Received 17 August 2021; revised 25 April 2022; accepted 25 September 2022.
Date of publication 21 October 2022; date of current version 7 June 2023.

Digital Object Identifier 10.1109/TETC.2022.3214931

NeuE: Automated Neural Network Ensembles for Edge Intelligence

YANG BAI, (Member, IEEE), LIXING CHEN, (Member, IEEE), AND JIE XU, (Senior Member, IEEE)

Yang Bai is with the Department of Automation, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
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
Jie Xu is with the Department of Electrical and Computer Engineer, University of Miami, Coral Gables, FL 33146 USA

CORRESPONDING AUTHOR: LIXING CHEN (lxchen@sjtu.edu.cn)

The work of L. Chen was supported in part by the National Natural Science Foundation of China under Grant 62202293.
The work of J. Xu was supported in part by NSF under Grants 2006630, 2044991, 2029858 and 2033681.

ABSTRACT Artificial Intelligence (AI) applications have been established in the mobile industry and are decisively determining the progress in entrepreneurial value creation. This article explores the potential of *Edge Computing* to enhance the performance of AI applications. In particular, a DNN ensemble formation (DEF) problem is studied which judiciously recruits members for DNN ensembles considering the device heterogeneity, computing resource limitation, and service deadline of edge computing systems, in an attempt to optimize the performance of edge AI services. We design a novel algorithm called Neural Ensemble (NeuE) to solve the DEF problem. NeuE involves an online learning process that learns the in-practice performance of DNN ensembles and adaptively forms DNN ensembles according to the features of admitted tasks. It leverages the framework of contextual multi-armed bandit and follows the constraints of computing resource limitation and service deadline. We also show theoretically that NeuE provides asymptotic optimality. However, NeuE suffers from poor scalability due to exponentially-growing ensemble decision space. We then propose a variant of NeuE, called NeuE-S, to expedite NeuE. NeuE-S identifies representative ensemble decisions using similarities of ensemble decisions and carries out learning with a reduced decision space. We show via theoretical analysis that NeuE-S drastically reduces the computation complexity with negligible performance loss. We implement our method on an edge computing testbed. The results show that our method dramatically improves the performance of edge AI services.

INDEX TERMS Edge computing, DNN ensemble formation, multi-armed bandit

I. INTRODUCTION

Deep learning has undoubtedly revolutionized artificial intelligence (AI) in a range of complicated domains, providing performance comparable to or even exceeding the human-level capability. Driven by the huge market of mobile and embedded devices, e.g., smartphones, self-driving cars, and smart home appliances, there is a continuous trend to push AI functionalities to these end-devices. On one hand, deep neural networks are made more compact with lighted libraries (e.g., Tensorflow Lite [1] and Core ML [2]) and DNN compression techniques (e.g., quantizing [3] and pruning [4]) to adapt the constrained computing resource at mobile and embedded devices. On the other hand, the mobile and embedded devices are now equipped with additional hardware acceleration, e.g., Graphics Processing Units (GPUs) or Neural Network Processing Units (NPUs) to support AI applications. While implementing AI applications on mobile devices is becoming feasible, it is unlikely to be a universal solution for all mobile devices due to the substantial heterogeneity in computing capacity and variations in device status [5]. Besides, running AI applications tends to incur large energy consumption [6], putting off its usage when the device battery is low. *Edge Computing* [7] is envisioned as a promising external booster to realize the full potential of mobile AI applications. It migrates the computation burden of AI application from mobile devices to edge servers deployed at the network edge — the AI service provider (ASP) configures its AI service (a counterpart of mobile AI applications) and related deep neural

networks (DNNs) at edge servers, and the users in the service area can offload their tasks to edge servers for processing. Furthermore, edge AI services can provide better performance with high-quality computing resources at edge servers. For example, powerful GPUs may be available for accelerating DNN processing, and complex DNN architectures that are computationally over-expensive for mobile devices can now be applied to improve the inference quality.

The goal of this paper is to improve the performance of edge AI services by exploiting the DNN ensemble techniques. The DNN ensemble technique [8], [9] has achieved state-of-the-art performances for many AI applications, e.g., the winners of ILSVRC [10] designed their methods based on the DNN ensemble technique. Several recent works [11], [12] also showed the DNN ensemble technique helps defend adversarial attacks and improve the robustness of inference results. The deployment of edge computing platforms grants ASPs access to powerful computing resources, making it possible to run resource consuming DNN ensemble technique at network edge. To use the DNN ensemble technique for edge AI services, the ASP will first form an ensemble of DNNs at an edge server, and then feed the received tasks to each of the DNNs in the ensemble, at last, the outputs of individual DNNs are combined to generate final results. This overall process seems straightforward, however, there are several issues to be addressed before the DNN ensemble technique can deliver what it is capable of on edge computing platforms.

1) The first issue is the space/time complexity control of DNN ensembles for edge AI services. While using DNN ensemble provides better service performance, it also incurs higher space and time complexity due to running multiple DNNs. The complexity control of DNN ensemble is of great importance for edge AI services from two aspects: 1) Computing resources at edge servers are limited compared to cloud [13], besides, ASPs may operate under budget constraints that only allow them to use a portion of edge computing resources. Therefore, the space complexity of DNN ensembles should be kept below the computing resource constraint at edge servers. 2) Running multiple DNNs incurs larger inference delays. Because edge AI services are often latency-critical, and therefore the time complexity of DNN ensemble should be judiciously managed to guarantee in-time result return.

2) The second issue is the task feature variation caused by user device heterogeneity and its unknown impact on the DNN performance. The user devices in the edge computing system can be extremely diverse, including smartphones, intelligent vehicles, security cameras, and etc. These devices are equipped with different hardware and are operated in different usage scenarios, which affects the task inputs to DNNs. Consider the image classification as an edge AI service, the device camera determines the image resolution, and the usage scenario affects the image brightness. We call these associated features the *context* of tasks. DNNs usually have different sensitivities to the change of task context. What is thornier is that the impact of context on the DNN performance is not fully known to ASP. However to address the task feature variations caused by device heterogeneity is an important designing goal to be considered.

3) The third issue is the obscured impact of individual DNNs on DNN ensembles. A DNN ensemble integrates individual prediction results of its members to generate a final prediction result, and hence the performance of DNN ensembles is determined by its constituting DNNs. However, it is difficult to characterize the inter-dependency between DNN members. The performance of an ensemble is affected by many factors, e.g., the quality of individual DNNs, diversity among DNNs, orthogonality of complementary of DNNs' training/validation datasets [14], and how these factor affects the performance of DNN ensembles is still unclear. Existing works [14], [15] provided several heuristic rules to form a good ensemble, e.g., picking DNNs that have the best individual performance [15] or DNNs that exhibit high diversity [16]. However, the performance of heuristic rules depends heavily on the applied data [15]. There is still no consensus in the community on how to build an optimal DNN ensemble from individual DNNs.

This paper defines DNN ensemble formation problem that jointly see to above three challenges. We design online learning algorithms to provide a solution to DNN ensemble formation problem. The crux is to learns the performance of DNN ensembles over the task feature space and identify the best-fit DNN ensemble for received user tasks. In addition, the proposed method controls the space and time complexity of DNN ensembles to guarantee that the formed DNN ensemble can be implemented with constrained edge computing platforms and return task results before service deadline. The key contributions of this paper are summarized as follows:

1) A novel DNN ensemble formation (DEF) problem is formulated which aims to improve the performance of edge AI services by identifying the best-fit DNN ensembles for user tasks. The DEF problem takes into account unique properties of edge computing platforms including device heterogeneity, edge resource constraint, and service deadline. In addition, the formulation of DEF presented in this paper is compatible with most edge computing platforms and AI services, providing a general solution for ASPs.
2) We address the DEF problem using the philosophy of "AI-for-AI" — leveraging AI techniques to form DNN ensembles for AI service provisioning. The DNN ensemble formation is automated by an online learning algorithm that adaptively forms a best-fit ensemble based on the context of received tasks. Our algorithm learns the performance of different *neural* network ensembles using a multi-armed bandit algorithm called *NeuralUCB* [17], and therefore, we call it Neural Ensemble (NeuE). NeuE judiciously balances the exploration (i.e., learning the performance of DNN ensembles) and exploitation (i.e.,

forming the best-fit DNN ensemble based on current knowledge) and achieves asymptotic optimality.

