoE has been an important topic for mobile app developers to enhance customer satisfaction and prevent customer churn. Efforts have been made in the field of mobile computing for analyzing and enhancing user QoE [9], [10]. However, most existing works

- *Yang Bai is with the Department of Automation, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: ybai@sjtu.edu.cn.*
- *Lixing Chen is with the Institute of Cyber Science and Technology, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China, and also with the Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, Shanghai 200240, China. E-mail: lxchen@sjtu.edu.cn.*
- *Shaolei Ren is with the Electrical and Computer Engineering, University of California, Riverside, CA 92521 USA. E-mail: sren@ece.ucr.edu.*
- *Jie Xu is with the Department of Electrical and Computer Engineer, University of Miami, FL 33146 USA. E-mail: jiexu@miami.edu.*

Manuscript received 11 December 2021; revised 6 June 2022; accepted 20 August 2022. Date of publication 20 September 2022; date of current version 7 April 2023.
The work of Lixing Chen was supported in part by the National Natural Science Foundation of China under Grant 62202293. The work of Shaolei Ren was supported in part by NSF under Grant CNS-2007115. The work of Jie Xu was supported in part by NSF under Grants 2006630, 2044991, 2029858, and 2033681.
(Corresponding author: Lixing Chen.)
Recommended for acceptance by G. Constantinides.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3208207, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3208207

TABLE 1
Performance of DNN Models

Model	Size	Accuracy	Latency
Mobilenet v1	**1.5MB**	61.2%	**3.6ms**
Mobilenet v2	3.4MB	70.8%	12ms
Inception v2	11MB	73.5%	59ms
Inception v3	23MB	**77.5%**	148ms
DenseNet	43.6MB	64.2%	195ms
NasNet M	21.4MB	73.9%	56ms

emphasize the impact of network conditions on mobile app QoE [11], [12], and optimize interactions between mobile devices and computing platforms (e.g., cloud data-centers or edge servers) to deliver higher QoE to users. By contrast, QoE enhancement for on-device DL inference is still an under-investigated topic. This paper studies On-device Inference Customization (OIC) for DL-based applications. Our goal is to customize on-device inference for individual users in a way that maximizes user perceived experience.

1.1 Motivations and Challenges

OIC is motivated by the heterogeneity of mobile devices, DNN models, usage scenarios, and user preferences, which jointly affect user perceived QoE in a complicated manner.

1.1.1 Device Heterogeneity

The mobile device ecosystem is very heterogeneous, some high-end devices have state-of-the-art CPUs along with dedicated graphic processing units (GPUs) and even purpose-built processing units to speed up DNN inferences, while many others are powered by CPUs of several years old [3], [13]. Consequently, there is no standard device model to optimize the DL inference for. The heterogeneity of device computing capacity results in a huge variability in service quality and user experience. Even with fine-tuned DNNs, the inference latency varies by a factor of 10+ across mobile devices [3].

1.1.2 DNN Model Heterogeneity

There often exist various DNN architectures that can be used by the application developer to address a learning problem. For example, in *image classification*, commonly-used DNN architectures include MobileNet [14], Inception [15], Yolo [16], and etc. The recent studies on DNN model compression [6] further provides more available DNN architectures that exhibit different trade-offs in a multi-dimension space of important metrics (e.g., size versus accuracy versus latency). Table 1 shows performance metrics of several DNN examples (measured on Google Pixel 3 with Android 10) provided by Tensorflow Lite [17]. We see that different DNNs require different computing resource (model size) and provide different inference performances (accuracy and latency). No single DNN can achieve optimality in all dimensions. Existing works [13], [18] have provided performance measurements of a broad range of DNN architectures on a variety of embedded platforms and mobile hardware, showing that the performance of DNNs varies significantly across devices.

1.1.3 Usage Scenario Heterogeneity

DNNs running on different mobile devices can be exposed to very different usage scenarios — different locations, illumination conditions, time, etc. These all account to drastically different distributions of user input data, which can result in very different inference accuracies even for the same DNN model. For example, variations of image illumination can create intra-class variability in image classification problems [5] and degrade the performance of DL algorithms. Besides environmental factors, the mobile device status (e.g., CPU/memory usage and battery level) also affects the inference quality. For example, when the battery level is low, mobile devices may switch to the battery saving mode and decrease CPU frequency, or when the CPU temperature surpasses the devices' thermal threshold, the device begins to throttle back its frequencies in an effort to reduce the heat. These actions lead to a larger DNN inference latency.

1.1.4 User Preference Heterogeneity

It should be noticed that users often have different sensitivity towards different performance metrics of DNN inferences. For example, some users are more energy-sensitive due to limited battery capacities, whereas others would like to trade energy consumption for lower inference latency and higher inference accuracy. In addition, the sensitivities of a user can also change under different circumstances, e.g., when the battery level becomes low, the user may become more energy-sensitive.

Therefore, how to design an OIC framework that can work efficiently with the joint impact of mobile device heterogeneity, DNN model heterogeneity, usage scenario heterogeneity, and user preferences heterogeneity would be a critical issue to be solved in this paper. In particular, OIC will need to address the following three challenges: The *first* challenge is the unknown and diverse user QoE patterns. The crux of OIC is finding an underlying mapping for each DNN that maps the device information, user preference, and usage scenario to the user QoE pattern. If these mappings are available at hand, then the best-fit DNN can be easily identified. Unfortunately, such mappings are unknown a priori, and therefore a learning mechanism is necessary to acquire the user QoE patterns. In particular, OIC will require the learning method to have a good generalization ability as user QoE patterns usually exhibit significant variability due to the heterogeneity in mobile devices and user preferences. The *second* challenge is the user-perceived inconvenience when learning user QoE patterns online. To customize the on-device inference for a particular user, we will need QoE data that precisely reflects the user's satisfaction about DNNs under various usage scenarios. Such QoE data can only be collected from users while they are using the application. This is often done by on-screen pop-ups that solicit the user about his/her experience. Such QoE solicitation strategy is commonly adopted, e.g., Skype asks a "How was your call quality?" question when a call is ended. Being users ourselves, we know that frequently receiving experience surveys can be annoying. Also, QoE solicitations may incur additional cost to application developers because incentive mechanisms [19] are often applied to motivate the QoE response. Therefore, a key designing topic of OIC is to keep

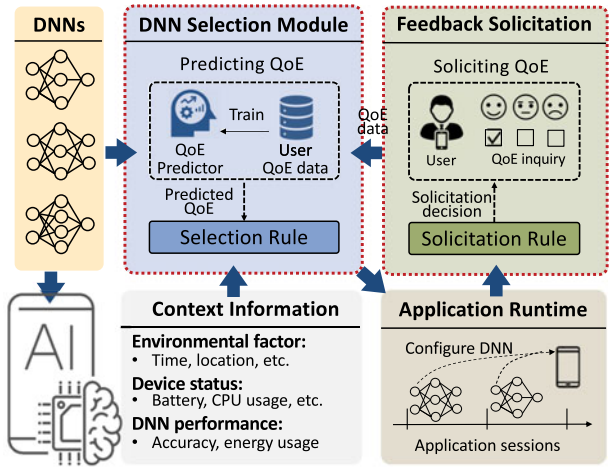


Fig. 1. Block diagram of OIC.

the solicitation inconvenience minimal during online learning. The *third* challenge is to guarantee the practicality of OIC. Despite the efficacy, other practical issues should also be considered to successfully deliver functionalities of OIC in practice. For example, the computational complexity of OIC must be low enough to work on mobile devices. OIC is also expected to take effect quickly and hence the learning process should be kept short. In addition, users may provide aggregated QoEs that reflect their experience for multiple DNNs used before rather than an individual DNN. All these practical issues should be taken care of when designing OIC.

1.2 Contributions

In this paper, we design a novel framework to address the above challenges. Fig. 1 depicts a block diagram of the proposed OIC method. It consists of two components: a *DNN Selection Module* that learns a QoE predictor for guiding the DNN selection; and a *Feedback Solicitation Scheme* (FSS) that determines when to pop up experience surveys. The contribution of this paper is summarized as follows:

1) We utilize online learning techniques to build the DNN selection module. NeuralUCB [20], a contextual multi-armed bandit algorithm, is employed to learn users' QoE patterns based on the side-information of mobile devices, DNNs, and usage scenarios. The predicted QoE pattern is then used to identify the best-fit DNN for the application. A salient feature of NeuralUCB is that it does not assume a certain distribution of QoE data and hence, attains generality across different users and devices. We further incorporate knowledge transferring techniques in NeuralUCB to speed up the customization process.

2) A novel Feedback solicitation scheme (FSS) is designed for NeuralUCB to reduce the user-perceived inconvenience caused by QoE solicitations during online learning. FSS jointly considers the learning efficiency of NeuralUCB and solicitation cost, and tries to reduce the number of QoE solicitations in a way that minimizes the total performance loss. FSS cuts the number of solicitations from T to $T^{2/3}$ (i.e., a 90% reduction for $T = 1000$) while keeping asymptotic optimality of NeuralUCB.

3) A learning strategy is further designed to apply OIC with the aggregated QoE where QoE ratings solicited from users reflect their experience over multiple DNNs used during a time span. The key to addressing the aggregated QoE is a feedback refinement approach that estimates individualized QoEs (i.e., QoEs for individual DNNs) from aggregated QoEs with the assistance of QoE predictor. The feedback refinement approach is a flexible add-on and also handles a mix of aggregated and non-aggregated QoEs.

4) The proposed method is evaluated on both numerical and real-world data. We collect context information and user QoE from human subjects when using a *image classification application* [21]. The results show that OIC can learn user QoE patterns with low solicitation costs and dramatically improve the user experience.

The rest of the paper is organized as follows. Section 2 discusses related works. Section 3 designs the DNN selection module. Section 4 develops feedback solicitation schemes. Section 5 designs a feedback refinement approach for the aggregated QoE. Section 6 carries out experiments and evaluations, followed by conclusions in Section 7.

