This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2022.3182728

Improving QoE of Deep Neural Network Inference on Edge Devices: A Bandit Approach

Bingqian Lu, Jianyi Yang, Jie Xu, Shaolei Ren

***Abstract*—Edge devices, including in particular mobile devices, have been emerging as an increasingly more important platform for deep neural network (DNN) inference. Typically, multiple lightweight DNN models generated using different architectures and/or compression schemes can fit into a device and thus, selecting an optimal one is crucial in order to maximize the users' quality of experience (QoE) for edge inference. The existing approaches to device-aware DNN optimization are usually time-consuming and not scalable in view of extremely diverse edge devices. More importantly, they focus on optimizing standard performance metrics (e.g., accuracy and latency), which may not translate into improvement of the users' actual subjective QoE. In this paper, we propose a novel automated and user-centric DNN selection engine, called** Aquaman**, which keeps users into a closed loop and leverages their QoE feedback to guide DNN selection decisions. The core of** Aquaman **is a neural network-based QoE predictor, which is continuously updated online. Additionally, we use neural bandit learning to balance exploitation and exploration, with a provably-efficient QoE performance. Finally, we evaluate** Aquaman **on a 15-user experimental study as well as synthetic simulations, demonstrating the effectiveness of** Aquaman**.**

***Index Terms*—Edge inference, quality of experience, bandit, neural network**

I. Introduction

Edge devices, such as smart phones and tablets, are gaining a strong momentum and emerging as a crucially important platform for deep neural network (DNN) inference [1], [2]. For example, DNN models have been commonly integrated with mobile apps for various functions (e.g., real-time style transfer in Facebook app [3]). Compared to cloud-based inference, running DNN inference directly on these edge devices (referred to as *edge inference*) is much less reliant on network connection and also better protects user privacy because of local data processing, thus driving the quick adoption of DNN-powered mobile inference.

To enable a satisfactory user experience of edge inference, numerous DNN optimization techniques for neural architecture search as well as model compression have been recently proposed [4]–[6]. Thus, given an inference task, a large number of diverse DNN models can be generated by navigating through (even a small part of) the huge design space in terms of different neural architectures and compression techniques (e.g., pruning, quantization, and knowledge distillation) [4], [7]–[10]. Generally, different DNN models can exhibit dramatically different tradeoffs among performance metrics, such as accuracy, inference latency and energy consumption [7], [11].

On the other hand, edge devices are extremely diverse, ranging from high-end devices with state-of-the-art CPUs/GPUs and dedicated accelerators to low-end devices powered by CPUs of several years old. For example, the existence of thousands of unique system-on-chips (SoCs) running on over ten thousand different types of mobile phones and tablets further explains the extreme heterogeneity of mobile devices. Importantly, not a single SoC dominates the market: only top 30 SoCs can each have more than 1% of the market share and, when combined, cover 51% of the whole market [3].

Consequently, *how to select the best DNN model out of many choices for each edge device* arises a significant challenge. The DNN selection/optimization for edge inference must be: (1) automated for scalability; and (2) optimizing user's quality of experience (QoE) of edge inference.

Many studies have considered optimally selecting the neural architecture, compression scheme, and/or compiler design for running DNN on a target device [5], [7], [12], [13]. The key idea is to formulate the DNN design as an optimization problem with a pre-determined objective function, such as minimizing a weighted sum of the accuracy loss, inference energy and latency. Nonetheless, the pre-determined objective function is essentially a *proxy* for subjective QoE, without necessarily reflecting the users' true QoE. While it is true that users are generally more satisfied if the latency/energy is lower and the accuracy is higher, using a weighted sum of the accuracy loss, inference energy and latency can be over-simplified for modeling QoE, because users' satisfaction may not be linear in these metrics (as supported by our experimental results in Section V). Even when the actual QoE is indeed a linear function of inference accuracy, energy and latency, the relative weights for these three metrics may not be known in advance and needs online learning based on users' QoE feedback.

More generally, the QoE can be a very complicated function of the DNN performance metrics. Importantly, the QoE function may not be known or accurately estimated in advance without sufficiently collecting the users' QoE feedback. Thus, properly selecting an objective function in advance to identify the QoE-optimal DNN model for a device is very challenging. Additionally, the same DNN model running on different devices can lead to dramatically different performance metrics. As a result, QoE-optimal DNN models can be very different for different devices, thus mandating an automated DNN selection algorithm rather than manual designs for each device.

B. Lu, J. Yang, and S. Ren are with the University of California, Riverside. E-mail: blu029@ucr.edu, jyang239@ucr.edu, and sren@ece.ucr.edu

J. Xu is with the University of Miami. E-mail: jiexu@miami.edu

In this paper, we advocate a different approach — *automated and user-centric DNN selection* — which keeps users into a closed loop and leverages their QoE feedback to guide DNN selections. Specifically, we focus on mobile devices as a concrete platform for edge inference and propose a novel provably-efficient DNN selection engine, called Aquaman, to achieve QoE-optimal mobile inference.

To address the key challenge of the *a priori* unknown QoE, we leverage a machine learning model to approximate the QoE function based on users' QoE feedback that serves as "labels" for training the model. The core of Aquaman is to exploit the universal approximation capability of a neural network-based QoE predictor, which estimates the expected QoE for each DNN selection decision and is updated online using users' QoE feedback. To avoid being trapped in a local optimum due to initially large prediction errors and continuously selecting sub-optimal DNN models, Aquaman employs the bandit-based reinforcement learning framework to balance exploitation and exploration. In particular, we can view each DNN model as an *arm* (or action), while the QoE is our reward. The key idea of bandit algorithms is to give higher priorities to those under-explored arms such that they can be explored more and potentially produce higher rewards in the long run. Concretely, we formulate the DNN selection problem into neural bandit learning with delayed feedback, which is also a novel setting in the emerging context of neural bandit [15]. We also provide rigorous analysis for Aquaman, proving that Aquaman can asymptotically maximize the users' average QoE even compared to an optimal oracle. Finally, we evaluate Aquaman by using an image classification app built on top of the official example of TensorFlow Lite [14]. We consider a 15-user experimental study as well as synthetic simulations, demonstrating that Aquaman outperforms the static DNN selection while approaching the optimal oracle in terms of the users' QoE.

Our main contributions can be summarized as follows:

- We advocate the motivation and necessity of QoE optimal edge inference and user-centric DNN selection approaches.
- We formulate automated user-centric DNN model selection as a multi-armed bandit problem with delayed feedback, and propose an efficient neural bandit approach to balance exploration and exploitation.
- We provide rigorous theoretical analysis, proving that Aquaman can asymptotically maximize the users' average QoE even compared to an optimal oracle.
- We experimentally evaluate our algorithm on a dataset collected via a user study as well as synthetic simulations, demonstrating its effectiveness over the static DNN selection while approaching the optimal oracle in terms of the users' QoE.

II. Motivation for User-Centric DNN Selection

We present two existing approaches to DNN selection and explain the necessity of a user-centric approach for QoE-optimal mobile inference.

Device-unaware DNN selection. A straightforward approach to DNN selection is to test on a number of mobile devices and then conservatively select a single model that can meet the performance requirement for most devices. Although simple, its drawbacks are also obvious: *first*, the selected DNN model is optimal only for certain devices and can perform poorly on others; and *second*, there is not a single DNN model that performs the best on all mobile devices.

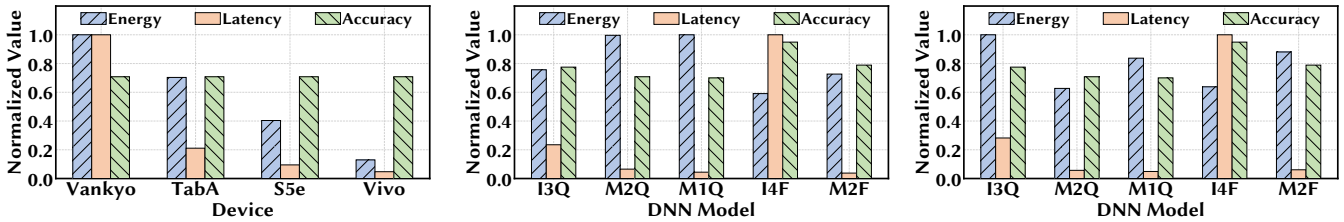
We take the model of MobileNet_V2_1.0_224_quant as an example and deploy it on four mobile devices for image classification (the details of our experimental setup are provided in Section V). We show in Fig. 1(a) the three widely-considered performance metrics for each inference execution: average energy consumption, average latency, and average inference accuracy. The "accuracy" metric in this work is measured in terms of how often the model correctly classifies an image. In our experiment, we run image classification 2000 times in our lab for each DNN model, and obtain the percentage of correct classification as our accuracy. While the model performs well on the high-end device Vivo V1838A, its performance is not satisfactory on low-end devices such as Vankyo Matrixpad z1 (whose average latency is around 1.2s). Our observation is also corroborated by a Facebook study [3], which finds that the same DNN model can result in a large performance variation by a factor of 10x on different devices.