3) We further design an extension of NeuE to expedite the learning and decision-making process with a large ensemble decision space. The complexity of NeuE depends heavily on the number of ensemble decisions which grows exponentially with the number of candidate DNNs. This makes NeuE very inefficient when the set of candidate DNNs is large. To address this issue, we propose a variant of NeuE, called NeuE-S, by mining the similarity of ensemble decisions. NeuE-S dynamically partitions the decision space into balls and uses only representative decisions from partitioned balls during decision-making. We show via theoretical analysis and experiment that NeuE-S drastically reduces the time complexity of NeuE without harming its asymptotic optimality.
4) We evaluate NeuE and NeuE-S on an edge computing testbed. The experiment is performed on the four real-world datasets Chest X-Ray [18], Caltech101 [19], MASATI [20], WIDER [21]. The experimental results show that NeuE outperforms other benchmark in terms of achieved utility, and NeuE-S can reduce complexity from exponential-time to polynomial-time with slight performance losses.

The rest of this paper is organized as follows. Section II reviews related works. Section III introduces the system model and defines the DEF problem. Section IV designs the NeuE algorithm. Section studies the extension NeuE-S. Section VI shows experimental results, followed by conclusions in Section VII.

II. RELATED WORK

A. ENSEMBLE LEARNING VERSUS ENSEMBLE FORMATION

Ensemble learning is a longstanding machine learning strategy that mainly involves two research topics: model training and output fusion. Model training studies how to train an ensemble model to reach desired performance. There are a variety of model training schemes for ensemble learning, e.g., Bagging [22], Boosting [23], AdaBoost [24], stacked generalization [25]. The training process of these algorithm involves manipulating training data to generate a set for weak base models. Output fusion investigates the process of integrating the base models' outputs into a single output. There are three main approaches for combining the outputs. 1) Algebraic combiners [9], [26]: algebraic combiners are non-trainable combiners, where outputs of base models are combined through an algebraic expression, such as minimum, maximum, weighted average, median, etc. 2) Voting based methods [14]: voting based methods, e.g., majority voting, weighted majority voting, operate on labels only, where the vote to a class is 1 or 0 depending on whether base model chooses the class. They then choose the class that receives the most votes. 3) Meta-learning methods [27]: In meta-learning, the individual outputs are inputs to the meta-learner that generates the final output. The key is training a good meta-learner. Ensemble learning has also been used in the DL community recent years. The authors in [8], [9] create a DNN ensemble by averaging the output of multiple individual DNNs, which far outperforms existing benchmarks in terms of inference accuracy. Other advanced ensemble/fusion rules are also investigated, e.g., authors in [26] utilize weighted averaging fusion rule and designs a learning algorithm to learn the optimal weight of each DNN.

The DEF problem considered in this paper is very different from classic ensemble learning. DEF neither considers model training nor fusion rule design, instead, it focuses how to form optimal DNN ensembles from a set of base DNN models. The DEF problem is related to DNN selection that aims to select one best single DNN. The authors in [28] shows that different DNNs have different accuracy and delay, and a DNN selector is learned to select the best DNN. The work [29] proposes a big/little DNN framework where a little DNN is used whenever possible and a big DNN is only users when the confidence of little DNN is below a threshold. However, forming DNN ensembles is much more complicated because it is difficult to characterize the inter-dependency among multiple DNNs and its impact on inference performance. The most related work is probably our previous work [16], which considers a DEF problem for edge AI services. However, the work [16] only offers a basic solution that rests on heuristic rules and simplified implementation scenarios: 1) The work [16] assumes that a well-performed DNN ensemble should include DNNs that have high individual accuracy and at the same time exhibit large diversity, and based on this assumption, a heuristic DNN formation rule that jointly considers the individual accuracy and diversity of DNN members. But the heuristic rule lacks strict theoretical guarantees, and cannot provide satisfactory performance in all cases. 2) The problem formulation in [16] directly restricts the maximum number of DNNs that can be included in the ensemble. This does not precisely capture the computing resource constraint at edge servers and may cause performance degradation in practice. 3) The performance of the learning algorithm proposed in [16] depends on the dimension of task context space, and becomes less efficient when the task context is large. This paper designs a novel DNN ensemble formation algorithm to address the above problems, it does not rely on any heuristic rules for DNN ensemble formation, and directly learns to find the optimal DNN ensemble. The DEF problem formulated in this paper does not make unrealistic assumptions about edge computing systems and thereby improving the practicality of the proposed method. In particular, the proposed learning algorithm can learn efficiently over a large task context space.

B. CONTEXTUAL MULTI-ARMED BANDIT ALGORITHMS

Contextual bandit algorithms have been applied in many real-world applications. The most studied model is linear contextual bandits [30], [31], which assumes that the expected reward is linear in the context. While successful in theory, the assumption of linear-reward often fails to hold in practice,

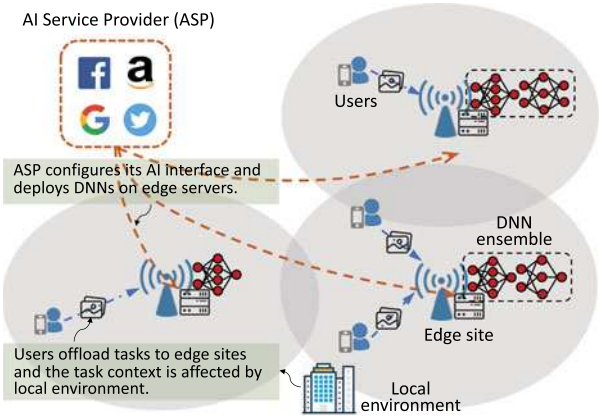


FIGURE 1. AI service provisioning on the edge computing platform.

which motivates the study of nonlinear contextual bandit [32], [33]. However, they still require fairly restrictive assumptions on the reward function. For example, the work [32] requires a Lipschitz continuous property in a proper metric space, and work [33] assumes the reward function belongs to some reproducing kernel Hilbert space. In order to overcome the above shortcomings, deep neural networks (DNNs), have been introduced to learn the underlying reward function in contextual bandit problems. The authors in [17] propose a new algorithm NeuralUCB that uses a neural network to learn the unknown reward function and follows the UCB strategy for exploration. Our method is inspired by the NeuralUCB algorithm to learn complicated mappings from task contexts to the utility of DNN ensembles. However, NeuralUCB can be very inefficient when the number of DNN ensembles is large, and therefore we exploit the similarity of DNN ensemble decisions to improve the scalability of our method.

III. AI SERVICES PROVISIONING ON EDGE COMPUTING PLATFORMS

A. IMPLEMENTATION SCENARIO

We exemplify the implementation scenario of AI service provisioning in the context of multi-access edge computing (MEC) [7], [34], illustrated in Figure 1. A MEC system consists of geographically distributed edge sites where each edge site has a wireless access point and an edge server. The users within the coverage of an edge site can send their tasks to the edge server via the wireless access point. ASP configures its service interfaces and deploys DNNs at edge servers to process the received tasks. Also, an edge server supports the coexistence of multiple ASPs. The key is to use virtualization techniques to create resource isolation, e.g., virtual machines or containers, on the edge server, and ASPs can deploy their service independently with allocated resources. The resource scheduling for multiple ASPs [35]–[37] is orthogonal to the theme of this paper.