2 Related Works

Deep Learning for Mobile Devices. The past few years have seen a surge in the investigation of DL techniques for mobile devices and embedded platforms. Promising results are appearing in many domains, including hardware, algorithms, and tools. 1) Hardware: Major mobile hardware vendors are working closely with the makers of DL frameworks to make sure that their processors are well supported. For example, the Apple Bionic chip [22] includes a group of specialized cores functioning as a neural processing unit dedicated to accelerating neural network operations. TensorFlow Lite for Microcontrollers [23] currently provides optimizations for Arm, Cadence, and Synopsys processors, and the list is still growing. 2) Algorithms: DNN compression methods [6] are investigated to compress large-scale DNN models into small DNNs that can be easily implemented on mobile devices. Based on their properties, DNN compression methods can be divided into four categories: parameter pruning/quantization, low-rank factorization, transferred/compact convolutional filters, and knowledge distillation. The parameter pruning/quantization methods [24] explore the redundancy of DNN parameters and remove the redundant and uncritical ones. Low-rank factorization techniques [25] use matrix/tensor decomposition to estimate informative parameters of DNNs. The transferred/compact convolutional filter approaches [25] design special structural convolutional filters to reduce the parameter space. The knowledge distillation methods [26] learn a distilled model and train a more compact neural network to reproduce the output of a larger network. 3) Tools and libraries: New DL tools and libraries, e.g., Tensorflow Lite [27], Core ML [28], and PyTorch Mobile have been designed to develop and implement DL intelligence on resource-constrained mobile devices.

QoE Enhancement in Mobile Computing. An array of studies has looked at mobile app QoE, focusing on establishing QoE models based on network Quality of Service (QoS) and/or contextual data [9]. The work [29] provides a mobile QoE model for web browsing, file upload, and file download using the data collected from 108 users in a controlled laboratory experiment. The network quality metrics include

throughput and delay. The authors in [30] modeled mobile QoE by examining a variety of specific popular mobile apps including YouTube, Facebook, and Google Maps. The work [31] used *in situ* passive device-based network measurements and user experience ratings to model the relationship between network quality (and non-network features) and QoE. The work [32] considered the impact of device capabilities on the QoE of mobile apps, and studied the QoE of web browsing, video streaming, and video telephony for different device parameters. However, most of the above works take the network quality as the key factor in the QoE model, which is starkly different from the QoE pattern in our problem as the on-device inference is independent of network data transmission. In addition, the above works build general QoE models for a large group of users, by contrast, our on-device inference customization aims to learn the QoE pattern for each individual user and adaptively select the best-fit DNNs for users.

DNN Model Selection. DNN selection aims to identify the best-fit DNN from a set of candidate DNNs that are trained with different architectures, parameters, and training data. Unlike DNN selections for general purposes that focus on accuracy and inference delay, DNN selections for mobile devices should take into account other important factors, e.g., computing resource requirement and energy consumption, due to the limited computing capacity and battery on mobile devices. Recent works consider the special needs of mobile devices when designing DNN selection schemes. For example, the work [5] presented an adaptive scheme to determine which DNN model to use for a given input by considering the desired accuracy and inference time. The authors achieve this by training a predictive model offline. The authors in [33] proposed OODIn for the optimized deployment of DL apps across heterogeneous mobile devices. OODIn uses a highly parametrized multi-layer design and performs a principled optimization of model-level and system-level parameters through a multi-objective formulation to adapt to device capabilities and the user-specified performance requirements in terms of accuracy, latency, throughput and energy. MobiSR [34] considered model compression techniques and traverses the design space to reach the highest performing trade-off between image quality and processing speed. The work [35] presented a design and implementation of an optimizing compiler and runtime scheduler to serve a stream of heterogeneous requests under resource constraints, such as memory, computation, and energy. Most of the above works optimize the on-device execution of DL-based mobile apps under the constraint of QoS requirements, e.g., accuracy, latency, and energy consumption. QoS metrics are objective, and the mapping between device status and QoS is deterministic. Therefore, the above works carry out learning and optimization based on offline-collected data. By contrast, our on-device inference customization is QoE-driven and QoE patterns vary significantly across users. Oftentimes there is no data available a priori to learn the QoE pattern for a particular. These issues motivate us to learn users' QoE patterns online. The most related work is probably [36], where the authors also studied a model selection scheme for QoE improvement. Compared with it, our work has two stark differences. First, the model selection in [36] is performed on Cloud, which finds an appropriate model for a user based on its device features. By contrast, our on-device inference customization method runs model selection on mobile devices, aiming to select models adaptively according to the user preference and usage scenario. Second, [36] used only static device information to guide the model selection while our work encompasses a variety of dynamic states (e.g., task features, device status, and environment changes) and discover complicated QoE patterns for individual users under various states.

3 Automated Customization of On-Device Inference

3.1 System Overview

3.1.1 Pre-Configuration

OIC is designed as a plug-in module for mobile applications, and it is deployed on the user device along with the application installation. The application will download multiple DNN models (hereinafter referred to as candidate DNNs), denoted by $\mathcal{M} = \{1, \ldots, M\}$, from the developer's cloud. The number of candidate DNNs M should not be too large to avoid excessive usage of the device storage. Only one of the stored DNN will be selected and loaded to the device memory (typically DRAM) for processing inference tasks, and therefore, OIC will not cause a large increase in the device memory usage. The candidate DNNs have different performances in terms of accuracy, latency, energy consumption, etc. The set of candidate DNNs $\mathcal{M}$ defines the action space of DNN selection. In a broader concept, the action space of DNN selection is not necessarily independent DNNs, it can also be different configurations of one DNN model. For example, a BranchyNet [37] enables a DNN to exit earlier (to save time) with a reduced inference delay but lower inference accuracy; SlimNets [38] and MutualNet [39] provide executable configurations with different DNN widths which can be adjusted according to resource constraints; Configurable DNN [40] allows dynamic adjustment of the number of channels in DNN based on the requirements of response time, energy usage, and accuracy. In such cases, the action space becomes the set of available configuration points (with different performance trade-offs) of a DNN.

3.1.2 Operations of On-Device Inference Customization

The operation timeline of OIC is discretized by *application sessions*, denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$. An application session begins when a user starts the application and ends when the application exits. The DNN selection module runs in the application loading phase during which it selects a DNN and configures it in the application. Note that the application loading phase originally exists for application initialization, object pooling, server connection, etc. Our method does not prolong this loading phase as we will show later in the experiment that the run-time of our method is less than 2ms. We for now assume that the selected DNN does not change during a session. An extended scene will be considered in Section 5 where the selected DNN changes during a session. The QoE solicitation happens at the end of application sessions. This is done by popping up an on-screen survey that inquires about the user experience during the application session. It is possible that users are not willing to complete the QoE survey

questions. In this case, the operator may apply incentive mechanisms [19] to motivate the QoE response. The feedback solicitation schemed designed later (in Section 4) will consider the cost of QoE solicitations due to the incentive payment and the inconvenience brought to users. The collected QoE will be stored in a database for learning user QoE patterns later.

3.1.3 QoE Predictor

A QoE predictor is built to guide the DNN selection. It establishes a mapping from the context of DNN and usage scenario to user QoE. Examples of context information includes: 1) *environmental context* which can be acquired via sensors equipped with the mobile device, e.g., the device location acquired via GPS signals and the illumination acquired via the ambient light sensor; 2) *device status*, e.g., battery level, CPU usage, and memory usage, which can be inspected by calling Java APIs; 3) *DNN performance*, e.g., size, accuracy, and latency of DNNs. The statistics of DNN performances are available on the TensorFlow Lite website [21]. Note that these statistics are *nominal* because they are measured on standard devices which may be different from the user device, and on a standard dataset which may be different from the user input data. Therefore, it is possible that the performance of a DNN on the user device can be very different from the nominal performance. While nominal performances do not reflect the actual performances of DNNs on user devices, it provides other useful information about the DNNs, e.g., the nominal inference delay can reflect the computational complexity of DNNs.

The contextual conditions of environment, device status, and DNN $m \in \mathcal{M}$ in application session t are collected in the context $x_{t,m}$. We let $\boldsymbol{x}_t = \{x_{t,m}\}_{m\in\mathcal{M}}$ collect context of all candidate DNNs. While we know in general that the user QoE is influenced by the above contextual conditions, we do not know the exact form of the QoE model. Therefore, OIC relies on an online learning algorithm to build a QoE predictor (the learning process will be explained later in this paper) to predict the user QoE delivered by DNNs $\hat{\boldsymbol{r}}_t = \{\hat{r}_{t,m}\}_{m\in\mathcal{M}}$, where $\hat{r}_{t,m}$ is the predicted QoE for DNN m given context $x_{t,m}$. If QoE predictions are accurate, then the best-fit DNN for session t can be easily determined by *greedy policy* $m_t = \arg\max_{m\in\mathcal{M}} \hat{r}_{t,m}$. The selected DNN m_t will be used to configure the mobile app for processing the inference tasks in application session t. When session t ends, the mobile app uses an on-screen popup to collect user's QoE in the application session. The overhead, e.g, RAM/CPU usage, energy consumption, and latency, for popping up on-screen questions and collecting user QoE scores is negligible. The solicitation cost is more about the user-perceived inconvenience caused by QoE solicitation questions. Clearly, the solicitation cost is less likely to be a constant because the inconvenience caused by QoE solicitations can be affected by users' emotions and usage circumstances, which are difficult to be sensed by mobile devices. Therefore, we assume the solicitation cost is drawn from a random distribution, and let the expectation of the random distribution be the unit solicitation cost λ.

Fig. 2 depicts the timeline of operations during an application session. Now a fundamental problem for OIC becomes learning an accurate QoE predictor for candidate DNNs. In the sequential, we utilize a multi-armed bandit (MAB) algorithm to learn QoE predictors in an online fashion.