We further test five different DNN models hosted on the official TensorFlow website and show their resulting performances on two mobile devices in Figs. 1(b) and 1(c), respectively. We notice that different models exhibit different performance tradeoffs on different devices. Among the five tested models on Vankyo Matrixpad z1, the MobileNet_V2_1.0_224 model is the most appealing one, because of its low energy and latency while achieving a reasonably high accuracy. We notice that the latency of I4F is much larger than other models in Figs. 1(b) and 1(c). The reasons are: first, I4F is a floating-point model compared to other quantized models; second, I4F has a more complicated model structure than the other floating-point model M2F. This aligns with the official latency measurements in [16].

Device-aware DNN selection. To overcome the drawbacks of the one-for-all approach, many prior studies have considered device-aware DNN optimization/selection [5], [7], [8], [10], [12], [13], [17]–[21]. While the existing approaches can produce an optimal DNN model for a given device, it lacks scalability to a large number of heterogeneous mobile devices. Specifically, even using prediction-assisted optimization, the often lengthy process of building an offline performance predictor is required for *each* target device [7], [13], [22].

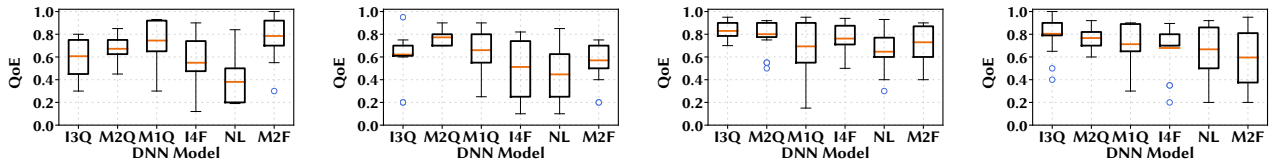
More importantly, optimizing a pre-determined objective function (such as a weighted sum of accuracy loss, inference energy and latency [7], [12]) does *not* necessarily lead to improvement in users' QoE. For example, for mobile devices with limited battery capacities, users may generally prefer more energy-efficient DNN models while willing to accept a lower accuracy and/or increased latency. In fact, the QoE can be a very complicated function that may not be accurately known in advance without sufficiently collecting the users' feedback.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2022.3182728



(a) MobileNet_V2_1.0_224_quant (b) Vankyo Matrixpad z1 (c) Samsung Galaxy Tab A

Fig. 1: (a) Performances of MobileNet_V2_1.0_224_quant on four devices: Vankyo Matrixpad z1 (Vankyo), Samsung Galaxy Tab A (TabA), Samsung Galaxy Tab S5e (S5e), and Vivo V1838A (Vivo). Energy and average latency are normalized to their maximum values on these four devices. (b) and (c) Performances of five DNN models on two devices: Inception_V3_quant (I3Q), MobileNet_V2_1.0_224_quant (M2Q), MobileNet_V1_0.75_192_quant (M1Q), Inception_V4 (I4F), and MobileNet_V2_1.0_224 (M2F). Energy and average latency are normalized to their respective maximum values of the five models on each device. In our experiment, we run image classification 2000 times in our lab for each DNN model, and obtain the percentage of correct classification as the accuracy here.



(a) Vankyo Matrixpad z1 (b) Samsung Galaxy Tab A (c) Samsun Galaxy Tab S5e (d) Vivo V1838A

Fig. 2: Distribution of user ratings regarding six DNN models on four devices. Each rating is normalized to 0-1. "NL" refers to NASNet_large in [14], and the abbreviations for the other five models are shown in Fig. 1.

A survey of users' QoE. To see how satisfied users feel when using DNN-based mobile inference, we recruit 15 participants into our survey to test six different DNN models on four different mobile devices (i.e., a total of 24 DNN-device pairs for each participant) and provide QoE feedback in the form of numeric ratings in 1-10. Each participant is asked to use each of the DNN models for a while and then provide a numerical rating for its QoE with each device-model pair on a scale of 1-10 (more details are in Section V). In addition to the five DNN models shown in Fig. 1, we include another model — NASNet_large (NL) — to verify our intuition that it should score the worst QoE, especially on low- and mid-end devices since it is an overly large and slow model for these devices. Despite the small survey size, our key point is that users' QoE is subjective and not a simple linear combination of the three widely-considered metrics (accuracy, energy and latency) in the existing DNN optimization approaches. Fig. 2 shows the survey results (normalized to 0-1), including the average rating, 25/75 percentile, minimum/maximum rating as well as outliers. Based on the average rating, each device has its own QoE-optimal DNN model: MobileNet_V2_1.0_224 on Vankyo Matrixpad z1, MobileNet_V2_1.0_224_quant on Samsung Galaxy Tab A, and Inception_V3_quant on Samsung Galaxy S5e and Vivo V1838A. Moreover, a common observation is that the overly large model NASNet_large yields almost the worst QoE on all the devices.

III. Overview of Aquaman

As illustrated in Fig. 3, Aquaman is implemented on the server/cloud side and works as a middleman between a pool of available DNN models and users' devices. Aquaman focuses on the selection of already pre-trained DNN models (each having different architectures and/or compression schemes) for mobile inference, and hence model training is orthogonal. The workflow of Aquaman is summarized as follows.

Initialization. The QoE function may not be accurately known in advance without enough feedback from the users. Thus, before running online, Aquaman first initializes a QoE prediction model that takes both DNN model and device features as input and outputs the estimated average QoE. Here, we leverage a fully-connected neural network-based QoE predictor due to its universal approximation capability to approximate any functional relationship (i.e., QoE in our study). In our work, Aquaman focuses on optimizing the users' average QoE. Nonetheless, Aquaman can be extended to optimize user-specific QoE by including user features (e.g., preferences and usage behaviors, if available) into the QoE predictor.

Online DNN selection. During the online stage, Aquaman selects DNN models for mobile devices and gradually updates its QoE predictor based on users' feedback.

QoE prediction. Whenever a mobile user requests a DNN model, Aquaman takes the DNN feature and user's device feature (e.g., CPU and RAM) as input into its QoE predictor and then estimates the resulting average QoE for each DNN in the model pool.

DNN selection. After predicting the QoE for each DNN on the given device, Aquaman selects the DNN that has the highest QoE upper confidence bound (UCB), balancing exploration and exploitation.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2022.3182728

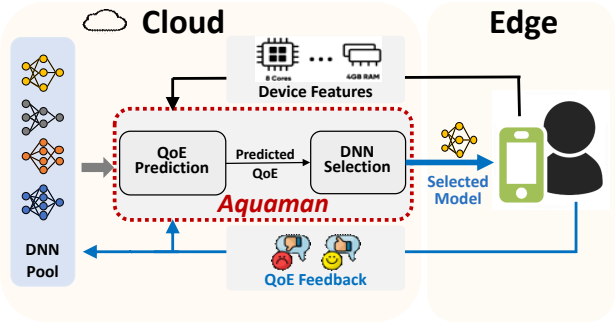


Fig. 3: Overview of **Aquaman**. Aquaman is implemented on the server/cloud side and works as a middleman between a pool of available DNN models and users' devices. Aquaman focuses on the selection of already pre-trained DNN models (each having different architectures and/or compression schemes) for mobile inference. Aquaman mainly consists of QoE prediction and DNN model selection: for a given device, the QoE predictor first outputs the QoE of each device-model pair; then, Aquaman selects the candidate model that generates the highest QoE upper confidence bound for the device.

QoE predictor update. QoE is a subjective score modeled as an *unknown* function of the device and model features. To learn the QoE model/predictor, users are kindly requested to provide feedback regarding their experiences with the selected DNN model in the mobile app after using the models for a while. This can be achieved by prompting a simple interface to solicit users' rating in a numeric scale or simply "like/not like", as commonly used by many of today's mobile apps. If users choose to provide QoE feedback, their QoE values will be leveraged by Aquaman to update the QoE predictor and improve future DNN selections, thus keeping users into a closed loop. Note that Aquaman aims at maximizing the average QoE, while individual QoE values are highly subjective. Thus, we can take the average of a few QoE values (from different users that have the same selected DNN and device) as one *average QoE* sample to update the QoE predictor.

IV. Formulation, Algorithm, and Analysis

A. Preliminaries on Multi-armed Bandits

The multi-armed bandit (MAB) problem models a process of sequential decision making with the tradeoff between exploration and exploitation. In each round t, an action is chosen from a pool of candidate actions (called arms), each of which corresponds to an *a priori* unknown reward. The actual reward is not known until the arm is chosen and played. The goal is to maximize the accumulated expected rewards over T rounds. With more rounds being played, some arms' rewards become better known to us. The tradeoff is a balance between sticking to the arm with the currently known maximal reward (exploitation), or exploring new arms which might give higher rewards in the long run (exploration). MAB models have been widely applied to practical problems, including as ad placement, source routing, computer game-playing, among others [23].

B. Problem Formulation

Consider that Aquaman needs to perform online DNN model selection over T rounds, each corresponding to a mobile device. Let $\mathcal{A}_t$ be the set of pre-trained DNN candidate models to be selected at round t and the number of candidate DNNs at round t is $|\mathcal{A}_t|$. Note that available DNN model set $\mathcal{A}_t$ can be volatile from round to round, because new DNN models may be added and/or only a subset of DNN models can be deployed on the given mobile device due to QoS constraints. The context/feature with respect to a DNN model $a \in \mathcal{A}_t$ at round t is denoted as $x_{t,a} \in \mathbb{R}^d$, which can be obtained by concatenating the mobile device features at round t (e.g. device's CPU and RAM capacity) and the features of DNN model a (e.g., DNN model size, million MACs, million parameters [7], [13]).