B. DNN ENSEMBLE AND COMPLEXITY CONSTRAINTS

An ASP stores a set of candidate DNNs, indexed by $\mathcal{M} = \{1, 2, \ldots, M\}$, on the edge server. Storing DNNs only consumes storage resource. Because the storage is cheap, the total number of candidate M can be possibly large. The ASP picks one or multiple DNNs from the candidate set $\mathcal{M}$ to form a DNN ensemble, denoted by $Y \subseteq \mathcal{M}$. The size of an ensemble can vary from 1 to $M = |\mathcal{M}|$. Let $\mathcal{Y}$ be the ensemble decision space that collects all possible ensemble decisions, then the size of $\mathcal{Y}$ is a Bell number of M:

$$|\mathcal{Y}| = \sum_{i=1}^{M} \binom{M}{i}. \tag{1}$$

From Equation (1), we see that the number of ensemble decisions grows exponentially with M, which needs to be carefully handled to avoid poor scalability. We will give detailed discussions about this when presenting our method.

The space and time complexity of DNN ensembles depends on the ensemble size and the scheme for running ensembles. For example, the edge server can load all DNNs in the ensemble to RAM and run them in parallel. However, doing that will require a considerable amount of computing resources as the computation of all DNNs happens at the same time. Alternatively, the edge server can run DNNs sequentially (e.g., one by one), which requires much less computing resource. But sequential execution tends to incur larger computation delay. Besides these two basic schemes, many others can also be applied to run DNN ensembles. Fortunately, our method does not require a specific scheme for running DNN ensembles. Given a certain implementation scheme, we let $c(Y)$ and $d(Y)$ be the computing resource usage and delay for running ensemble Y, respectively. Intuitively, $c(\cdot)$ and $d(\cdot)$ are non-decreasing functions of the ensemble size $|Y|$. Note that the computing resources allocated to an ASP can be limited. Let $\bar{c}$ denote the computing capacity available to ASP at the edge server, for a feasible ensemble Y, its resource usage should not exceed the computing capacity $\bar{c}$, i.e., $c(Y) \leq \bar{c}$. In addition, if the user tasks are associated with the deadline requirement $\bar{d}$, then the runtime of a feasible ensemble should not violate the deadline requirement, i.e., $d(Y) \leq \bar{d}$.

C. DNN ENSEMBLE FORMATION PROBLEM

Next, we formally define the DEF problem. Due to the volatile mobile environment, the context of user tasks varies across time. Our method proposes to reconfigure the ensemble adaptively according to the changes in task context. The operational timeline is discretized into time slots $t = 1, 2, \ldots, T$ (e.g., a few seconds per slot). In each time slot t, we let $\mathbf{x}^t = \{x_1^t, x_2^t, \ldots, x_N^t\}$ be the set of user tasks received at the edge server.

Suppose DNN ensemble Y^t is formed in time slot t, all received tasks in $\mathbf{x}^t$ will be forwarded to each DNN in Y^t. Consider an arbitrary task $x_n^t \in \mathbf{x}^t$ and an arbitrary DNN $m \in Y^t$, we denote the prediction result of DNN m for task x_n^t by y_{nm}^t. Because all DNNs in Y^t are used to process task x_n^t, a fusion rule π will be used to combine individual results to a final decision $\hat{y}_n^t(Y^t)$, i.e., $\hat{y}_n^t(Y^t) \leftarrow \pi(\{y_{nm}^t\}_{m \in Y^t})$. The fusion rule

is usually different for different AI services due to different output formats. Even for the same AI application, there are various fusion rules, for example in classification problem, commonly-used fusion rules include majority voting [38], confidence averaging [39], meta-learning [40], etc. It is worth highlighting that the method proposed in this paper is not confined to a specific AI application and is compatible with most fusion rules.

ASP gains utility by completing user tasks. For task $x_n^t \in \mathbf{x}^t$, we let $c_n^t = \mathbf{1}\{\hat{y}_n^t(Y^t) = y_n^t\}$ denote the prediction correctness of ensemble Y^t where $\mathbf{1}\{\cdot\}$ is the indicator function and y_n^t is the ground truth of task x_n^t. Given the set of received tasks $\mathbf{x}^t$, the reward in time slot t is

$$u^t(\mathbf{x}^t, Y^t) = \sum\nolimits_{x_n^t \in \mathbf{x}^t} \mathbf{1}\{\hat{y}_n^t(Y^t) = y_n^t\} \tag{2}$$

ASP aims to maximize the cumulative reward in a total of T time slots by finding a sequence of DNN ensemble decisions $\{Y^t\}_{t=1}^T$. The objective of the DEF problem is defined as:

$$\mathscr{P}1: \max_{\{Y^t\}_{t=1}^T} \sum\nolimits_{t=1}^T u^t(\mathbf{x}^t, Y^t) \tag{3a}$$

$$\text{s.t.} \quad c(Y^t) \le \bar{c}, \forall t \tag{3b}$$

$$d(Y^t) \le \bar{d}, \forall t \tag{3c}$$

$$Y^t \in \mathcal{Y}, \forall t \tag{3d}$$

Recall that (3b) and (3c) are constraints posed by the computing capacity and response deadline. Although $\mathscr{P}1$ is given in an off-line form, it can only be solved in an online manner because the task set $\mathbf{x}^t$ is not revealed before time slot t. The online decision-making would be simple if we know utility mapping $u(\mathbf{x}^t, Y^t)$ — the optimal ensemble in each time slot t can be easily identified using

$$Y^{*t} = \max_{Y \in \mathcal{Y}} u(\mathbf{x}^t, Y). \tag{4}$$

However, such a utility function is often unknown in practice. The performance of DNN ensembles for user tasks is revealed only during implementation. Moreover, evaluating the general performance of DNN ensembles is not enough because we also need to analyze the impact of task context on the performance of DNN ensembles. This further increases the difficulty of obtaining the offline utility function. As a result, $\mathscr{P}1$ cannot be solved merely as an online optimization problem. Learning the performance of DNN ensembles is a necessary component to be incorporated in our method. In the next section, we will cast $\mathscr{P}1$ into a multi-armed bandit problem and provide an online learning algorithm to solve the DEF problem.

IV. DNN ENSEMBLE FORMATION VIA CONTEXTUAL MULTI-ARMED BANDIT

We utilize contextual multi-armed bandit to provide a solution to the DEF problem. Before presenting the designed method, we first need to define the context-parameterized utility.

A. CONTEXT-PARAMETERIZED UTILITY

We consider simple task contexts that can be obtained without processing the task, e.g., in the experiment, we take the patient data associated with medical images as the task context. In this case, using context will not incur extra computation burdens. Upon the arrival of user tasks, ASP first observes the associated context. Let $\omega_{\mathbf{x}^t} \in \Omega_X$ (Ω_X is the context space) denote the context of received tasks $\mathbf{x}^t$, we slightly abuse the notation of utility function $u^t(\mathbf{x}^t, Y)$ by defining the context-parameterized utility $u^t \sim u(\omega_{\mathbf{x}^t}, Y)$, i.e., the utility of using DNN ensemble Y for received tasks $\mathbf{x}^t$ is sampled from a unknown distribution $u^t(\omega_{\mathbf{x}^t}, Y)$ parameterized by the context of tasks $\omega_{\mathbf{x}^t}$. We further define $\mu(\omega_{\mathbf{x}^t}, Y) = \mathbb{E}[u(\omega_{\mathbf{x}^t}, Y)]$ as the expected utility given task context $\omega_{\mathbf{x}^t}$ and DNN ensemble Y. If the utility function is known a priori, then the optimal DNN ensemble in time slot t can be found by

$$Y^{*t} = \operatorname*{argmax}_{Y \in \mathcal{Y}} u^t(\omega_{\mathbf{x}^t}, Y). \tag{5}$$

B. DNN ENSEMBLE FORMATION WITH NEURALUCB

The core of our method is to learn the context-parameterized utility function online and use the learned knowledge to guide the DNN ensemble formation for utility maximization. A *exploration-and-exploitation* dilemma is worth highlighting to achieve this goal. Note that the performance of a DNN ensemble is revealed only after it is used for processing user tasks. The learner needs to observe an adequate amount of utility (u^t) and context ($\omega_{\mathbf{x}^t}$) data for each ensemble Y to learn an accurate utility function $u^t(\omega_{\mathbf{x}^t}, Y)$. An accurate utility estimation is a precondition for identifying the best-fit DNN ensemble using the rule in (5). Selecting DNN ensembles based on inaccurate utility predictions can lead to arbitrarily low utility. Therefore, the learner needs to judiciously balance 1) *Exploration*, i.e., selecting a DNN ensemble to collect its utility for better estimation of utility function; and 2) *Exploitation*, i.e., selecting the optimal DNN ensemble based on the learned utility function. Such a problem fulls into multi-armed bandit (MAB) learning and we use contextual MAB to provide a solution.