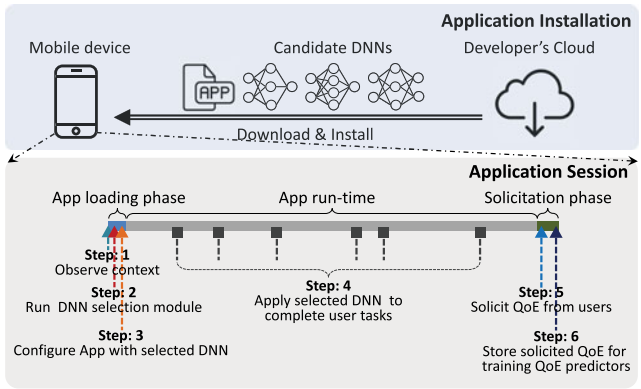


Fig. 2. Operations during an application session.

3.2 Learning QoE Predictors

Note that the greedy policy for selecting the best-fit DNN is only plausible when the QoE predictor give accurate enough results, otherwise, the selected DNN may deliver much worse QoE than the expectation. Obtaining a high-quality QoE predictor requires adequate collections of context-QoE data for candidate DNNs. Note that the user QoE for a DNN can be solicited only when the DNN is used during a session, and therefore the purpose of selecting a DNN can be either *exploration*, i.e., to collect user QoE delivered by the selected DNN for better QoE prediction in the future, or *exploitation*, i.e., to select the DNN that is expected to deliver the highest QoE to the user. An important designing goal in MAB problems is balancing the tradeoff between exploration and exploitation.

We consider that the QoE value r_t in session t is randomly sampled from an unknown distribution $\mathcal{R}$ parameterized by the context x_{t,m_t}, i.e., $r_t \sim \mathcal{R}(x_{t,m_t})$. Note that we let r_t be sampled from distribution $\mathcal{R}(x_{t,m_t})$ instead of being a deterministic value because the user QoE can be influenced by other contextual conditions that cannot be included in the context x_{t,m_t}. We do not restrict distribution $\mathcal{R}(x_{t,m_t})$ to a certain type as QoE patterns vary significantly across users. This requires our MAB algorithm to attain good generalization ability for handling any possible QoE distributions the users may have.

3.2.1 NeuralUCB

OIC treats the user's QoE pattern as a blockbox and employs *NeuralUCB* [20] to learn the mapping from context to QoE. NeuralUCB constructs a neural network to approximate the QoE pattern and has the capability of identifying and representing general dependencies between contextual conditions and QoE without a priori specifying which particular form of distribution to look for. In particular, NeuralUCB can directly take continuous contexts as input and avoids information loss caused by discretization.

The neural network can act either as a classification model for label-form QoEs (e.g., *"satisfactory"* or *"not satisfactory"*) or a regression model for rating-form QoEs (e.g., $r_t \in [0, 1]$). The neural network constructed in NeuralUCB is referred to as *QoE Predicting Network (QPN)* and works as a QoE predictor in DNN selection module. The parameter vector of QPN is

denoted by $\boldsymbol{\theta}$. The input to QPN is the context $x_{t,m}$ of a DNN $m \in \mathcal{M}$ and the output of QPN is the predicted QoE delivered by DNN m, denoted by $\hat{r}(x_{t,m}, \boldsymbol{\theta})$. The goal of NeuralUCB is to maximize the cumulative user QoE $\sum_{t=1}^{T} r_{t,m_t}$ by selecting an appropriate DNN m_t in each session t.

Algorithm 1. NeuralUCB for OIC

1: **Input**: time horizon T, algorithm parameter γ, the number of nodes in QPN hidden layers h.
2: **Initialization**: Randomly initialize $\boldsymbol{\theta}$, initialize $\boldsymbol{Z_0} = \boldsymbol{I}$
3: **for** application session $t = 1, \ldots, T$ **do**
4: **for** each DNN $m \in \mathcal{M}$ **do**
5: Observe the context of DNN m, $x_{t,m}$
6: Predict QoE for DNN m: $\hat{r}_{t,m} \leftarrow \hat{r}(x_{t,m}; \boldsymbol{\theta})$
7: Computing the gradient of QPN parameter $\boldsymbol{\theta}$ at $x_{t,m}$: $\boldsymbol{g}_{t,m} \leftarrow \nabla_\theta \hat{r}(x_{t,m}; \boldsymbol{\theta})$
8: **end for**
9: Compute $u_{t,m} \leftarrow \hat{r}_{t,m} + \gamma \sqrt{\boldsymbol{g}_{t,m}^\top \boldsymbol{Z}_{t-1}^{-1} \boldsymbol{g}_{t,m}/h}, \forall m$
10: Select the DNN $m_t = \arg\max_{m \in \mathcal{M}} u_{t,m}$
11: Solicit QoE r_{t,m_t} from the user and store the context-QoE data (x_{t,m_t}, r_{t,m_t}) in data set $\mathcal{X}$
12: Update QPN: $\boldsymbol{\theta} \leftarrow \texttt{TrainQPN}(\mathcal{X})$ ▷Algorithm 2
13: Compute $\boldsymbol{Z}_t \leftarrow \boldsymbol{Z}_{t-1} + \boldsymbol{g}_{t,m_t} \boldsymbol{g}_{t,m_t}^\top / h$;
14: **end for**

Algorithm 2. Subroutine: $\texttt{TrainQPN}(\mathcal{X})$

1: **Input**: learning rate η, context-QoE dataset $\mathcal{X}$, number of gradient descent update steps J;
2: Define $L(\boldsymbol{\theta}) = \sum_{(x_{\tau,m_\tau}, r_{\tau,m_\tau}) \in \mathcal{X}} (\hat{r}(x_{\tau,m_\tau}; \boldsymbol{\theta}) - r_{\tau,m_\tau})^2$
3: **for** $j = 1, \ldots, J-1$ **do**
4: $\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \eta \nabla L(\boldsymbol{\theta}_j)$
5: **end for**
6: **return** $\boldsymbol{\theta}_J$

The pseudocode of NeuralUCB for OIC is given in Algorithm 1. In each session, a DNN is selected based on $u_{t,m} := \hat{r}(x_{t,m}, \boldsymbol{\theta}) + \gamma \sqrt{\boldsymbol{g}_{t,m}^\top \boldsymbol{Z}_{t-1}^{-1} \boldsymbol{g}_{t,m}/h}$ (Line 9 and 10). The first term of $u_{t,m}$ is the user QoE predicted by QPN and the second term of $u_{t,m}$ characterizes the uncertainty of the prediction. If the QoE prediction exhibits large uncertainty for DNN m, then NeuralUCB has a tendency of selecting DNN m in order to collect its context-QoE data for improving the prediction performance. Otherwise, $u_{t,m}$ is dominated by the predicted QoE value and NeuralUCB selects the DNN that is expected to deliver the highest QoE. The parameter γ is used to adjust the importance of exploration and exploitation.

*Remark on Complexity of NeuralUCB.*Note that OIC runs on mobile devices, it should be assured that the computational complexity of NeuralUCB is acceptable to resource-constrained mobile devices. The computational cost of NeuralUCB lies mainly in training and running QPN. Later in the experiment, we will show that a simple neural network (3-layer fully connected network with 8, 16, 8 nodes) can achieve good performances. Therefore, the space and time complexity of NeuralUCB is low.

3.2.2 Speed Up Learning With Knowledge Transferring

The standard NeuralUCB algorithm learns without any a priori knowledge. It initializes QPNs with random parameters and gradually trains the QPN to approximate the QoE pattern of a particular user. However, certain general knowledge of user QoE patterns is actually available. Although QoE patterns differ across users, the users still share some preferences over certain performance metrics, e.g., all users will prefer higher inference accuracy, lower inference delay, and less energy consumption. In observation of this, we incorporate knowledge transferring [41] into NeuralUCB to speed up the customization process. With knowledge transferring, a pre-trained QPN is used in the initialization of NeuralUCB. The pre-trained QPN contains the general knowledge about user QoE patterns. For example, application developers can collect QoEs of DNNs from a group of test users on standard mobile devices, and train a QPN that works generally well for all tested users. In this way, OIC is able to give better DNN selection decisions compared to a randomly initialized in the early stage. As the online learning proceeds, the pre-trained QPN is retrained to approximate the QoE pattern of a particular user. Using knowledge transferring dramatically accelerates the customization process, and improves the average QoE of the whole learning process.

3.2.3 Performance Guarantee of NeuralUCB

The performance of NeuralUCB is measured by *regret* which defines the performance loss compared to an Oracle. The Oracle knows users' QoE pattern a prior and is able to selects the optimal DNN, $m_t^* = \arg\max_{m \in \mathcal{M}} r_{t,m}$, in every session t. The regret is formally defined as

$$R_T = \mathbb{E}\Big[\sum_{t=1}^{T} r_{t,m_t^*} - r_{t,m_t}\Big]. \tag{1}$$

The performance guarantee of NeuralUCB (in terms of regret upper bound) has been analyzed in [20], showing that NeuralUCB gives a sublinear regret $\mathcal{O}(\sqrt{T})$. This means that NeuralUCB is *asymptotically optimal* compared to Oracle. However, standard NeuralUCB solicits QoE feedback from users after every session. The users may feel these survey questions annoying if they appear constantly. In certain cases, the application developer also needs incentive schemes to motivate the user response. Therefore, a feedback solicitation scheme needs to be designed to control the solicitation cost (e.g., incentive payment and user inconvenience) of the online learning algorithm.