Considering the QoE $y_{t,a}$ is a random variable depending on context $x_{t,a}$, we model it as

$$y_{t,a} = h(x_{t,a}) + \eta_t \tag{1}$$

where the average QoE function $h(x_{t,a})$ is a deterministic yet *unknown* function of $x_{t,a}$ with a normalized range $[0, 1]$, and η_t is a ν-sub-Gaussian noise with zero mean conditioned on filtration $\mathcal{F}_{t-1} = \{x_\tau, \eta_{\tau-1}, a_\tau, \tau = 1, \cdots, t-1\}$, i.e. $\forall \varsigma \in \mathbb{R}, \mathbb{E}\left[e^{\varsigma \eta_t} \mid \mathcal{F}_{t-1}\right] \leq \exp\left(\varsigma^2 \nu^2 / 2\right)$.

There is usually a random delay in QoE feedback since users cannot give useful feedback until their models are used for a while. Formally, assuming that if a DNN model is selected for a target mobile device at round s, its corresponding QoE feedback is received after d_s rounds. Then, the set of rounds with QoE fed back at the beginning of round t is expressed as $\mathcal{T}_t^{\mathrm{r}} = \{s \mid s = t - d_s\}$. Also, we assume that the QoE feedback for any $s = 1, \cdots, T$ satisfies $d_s \leq d_{\mathrm{m}}$, i.e., the largest delay is d_{m} after which users are no longer requested for QoE feedback. A larger QoE feedback delay will cause less QoE data collected, whose impact will be analyzed in Theorem IV.1. In practice, some users may not provide any QoE feedback (i.e., missing feedback). This will slow down the QoE predictor updating in Aquaman but does not affect the asymptotic optimality of Aquaman.

The goal of Aquaman is to select DNN models to optimize the average QoE. This is also equivalent to minimizing the *regret*, which is the difference between the optimal average QoE and the average QoE achieved by Aquaman. More formally, we consider the cumulative *regret* as the performance metric for DNN model selection, which is expressed as

$$\begin{aligned} R_T &= \mathbb{E}\left[\sum_{t=1}^{T} \left(y_{t,a_t^*} - y_{t,a_t}\right)\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T} y_{t,a_t^*}\right] - \mathbb{E}\left[\sum_{t=1}^{T} y_{t,a_t}\right], \end{aligned} \tag{2}$$

where $a_t^* = \arg\max_{a \in \mathcal{A}_t} \mathbb{E}\left[y_{t,a}\right] = \arg\max_{a \in \mathcal{A}_t} h(x_{t,a})$ is the optimal DNN model with respect to the expected QoE chosen by the oracle and a_t is the DNN model selected by Aquaman at round t. If the cumulative regret R_T increases sub-linearly with T, the resulting average QoE achieved by Aquaman will be asymptotically optimal as $T \to \infty$ [24].

In our problem, we train a QoE predictor to predict the extent of user satisfaction towards DNN models at an individual device level. That is, we optimize the average QoE among each group of users who use the same type of device, while individual users' personal preferences are modeled as noise in Eqn. (1). Nonetheless, we can extend our QoE predictor by including user-level features (e.g., user's gender, hobby, etc.) if available. This will make the DNN model selection more personalized to individual users (not only devices), although this requires user-level features and might raise user privacy concerns that are beyond the scope of our study.

C. Algorithm for Online DNN Selection

At the beginning of each round, the QoE predictor is updated when at least one new QoE feedback is received, i.e., the set of rounds with QoE feedback at the beginning of round t is $\mathcal{T}_t^{\mathrm{r}} = \{s \mid s = t - d_s\} \neq \emptyset$. Considering the powerful representation capacity of neural networks, a fully-connected neural network is used as the QoE predictor to approximate the *unknown* QoE function $h(x_{t,a})$. Let $f(x, \theta)$ be the QoE neural network with respect to input x and network parameter $\boldsymbol{\theta}$. The network can be trained by Algorithm 2.

The input of Algorithm 2 is prepared as follows. Denote the set of rounds whose delayed QoE feedback is received before round t as $\mathcal{T}_t = \bigcup_{i=0}^{t} \mathcal{T}_i^{\mathrm{r}}$. The training data of the QoE predictor neural network is an input matrix $\mathbf{X}_t$ of size $|\mathcal{T}_t| \times d$, with each row being the context vector x_{s,a_s} for $s \in \mathcal{T}_t$ and a $|\mathcal{T}_t|$-dimensional output vector $\mathbf{y}_t$, with each element being the corresponding received QoE feedback y_{s,a_s}, for $s \in \mathcal{T}_t$. With input $\mathbf{X}_t$ and $\mathbf{y}_t$, the QoE predictor is trained by, e.g., gradient descent with a loss function

$$L(\boldsymbol{\theta}_t) = \frac{1}{2} \sum_{s \in \mathcal{T}_t} (f(x_{s,a_s}; \boldsymbol{\theta}_t) - y_{s,a_s})^2 + \frac{1}{2} \lambda ||\boldsymbol{\theta}_t - \boldsymbol{\theta}_0||_2^2, \tag{3}$$

where $\boldsymbol{\theta}_0$ is the initialized neural network parameters.

With $\boldsymbol{\theta}_t$ being the updated QoE neural network parameter, the QoE at round t is predicted as $f(x_{t,a}, \boldsymbol{\theta}_t)$ where $x_{t,a}$ is the context which concatenates device features with DNN model features. To balance the exploitation and exploration without being trapped in a local optimum, Aquaman selects the DNN model to maximize the corresponding QoE UCB, which is approximated as [15]:

$$p_{t,a} = f(x_{t,a}, \boldsymbol{\theta}_t) + \gamma_{t-1} \|\mathbf{g}(x_{t,a}, \boldsymbol{\theta}_t)\|_{\mathbf{Z}_t^{-1}} \tag{4}$$

where $\mathbf{g}(x_{t,a}, \boldsymbol{\theta}_t)$ is gradient of the neural network of the QoE predictor, $\mathbf{Z}_t = \lambda \mathbf{I} + \sum_{s \in \mathcal{T}_t} \mathbf{g}(x_{s,a}; \boldsymbol{\theta}_t)^T \mathbf{g}(x_{s,a}; \boldsymbol{\theta}_t)$, and γ_{t-1} is a hyperparameter to balance exploration (the first term) and exploitation (the second term): the larger γ_{t-1}, the more emphasis on exploration. Then, the DNN model is selected as:

$$a_t = \arg\max_{a \in \mathcal{A}_t} p_{t,a}, \tag{5}$$

where a_t is the selected model with the largest QoE UCB, and $\mathcal{A}_t$ is the candidate DNN model pool at time t.

Algorithm 1 Aquaman: Online DNN Model Selection with Delayed QoE Feedback

Initialize Neural network parameter is initialized as $\boldsymbol{\theta}_0$. Let $\mathbf{Z}_0 = \lambda \mathbf{I}$. $\mathbf{X}_0$ and $\mathbf{y}_0$ are empty matrix and empty vectors.
for $t = 1, ..., T$ **do**
  if the set of feedback rounds at round t is $\mathcal{T}_t^{\mathrm{r}} \neq \emptyset$ **then**
    $\forall s \in \mathcal{T}_t^{\mathrm{r}}$, append x_{s,a_s} to $\mathbf{X}_t$ and append y_{s,a_s} to $\mathbf{y}_t$.
    Update $\boldsymbol{\theta}_t$ by Algorithm 2;
    Update $\mathbf{Z}_t = \mathbf{Z}_{t-1} + \sum_{s \in \mathcal{T}_t^{\mathrm{r}}} \mathbf{g}(x_{s,a}; \boldsymbol{\theta}_t)^T \mathbf{g}(x_{s,a}; \boldsymbol{\theta}_t)$
  else
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$; $\mathbf{Z}_t = \mathbf{Z}_{t-1}$
  end if
  if the set of feedback rounds before round t is $\mathcal{X}_t = \emptyset$ **then**
    Randomly choose a DNN model a_t;
  else
    Compute UCB $p_{t,a}$ for $a \in \mathcal{A}_t$ by Eqn. (9)
    Select DNN model $a_t = \arg\max_{a \in \mathcal{A}_t} p_{t,a}$;
  end if
end for

Algorithm 2 QoE Predictor Neural network Updating

Input Step size ρ, number of gradient descent steps G, current context matrix $\mathbf{X}_t$, current reward matrix $\mathbf{y}_t$, initialized network parameter $\boldsymbol{\theta}_0$
Compute the gradient $\nabla L(\boldsymbol{\theta}_t)$ of loss function $L(\boldsymbol{\theta}_t)$ in Eqn. (3);
for $j = 0, 1, 2, ..., G-1$ **do**
  $\boldsymbol{\theta}_t^{j+1} = \boldsymbol{\theta}_t^j - \rho \nabla L(\boldsymbol{\theta}_t^j)$
end for
return $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t^G$.