The existing contextual MAB methods can be categorized into two groups. The first category is to construct a mapping function with a specific form and then estimate parameters in the constructed function. A noticeable deficiency of this method is that it requires knowing the form of the utility function, which does not hold in all cases. Besides, the constructed utility function may not correctly reflect the properties of the true utility function. For example, a widely-used contextual MAB algorithm, LinUCB [41], assumes that the utility function is a linear mapping of context, which is often false in practice. The second category is to partition the context space into multiple sub-spaces and learns the expected utility for each of these sub-spaces [42]. For this type of method, the curse of dimensionality is a serious problem. When the dimension of context space becomes high, the algorithm is extremely

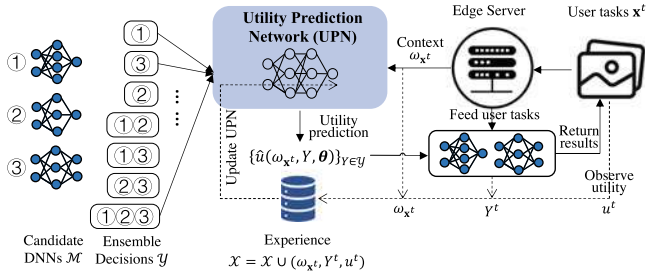


FIGURE 2. Illustration of NeuE. It depicts one decision cycle in a time slot, starting from the arrival of user tasks at the right top corner.

inefficient. In our problem, the context-parameterized utility function is less likely to be linear and the task context space can also be high. To address these issues, we employ a novel contextual MAB algorithm, NeuralUCB [17], as our learning engine.

Next, we show our online learning algorithm for the DEF problem. We call our algorithm NeuE (Neural Ensemble) because it leverages "Neural"UCB, to construct ensembles of "neural" networks. Following NeuralUCB, NeuE builds a neural network to approximate the utility function $u(\omega_{\mathbf{x}^t}, Y)$. We name this neural network *utility predicting network* (UPN) to distinguish it from the deep neural networks used for processing AI tasks. UPN has the capability of representing general non-linear dependencies between the context information and utility without a priori specifying which particular form of dependencies to look for. The input to UPN is the context of received tasks $\omega_{\mathbf{x}^t}$ and the ensemble decision Y; and the output is the predicted utility, denoted by $\hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the parameter vector of UPN.

The pseudocode of NeuE is given in Algorithm 1. Next, we discuss in detail the procedures of NeuE as depicted in Figure 2. Upon the arrival of user tasks $\mathbf{x}^t$ in time slot t, NeuE first observes the context of received tasks $\omega_{\mathbf{x}^t}$, which will then be used to estimates the utility of DNN ensemble decisions. For each DNN ensemble decision $Y \in \mathcal{Y}$ that satisfies $c(Y) \leq \bar{c}$ and $d(Y) \leq \bar{d}$, NeuE constructs an input pair $\{\omega_{\mathbf{x}^t}, Y\}$ and feeds it to UPN to get the utility prediction $\hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$. After that, NeuE calculates *Lambda Indicator* λ_Y^t of ensemble decisions:

$$\lambda_Y^t = \begin{cases} 0, & c(Y) > \bar{c} \text{ or } d(Y) > \bar{d} \\ \hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta}) + \frac{\gamma}{\sqrt{h}} \sqrt{(\boldsymbol{g}_Y^t)^\top \boldsymbol{Z}^{-1} \boldsymbol{g}_Y^t}, & \text{otherwise} \end{cases} \tag{6}$$

where $\boldsymbol{g}_Y \leftarrow \nabla_{\boldsymbol{\theta}} \hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$ is the gradient of UPN parameters of $\boldsymbol{\theta}$ at $\{\omega_{\mathbf{x}^t}, Y\}$, $\boldsymbol{Z}$ is an algorithm parameter that is iteratively updated based on $\boldsymbol{g}_Y$ (Lane 15 in Algorithm 1), and h is a constant determined by the network architecture of UPN. Given the Lambda indicator, the best-fit ensemble for each time slot t is determined by

$$Y^t = \underset{Y\in\mathcal{Y}}{\operatorname{argmax}} \ \lambda_Y^t. \tag{7}$$

Algorithm 1. NeuE

1: **Input**: time horizon T, ensemble decision set $\mathcal{Y}$, algorithm parameter γ, the number of nodes in the hidden layers of UPN h.
2: **Initialization**: Randomly initialize the UPN parameter $\boldsymbol{\theta}$, initialize $\boldsymbol{Z} \leftarrow \boldsymbol{I}$
3: **for** $t = 1, \ldots, T$ **do**
4: Observe the context of received tasks $\omega_{\mathbf{x}^t}$
5: **for** each ensemble $Y \in \mathcal{Y}$ **do**
6: **if** $c(Y) \leq \bar{c}$ and $d(Y) \leq \bar{d}$ **then**
7: Predict utility of ensemble Y for tasks $\mathbf{x}^t$ with current UPN $\hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$
8: Computing the gradient of UPN parameter $\boldsymbol{\theta}$ at $\{\mathbf{x}^t, Y\}$: $\boldsymbol{g}_Y^t \leftarrow \nabla_{\boldsymbol{\theta}} \hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$
9: Assign $\lambda_Y^t \leftarrow \hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta}) + \frac{\gamma}{\sqrt{h}} \sqrt{(\boldsymbol{g}_Y^t)^\top \boldsymbol{Z}^{-1} \boldsymbol{g}_Y^t}$
10: **else**
11: Assign $\lambda_Y^t \leftarrow 0$
12: **end if**
13: **end for**
14: Select ensemble $Y^t = \operatorname{argmax}_{Y\in\mathcal{Y}} \lambda_Y^t$
15: Update $\boldsymbol{Z} \leftarrow \boldsymbol{Z} + \boldsymbol{g}_{Y^t}^t (\boldsymbol{g}_{Y^t}^t)^\top / h$
16: Apply DNN ensemble Y^t to process received tasks and observe the achieved utility u^t
17: Store experience in $\mathcal{X}$, $\mathcal{X} \leftarrow \mathcal{X} \cup \{\omega_{\mathbf{x}^t}, Y^t, u^t\}$
18: Update UPN parameter $\boldsymbol{\theta}$ using $\mathcal{X}$:
$\boldsymbol{\theta} \leftarrow \texttt{TrainUPN}(\mathcal{X})$ ▷Use Algorithm 2
19: **end for**

Algorithm 2. Subroutine: `TrainUPN`

1: **Input**: learning rate η, experience $\mathcal{X}$, number of gradient descent updates J;
2: Define $L(\boldsymbol{\theta}) = \sum_{\{\omega_{\mathbf{x}^\tau}, Y^\tau, u^\tau\}\in\mathcal{X}} (\hat{u}(\omega_{\mathbf{x}^\tau}, Y^\tau; \boldsymbol{\theta}) - u^\tau)^2$
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 Lambda indicator is designed to balance the exploration and exploitation trade-off. To be specific, the first term $\hat{u}(\omega_{\mathbf{x}^t}, Y; \boldsymbol{\theta})$ in the Lambda indicator (6) is the utility predicted by UPN and the second term is used to characterize the uncertainty of the utility prediction. If the utility prediction for ensemble Y exhibits a large uncertainty, then the rule in (7) has a tendency to select ensemble Y. In this case, the learner is able to collect utility at the end of the time slot and use it to retrain UPN, such that UPN can produce more accurate results if similar contexts appear in the future. If the uncertainty of utility prediction is small, then λ_Y^t is dominated by the predicted utility, and the ensemble selected by (7) is expected to deliver the highest utility. The parameter γ is used to adjust the importance of exploration and exploitation.