4 Feedback Solicitation

The goal of feedback solicitation scheme (FSS) is to reduce the number of QoE solicitations while keeping the learning performance of NeuralUCB. Let $Q(t)$ denote the event of QoE solicitation in session t, then the solicitation cost is defined as $\lambda \cdot \mathbf{1}\{Q(t) = \text{True}\}$, where λ is the unit cost of one executed solicitation. We define the *reward* in session t as $\tilde{r}_{t,m} = r_{t,m} - \lambda \cdot \mathbf{1}\{Q(t) = \text{True}\}$. The performance of FSS is measured by *modified regret* (m-regret)

$$\begin{aligned}
\tilde{R}_T &= \mathbb{E}\Big[\sum_{t=1}^{T} \tilde{r}_{t,m_t^*} - \tilde{r}_{t,m_t}\Big] \\
&= \mathbb{E}\Big[\sum_{t=1}^{T} r_{t,m_t^*} - r_{t,m_t} + \lambda \cdot \mathbf{1}\{Q(t) = \text{True}\}\Big] \\
&= \underbrace{\mathbb{E}\Big[\sum_{t=1}^{T} r_{t,m_t^*} - r_{t,m_t}\Big]}_{\text{Learning regret}} + \underbrace{\lambda \cdot \sum_{t=1}^{T} \mathbf{1}\{Q(t) = \text{True}\}}_{\text{Solicitation cost}}
\end{aligned}$$

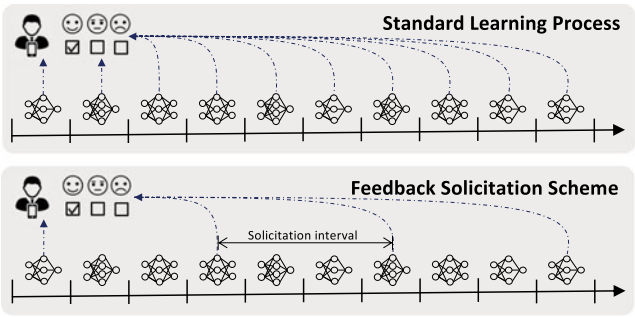


Fig. 3. Illustration of feedback solicitation scheme.

where $\tilde{r}_{t,m_t^*}$ is the reward achieved by Oracle in session t and its value equals r_{t,m_t^*} because Oracle knows the user QoE pattern and does not need QoE solicitations. The m-regret consists of two parts. The first term is the same to the standard regret in (1), which measures the learning performance and we call it *learning regret*. The second term measures the *solicitation cost*. Intuitively, with more frequent QoE solicitations, we are able to learn the user QoE pattern faster, thereby reducing the learning regret. However, frequently soliciting QoEs from users causes a high solicitation cost. Consider the standard learning process of NeuralUCB, although it guarantees a sublinear learning regret $\mathcal{O}(\sqrt{T})$, its solicitation cost increases linearly with the number of application sessions $\mathcal{O}(T)$, resulting in a linear m-regret.

4.1 Feedback Solicitation Scheme

The goal of the feedback solicitation scheme (FSS) is to achieve a sublinear m-regret. To have this done, we need to guarantee that both the learning regret and solicitation cost are sublinear. To obtain a sublinear solicitation cost, the learner needs to reduce the frequency of QoE solicitation, and ensure that the number of QoE solicitations grows sublinearly over time. However, reducing QoE solicitations makes online learning less efficient and enlarges the learning regret of NeuralUCB. Therefore, there is a tradeoff between learning regret and solicitation cost that needs when minimizing the m-regret. Theoretically, we could expect that the lowest m-regret is achieved when the learning regret and solicitation cost reach a balance. In the following proposition, we give out our design of FSS that strikes the balance between learning regret and solicitation cost. Detailed analysis can be found in Appendix A in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2022.3208207

Proposition 1 (Feedback Solicitation Scheme). *Let T be the total number of application sessions, the frequency of QoE solicitation for NeuralUCB should be set to $T^{-1/3}$ for minimizing m-regret.*

Proposition 1 indicates that the QoE solicitation is performed periodically with a fixed frequency $T^{-1/3}$, i.e., QoE solicitation happens every $T^{1/3}$ sessions. If NeuralUCB runs $T = 1000$ sessions, OIC only solicits QoE for $1000^{2/3} = 100$ times, i.e., a 90% reduction of solicitation cost. Fig. 3 illustrates the designed FSS. The solicitation frequency is determined according to the regret upper bound $\mathcal{O}(\sqrt{T})$ of NeuralUCB. FSS aims to guarantee a sublinear m-regret such that NeuralUCB is still asymptotic optimal with reduced QoE solicitations. The theorem below gives an upper bound of m-regret of NeuralUCB with FSS.

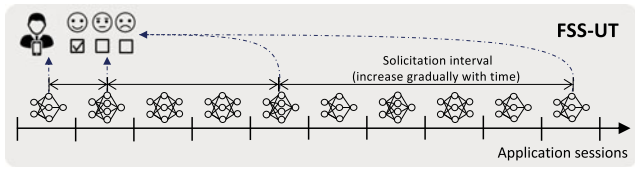


Fig. 4. Illustration of FSS-UT.

Theorem 1 (m-regret of NeuralUCB with FSS). *Suppose NeuralUCB is applied with FSS designed in Proposition 1. For any time horizon T, the upper bound of its m-regret is $\mathcal{O}(\tilde{R}_T) = \mathcal{O}(T^{2/3})$.*

Proof. See Appendix A in the supplementary file, available online. □

The above theorem indicates a sublinear upper bound $\mathcal{O}(T^{2/3})$ for m-regret of NeuralUCB-FSS. This means that our method is able to learn an asymptotically optimal DNN selection policy while keeping the number of QoE solicitations as low as $T^{2/3}$. Note that the regret upper bound given in Theorem 1 is for the worst case with an arbitrarily bad initial QPN. With knowledge transferring, the regret incurred by NeuralUCB is actually much lower, which is verified by our experimental results in Section 6.

4.2 FSS With Unknown Time Horizon

One limitation of FSS is that it needs to know the time horizon T in advance to determine the solicitation frequency. However, the time horizon is often unknown in practice, in which case FSS cannot be applied directly. In the sequential, we show a variant of FSS called FSS-UT that is able to work with an unknown time horizon T. FSS-UT keeps a counter c that counts the number of QoE solicitations executed up to application session t. If the counter satisfies $c \leq t^{1-\alpha}$, then a QoE solicitation will be executed in session t. Fig. 4 illustrates the designed FSS-UT. FSS-UT starts with a relatively small solicitation interval to ramp up QoE data for training the QoE predictor, and then gradually increase the solicitation interval over time. FSS-UT guarantees that $t^{1-\alpha}$ QoE solicitations will be executed up to session t. If we let $\alpha = 1/3$, the number of QoE solicitations executed by FSS-UT in T sessions (i.e., $T^{2/3}$) is the same as that of FSS (Proposition 1), and in this case, the solicitation cost of FSS-UT is the same as FSS. The theorem below provides a performance guarantee of FSS-UT in terms of m-regret.

Theorem 2 (m-regret of NeuralUCB with FSS-UT). *Suppose NeuralUCB is applied with FSS-UT using parameter α. For any time horizon T, the upper bound of m-regret is $\mathcal{O}(\tilde{R}_T) = \mathcal{O}(T^z)$ with $z = \max\{\frac{\alpha+1}{2}, 1-\alpha\}$.*

Proof. see Appendix B in the supplementary file, available online. □

In the above theorem, $\mathcal{O}(T^{(\alpha+1)/2})$ is the order of learning regret, $\mathcal{O}(T^{1-\alpha})$ is the order of solicitation cost, and the balance is achieved at $\alpha = 1/3$. Therefore, the lowest order of m-regret is $\mathcal{O}(T^{2/3})$.

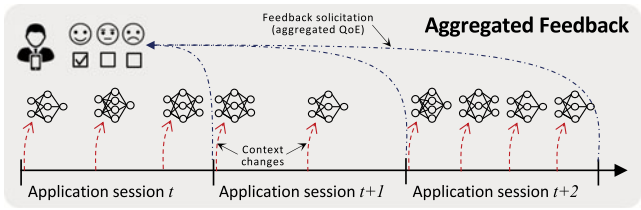


Fig. 5. Illustration of aggregated QoEs.

5 Learning With Aggregated QoE

The previous sections have shown the online learning algorithm, NeuralUCB, and feedback solicitation schemes for solicitation cost reduction. However, the proposed methods rely on two underlying assumptions: 1) only one DNN is selected in each application session and the selected DNN is used during the whole session; 2) the user QoE, if solicited, accurately reflects the QoE of a single DNN used in the session. In practice, an application session can be quite long and the context may change during a session. For example, while using the application, the user may move to different locations and the device battery level will decrease. This requires the application to reconfigure its DNN adaptively according to context changes. In this case, a QoE feedback solicited at the end of an application session may reflect the user experience over all (potentially different) DNNs used in that session, and we call this kind of QoE value aggregated QoE. Fig. 5 gives an illustration of aggregated QoEs. The standard NeuralUCB presented previously cannot handle the aggregated QoE. In this section, we develop a feedback refinement approach as a subroutine of NeuralUCB to deal with aggregated QoEs.

The goal of feedback refinement is to estimate QoEs for individual DNNs (referred to as individualized QoE) based on aggregated QoEs collected from users. A consideration here is that the estimated individualized QoEs are subjected to the constraint that a combination (e.g., a weighted sum) of individualized QoEs should equal the aggregated QoE.

Before presenting the feedback refinement approach, we need a modelling of aggregated QoEs. In application session t, we let K_t denote the times of context change. The set of contexts appeared in session t is collected in $\boldsymbol{x}_t^{\texttt{AG}} = \{\boldsymbol{x}_t^k\}_{k=1}^{K_t}$ where $\boldsymbol{x}_t^k = \{x_{t,m}^k\}_{m\in\mathcal{M}}$ is the kth context of candidate DNNs. Let m_t^k denote the selected DNN given context $\boldsymbol{x}_t^k$. The set of DNNs selected in session t is denoted by $\boldsymbol{m}_t^{\texttt{AG}} = \{m_t^k\}_{k=1}^{K_t}$. The aggregated QoE collected at the end of session t is denoted by $r_t^{\texttt{AG}}$. The contexts $\boldsymbol{x}_t^{\texttt{AG}}$, selected DNNs $\boldsymbol{m}_t^{\texttt{AG}}$, and aggregated QoE $r_t^{\texttt{AG}}$ are stored in memory $\mathcal{X}^{\texttt{AG}} \leftarrow \mathcal{X}^{\texttt{AG}} \cup (\boldsymbol{x}_t^{\texttt{AG}}, \boldsymbol{m}_t^{\texttt{AG}}, r_t^{\texttt{AG}})$.