D. Theoretical Analysis

Theorem IV.1. *Assuming that in neural bandit with delayed QoE feedback, the feedback delay satisfies $d_t \leq d_{\mathrm{m}}$, and neural network satisfying the assumptions in the appendix is used as the QoE predictor. Then, with probability $1 - \delta, \delta \in (0, 1)$, the cumulative regret of* Aquaman *satisfies*

$$R_T \leq 2\gamma_T \sqrt{T d_{\mathrm{m}} \left(2\tilde{d} \log(1 + \lfloor T/d_{\mathrm{m}} \rfloor K/\lambda) + 3\right)} + 2d_{\mathrm{m}}, \tag{6}$$

where $\lfloor \cdot \rfloor$ is the floor function, $\tilde{d}$ is the effective dimension of the neural tangent kernel matrix which is defined in Definition 4.3 of [15] and there exists a constant C such that $\gamma_T \leq C\nu \sqrt{\tilde{d} \log(1 + TK/\lambda) + 2 - 2\log\delta}$ where ν is the parameter of sub-Gaussian noise. □

Theorem IV.1, whose proof is in the appendix, extends the performance analysis of the standard neural bandit setting with no feedback delay in [15]. The key point is that, despite the QoE feedback delay, the cumulative regret is sub-linear in T, meaning that Aquaman is asymptotically optimal in terms of the average QoE as $T \to \infty$. Like in the existing

bandit learning literature [15], [25], the regret upper bound quantifies the *worst-case* QoE gap between Aquaman and the optimal oracle, and is mostly accurate in asymptotic cases when $T \to \infty$. Even when T is finite, however, we can empirically observe the shrinking gap between online learning-based Aquaman and the optimal oracle (i.e., the average QoE achieved by Aquaman is approaching that of the oracle).

Now, let us discuss how Aquaman handles delayed QoE feedback. For each round, if there is delayed feedback arriving, Aquaman accumulates it to the previous feedback history and uses the entire history as training data to update the QoE predictor. For the training data, input or feature is the context vectors for each previous rounds, while the labels are the corresponding (potentially delayed) QoE feedback. For the initial rounds when no feedback has arrived yet, Aquaman faces cold-start issues like any other recommendation systems and hence focuses more on exploration. When QoE feedback arrives, we can use it to supervise the learning of the QoE predictor. Naturally, the larger the feedback delay, the potentially worse the QoE (mostly at the beginning). Nonetheless, as time goes on, the impact of feedback delays also becomes less significant, because enough QoE feedback, albeit with delays, has been collected.

E. Practical Considerations

QoS constraint. In practice, a quality-of-service (QoS) constraint may be imposed in terms of, e.g., latency constraint for mobile inference. To handle the QoS constraint, we first estimate the resulting QoS-related performance for DNN models on an incoming device. Unlike QoE, the QoS-related performance does not depend on users' subjective evaluation, and hence can be measured offline based on the DNN developer's own device pool or estimated using performance models [7], [13], [26]. Then, Aquaman will only select those DNNs that meet the QoS constraint.

Updating QoE predictor online. For the ease of analysis, Algorithm 1 states that the QoE predictor is updated whenever new QoE feedback from an individual user is received. In practice, as described in Section III, we can calculate the average QoE feedback from multiple users using the same device and DNN model and consider the average value as one QoE sample to reduce the variation due to individual users' subjective QoE values. Moreover, we can update the QoE predictor neural network using stochastic gradient decent whenever receiving a batch of average QoE values.

Selection of multiple DNNs. In practice, a mobile app may include multiple DNN models, each for one function in the app. This can be addressed by either viewing multiple DNN models as a "super-DNN" ensemble using our current design of Aquaman, or implementing multiple Aquaman in parallel each for one DNN selection. To reduce annoyance to users, the users' QoE feedback is shared for all the selected DNNs.

Malicious user feedback. For any online services, malicious user feedback is an unavoidable problem [27]. In practice, Aquaman can take the average of multiple individual QoE values as one data sample. Thus, the QoE predictor is expected to work well, provided that malicious users are not dominant. Additionally, this issue can be mitigated by using robust online learning [25] and/or removing likely malicious feedback via anomaly detection, which is beyond our focus.

Computational complexity and overhead. Aquaman serves as a middleman selection engine at the server or cloud side. There are many large-scale recommendation systems (e.g., YouTube recommender, Amazon recommender) running in the cloud and serving billions of user requests each day. These systems are also frequently updated based on users' feedback (e.g., click rates). Compared to them, DNN model selection in Aquaman occurs much less frequently, and training our QoS predictor has a much lower complexity. Thus, the computational complexity of Aquaman is affordable and less of a concern. The runtime overhead of Aquaman involves communications (for collecting device features) and QoE prediction. While the communications overheads are addressed by some bandit studies considering rate-limited channels [28], they are not a major concern for our problem. This is because we only need to collect device features (e.g., CPU speed, number of cores, RAM frequency and battery size) for each incoming request. Compared to downloading the DNN model itself which is typically in the order of megabytes or more, transmitting the device features to Aquaman takes negligible bandwidth. Additionally, upon receiving the device features, running the QoE predictor only takes one forward inference, which is in the order of milliseconds. Therefore, the total runtime overhead is negligible for Aquaman.

V. Evaluation Methodology

A. Experimental Setup

Experiment Platform. We set up our experiment platform by building an image classification app for Android based on the official example provided by Tensorflow Lite [14]. We can modify the app by deploying different pre-trained DNN models. We use Android Studio Profiler [29] to measure the energy and latency performance of a DNN running on a mobile device. For each DNN-device pair, we run more than 2,000 inferences to obtain average performance of energy consumption, inference latency, and inference accuracy. Some examples of performance profiling results for five DNN models and four mobile devices are shown in Fig. 1.

Devices and DNN Models. To profile DNN model runtime metrics and collect QoE data, we deploy six DNN models on four different devices. The models are: MobileNet_V2_1.0_224_quant, Inception_V3_quant, MobileNet_V2_1.0_224_quant, MobileNet_V1_0.75_192_quant, Inception_V4, and MobileNet_V2_1.0_224. The devices are Vankyo Matrixpad z1, Samsung Galaxy Tab A, Samsung Galaxy Tab S5e, and Vivo V1838A.

QoE Predictor. Our QoE predictor consists of four fully-connected layers. It takes features of DNN-device pair as input, and outputs the predicted QoE. The input features include device features (i.e., CPU speed, number of cores, RAM size, RAM frequency and battery size) and DNN model features (i.e., million MACs, million parameters, model size, number of nodes and number of layers). After predicting the QoE, the QoE UCB can be computed to select the DNN model

according to Eqn. 5. We train the QoE predictor for 100 epochs for every 100 QoE feedback collected, and our learning rate is set as 0.001.

Datasets. We conduct two types of experiments: experiment on the actually collected user QoE data, and experiment on our synthetic data. For our collected QoE dataset, we run Aquaman for T=20000 rounds. In each round, one of four mobile devices arrives randomly. More details of conducting our user survey, and utilizing the survey results to build the training dataset and gain QoE labels are introduced in Sections V-B and VI-A, respectively. Furthermore, our methodology to synthesize the dataset is presented in Section V-C.

B. User Study

As stated in Section II, we recruit 15 participants into our survey to test six different DNN models on four different mobile devices (i.e., a total of 24 DNN-device pairs for each participant) and provide QoE feedback in the form of numeric ratings in 1-10. Each participant is asked to use each of the DNN models for a while and then provide a numerical rating for its QoE with each device-model pair on a scale of 1-10. The survey result is shown in Fig. 2.

C. Generation of Synthetic Data

Because of limited resources, it is practically impossible for us to evaluate Aquaman on a large scale, which requires interaction with many more mobile users. To circumvent this hurdle, we resort to synthetic data generated as follows.

The idea for generating synthetic QoE values is to utilize our user study to build a synthetic QoE generator that outputs user's average QoE given a DNN-device pair. This synthetic QoE generator is unknown to Aquaman during the evaluation. Specifically, we first build a synthetic performance generator based on our profiling results, and then feed the synthetic performance values to our synthetic QoE generator for producing QoE. The reason we use this indirect approach to generating synthetic QoE instead of directly utilizing DNN-device features as input is that the QoE data we have is not adequate. We consider three widely-considered performance metrics — average energy consumption, inference latency, and inference accuracy — in our synthetic data generation. That is, we mainly consider that the users' QoE depends on these three runtime metrics of a DNN model on a device, which further depend on features of the device-model pair. Thus, the QoE is a function of the DNN and device features. While other factors such as latency variability can affect a user's QoE, they are essentially captured by the noise term in our QoE function in Eqn. (1). To build our synthetic performance generator, we augment the energy/latency performance predictor used the existing research [7], [13] by including device features as additional input. Here, the device features include CPU speed, number of cores, RAM size, RAM frequency and battery size, while additional features such as GPU and operating system version can also be added. To represent DNN models, one can use high-level features such as million MACs, million parameters, model size, number of nodes and number of layers. Alternatively, a finer-grained DNN embedding representation on a block basis can also be used as input for performance generator.