After ASP determines the ensemble decision Y^t, the edge server runs all DNNs in Y^t to process the received tasks $\mathbf{x}^t$ and gives final prediction results $\hat{\mathbf{y}}^t = \{\hat{y}_1^t, \hat{y}_2^t, \ldots, \hat{y}_N^t\}$ which are then returned to users. At last, ASP observes the utility u^t

achieved by Y^t. The user utility can be determined by many factors, e.g., the accuracy of returned results, service delay, and energy consumption. NeuE can handle any form of utility function as long as the utility value can be observed at the end of each time slot. The task context $\omega_{\mathbf{x}^t}$, the formed ensemble Y^t, and the observed utility u^t will be stored in an experience database $\mathcal{X}$, $\mathcal{X} \leftarrow \mathcal{X} \cup \{\omega_{\mathbf{x}^t}, Y^t, u^t\}$. The data collected in experience $\mathcal{X}$ will be used to train UPN. The training of UPN depends on the training frequency $f \leq 1$, which means that the UPN is trained every $1/f$ time slots. Given the time horizon T, the UPN will be trained $f \cdot T$ times. The training process of UPN is presented in Algorithm 2. It uses the standard gradient descent method.

C. PERFORMANCE ANALYSIS OF NEUE

Next, we provide performance guarantees of NeuE. The performance of NeuE is measured by the utility gap, termed as *regret*, between NeuE and an oracle algorithm that always selects the optimal DNN ensemble decision in each time slot. Regret is formally defined as

$$R(T) = \sum_{t=1}^{T} u^t(\omega_{\mathbf{x}^t}, Y^{*t}) - u^t(\omega_{\mathbf{x}^t}, Y^t). \tag{8}$$

where Y^{*t} and Y^t are the DNN ensemble decisions selected by oracle and NeuE, respectively. The lemma below provides a performance guarantee for NeuE in terms of regret.

Lemma 1. (Regret of NeuE). The upper regret bound of NeuE is $\mathcal{O}(R(T)) = \mathcal{O}(\sqrt{T})$, where T is the total number of time slots that NeuE runs.

The proof for Lemma 1 can be found in [17]. It gives a sublinear regret $\mathcal{O}(\sqrt{T})$, meaning that NeuE is *asymptotically optimal* compared to the oracle algorithm.

*Scalability Issue of NeuE.*The complexity of NeuE mainly lies in running UPN for utility prediction. Note that for each ensemble decision, NeuE runs UPN one time to get the predicted utility. Therefore, the complexity of NeuE is decided by the total number of DNN ensemble decisions $|\mathcal{Y}|$. However, as shown in (1), $|\mathcal{Y}|$ grows exponentially over the number of candidate DNNs M, resulting in extremely-high computational complexity when the number of candidate DNNs is large. Although the computing resource constraints (3b) and service deadline (3c) helps reduce the decision space of DNN ensemble decisions, it cannot settle the scalability issue completely as the set of candidate DNNs can still be large due to the cheap storage. In the next section, we propose a variant of NeuE to improve its scalability by leveraging the similarity of ensemble decisions.

V. EXPEDITING NEUE USING THE SIMILARITY OF DNN ENSEMBLES

Our method is motivated by the intuition – if the constituting DNNs of two ensembles are similar, then these two ensembles would be likely to deliver similar utility. To define the similarity of DNN ensembles, we introduce the context of ensemble decisions. Let $\omega_Y \in \Omega_Y$ denote the context of ensembles with Ω_Y being the context space of the ensemble decision. The similarity of two ensemble decisions $Y, Y' \in \mathcal{Y}$ is measured by a distance metric $L_Y(\omega_Y, \omega_{Y'})$. For example, the simplest context could be a M-dimension binary vector where m-th entry indicates whether DNN m is included in the ensemble or not, and euclidean distance $\|\omega_Y - \omega_{Y'}\|$ could be used to measure the similarity. We note that other information, e.g., the properties of DNNs' training data, can also be included in the context of ensemble decisions. Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TETC.2022.3214931, provides concretes of ensemble decisions. With the above definitions, our intuition is captured by the Lipschitz condition.

Definition 1 (Lipschitz Condition). Given the context space Ω_X of tasks and the context space Ω_Y of ensemble decision, the context-parameterized utility function $u(\omega_{\mathbf{x}}, Y)$ is Lipschitz with the distance metric L_Y, if the below inequality holds $\forall \omega_{\mathbf{x}} \in \Omega_X \, \forall Y, Y' \in \mathcal{Y}$, and $\forall \omega_Y, \omega_{Y'} \in \Omega_Y$

$$|u(\omega_{\mathbf{x}}, Y) - u(\omega_{\mathbf{x}}, Y')| \leq L_Y(\omega_Y, \omega_{Y'}) \tag{9}$$

We leverage the similarity property stated in the Lipschitz condition to speed up NeuE. The proposed variant is called NeuE-S (NeuE + Similarity), it reduces the computational complexity by using only representative DNN ensemble decisions instead of the entire decision set. To define the representative decisions, we first introduce the covering number and covering dimension of context space Ω_Y.

Definition 2 (Covering Number and Covering Dimension). Let (Ω_Y, L_Y) be a metric space. Covering number $C(\Omega_Y, L_Y, r)$ is the smallest number of sets needed to cover Ω_Y and in each set of the covering, any two points have a distance less than r. The covering dimension of (Ω_Y, L_Y), denoted by $D(\Omega_Y, L_Y)$, is defined based on the covering number:

$$\inf \{d : \exists c \geq 0, \forall r \in (0, 1], C(\Omega_Y, L_Y, r) \leq c r^{-d}\}.$$

We also define D' and c such that $D' > D$ and $C(\Omega_Y, L_Y, r) \leq c r^{-D'}$. The existence of such constant c is guaranteed by the definition of covering dimension.

Now, we are ready to present NeuE-S. NeuE-S splits the timeline into phases $i = 0, 1, 2, \ldots$, with the i-th phase having 2^i time slots. Suppose NeuE-S enters phase i, it first partitions the ensemble decision space $\mathcal{Y}$ into disjoint sets $\tilde{\mathcal{Y}}_1, \tilde{\mathcal{Y}}_2, \ldots, \tilde{\mathcal{Y}}_{K_i}$, where the number of disjoint sets in phase i is K_i

$$K_i = c \cdot 2^{\frac{D'i}{D'+2}}, \tag{10}$$

and the diameter of each partitioned set should not be larger than r_i

$$r_i = 2^{-\frac{i}{D'+2}}. \tag{11}$$

The existence of such partitions $\{\tilde{\mathcal{Y}}_k\}_{k=1}^{K_i}$ follows from that the covering dimension of Ω_Y is D. NeuE-S then finds a subset $\mathcal{Y}' \subseteq \mathcal{Y}$ based on partitioned $\{\tilde{\mathcal{Y}}_k\}_{k=1}^{K_i}$. Specifically, NeuE-S will pick at least one DNN ensemble decision from $\tilde{\mathcal{Y}}_k, \forall k$ and put them in $\mathcal{Y}'$, and hence we will have $|\mathcal{Y}'| \geq K_i$.