The feedback refinement approach is a two-step loop towards convergence: 1) individualized QoEs are first estimated with the assistance of the QoE predictor, and then 2) estimated QoEs will be used in return to update the QoE predictor. For example, suppose OIC need to update QPN in session t with the collected aggregated QoEs, the feedback refinement approach iterates between two steps:

Step 1: Given memory $\mathcal{X}$, we estimate individualized QoEs with the current QPN $\hat{r}(\cdot;\boldsymbol{\theta})$. For each sample $(\boldsymbol{x}_\tau^{\texttt{AG}}, \boldsymbol{m}_\tau^{\texttt{AG}}, r_\tau^{\texttt{AG}}) \in \mathcal{X}^{\texttt{AG}}$ with some $\tau \le t$, we compute its group residual

$$\delta_\tau = r_\tau^{\texttt{AG}} - \frac{1}{K_\tau}\sum\nolimits_{k=1}^{K_\tau} \hat{r}\left(x_{\tau,m_\tau^k}^k;\boldsymbol{\theta}\right). \tag{2}$$

Then, the individualized QoE for DNN m_τ^k is estimated by

$$r_\tau^k = \hat{r}\left(x_{\tau,m_\tau^k}^k;\boldsymbol{\theta}\right) + \delta_\tau. \tag{3}$$

It can be easily verified that the average of individualized QoEs estimated by (3) equals the aggregated QoE.

Step 2: The estimated individualized QoEs r_τ^k and context $x_{\tau,m_\tau^k}^k$ are stored in dataset $\mathcal{X}$ for updating the QPN with `TrainQPN` (Algorithm 2). The only difference is that we use estimated individualized QoEs from Step 1 instead of ground-truth QoEs collected from users. The updated QPN will be used in the next iteration.

Algorithm 3. Feedback Refinement Approach

while group residual does not converge **do**
  for $(\boldsymbol{x}_\tau^{\texttt{AG}}, \boldsymbol{m}_\tau^{\texttt{AG}}, r_\tau^{\texttt{AG}}) \in \mathcal{X}^{\texttt{AG}}$ **do**
    Compute residual $\delta_\tau = r_\tau^{\texttt{AG}} - \frac{1}{K_\tau}\sum_{k=1}^{K_\tau} \hat{r}(x_{\tau,m_\tau^k}^k;\boldsymbol{\theta})$
    Get individualized QoE: $r_\tau^k = \hat{r}(x_{\tau,m_\tau^k}^k;\boldsymbol{\theta}) + \delta_\tau$
    Store $(x_{\tau,m_\tau^k}^k, r_\tau^k)$ in $\mathcal{X}$
  end for
  $\boldsymbol{\theta} \leftarrow$ `TrainQPN`$(\mathcal{X})$
end while

The convergence of the above process is proven in [42]. The feedback refinement approach can be easily extended to a scenario where collected QoEs contain both aggregated QoEs and non-aggregated QoEs. In this case, aggregated QoEs are stored in $\mathcal{X}^{\texttt{AG}}$ and will be used to estimate individualized QoEs. The non-aggregated QoEs are stored in $\mathcal{X}$ and are directly used to train QPN. Intuitively, if the fraction of aggregated QoEs is smaller, OIC can achieve better performance. In addition, FSS designed in Section 4 can also be applied in the scenario of aggregated QoE to reduce the solicitation cost.

6 Experiments and Results

Due to the lack of available large-scale real-world datasets, we first run experiments on synthetic data to show that OIC is able to customize appropriate DNN selection policies for a large group of users that have various QoE patterns. We then collect real-world data from real users, which are used to evaluate the performance of OIC for realistic QoE patterns.

6.1 Numerical Experiments

We first construct a mapping from context to user QoE. The user QoE is formulated as a weighted sum of *accuracy* and *delay*: the QoE of user i is defined as $\texttt{QoE}_i = w_i^{\texttt{a}} a_i + w_i^{\texttt{d}} d_i$, where $w_i^{\texttt{a}}$ is the weight for the service accuracy a_i and $w_i^{\texttt{d}}$ is the weight for the service delay. These weights are different for different users to capture the heterogeneous user preference. Both service performance (accuracy a_i and delay d_i) and the user weights ($w_i^{\texttt{a}}$ and $w_i^{\texttt{d}}$) depend on the context information that includes *brightness*, *location*, *CPU temperature*, *time*, and *battery level*, *DNN nominal accuracy*, and *DNN nominal delay*. Three DNN models are considered: MobileNet-v2 (nominal accuracy 70.8%, nominal delay 12ms), Inception-v2 (nominal accuracy 73.5%, nominal delay 59ms), and Inception-v3 (nominal accuracy 77.5%, nominal delay

148ms). The nominal accuracy and nominal delay are from TensorFlow Lite website [21], which are measured on ILSVRC 2012 image classification tasks [43] with Pixel 3 smartphone (Android 10). Note that the nominal performance can be different from the in-use performance because the user device and classification tasks are different.

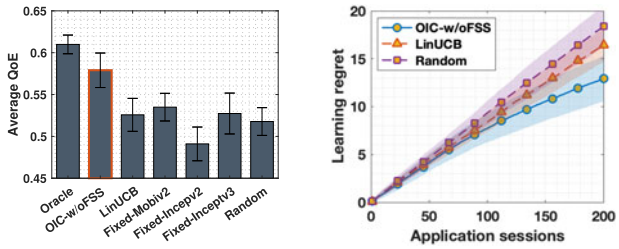
The inference accuracy for user i is determined by the nominal accuracy ($\texttt{nacc}_m$) of selected DNN m and the context ambient brightness ($\texttt{brt}$): $a_i = \texttt{brt}^{-2\cdot\texttt{nacc}_m/\texttt{brt}^{-2}}$. This function indicates that a DNN with higher nominal accuracy is less sensitive to the brightness change. The users may have different requirement on the inference accuracy at different location, and therefore we parameterize the weight by the location context ($\texttt{loc}$): $w_i^{\texttt{a}} = w_i(\texttt{loc})$. The variable $\texttt{loc}$ has 10 discrete values, and the value of $w_i(\texttt{loc})$ is sampled from a normal distribution $\mathcal{N}(\mu(\texttt{loc}), \delta^2(\texttt{loc}))$ where the mean value $\mu(\texttt{loc})$ and the standard deviation $\delta(\texttt{loc})$ are from a uniform distribution [0,2]. The inference delay is affected by the CPU temperature ($\texttt{ctemp} \in [0, 1]$). Usually, the CPU frequency diminishes at the rate of approximately 150 Hz per degree Celsius, and therefore the inference delay may become larger when the CPU temperature is high. Formally, the inference delay is calculated by $d_i = \texttt{ctemp} \cdot \texttt{del}_{mi}$ where $\texttt{del}_{mi}$ is the expected delay of DNN m on user i's device. Because the actually inference delay varies across device, we let $\texttt{del}_{mi} \sim \texttt{N}(\texttt{ndel}_m, \delta^2)$ be sampled from a normal distribution with the nominal delay of DNN m, $\texttt{ndel}_m$, as the mean. The weight for the service delay $w_i^{\texttt{d}}$ depends on the *time* and *battery level*: for example, the user may require faster response during a certain time span; and when the battery level is low, the user may want the computation to be completed in a shorter time to save energy (assuming the CPU frequency is fixed). The time of the delay is discretized into 24 values. For user i, we choose a weight $w_i(\texttt{time}) \in \mathcal{N}(1, 0.5)$. The state of battery level has two values, $\texttt{battery} \in \{1 : \text{'high'}, 2 : \text{' low'}\}$. For each user, we set $w_i(\texttt{battery} = 1) = 1$ and choose $w_i(\texttt{battery} = 0) \in \mathcal{N}(3, 1)$. The weight for the service delay is determined by $w_i^{\texttt{d}} = w_i(\texttt{time}) \cdot w_i(\texttt{battery})$.

We simulate 50 users ($i = 1, 2, \ldots, 50$), and run our OIC method for each user. The QPN used in NeuralUCB has three fully-connected hidden layers with 8, 16, 8 nodes, respectively. During training the continuous contexts and QoE is normalized to [0,1]. The size of QPN is 4 KB; the inference delay of QPN is $0.50 \pm 0.16ms$; the training delay of QPN is $100.8 \pm 3.5ms$. Therefore, the complexity of NeuralUCB is acceptable to mobile devices. The results shown below give the average performance for all 50 users.

6.1.1 Performance Comparisons

To show the superiority of OIC, we compare OIC with four benchmarks. 1) *Oracle*: Oracle knows user QoE patterns and selects the best-fit DNN in each session. 2) *LinUCB*: LinUCB [44] is a widely-used contextual multi-armed bandit algorithm. It assumes that user QoE is a linear function of context information. 3) *Fixed-DNN*: one single DNN is used for on-device inference all the time. 4) *Random*: a DNN is selected randomly in each application session.

Fig. 6 shows the performance of the proposed framework and other benchmarks in terms of average QoE (the scale is [0,1]) and regret. We first analyze the learning capability of OIC by excluding the impact of FSS (OIC-w/oFSS). Fig. 6a compares the average QoE (averaged over sessions and users) of our method and other benchmarks. As expected, Oracle offers the best performance because it knows user QoE patterns and selects the best-fit DNN in each session. Among the others, OIC-w/oFSS delivers the highest QoE to users. By comparing OIC-w/oFSS with LinUCB, we can infer that on-device inference customization does not always work if the learning mechanism is inappropriate. Because the constructed QoE function is non-linear, LinUCB is unable to learn user QoE patterns precisely. Selecting DNNs based on inaccurate QoE predictions, LinUCB performs even worse than Fixed-DNN.