We use kernel ridge regression to fit the relationship between DNN-device features and the resulting performance metrics due to limited mobile devices we have. Next, based on the energy/latency/accuracy values, we build the synthetic QoE generator denoted by $QoE = m_{QoE}(energy, latency, accuracy)$. Again, because of limited QoE data we have, we consider regression methods to approximate $QoE = m_{QoE}(energy, latency, accuracy)$: linear regression, and kernel ridge regression with RBF and Laplacian kernels. Fig. 4 shows the average QoE fitting results, along with importance of different performance metrics. We see that Laplacian kernel ridge regression can almost fully fit into our limited average QoE data, whereas linear regression performs poorly (which also implies that the existing DNN design maximizing linear combination of accuracy, energy and latency [7], [13] may not lead to optimal QoE). We use permutation importance (a.k.a. mean decrease accuracy) as the indicator of feature importance [30], [31]. Fig. 4(d) shows that different regression methods impose different feature importance, but all the three methods gives the top priority to the latency metric feature.

Finally, we note that Aquaman is oblivious of the QoE generator we use; instead, it tries to learn it online (and learn the true user QoE if deployed in the real world).

D. Baseline Approaches

We consider the following representative baselines.

• Static: It selects a single DNN model regardless of the actual mobile devices. The single DNN model is chosen such that the average QoE of all the users is maximized. The assumption is that Static knows a priori which model will result in the maximum average QoE of all the users.

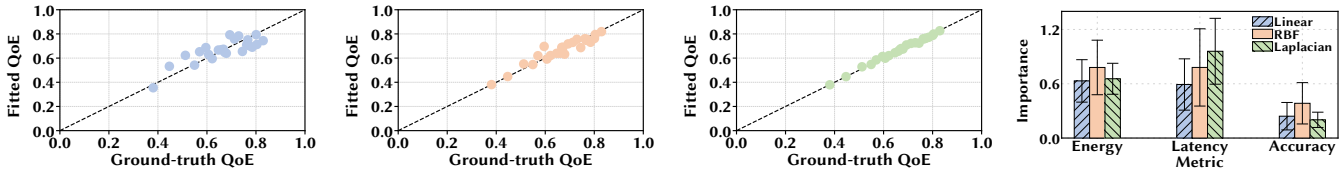
• Oracle: The oracle is assumed to know the best DNN that results in the highest expected QoE for each mobile device. In reality, there does not exist such an oracle, and no practical algorithms can outperform the oracle in terms of the QoE for mobile inference.

In principle, the existing device-aware DNN optimization/selection approaches [5], [7], [8], [10], [12], [13], [17]–[21], [32] could also result in the QoE-optimal DNN model, had the exact QoE function been known for each mobile device in advance. Nonetheless, this is equivalent to assuming the Oracle baseline. On the other hand, if we fix a proxy objective function in the DNN selection process to select a single DNN, it can be even worse than Static, since the proxy objective function may not reflect the actual QoE whereas Static directly optimize for the average QoE. For these reasons, we do not compare Aquaman against the existing approaches that focuses on optimizing (proxy) objective functions.

VI. Evaluation Results

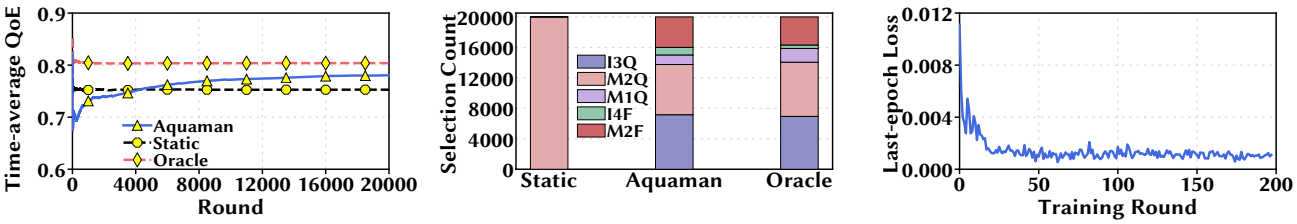
We conduct two types of experiments: on the actually collected user QoE, and on our synthetic dataset (i.e., simulation).

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2022.3182728



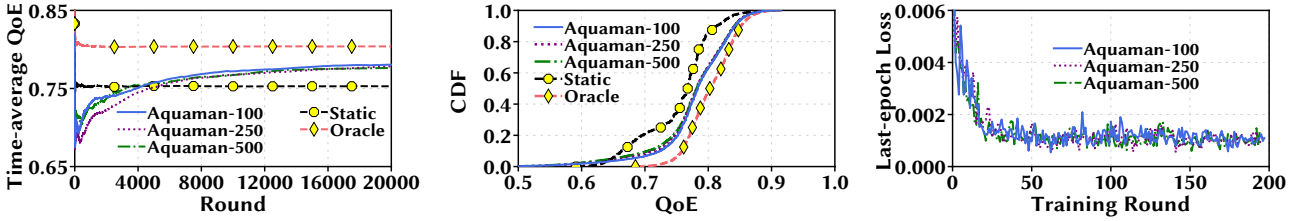
(a) Linear regression (b) Kernel ridge regression (RBF) (c) Kernel ridge regression (Laplacian) (d) Metric importance

Fig. 4: (a)(b)(c) Average normalized QoE fitting results for three regression methods, compared to the average QoE of 15 users. RBF: $\alpha = 0.001$ and $\gamma = 4$. Laplacian: $\alpha = 0.01$ and $\gamma = 1$. (d) Importance of different performance metrics.



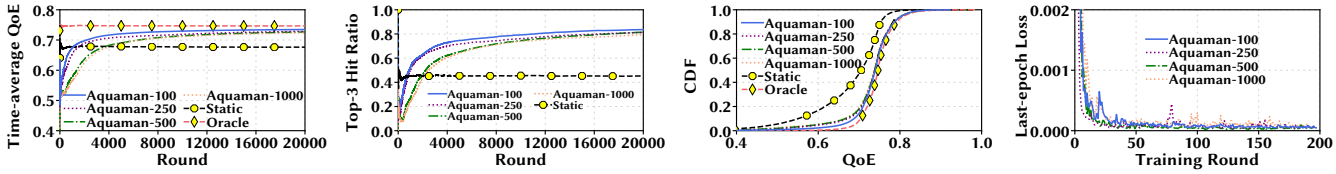
(a) Time-average QoE (b) Selection count of each DNN. (c) Training Loss

Fig. 5: Experiment on collected data. Feedback delay is uniformly distributed on 1-100 rounds.



(a) Time-average QoE (b) QoE CDF (c) Training loss

Fig. 6: Experiment on collected data. "Aquaman-x" means the QoE feedback delay is uniformly distributed between 1 and x.



(a) Time-average QoE (b) Ratio of selecting top-3 DNNs. (c) QoE CDF (d) Training Loss

Fig. 7: Simulation on synthetic data. "Aquaman-x" means the QoE feedback delay is uniformly distributed between 1 and x.

A. *Experiment on the User Study*

We first evaluate Aquaman using the collected QoE data from our user study. While users provide QoE ratings to six DNN models in our survey, we exclude the NASNet_large model from consideration as it is overly slow and has the worst QoE on three of the four devices in our experiment. Thus, for each device, Aquaman selects one out of five pre-trained DNN models. Moreover, our collected user QoE on each DNN-device pair is only limited to 15 samples. Thus, instead of using average QoE of 15 users as one sample, we utilize bootstrapping to randomly pick 8 samples out of 15 QoE data each time and use their average value as one ground-truth QoE sample for training. We run Aquaman for T=20000 rounds. For each round (with a given device), the QoE for each of the five DNN-device pairs is calculated as the average of eight random samples out of 15 QoE data collected by our user study. The training details and other settings are reported in Section V.

We present the results in Figs. 5 and 6(a), where Fig. 5 is for the case when users' QoE feedback delay is distributed uniformly between 1 and 100, while Fig. 6(a) shows more results on different feedback delays. While the absolute value of QoE improvement is subjective, we see clearly that Aquaman gradually improves the time-average QoE, approaching Oracle and outperforming the Static baseline, because our QoE predictor becomes more accurately with more feedback collected and online update. The count breakdown of every single DNN model in Fig. 5(b) shows that the selection count of each candidate DNN model should be similar to the Oracle, and largely different from the Static since it selects merely

one model for all the devices arriving. As for the training loss of our QoE predictor, we show in Fig. 5(c) the last-epoch loss. The total training rounds is about 200 because we train the neural network whenever the number of collected QoE feedback reaches 100 and the total learning rounds is T=20000. Additionally, in Figs. 5(a), 6(a) and 6(b), the time-average QoE is calculated in terms of all devices. As Aquaman takes the device features into account when selecting DNN models, it also achieves higher QoE than Static for each individual type of device.

To see the impact of QoE feedback delays on Aquaman, we show in Fig. 6 the time-average QoE, CDF of QoE, and training loss of Aquaman under uniformly distributed delays of 1-100, 1-250 and 1-500, respectively . It shows that Aquaman is still better than Static.

B. Simulation on Synthetic Data

1) Synthesising DNNs and Mobile Devices: For synthetic evaluation, each DNN is represented by a feature vector, and so is each mobile device. Besides the five DNN models used in Section VI-A, we synthesize another 10 DNN models. We still run Aquaman for T=20000 rounds, in each of which we synthesize a device feature vector to denote a group of mobile devices. Combining the device feature and with DNN feature as input, our synthetic QoE generator produces the average QoE value. To mimic real cases, we will also add noise to the synthetic QoE value.