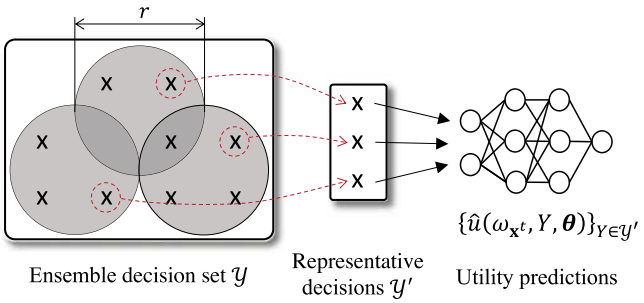


FIGURE 3. Illustration of the set cover problem and NeuE-S.

It is guaranteed that an arbitrary ensemble decision $Y \in \mathcal{Y}$ is within distance r_i to a point in $\mathcal{Y}'$. We call $\mathcal{Y}'$ the representative decision set of $\mathcal{Y}$.

Algorithm 3. NeuE-S

1: **Input**: ensemble decision set $\mathcal{Y}$, distance metric L_Y, constant D';
2: **for** phase $i = 1, 2, \ldots$ **do**
3: Calculate $K_i = c \cdot 2^{\frac{D'i}{D'+2}}$ and $r_i = 2^{-\frac{i}{D'+2}}$
4: Create partition on $\mathcal{Y}$ by solving the set cover problem: $\{\tilde{\mathcal{Y}}_k\}_{k=1}^{K_i} \leftarrow \texttt{SetCover}(\mathcal{Y}, r_i)$.
5: Pick at least on decision from $\tilde{\mathcal{Y}}_k, \forall k$ to generate the representative decision set $\mathcal{Y}'$.
6: Run NeuE (Algorithm 1) with input $\mathcal{Y}'$ and the time horizon $T = 2^i$.
7: **end for**

In each phase, NeuE-S runs NeuE with representative ensemble decisions. NeuE-S only needs to run UPN prediction for $|\mathcal{Y}'|$ ensemble decisions to identify the optimal ensemble decision. Figure 3 illustrates the core philosophy of NeuE-S.

We want $|\mathcal{Y}'|$ to be as small as possible in each phase such that the computing complexity of NeuE-S is minimized. This equals a *set cover* problem that aims to partition the ensemble decision space with the smallest number (i.e., K_i) of metric balls. Solving the set cover problem is NP-hard. Fortunately, our method does not require an exact solution, an approximate solution would be efficient enough to reduce the computation complexity of NeuE. In the experiment, we use a greedy algorithm [43] to solve the set cover problem. The greedy algorithm takes $\mathcal{Y}$ and r_i as inputs and outputs disjoint sets $\{\tilde{\mathcal{Y}}_k\}_{k=1}^{K_i}$. It only requires polynomial time and provides a rigorous performance guarantee (see reference [43] for more details). Algorithm 3 gives the pseudocode of NeuE-S.

We can also rigorously prove the performance of NeuE-S in terms of regret upper bound.

Theorem 1 (Regret upper bound of NeuE-S). For any $T \geq 0$, *the regret of NeuE-S is upper bounded by* $\mathcal{O}(T^{\frac{D'+1}{D'+2}})$.

Proof. See in Appendix B, available in the online supplemental material. □

The above theorem indicates that NeuE-S loosen the regret upper bound of NeuE from $\mathcal{O}(\sqrt{T})$ to $\mathcal{O}(T^{\frac{D'+1}{D'+2}})$. Although the upper bound of NeuE-S becomes less sharp, it is still sublinear and hence the asymptotic optimality is guaranteed. Note that the regret upper bound in Theorem 1 is given for the worst case, we can show via experiment (see details in Section VI) that the actual regret of NeuE-S is much smaller and using representative decisions does not degrade the performance much.

NeuE-S is an efficiently variant of NeuE that can handle the large ensemble decision space of the DEF problem with slight performance degradations. AI service providers (ASP) can pick an appropriate scheme based on their needs in the real-world implements. For example, if the number of candidate DNNs is moderate, the ASP can use NeuE, and if the number of candidate DNNs is large, then the ASP may choose NeuE-S.

VI. EXPERIMENTS

A. EXPERIMENT SETUP

We run the proposed method on an edge computing testbed. A DELL workstation is used as an edge server, which is equipped with a 64-bit 8 Intel i7-4770 CPU cores running at 3.40 GHz, and one NVIDIA Geforce GTX 1080 Ti GPU. The experiment is run on Ubuntu 16.04 LTS system with Pytorch v1.1.0, CUDA v9.0, cuDNN v7.0. AI tasks are fed into the edge server in a batch-wise manner. The size of the task batch in each time slot is 10.

We evaluate our method on four real-world datasets: Chest X-Ray [18] (CXR), Caltech101 [19], MASATI [20], WIDER [21]. These four datasets represent two types experiment settings in terms of how task contexts are generated. CXR dataset has natural context associated with the inference tasks, while artificial contexts are generated for Caltech101, MASATI, and WIDER. We use 8 types of DNNs including DenseNet, GoogleNet, ShuffleNet, ResNet, EfficientNet, MobileNet, Inception, RegNet. The pre-trained model (pre-trained on the ImageNet dataset [44]) of above 8 DNN types can be found on the PyTorch website [45]. Next, we show how to generated candidate DNNs on these datasets.

Chest X-Ray (CXR) Dataset. The CXR dataset is comprised of 112,120 X-ray images with disease labels from 30,805 unique patients. There are 15 classes (14 diseases, and one for "No findings"), and images can be labeled as "No findings" or one or more disease classes. *A Multi-label classification* is performed on the CXR dataset. The output of DNN ensembles is a vector of binary values $\hat{\mathbf{y}}$, where $\hat{y}_l \in \{1 : \text{'positive'}, 0 : \text{'negative'}\}$ is predicted label for l-th disease. The utility of a task is measured by the Hamming loss, calculated by $h_{\texttt{loss}} = \sum_{l=1}^{L} w_l \cdot |y_l - \hat{y}_l|$ where $\mathbf{y} = \{y_l\}_{l=1}^{L}$ is the ground-truth label of the task and $\mathbf{w} = \{w_l\}_{l=1}^{L}$ is the importance weight for disease classes. The importance weight can change across tasks. The utility is higher when the Hamming loss is smaller. The CXR dataset is split into 3 parts: training dataset (78,468 images, used for generating candidate DNNs), validation dataset (11,219 images, used for evaluating candidate DNNs), and testing dataset (22,433 images, used for evaluating the proposed methods). Each X-ray image in the CXR dataset has context associated with it, which includes

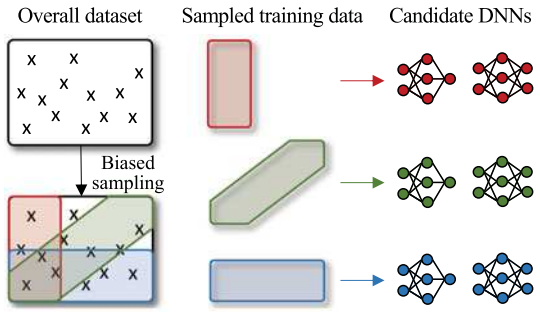


FIGURE 4. Generation of candidate DNNs. In this example, two types of DNN architecture are used for each training dataset.

patient data (e.g., patient age, patient gender, etc.) and image features (e.g., pixel spacing and X-ray orientation). This information can be used as task contexts to run NeuE. To imitate the scenario that DNNs may come from different data sources, we split the CXR dataset into different training datasets in a non-i.i.d. fashion. Specifically, we first split the entire training data into two subsets. One subset has more images in diseases *Effusion, Infiltration, Pneumonia, Consolidation, Edema*, the other subset has more images in disease *Atelectasis, Cardiomegaly, Mass, Nodule, Pneumothorax, Emphysema, Fibrosis, Pleural-thickening, Hernia*. Each subset is further divided into two parts based on the X-ray orientation (`Anterior-Posterior` or `Posterior-Anterior`). We have 4 different training datasets and use 8 types of DNN. For each combination of training dataset and DNN type, we will train a DNN model. In this way, we generate 32 DNNs in total. Because running NeuE with 32 candidate DNNs (i.e., 4.295×10^9 DNN ensemble decisions) is computationally expensive, we only use 3 best DNNs for each training dataset, which results in 12 candidate DNNs (i.e., 4095 DNN ensemble decisions). The task context used by NeuE includes the patient's age, X-ray orientation, and disease importance weight. Figure 4 illustrates how the candidate DNNs are generated.