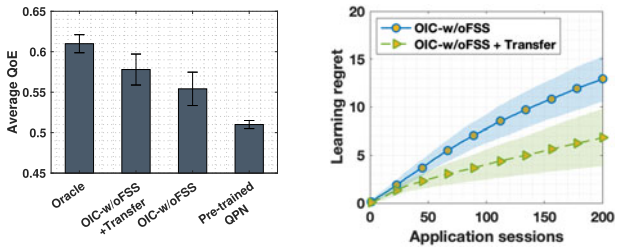
(a) Average QoE. (b) Regret.

Fig. 6. Performance comparison with benchmarks.

Fig. 6b shows the regret of OIC-w/oFSS, LinUCB, and Random in 200 sessions. We can see that the regrets of LinUCB and Random grow linearly with the number of sessions. By contrast, the regret of OIC-w/oFSS exhibits a clear sublinear trend. Recall that regret measures the cumulative performance loss compared to Oracle. A sublinear regret means that the performance loss of our method is very small at the late stage of the learning process. Therefore we say that our method achieves asymptotic optimality.

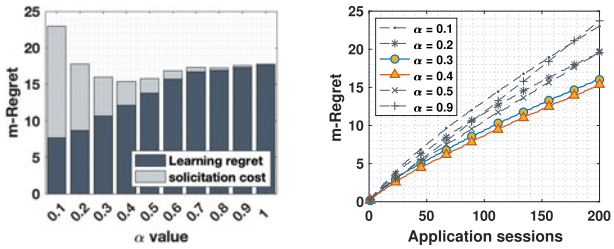
6.1.2 Efficacy of Knowledge Transfer

It can be observed from Fig. 6a that the gap between Oracle and OIC-w/oFSS in average QoE is still large. We implement the knowledge transfer technique to reduce this gap. The first step to carry out knowledge transfer is building a pre-trained QPN. We collect 10 QoE samples from each user, and a total of 500 QoE samples are used to pre-train a QPN. Each user is initialized with this pre-trained QPN and then runs OIC-w/oFSS. Fig. 7 reports the performance of OIC-w/oFSS with knowledge transfer. Fig. 7a compares the averaged QoE achieved by the proposed method and benchmarks. The benchmark 'Pre-trained QPN' in Fig. 7a uses the pre-trained QPN all the time without further training. The QoE gap between OIC-w/oFSS + Transfer and OIC-w/oFSS shows that incorporating knowledge transfer in online learning provides obvious performance improvements. In addition, we see that Pre-trained QPN delivers a much lower average QoE, and this indicates that the pre-trained QPN needs to be further re-trained to better capture the QoE pattern of individual users. Fig. 7b compares the performance in terms of regret. We can see that OIC-w/oFSS+Transfer exhibits a stronger sublinearity in the regret curve, which means that it can find the optimal strategy faster. In the rest of the performance analysis, we run OIC with knowledge transfer by default.



(a) Average QoE. (b) Regret.

Fig. 7. Efficacy of knowledge transfer.



(a) Decomposition of m-regrets. (b) m-regret curves.

Fig. 9. Performance of OIC with FSS-UT.

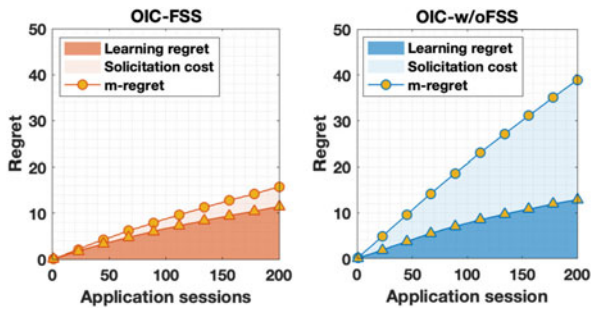
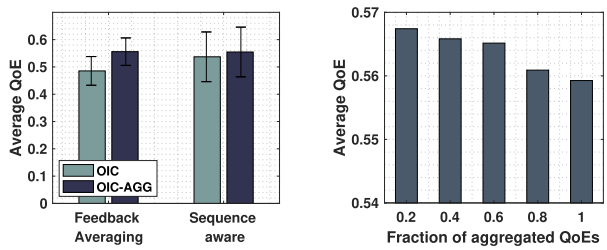


Fig. 8. m-Regret of FSS with known time horizon.



(a) Average QoE. (b) Mixed QoE feedback.

Fig. 10. Performance of NeuralUCB-AGG.

6.1.3 Performance of Feedback Solicitation Schemes

Fig. 8 shows m-regret of OIC when it is applied with and without FSS. We run experiments with $T = 200$ sessions. According to the design, OIC-FSS only needs $\lceil 200^{2/3} \rceil = 35$ solicitations which is much smaller than that of standard NeuralUCB (200 solicitations). Fig. 8 compares m-regret of OIC-FSS and NeuralUCB. The unit solicitation cost λ is set to 0.13. We see that m-regret of OIC-w/oFSS increases almost linearly due to its high solicitation cost. By contrast, OIC-FSS achieves a sublinear m-regret. In particular, using FSS does not harm the learning efficiency as we can see that the learning regret of OIC-FSS is similar to that of OIC-w/oFSS. This means that FSS keeps the asymptotic optimality of NeuralUCB and reduces the increase of solicitation cost to a sublinear rate.

6.1.4 Performance of FSS-UT

Fig. 9 shows the performance of FSS-UT in terms of reward and m-regret. We vary the parameter α in FSS-UT from 0.1 to 0.9. Fig. 9a provides a decomposition of m-regret after 200 sessions. We see that the solicitation cost decreases with α and the learning regret increase with α. The lowest m-regret is achieve at $\alpha = 0.4$ which is close to our theoretical analysis $\alpha = 0.33$. Fig. 9b shows the m-regret curves for FSS-UT, we see that setting when α is appropriately chosen (i.e., $\alpha = 0.3$ or 0.4), our method can achieve sublinear m-regrets, otherwise, the m-regret is nearly linear.

6.1.5 Feedback Refinement for Aggregated QoEs

We use two schemes to generate aggregated QoE.

1) The first scheme is *feedback averaging*. It assumes that all used DNNs in a session have the same impact on the aggregated QoE. Feedback averaging generates aggregated QoEs $r_t^{\texttt{AG}}$ using: $r_t^{\texttt{AG}} = 1/K_t \sum_{t=1}^{K_t} r_t^k$ where r_t^k is the QoE for kth DNN used in session t.

2) The second scheme is *sequence-aware aggregation*. It assumes that the impact of DNNs on the aggregated QoE depends on the selection sequence. We let DNNs picked later in the session have a higher impact: $r_t^{\texttt{AG}} = \sum_{k=1}^{K} w_k r_t^k$ with weights $w_k = 2^k / \sum_{i=1}^{K} 2^i$.

Fig. 10a compares the performance of OIC and OIC-AGG (NeuralUCB with the feedback refinement approach) on aggregated QoEs. Standard NeuralUCB regards the aggregated QoE as the QoE of the last selected DNN in a session. In general, we see that OICB-AGG delivers higher QoE on aggregated QoEs. We can see that the improvement provided by OIC-AGG is smaller with sequence-aware aggregation. This is because aggregated QoEs generated by sequence-aware aggregation are dominated by the QoE of the last selected DNN, and hence standard OIC can still work well. Fig. 10b shows the performance of OIC-AGG with mixed feedback where solicited QoEs contain both aggregated QoEs (generated by feedback averaging) and non-aggregated QoEs. We see that OIC-AGG achieves higher user QoE when the fraction of aggregated QoEs is low. This is because non-aggregated QoEs help estimate individualized QoE in the feedback refinement approach.

6.2 Experiments on Real-World Dataset

We also collect context and QoE data from real-world human users to evaluate the proposed method. In this part, we focus on the customization performance and show how our method adapts DNN selection decisions to different user preferences and usage scenarios.

User QoE Collection. We collect user QoE data on smartphones as they are the most widely used mobile devices. Two types of smartphones, Motorola X^4 and OnePlus One are used in our experiments. Motorola X^4 is equipped with a Qualcomm Snapdragon 625 processor, an octa-core CPU, an Adreno 508 GPU, and a 3GB RAM; OnePlus One is equipped with a Qualcomm Snapdragon 801 processor, a quad-core CPU, an Adreno 330 GPU, and a 3GB RAM. Both

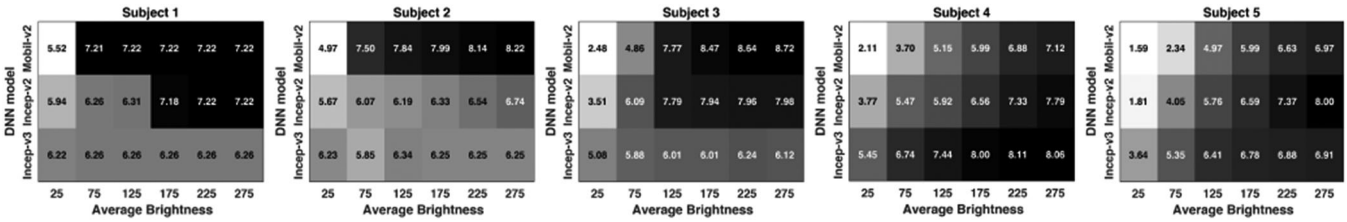


Fig. 11. User QoE v.s. DNN models v.s. Environmental changes. The scale of QoE is [0,10]