2) Results: We show the results in Fig. 7, by considering that the synthetic QoE is generated using the RBF kernel ridge regression (Section V-C). The time-average QoE and QoE CDF achieved by Aquaman gradually approaches those of Oracle with an increasingly smaller gap. The reason is that by exploiting the history information, Aquaman can learn the QoE predictor neural network parameter $\boldsymbol{\theta}_t$ with an improved accuracy over time, which is helpful for future DNN model selection. On the contrary, Static constantly yields the lowest QoE, because it does not adapt its DNN model selection to an incoming mobile device's feature. We also show in Fig. 7(b) the percentage of time when the selected DNN is among the top-3 models in terms of the average QoE. It can be observed that Aquaman is much more likely to select a top-3 DNN model than Static. Note that Aquaman keeps on exploring in order to avoid being trapped in local optimum. Hence, the top-3 DNN hit ratio is increasing by using Aquaman but not 100%. Like in Section VI-A, we see that Aquaman is not sensitive to the QoE feedback delays.

We also consider the case when the synthetic QoE values are generated using the Laplacian kernel and a random mixture of two different kernels (Section V-C). The results are similar: because of the strong prediction power of neural networks, Aquaman learn the average QoE for a DNN-device pair over time and hence gradually improve the average QoE for mobile inference. Thus, we omit the results for space limitation.

C. Complexity Analysis

The QoE predictor consists of four fully-connected layers in our experiment, and is updated for 100 epochs whenever a batch of 100 QoE feedback values are collected. The training is very fast and each takes less than 10 seconds on Google Colab platform configured with a regular CPU. For online inference, given each incoming device, we only need to run one forward inference, which is in the order of milliseconds. Even considering large-scale deployment, due to the smaller input features and much less frequent model updating, Aquaman is still much less computationally demanding than industry-level recommendation systems that are frequently updated and serve tens of thousands of users every minute.

VII. Related Work

To turn DNN-based mobile inference into reality, it is crucial to reduce the size of otherwise overly large and computationally prohibitive DNN models by using efficient model compression techniques, such as pruning and quantization [33], matrix factorization [34], compact convolution filters [35], and knowledge distillation [36], among many others. Furthermore, automated neural architecture search is also necessary to identify an appropriate network architecture for mobile inference [7]. The survey [37] comprehensively covers accelerating the training process for large machine learning models in IoT. These studies are complementary to Aquaman: the lightweight DNN models produced by these studies can be included into the DNN pool and selected by Aquaman for QoE-optimal mobile inference.

For device-aware DNN optimization, latency/energy predictors have been utilized for optimization speed-up [7], [38]. Nonetheless, these average latency/energy performance predictors do not incorporate device features. As a result, for every new device, new latency/energy predictors need to be built, which can be time-consuming. More crucially, these studies as well as other relevant approaches [12] focus on optimizing an objective function, which may not improve the users' actual QoE. By contrast, we advocate a scalable user-centric DNN selection approach which keeps users into a closed loop and leverages their QoE feedback to optimize DNN selections.

There have also been studies on runtime DNN model selection/adaptation in view of time-varying environmental/input conditions [39], [40]. While they focus on dynamic selection/adaptation of already-deployed DNN models on mobile devices, Aquaman is different and focuses on DNN model selection during the deployment stage. Moreover, [41] studies model selection and switching between a subset of the machine learning models from a superset of models for Industrial IoT. The goal is to maximize the level of model trustworthiness, which is orthogonal to our study.

Bandit is a classic online learning setting [24], [42]–[44], and our work extends the neural bandit [15] by considering delayed feedback. A recent position paper [45] briefly addresses DNN selection for mobile inference using a linear function as a toy example. By contrast, we propose a provably-efficient online algorithm, leverage a neural network-based QoE predictor with strong representation power, and conduct both experiments and synthetic simulations for evaluation. In a different context, [46] proposes to build a neural network to predict user QoE for video applications, whereas we not only

predict QoE but also propose a bandit algorithm to balance exploration and exploitation.

VIII. Conclusion

In this paper, we propose an automated and user-centric DNN selection engine, called Aquaman, which leverages QoE feedback to optimize DNN selection decisions. Aquaman consists of two integrated parts: QoE prediction and DNN model selection. To balance exploitation and exploration, Aquaman selects DNN models for diverse devices based on the QoE UCB, resulting in provably-efficient QoE performance compared to the oracle. Finally, we evaluate Aquaman on a user study as well as synthetic simulations. We demonstrate the effectiveness of Aquaman by showing that it outperforms the static DNN selection approach while being close to the oracle in terms of users' QoE.

Acknowledgement

Bingqian Lu and Shaolei Ren were supported in part by the U.S. NSF under grants CNS-1910208 and CNS-2007115. Jie Xu was supported in part by the U.S. NSF under grants CNS-2006630 and CNS-2044991.

Appendix
Proof of Theorem IV.1

Assumptions. Without loss of generality, we assume that each hidden layer has the same width m, and the parameter matrix of layer l at time t is $\mathbf{W}_{l,t}$, then the dimension of $\mathbf{W}_{l,t}$ is $\mathbf{W}_{1,t} \in \mathbb{R}^{d\times m}, \mathbf{W}_{l,t} \in \mathbb{R}^{m\times m}$ for $l = 2, ..., L-1$, and $\mathbf{W}_{L,t} \in \mathbb{R}^{m\times 1}$. Assume that the width m can be sufficiently large. In the neural network, each hidden layer is followed by activation operations $\sigma(\cdot)$, such as Rectified Linear Unit (ReLU) function defined as $\sigma(x) = \max(0, x)$. Thus the predicted QoE of arm a in round t is

$$f(x_{t,a};\boldsymbol{\theta}_t) = \sqrt{m}\sigma(\sigma(\sigma(x_{t,a}\mathbf{W}_{t,1})\mathbf{W}_{t,2})\cdots)\mathbf{W}_{t,L-1})\mathbf{W}_{t,L}. \tag{7}$$

We vectorize each of $\mathbf{W}_{l,t}$ and get vector $\mathbf{W}^{'}_{l,t} \in \mathbb{R}^{k\times 1}$, where $k = m \times d$ if $l = 1$, $k = m \times m$ if $l = 2, ..., L-1$, and $k = m$ if $l = L$. Thus neural network parameter at round t in Algorithm 2 can be written as $\boldsymbol{\theta}_t = [\mathbf{W}^{'}_{1,t}; \mathbf{W}^{'}_{2,t}; ...; \mathbf{W}^{'}_{L,t}] \in \mathbb{R}^{(m\times d+m\times m+...+m\times m+m)\times 1}$.

The same initialization method of the neural network as in [15] is adopted. $\mathbf{W}_{l,0}$ is initialized as $\begin{pmatrix}\mathbf{W} & \mathbf{0}\\ \mathbf{0} & \mathbf{W}\end{pmatrix}$ for $0 \le l \le L-1$ with each entry of $\mathbf{W}$ sampled from Gaussian distribution $\mathcal{N}(0, 4/m)$. $\mathbf{W}_{L,0}$ is initialized as $\left[\mathbf{w}^T, -\mathbf{w}^T\right]$ with each entry of $\mathbf{w}$ sampled from Gaussian distribution $\mathcal{N}(0, 2/m)$. And thus the initialized neural network parameters $\boldsymbol{\theta}_0$ is acquired.

The neural network is trained by gradient descent with loss function

$$L(\boldsymbol{\theta}_t) = \frac{1}{2}\sum_{s\in\mathcal{T}_t}(f(x_{s,a_s};\boldsymbol{\theta}_t) - y_{s,a_s})^2 + \frac{1}{2}m\lambda||\boldsymbol{\theta}_t - \boldsymbol{\theta}_0||_2^2, \tag{8}$$

where $\boldsymbol{\theta}_0$ is the initialized neural network parameters and m is the width of the neural network. UCB for each arm $p_{t,a}$ in algorithm 1 can be computed as:

$$p_{t,a} = f(x_{t,a},\boldsymbol{\theta}_t) + \gamma_{t-1}\|\mathbf{g}(x_{t,a},\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}} \tag{9}$$

where $\mathbf{g}(x_{t,a},\boldsymbol{\theta}_t)$ is gradient of the neural network of the QoE predictor, $\mathbf{Z}_t = \sum_{s\in\mathcal{T}_t}\mathbf{g}(x_{s,a};\boldsymbol{\theta}_t)^T\mathbf{g}(x_{s,a};\boldsymbol{\theta}_t)/m$, and the parameter γ_t is set in the same way as the exploration rate in Algorithm 1 of [15] to get a provable sub-linear regret.