Caltech/MASATI/WIDER Dataset. These three datasets are widely-used and therefore we save the introduction of these datasets. *Image classification problems* are performed on the Caltech/MASATI/WIDER dataset. The utility of image classification problem is measured by accuracy, i.e., $\frac{1}{N}\sum_{n=1}^{N}\mathbf{1}\{\hat{y}_n = y_n\}$. These three datasets do not have natural context associated with images. Therefore, we add artificial context to the original dataset. Specifically, we add different levels of noise to images as the task context. The original data source is denoted by ORI. Two noisy data sources are created by adding white Gaussian noise with variance 0.1 (data source NOS-1) and 0.2 (data source NOS-2), respectively. The training data contains 60% of the images, and the rest 40% are used as test data to run the proposed algorithm. Similarly, for each combination of DNN architecture and data source, we will train a DNN model. Therefore, We have 24 DNNs for each dataset. Because running NeuE with 24 candidate DNNs is computationally expensive, we use 4 best DNNs for each data source, which also results in 12 candidate DNNs for each dataset. NeuE uses the noise of images as the task context.

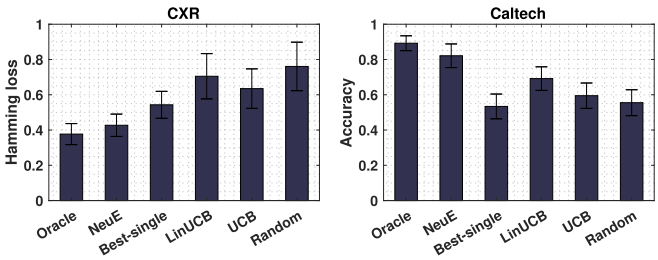


FIGURE 5. Utility of NeuE and benchmarks.

B. RESULTS AND EVALUATIONS

We compare NeuE with 5 benchmarks:

1) Oracle: Oracle knows the context-parameterized utility function of DNN ensembles. It selects a DNN ensemble that is optimal for the received tasks in each time slot. Note that Oracle cannot be applied in practice.
2) UCB1: UCB1 is a classic MAB algorithm. This scheme neglects the context associated with tasks and learns the general performance of DNN ensembles.
3) LinUCB: LinUCB [30] is a widely-used contextual multi-armed bandit algorithm. It assumes that the performance of DNN ensembles is a linear function of the task context.
4) Best-single: This scheme selects one DNN that has the highest predicted accuracy for received tasks in each time slot. The learning and decision-making process of Best-single is the same as NeuE, but it only selects one DNN. This scheme is used to evaluate the benefit of DNN ensemble technique.
5) Random: This scheme also uses DNN ensemble technique, however, it randomly picks candidate DNNs into the ensemble.

Because the proposed methods and benchmarks show similar performances on Caltech, MASATI, and WIDER, we only present the experimental results for CXR and Caltech in this section, other results can be found in Appendix C, available in the online supplemental material.

B.1 UTILITY COMPARISON

Figure 5 compares the utility achieved by NeuE and the other five benchmarks. For the CXR dataset, the utility is given in terms of Hamming loss, and a lower Hamming loss indicates a higher utility. For the Caltech dataset, the utility is given in terms of accuracy, and a higher accuracy indicates a higher utility. As expected, Oracle achieves the highest utility on two datasets. Among the others, we see that NeuE outperforms other benchmarks, and achieves close-to-oracle performance. In particular, NeuE increases the accuracy by 28.73% compared to Best-single on the Caltech dataset. We can even see that Random achieves higher accuracy than Best-single on the Caltech dataset. This indicates that using the DNN ensemble technique effectively improves the performance of AI services.

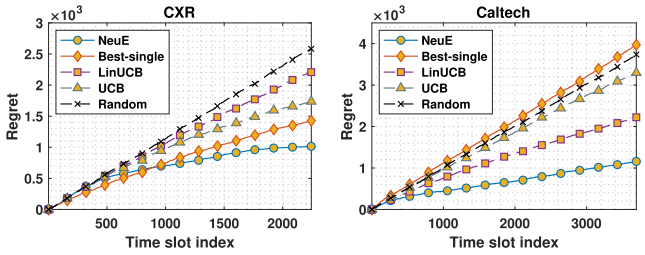


FIGURE 6. **Regret of NeuE and benchmarks.**

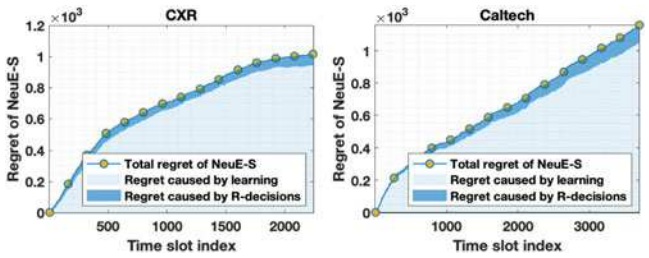


FIGURE 8. **Regret analysis of NeuE-S.**

B.2 REGRET ANALYSIS

Figure 6 depicts the regret of NeuE and benchmarks. The results show that NeuE is able to reach a sublinear regret, meaning that our algorithm can achieve asymptotic optimality. By contrast, the regret of UCB and LinUCB increases linearly over time. This indicates that UCB and LinUCB cannot identify the optimal ensemble decision effectively.

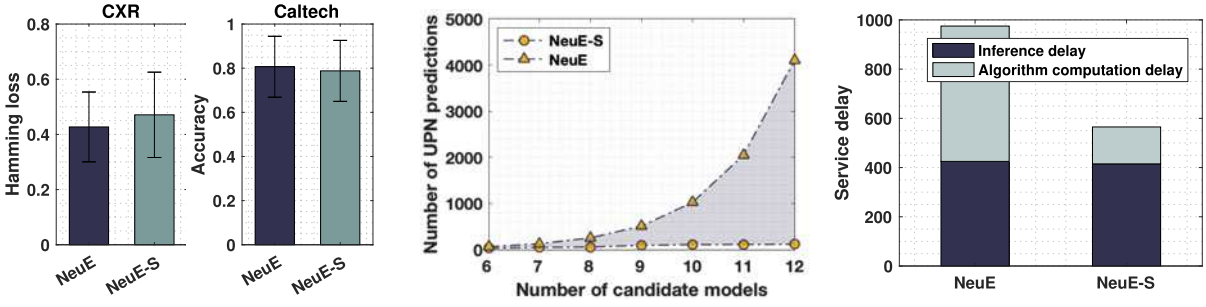
B.3 PERFORMANCE OF NEUE-S

Figure 7 compares the performance of NeuE and Neue-S in terms of utility and algorithm complexity. Figure 7(a) gives the utility achieved by two algorithms, we can see that the performance of NeuE-S is slightly lower than that of NeuE. This means that using representative decisions will not harm the performance of NeuE much. Figure 7(b) compares the computing complexity of NeuE and NeuE-S. The computing complexity is measured by the number of UPN predictions required to complete DNN ensemble formation in each time slot (the computing capacity constraint and service deadline are not considered in this figure). We can see clearly that the computing complexity of NeuE increases exponentially with the number of candidate DNNs. For example, the current experiments run with 12 candidate DNNs, which means that NeuE will need to run UPN prediction 4095 times in one time slot. By contrast, the computing complexity NeuE-S stays low even when the number of candidate DNNs is 12. This indicates that NeuE-S can significantly speed up the decision-making process with negligible utility losses. Figure 7(c) compares the service delay of NeuE and NeuE-S. The service delay consists of two parts, the algorithm computation delay and the inference delay. The algorithm computation delay is the time used by NeuE or NeuE-S to get the optimal DNN ensemble, and the inference delay is the time for running the DNN ensemble to get the prediction results of the admitted tasks. Figure 7(c) shows the average service delays (per task) of NeuE and NeuE-S on the CXR dataset with 200 MB computation capacity. We can see that the algorithm computation delay accounts for more than 50% of the total service delay for NeuE. By contrast, the algorithm computation delay for NeuE-S is much lower.