smartphones run on the Android 8.1.0 system. We install a DL-based image classification app provided by TensorFlow Lite [21] on these smartphones. The image classification app captures images in real-time using the device camera, then classifies the captured images and displays top-3 classification results. We use three DNN models, MobileNet-v2, Inception-v2, and Inception-v3, as candidate DNNs. These models are pre-trained by TensorFlow and can be downloaded from *TensorFlow Hub* [17]. The ImageClassifier API from *TensorFlow Lite Task Library* is used to deploy a selected DNN into the image classification app. Interested readers are referred to [21] for a detailed description of the image classification app. We recruit five human subjects and collect their QoE scores when using the image classification app. Subject 1, 2, 3 use OnePlus One and subject 4, 5 use Motorola X^4. We ask the human subjects to use the image classification app in various scenarios, e.g., in-house, college campus, classroom, and laboratory. We also encourage the subjects to try different usage patterns, e.g., running the app consecutively for a long time (which may cause an increase in device CPU temperature), or running the app occasionally (which allows the app to be executed under more diverse device status and external environments). The QoE data collection is performed session-wise, at the beginning of each application session, a DNN is randomly selected and deployed in the image classification application, then the subjects record the current contextual conditions. We use a 7-dimension context: the ambient brightness, location of the device, time of the day, battery level, CPU temperature, nominal accuracy of selected DNN, and nominal delay of selected DNN. The subjects are asked to use the image classification app to classify objects in the surrounding environment during the session (around 2 minutes). At the end of the session, subjects provide their QoE scores on a scale of 0 to 10. The subjects are instructed to give a QoE rating that reflects the overall satisfaction of the received inference quality, including but not limited to the actual DNN inference delay/accuracy, energy consumption, device temperature, etc. The subjects are also allowed to take any factors into consideration when giving their QoE scores. The context, selection DNN, and QoE score are stored in a dataset. Each subject runs 200 application sessions, which gives 200 data samples. However, we cannot directly run our method on the collected data due to the lack of counterfactual QoE data. Specifically, each collected data sample records the context information and QoE for a used DNN. When we feed the context of a data sample into OIC, the DNN selected by our method can be different from the DNN we used for collecting the user QoE. As a result, we cannot obtain the ground-truth user QoE for the selected DNN. In this experiment, we utilize the counterfactual prediction method in [45] to generate the counterfactual QoE data. The average error of the predicted counterfactual QoE is 0.294 ± 0.083, 0.3072 ± 0.107, 0.269 ± 0.079, 0.241 ± 0.983, 0.313 ± 0.113 for subject 1, 2, 3, 4, 5, respectively.

6.2.1 Dependency Between QoE and Context

Before analyzing the performance of OIC, we first show the dependency between user QoE and context to support our motivations and claims. Fig. 11 depicts the average QoE of subjects using different DNNs under different contextual conditions (brightness as an example). The results validate some of our claims: 1) The device affects user QoE. We can see that subjects have similar QoE patterns if they use the same device. This because the device computing capacity determines the DNN inference delay which is one of the most key factors that affect user experience. 2) The user preferences are different. Even for subjects using the same device, their QoE patterns exhibit noticeable differences. For example, subject 4 reports a higher QoE using inception-v3 while subject 5 prefers inception-v2. 3) The usage scenario has a significant impact on user QoE. There is a drastic reflection on the QoE variation as we change the brightness of the environment. In particular, we can also observe that users and DNNs have different sensitivity to environmental changes, e.g., the QoE of subject 2 on inception-v2 is relatively stable with different brightness values while the QoE of subject 3 on inception-v2 varies significantly across brightness.

6.2.2 Runtime Performance of DNNs and QPN

Table 2 presents the runtime performance (in terms of inference delay, memory usage, and storage usage) of candidate DNNs and QPN on two smartphones used in the experiment. We see that the actual performance of DNNs on the user device is significantly different from the nominal performances presented in Table 1. It can be observed that the inference/training delay of QPN is around 5.2ms/30.7ms on Motorola X^4 and 10.3ms/50.1ms on OnePlus One. Note that OIC is performed in the app loading phase at the beginning of each application session, a 50ms-delay is almost negligible. The memory usage and storage usage of QPN are also very low. Therefore, OIC is light-weighted enough to be implemented on most mobile devices.

6.2.3 Comparison on Average QoE

Fig. 12 shows (session-)average QoE of 5 subjects when using OIC and other benchmarks. Similar to that in the numerical experiment, OIC outperforms other benchmarks and achieves a close-to-oracle performance. For certain subjects, the performance of Fixed-DNN is close to OIC, e.g.,

TABLE 2
Performance of DNNs and QPN at User Devices

Device & Models \ Metrics		Inference delay (ms)	Memory usage (MB)	Storage usage
Motorola X[4]	Mobi-v2	27.3±3.5	7.3±0.8	1.5MB
	Incep-v2	203.5±39.4	14.6±1.4	11MB
	Incep-v3	489.2±82.8	26.6±3.5	23MB
	QPN	5.2±1.0 (Inference) 30.7 ± 5.8 (Training)	1.1±0.5 (Inference) 7.8 ± 0.7 (Training)	4KB
OnePlus One	Mobi-v2	41.1 ± 6.3	7.2 ± 1.1	1.5MB
	Incep-v2	286.4±57.2	14.6 ± 2.2	11MB
	Incep-v3	861.6 ± 104.9	26.7 ± 5.1	23MB
	QPN	10.3±1.6 (Inference) 50.1 ± 9.2 (Training)	1.1±0.5 (Inference) 7.6 ± 0.8 (Training)	4KB

Fixed-Mobiv2 for Subject 1 and Fix-Incepv3 for Subject 5. However, these subjects still exhibit different QoE patterns (showing different preference over DNNs), and therefore, OIC is still needed for learning user's QoE patterns.

6.2.4 Heterogeneity of User Preference

Fig. 13 shows selected DNNs for 5 subjects in 200 sessions. For ease of interpretation. Based on the information presented in Table 2, we can see that DNNs' inference delay on OnePlus One (used by Subject 1, 2, 3) is much higher than that on Motorola X4 (used by Subject 4, 5). The accuracy of DNNs does not vary much due to device capability (Mobiv2: 70.8%, Incep-v2: 73.5%, Incep-v3:77.5%). In Fig. 13, we see that Mobiv2 is often selected for the subject 1, 2, and 3, meaning that these users prefer low inference delay, and Incepv3 is often selected for the subject 4 and 5, meaning that these users prefer high inference accuracy. OIC is able to adapt its DNN selection policy to different user preferences.

6.2.5 Adaptation to Environment Changes

We next take ambient brightness as an example to show the impact of context on the DNN selection. Fig. 14 shows in what ambient brightness a DNN is more likely to be selected. The color scale denotes the fraction (dimensionless quantity) that a DNN is selected for an ambient brightness range. For user 1, 2, and 3 (prefer low latency), we can see that when the ambient brightness is relatively low (below 100), Incepv2 and Incepv3 are more likely to be selected. This is because low brightness causes accuracy degradation, and in this case, the users may trade low inference delay

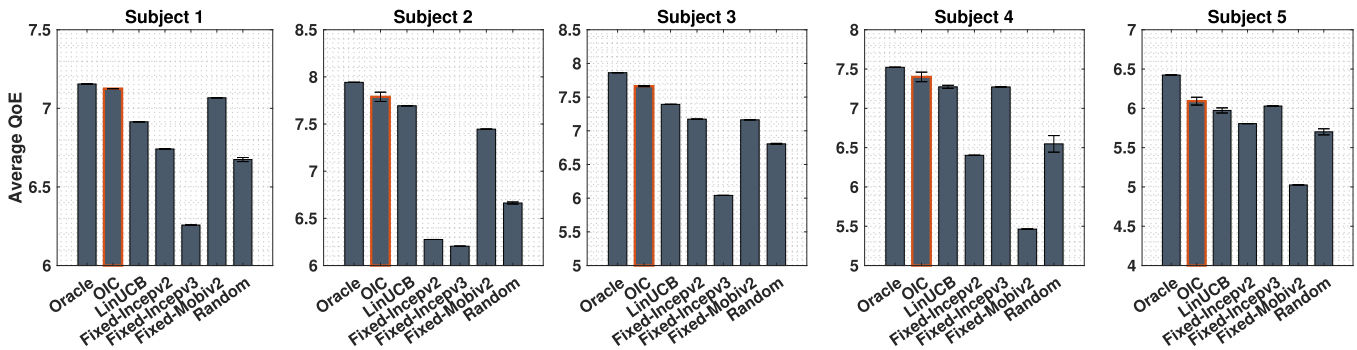


Fig. 12. Average QoE of subjects.

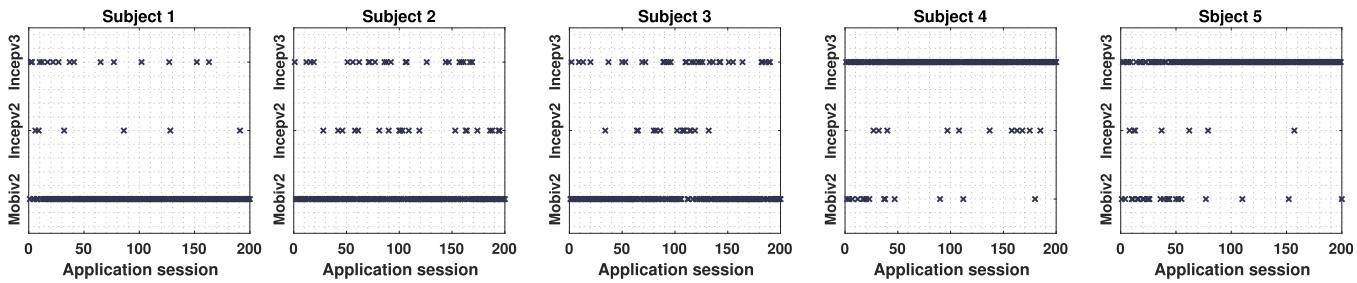


Fig. 13. Selected DNNs for subjects.