Proof. Since the maximum feedback delay is d_{m}, we consider the cumulative regret of the first d_{m} rounds and the cumulative regret from $d_{\mathrm{m}}+1$ to T rounds separately. That means the regret can be written as:

$$\begin{aligned} R_T &= \sum_{t=1}^{T}[h(x_{t,a_t^*}) - h(x_{t,a_t})] \\ &= \sum_{t=1}^{d_{\mathrm{m}}}[h(x_{t,a_t^*}) - h(x_{t,a_t})] + \sum_{t=d_{\mathrm{m}}+1}^{T}[h(x_{t,a_t^*}) - h(x_{t,a_t})] \end{aligned} \tag{10}$$

Since we assume that QoE of any model satisfies $0 \le h(x_{t,a}) \le 1$, the starting regret $\sum_{t=1}^{d_{\mathrm{m}}}[h(x_{t,a_t^*}) - h(x_{t,a_t})] \le d_{\mathrm{m}}$. Denote the continuing regret as $R_T^c = \sum_{t=d_{\mathrm{m}}+1}^{T}[h(x_{t,a_t^*}) - h(x_{t,a_t})]$. According to Lemma 5.3 in [15], with probability $1-\delta, delta \in (0,1)$, we have

$$\begin{aligned} R_T^c \le\ & 2\sum_{t=d_{\mathrm{m}}+1}^{T}\gamma_t \min\left\{\left\|\mathbf{g}(x_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}}, 1\right\} \\ & + C_1\left(Sm^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{7}{6}}\lambda^{-\frac{1}{6}}L^{\frac{7}{2}} + m^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{5}{3}}\lambda^{-\frac{2}{3}}L^3\right), \end{aligned} \tag{11}$$

where C_1 is a constant and S is a constant about the neural tangent kernel matrix defined in Theorem 4.5 of [15]. So the challenge is to bound $\sum_{t=d_{\mathrm{m}}+1}^{T}\|\mathbf{g}(x_{t,a};\theta_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}}$ where $Z_t = \lambda\mathbf{I} + \sum_{s\in\mathcal{T}_t}\mathbf{g}\left(x_{s,a_s},\theta_{s+d_s}\right)\mathbf{g}^T\left(x_{s,a_s},\theta_{s+d_s}\right)/m$ in the case of delayed feedback. For compactness, we denote gradients as $\mathbf{g}_t = \mathbf{g}(x_{t,a_t};\theta_t)$ and delayed gradients as $\bar{\mathbf{g}}_t = \mathbf{g}(x_{t,a_t};\theta_{t+d_t})$.

Devide the $T - d_{\mathrm{m}}$ rounds into d_{m} groups, each with $I = \frac{T-d_{\mathrm{m}}}{d_{\mathrm{m}}}$ elements. Thus, the nth round set, $n \in \mathbb{Z}^+, n \in [1, d_{\mathrm{m}}]$, is $\Omega^n = \{d_{\mathrm{m}}+n, 2d_{\mathrm{m}}+n, \cdots, Id_{\mathrm{m}}+n\}$. Correspondingly, the delayed gradients are also devided into d_{m} groups, each with I elements. In this way, the nth context group is $\left\{\mathbf{g}_{d_{\mathrm{m}}+n}, \mathbf{g}_{2d_{\mathrm{m}}+n}, \cdots, \mathbf{g}_{Id_{\mathrm{m}}+n}\right\}$. For each group n, define I matrices as

$$\begin{aligned} V_i^n &= \lambda\mathbf{I} + \sum_{s=1}^{i}\bar{\mathbf{g}}_{sd_{\mathrm{m}}+n}\bar{\mathbf{g}}_{sd_{\mathrm{m}}+n}^T/m, \\ & i, n \in \mathbb{Z}^+, \quad n \in [1, d_{\mathrm{m}}], \quad i \in [1, I]. \end{aligned} \tag{12}$$

By directly using Lemma 11 in [47], there exists a constant C_2 such that

$$\begin{aligned} \sum_{i=1}^{I}\min\left\{\|\bar{\mathbf{g}}_{id_{\mathrm{m}}+n}/\sqrt{m}\|^2_{(V_{i-1}^n)^{-1}}, 1\right\} &\le 2\log\frac{\det(V_I^n)}{\det(\lambda\mathbf{I})} \\ &\le 2\tilde{d}\log(1+IK/\lambda) + 2 + C_2 m^{-1/6}\sqrt{\log m}L^4T^{5/3}\lambda^{-1/6} \end{aligned} \tag{13}$$

where K is the number of candidate arms, the second inequality is from Lemma 5.4 in [15] and $\tilde{d}$ is the effective dimension of the neural tangent kernel matrix which is defined in Definition 4.3 of [15].

Since the feedback delay is not larger than d_{m}, we have $\forall t > d_{\mathrm{m}}, \mathcal{T}_{t-d_{\mathrm{m}}} \subseteq \mathcal{T}_t$. Let $\Omega_i^n = \{d_{\mathrm{m}}+n, 2d_{\mathrm{m}}+n, \cdots, id_{\mathrm{m}}+n\}$ for $i, n \in \mathbb{Z}^+, i \leq I$. If $t = id_{\mathrm{m}}+n$, then $\Omega_{i-1}^n \subset \mathcal{T}_{t-d_{\mathrm{m}}} \subseteq \mathcal{T}_t$. Since Z_t and V_i^n are both positive-definite matrix, for $t = id_{\mathrm{m}}+n$, $i \leq I$, we have

$$\begin{aligned}\|\bar{\mathbf{g}}_t/\sqrt{m}\|_{Z_t^{-1}} &= \bar{\mathbf{g}}_t^T \left(\lambda\mathbf{I} + \sum_{s\in\mathcal{T}_t} \bar{\mathbf{g}}_s\bar{\mathbf{g}}_s^T/m\right)^{-1} \bar{\mathbf{g}}_t/m \\ &\leq \bar{\mathbf{g}}_t^T \left(\lambda\mathbf{I} + \sum_{s\in\Omega_{i-1}^n} \bar{\mathbf{g}}_s\bar{\mathbf{g}}_s^T/m\right)^{-1} \bar{\mathbf{g}}_t/m \\ &= \|\bar{\mathbf{g}}_t/\sqrt{m}\|_{(V_{i-1}^n)^{-1}}\end{aligned} \tag{14}$$

Therefore, we have the following

$$\begin{aligned}\sum_{t=d_{\mathrm{m}}+1}^{T} ||\bar{\mathbf{g}}_t/\sqrt{m}||_{Z_t^{-1}}^2 &= \sum_{n=1}^{d_{\mathrm{m}}}\sum_{i=1}^{I} \|\bar{\mathbf{g}}_{id_{\mathrm{m}}+n}/\sqrt{m}\|_{\mathbf{Z}_{id_{\mathrm{m}}+n}^{-1}} \\ &\leq \sum_{n=1}^{d_{\mathrm{m}}}\sum_{i=1}^{I} \|\bar{\mathbf{g}}_{id_{\mathrm{m}}+n}/\sqrt{m}\|_{(V_{i-1}^n)^{-1}}.\end{aligned} \tag{15}$$

By Eqn. (13), we have

$$\begin{aligned}&\sum_{t=d_{\mathrm{m}}+1}^{T} \min\left\{||\bar{\mathbf{g}}_t/\sqrt{m}||_{Z_t^{-1}}^2, 1\right\} \\ \leq&\sum_{n=1}^{d_{\mathrm{m}}}\sum_{i=1}^{I} \min\left\{\|\bar{\mathbf{g}}_{id_{\mathrm{m}}+n}/\sqrt{m}\|_{(V_{i-1}^n)^{-1}}\right\} \\ \leq& d_{\mathrm{m}}\left(2\tilde{d}\log(1+IK/\lambda) + 2 + C_2 m^{-1/6}\sqrt{\log m}L^4 I^{5/3}\lambda_{-1/6}\right)\end{aligned}$$

Next, since $\|\mathbf{g}_t\|_{\mathbf{Z}_t^{-1}} \leq \|\bar{\mathbf{g}}_t\|_{\mathbf{Z}_t^{-1}} + \|\mathbf{g}_t - \bar{\mathbf{g}}_t\|_{\mathbf{Z}_t^{-1}}$ according to triangle ineuqality, it is necessary to bound $\|\mathbf{g}_t - \bar{\mathbf{g}}_t\|_{\mathbf{Z}_t^{-1}}$.

By triangle inequality, with probability at least $1-\delta$, there exists a constant C_3 such that

$$\begin{aligned}\|\mathbf{g}_t - \bar{\mathbf{g}}_t\|_2 &\leq \|\mathbf{g}_t - \mathbf{g}(x_{t,a_t}, \theta_0)\|_2 + \|\bar{\mathbf{g}}_t - \mathbf{g}(x_{t,a_t}, \theta_0)\|_2 \\ &\leq C_3\sqrt{\log m}\left(\tau_1^{1/3} + \tau_2^{1/3}\right) L^3 \|\mathbf{g}(x_{t,a_t}, \theta_0)\|_2\end{aligned}$$

where $\tau_1 = 2\sqrt{t/(m\lambda)}$ and $\tau_2 = 2\sqrt{(t+d_t)/(m\lambda)}$, and the second inequality holds due to Lemma B.5 in [15]. Further, since the maximum eigenvalue of Z_t^{-1} is λ^{-1}, we have probability at least $1-\delta$,

$$\begin{aligned}&\left\|\left(\mathbf{g}_t - \bar{\mathbf{g}}_t\right)/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}} \\ \leq&\lambda^{-1/2}\left\|\left(\mathbf{g}_t - \bar{\mathbf{g}}_t\right)/\sqrt{m}\right\|_2 \\ \leq&\lambda^{-1/2}C_2\sqrt{\log m}\left(\tau_2^{1/3}\right) L^{7/2} \\ \leq&2C_3 m^{-1/6}\sqrt{\log m}\,(t+d_{\mathrm{m}})^{1/6}\,\lambda^{-2/3}L^{7/2},\end{aligned} \tag{16}$$

where the second inequality comes from Eqn. (16), Lemma B.6 in [15] and the fact that $\tau_2 \geq \tau_1$. Now, with probability at least $1-\delta$, we have

$$\begin{aligned}&\sum_{t=d_{\mathrm{m}}+1}^{T} \min\left\{\left\|\mathbf{g}_t/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}}^2, 1\right\} \\ \leq&\sum_{t=d_{\mathrm{m}}+1}^{T} \left[\min\left\{\left\|\bar{\mathbf{g}}_t/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}}^2, 1\right\} + \left\|\left(\mathbf{g}_t - \bar{\mathbf{g}}_t\right)/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}}^2\right] \\ \leq& d_{\mathrm{m}}\left(2\tilde{d}\log(1+IK/\lambda) + 2 + C_2 m^{-1/6}\sqrt{\log m}L^4 I^{5/3}\lambda^{-1/6}\right) \\ &+ 2C_3 m^{-1/6}\sqrt{\log m}\,(T+d_{\mathrm{m}})^{7/6}\,\lambda^{-2/3}L^{7/2}.\end{aligned}$$