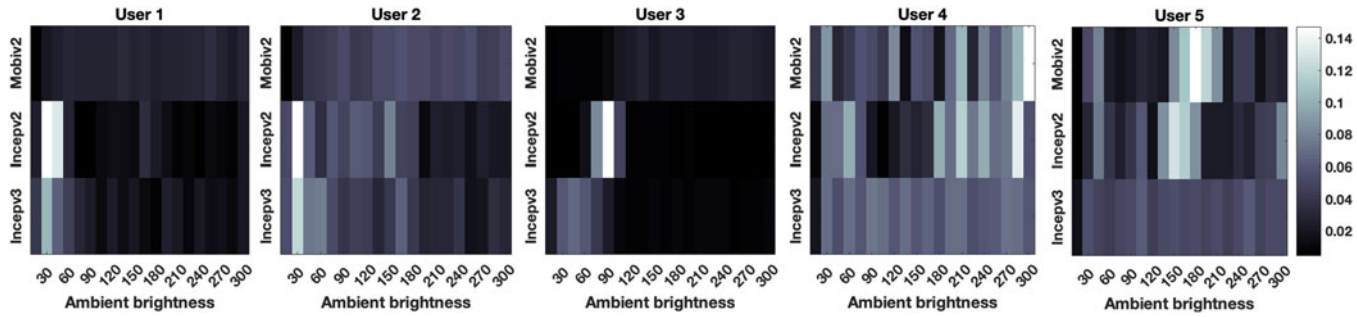


Fig. 14. Impact of ambient brightness on DNN selection.

(i.e., selecting Mobiv2) for better inference accuracy (i.e., selecting Incepv2 and Incepv3). For user 5, we see that Mobiv2 and Incepv2 are more likely to be selected when the brightness is appropriate (between 120-200). This is because all three DNNs can achieve high accuracy when the brightness is appropriate, and hence DNNs with low inference delay are preferred. These results indicate that OIC can adapt to environmental changes.

7 Conclusion

In this article, we designed automated on-device inference customization for DL-based applications. Our goal is to improve the user-perceived experience by selecting the best-fit DNN for configuring the DL-based mobile app. Two main designing topics, QoE prediction and feedback solicitation scheme, are investigated. We utilized NeuralUCB as the learning tool for QoE prediction. NeuralUCB has an excellent generalization ability and is extremely suitable for handling heterogeneous user QoE patterns. The feedback solicitation scheme (FSS) was studied to mitigate the solicitation cost during online learning. With FSS, we dramatically reduced the number of QoE solicitations required by NeuralUCB without harming its asymptotic optimality. While our methods are presented for smartphones, it is compatible with most IoT devices as long as they run DL-based services and have the demand for on-device inference customization. However, there are still many future works to be done. For example, the real-world experiment is still limited, implementing OIC with more DL-based services, device types, and usage scenarios is a meaningful step to evaluate the practicality of the proposed method. The solicitation cost currently is formulated as a pre-determined random variable, carrying out user surveys to obtain the realistic solicitation cost would be a beneficial add-on extra.

References

[1] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 565–576, Feb. 2021.

[2] D. Freire-Obregón, F. Narducci, S. Barra, and M. Castrillón-Santana, "Deep learning for source camera identification on mobile devices," *Pattern Recognit. Lett.*, vol. 126, pp. 86–91, 2019.

[3] C.-J. Wu et al., "Machine learning at facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect.*, 2019, pp. 331–344.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," 2017, *arXiv:1701.01090*.

[5] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," in *Proc. 19th ACM SIGPLAN/SIGBED Int. Conf. Lang., Compilers, Tools Embedded Syst.*, 2018, pp. 31–43 .

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[7] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," 2017, *arXiv:1710.05420*.

[8] Y. Chen, K. Wu, and Q. Zhang, "From QoS to QoE: A tutorial on video quality assessment," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 1126–1165, Second Quarter 2014.

[9] E. Boz, B. Finley, A. Oulasvirta, K. Kilkki, and J. Manner, "Mobile QoE prediction in the field," *Pervasive Mobile Comput.*, vol. 59, 2019, Art. no. 101039.

[10] S.-T. Hong and H. Kim, "Qoe-aware computation offloading to capture energy-latency-pricing tradeoff in mobile clouds," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2174–2189, 2018.

[11] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements," in *Proc. 15th Workshop Mobile Comput. Syst. Appl.*, 2014, pp. 1–6.

[12] B. Finley, E. Boz, K. Kilkki, J. Manner, A. Oulasvirta, and H. Hämmäinen, "Does network quality matter? a field study of mobile user satisfaction," *Pervasive Mobile Comput.*, vol. 39, pp. 80–99, 2017.

[13] M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, and N. D. Lane, "Smart at what cost? characterising mobile deep neural networks in the wild," in *Proc. 21st ACM Internet Meas. Conf.*, 2021, pp. 658–672.

[14] A. G. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4278–4284.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[17] Google, Pretrained image classification models on tensorflow hub, 2020. [Online]. Available: https://tfhub.dev/tensorflow/collections/lite/task-library/image-classifier

[18] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "Embench: Quantifying performance variations of deep neural networks across modern commodity devices," in *Proc. 3rd Int. Workshop Deep Learn. Mobile Syst. Appl.*, 2019, pp. 1–6.

[19] R. T. Carson and T. Groves, "Incentive and informational properties of preference questions," *Environ. Resour. Econ.*, vol. 37, no. 1, pp. 181–210, 2007.

[20] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with upper confidence bound-based exploration," 2019, *arXiv:1911.04462*.

[21] Google, Tensorflow lite hosted DNN models, 2020. [Online]. Available: https://www.tensorflow.org/lite/examples/image_classification/overview

[22] D. Sima, "Apple's mobile processors," 2018. [Online]. Available: https://users.nik.uni-obuda.hu/sima/letoltes/Processor_families_Knowledge_Base_2019/Apple%27s_processor_lines_2018_12_23.pdf

[23] TensorFlow, Tensorflow lite for microcontrollers, 2022. [Online]. Available: https://tensorflow.google.cn/lite/microcontrollers/get_started_low_level

[24] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4820–4828.

[25] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2013, pp. 6655–6659.

[26] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2654–2662.

[27] Google Inc., "TensorFlow Lite: Deploy machine learning models on mobile and edge devices," 2022. [Online]. Available: https://www.tensorflow.org/lite

[28] A. Inc. Core ML: Integrate machine learning models into your app, 2022. [Online]. Available: https://developer.apple.com/documentation/coreml

[29] J. Hosek et al., "Predicting user QoE satisfaction in current mobile networks," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 1088–1093.

[30] P. Casas, R. Schatz, F. Wamser, M. Seufert, and R. Irmer, "Exploring QoE in cellular networks: How much bandwidth do you need for popular smartphone apps?," in *Proc. 5th Workshop All Things Cellular: Operations, Appl. Challenges*, 2015, pp. 13–18.

[31] P. Casas et al., "Predicting QoE in cellular networks using machine learning and in-smartphone measurements," in *Proc. 9th Int. Conf. Qual. Multimedia Experience*, 2017, pp. 1–6.

[32] M. Dasari, S. Vargas, A. Bhattacharya, A. Balasubramanian, S. R. Das, and M. Ferdman, "Impact of device performance on mobile internet qoe," in *Proc. Internet Meas. Conf.*, 2018, pp. 1–7.

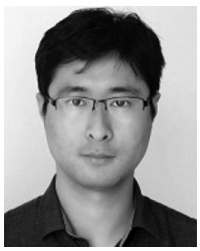
[33] S. I. Venieris, I. Panopoulos, and I. S. Venieris, "OODIn: An optimised on-device inference framework for heterogeneous mobile devices," in *Proc. IEEE Int. Conf. Smart Comput.*, 2021, pp. 1–8.

[34] R. Lee, S. I. Venieris, L. Dudziak, S. Bhattacharya, and N. D. Lane, "MobiSR: Efficient on-device super-resolution through heterogeneous mobile processors," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.
[35] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2016, pp. 123–136.
[36] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," in *Proc. IEEE 1st Int. Conf. Cogn. Mach. Intell.*, 2019, pp. 184–193.
[37] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.
[38] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, "Mutualnet: Adaptive convnet via mutual learning from network width and resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 299–315.
[39] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 1–12.
[40] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synth.*, 2016, pp. 1–10.
[41] G. Konidaris and A. Barto, "Autonomous shaping: Knowledge transfer in reinforcement learning," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 489–496.
[42] A. Bhowmik, M. Chen, Z. Xing, and S. Rajan, "EstimAgg: A learning framework for groupwise aggregated data," in *Proc. SIAM Int. Conf. Data Mining*, 2019, pp. 477–485.
[43] Large scale visual recognition challenge (ILSVRC), 2017. [Online]. Available: http://www.image-net.org/challenges/LSVRC/
[44] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 661–670.
[45] J. Hartford, G. Lewis, K. Leyton-Brown, and M. Taddy, "Deep iv: A flexible approach for counterfactual prediction," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1414–1423.

Yang Bai (Member, IEEE) received the BS degree from Northeastern University, Shenyang, China, the MS degree from the College of Engineering, University of Miami, and the PhD degree from the College of Engineering, University of Miami, USA, in 2021. She is a research associate with the Department of Automation, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China. Her primary research interests include intelligent edge systems and industrial IoT.

Lixing Chen (Member, IEEE) received the BS and ME degrees from the College of Information and Control Engineering, China University of Petroleum, Qingdao, China, in 2013 and 2016, respectively, and the PhD degree in electrical and computer engineering from the University of Miami, in 2020. He is a tenure-track assistant professor with the Institute of Cyber Science and Technology, Shanghai Jiao Tong University, China. His primary research interests include mobile edge computing and machine learning for networks.

Shaolei Ren (Senior Member, IEEE) received the BE degree from Tsinghua University, in 2006, the MPhil degree from the Hong Kong University of Science and Technology, in 2008, and the PhD degree from the University of California, Los Angeles, in 2012, all in electrical and computer engineering. He is an assistant professor of electrical and computer engineering with the University of California, Riverside. His research interests include cloud computing, data centers, and network economics. He was a recipient of the U.S. NSF Faculty Early Career Development (CAREER) Award, in 2015.



Jie Xu (Senior Member, IEEE) received the BS and MS degrees in electronic engineering from Tsinghua University, Beijing, China, in 2008 and 2010, respectively, and the PhD degree in electrical engineering from University of California, Los Angeles, in 2015. He is currently an associate professor with the Department of Electrical and Computer Engineering, University of Miami. His research interests include mobile edge computing/intelligence, machine learning for networks, and network security. He received the NSF CAREER Award, in 2021.