By Eqn. (11), we have with probability at least $1-\delta$,

$$\begin{aligned}R_T^c &\leq 2\sqrt{T}\gamma_T\sqrt{\sum_{t=d_{\mathrm{m}}+1}^{T} \min\left\{\left\|\mathbf{g}(x_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\right\|_{\mathbf{Z}_t^{-1}}^2, 1\right\}} \\ &\quad + C_1\left(Sm^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{7}{6}}\lambda^{-\frac{1}{6}}L^{\frac{7}{2}} + m^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{5}{3}}\lambda^{-\frac{2}{3}}L^3\right) \\ &\leq 2\gamma_T\sqrt{Td_{\mathrm{m}}\left(2\tilde{d}\log(1+IK/\lambda) + 2 + U\right)} \\ &\quad + 2\gamma_T\sqrt{2TC_3 m^{-1/6}\sqrt{\log m}\,(T+d_{\mathrm{m}})^{7/6}\,\lambda^{-2/3}L^{7/2}} \\ &\quad + C_1\left(Sm^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{7}{6}}\lambda^{-\frac{1}{6}}L^{\frac{7}{2}} + m^{-\frac{1}{6}}\sqrt{\log m}T^{\frac{5}{3}}\lambda^{-\frac{2}{3}}L^3\right) \\ &\leq 2\gamma_T\sqrt{Td_{\mathrm{m}}\left(2\tilde{d}\log(1+IK/\lambda) + 3\right)} + 1,\end{aligned}$$

where $U = C_2 m^{-1/6}\sqrt{\log m}L^4 I^{5/3}\lambda^{-1/6}$ and the third equation holds with a sufficiently large m. Recalling $I = \lfloor T/d_{\mathrm{m}} \rfloor$, we have with probability at least $1-\delta$,

$$\begin{aligned}R_T &= \sum_{t=1}^{d_{\mathrm{m}}}[h(x_{t,a_t^*}) - h(x_{t,a_t})] + R_T^c \\ &\leq 2\gamma_T\sqrt{Td_{\mathrm{m}}\left(2\tilde{d}\log(1+\lfloor T/d_{\mathrm{m}}\rfloor K/\lambda) + 3\right)} + 2d_{\mathrm{m}}.\end{aligned} \tag{17}$$

By Lemma 5.4 in [15], there exists a constant C_4 such that $\gamma_T \leq C_4\nu\sqrt{\tilde{d}\log(1+TK/\lambda) + 2 - 2\log\delta}$ where ν is the parameter of sub-Gaussian noise. □

REFERENCES

[1] L. Xiao, Y. Ding, D. Jiang, J. Huang, D. Wang, J. Li, and H. V. Poor, "A reinforcement learning and blockchain-based trust mechanism for edge networks," *IEEE Transactions on Communications*, 2020.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[3] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at Facebook: Understanding inference at the edge," in *HPCA*, Washington, DC, USA, February 2019.

[4] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *ICLR*, New Orleans, LA, USA, May 2019.

[5] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *CVPR*, Long Beach, CA, USA, June 2019.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, 2016.

[7] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint search for network architecture, pruning and quantization policy," in *CVPR*, Virtually, June 2020.

[8] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *AAAI*, New York, New York, USA., February 2020.

[9] W. Liu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, Virtually, March 2020.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, San Juan, Puerto Rico, May 2016.

[11] Y. Liu, Y. Zhu, and J. James, "Resource-constrained federated learning with heterogeneous data: Formulation and analysis," *IEEE Transactions on Network Science and Engineering*, 2021.

[12] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *MobiSys*, Munich, Germany, June 2018.

[13] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *CVPR*, Long Beach, CA, USA, June 2019.

[14] Google, "Tensorflow lite image classification app," https://www.tensorflow.org/lite/models/image_classification/overview.

[15] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with upper confidence bound-based exploration," in *ICML*, Virtually, July 2020.

[16] Google, "Tensorflow lite image classification hosted models," https://www.tensorflow.org/lite/guide/hosted_models.

[17] H. Cai, L. Zhu, and S. Han, "ProxylessNas: Direct neural architecture search on target task and hardware," in *ICLR*, Long Beach, CA, USA, June 2019.

[18] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in DNN weight pruning for real-time execution on mobile device," in *AAAI*, New York, New York, USA, February 2020.

[19] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2923–2960, Fourthquarter 2018.

[20] B. H. Ahn, P. Pilligundla, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Chameleon: Adaptive code optimization for expedited deep neural network compilation," in *ICLR*, Virtually, April 2020.

[21] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," in *ICCAD*, Westminster, Colorado, USA, November 2019.

[22] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *MobiSys*, Virtually, July 2021.

[23] S. Bubeck, N. Cesa-Bianchi *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[24] D. Bouneffouf and I. Rish, "A survey on practical applications of multi-armed and contextual bandits," in *arXiv*, 2019.

[25] J. Kirschner, I. Bogunovic, S. Jegelka, and A. Krause, "Distributionally robust bayesian optimization," in *AISTATS*, Virtually, September 2020.

[26] B. Lu, J. Yang, W. Jiang, Y. Shi, and S. Ren, "One proxy device is enough for hardware-aware neural architecture search," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, dec 2021.

[27] A. Mukherjee, B. Liu, and N. Glance, "Spotting fake reviewer groups in consumer reviews," in *WWW*, New York, New York, USA, April 2012.

[28] F. Pase, D. Gunduz, and M. Zorzi, "Contextual multi-armed bandit with communication constraints," 2021.

[29] Google, "Android profiler," https://developer.android.com/studio/profile/android-profiler.

[30] ELI5, "Permutation importance," https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html.

[31] L. Breiman, "Random forests," *Machine learning*, 2001.

[32] Y. Wang, "Towards ultra-efficient dnn inference acceleration on edge devices for wellbeing applications," in *HealthDL*, Toronto, Ontario, Canada, June 2020.

[33] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," 2017. Available at: https://arxiv.org/abs/1709.02260.

[34] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NeurIPS*, Montreal, Quebec, Canada, December 2014.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. Available at: https://arxiv.org/abs/1704.04861.

[36] R. Sharma, S. Biookaghazadeh, B. Li, and M. Zhao, "Are existing knowledge transfer techniques effective for deep learning with edge devices?" in *EDGE*, San Francisco, CA, USA, July 2018.

[37] H. Wang, Z. Qu, Q. Zhou, H. Zhang, B. Luo, W. Xu, S. Guo, and R. Li, "A comprehensive survey on training acceleration for large machine learning models in iots," *IEEE Internet of Things Journal*, 2021.

[38] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "*NeuralPower*: Predict and deploy energy-efficient convolutional neural networks," in *ACML*, 2017.

[39] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," Philadelphia, PA, USA, June 2018.

[40] S. S. Ogden and T. Guo, "Characterizing the deep neural networks inference performance of mobile applications," in *arXiv*, 2019, https://arxiv.org/abs/1909.04783.

[41] B. Qolomany, I. Mohammed, A. Al-Fuqaha, M. Guizani, and J. Qadir, "Trust-based cloud machine learning model selection for industrial iot and smart city services," *IEEE Internet of Things Journal*, 2020.

[42] W. Chen, Y. Wang, and Y. Yuan, "Combinatorial multi-armed bandit: General framework, results and applications," in *ICML*, Atlanta, GA, USA, June 2013.

[43] V. Saxena, J. Jaldén, J. E. Gonzalez, M. Bengtsson, H. Tullberg, and I. Stoica, "Contextual multi-armed bandits for link adaptation in cellular networks," in *Workshop on Network Meets AI & ML (NetAI)*, Beijing, China, August 2019.

[44] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *WWW*, Raleigh, NC, USA, April 2010.

[45] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," in *CogMI*, Los Angeles, California, USA, December 2019.

[46] H. Zhang, L. Dong, G. Gao, H. Hu, Y. Wen, and K. Guan, "Deepqoe: A multimodal learning framework for video quality of experience (qoe) prediction," *IEEE Transactions on Multimedia*, 2020.

[47] Y. Abbasi-yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits," in *NIPS*, Granada, Spain, December 2011.