Figure 8 depicts the regret of NeuE-S on CXR and Caltech datasets. Overall, we can see that the regret curve of NeuE-S is sublinear on both datasets. We also provide a regret decomposition in Figure 8, splitting the total regret into two parts, the regret caused by using representative decisions (R-decisions) and the regret caused by online learning. We can see clearly that using representative decisions only accounts for a very small portion of the total regret.

B.4 IMPACT OF COMPUTING CAPACITY

Figure 9 shows the impact of computing capacity on NeuE-S. We allocate different amounts of GPU RAM to run NeuE-S, Figure 9 shows the performance of NeuE-S, Oracle, and Random when the amount of allocated GPU RAM changes from 100 MB to 400 MB. In general, we see that all three methods achieve higher utilities (lower Hamming loss for the CXR dataset, and higher accuracy for the Caltech dataset) with more computing resources. This is because the ASP can recruit more DNNs in the ensemble which tends to give more stable prediction results. It is also worth noticing that increasing the computing capacity has a diminishing effect, i.e., when the computing capacity is already large, further



(a) Utility. (b) Complexity. (c) Service delay.

FIGURE 7. **Comparison of utility, complexity, and service delay between NeuE and NeuE-S.**

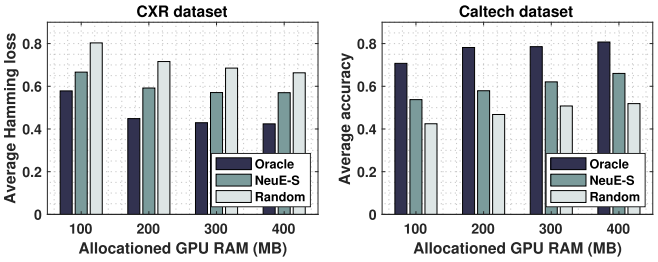


FIGURE 9. Impact of computing capacity.

increasing the computing capacity will not provide large utility increases.

B.5 IMPACT OF SERVICE DEADLINE

Figure 10 shows the impact of service deadline on NeuE-S. We vary the service deadline from 400 ms to 1200 ms (the computing capacity is fixed at 400 MB), Figure 10 shows the performance of NeuE-S, Oracle, and Random with different service deadline constraints. Similar to the impact of computing capacity, we see that all three methods achieve higher utilities with looser service deadline, because the ASP can recruit more DNNs in the ensemble which tends to give more stable prediction results. In addition, loosening the service deadline also exhibits a diminishing effect.

B.6 IMPACT OF TASK CONTEXT ON ENSEMBLE FORMATION

We next take the Caltech dataset as an example the show how task context affects DNN ensemble formation. Figure 11 shows the probability that a DNN is included in the formed ensemble under different image noise levels. First, we can see in general that certain DNNs, e.g., DNN 1, is more likely to be selected in ensembles, mainly because it has better general performance compared to others. In particular, we can see that task context has a noticeable on ensemble formation. For example, DNN 1 is more likely to be selected when the image noise level is moderate (0.2-0.8); DNN 2 is more likely to be selected when the image noise level is low (0-0.2); and DNN 6 is more likely to be selected when the image noise level is high (0.6-1.0).

VII. CONCLUSION

This paper investigates a DNN ensemble formation (DEF) problem for edge computing systems, which aims to identify the best-fit DNNs for users' tasks. An online learning algorithm, called NeuE, is proposed to offer a solution to the DEF problem. NeuE does not rely on heuristic rules for DNN formation, instead, it directly learns the performance of DNN ensembles and adaptive reconfigure members in the DNN ensemble. In particular, the proposed method takes into account several unique properties of edge computing platforms including device heterogeneity, computing resource constraint, and the service deadline. We further propose a variant, called NeuE-S, of NeuE to improve scalability by leveraging the similarity of DNN ensemble decisions. NeuE-S drastically reduces the computing complexity of NeuE with a negligible performance loss. The proposed method can be generalized to work in many other scenarios that involve optimizing members of ensemble classifiers.

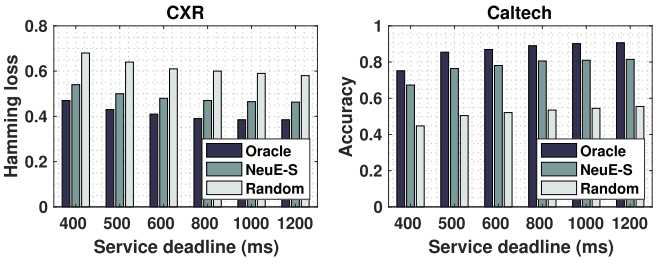


FIGURE 10. Impact of service deadline.

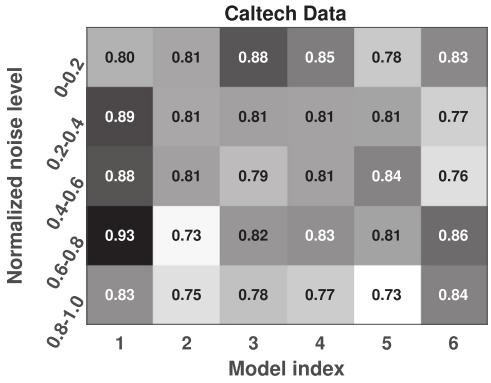


FIGURE 11. Impact of task context on the DNN ensemble formation.

REFERENCES

[1] Google Inc., "Android to launch TensorFlow lite for mobile machine learning," 2022. [Online]. Available: https://www.tensorflow.org/lite/android

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

[3] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 365–382.

[4] T. Zhang et al., "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 184–199.

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

[6] 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 *Proc. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 389–400.

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[10] Large scale visual recognition challenge (ILSVRC), 2017. [Online]. Available: http://www.image-net.org/challenges/LSVRC/

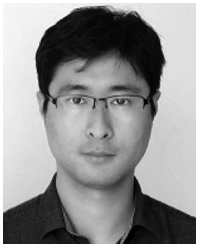
[11] A. Kurakin et al., "Adversarial attacks and defences competition," in *The NIPS'17 Competition: Building Intelligent Systems*. Berlin, Germany: Springer, 2018, pp. 195–231.

[12] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu, "Improving adversarial robustness via promoting ensemble diversity," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 4970–4979.

[13] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[14] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Mach. Learn.*, vol. 51, no. 2, pp. 181–207, 2003.
[15] S. Rothe and D. Söffker, "Comparison of different information fusion methods using ensemble selection considering benchmark data," in *Proc. IEEE Int. Conf. Inf. Fusion*, 2016, pp. 73–78.
[16] Y. Bai, L. Chen, M. Abdel-Mottaleb, and J. Xu, "Automated ensemble for deep learning inference on edge computing platforms," *IEEE Internet of Things J.*, vol. 9, no. 6, pp. 4202–4213, Mar. 2022.
[17] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with upper confidence bound-based exploration," 2019, *arXiv: 1911.04462*.
[18] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "ChestX-ray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2097–2106.
[19] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006.
[20] A. Gallego, A. Pertusa, and P. Gil, "Automatic ship classification from optical aerial images with convolutional neural networks," *Remote Sens.*, vol. 10, no. 4, 2018, Art. no. 511.
[21] S. Yang, P. Luo, C. C. Loy, and X. Tang, "WIDER FACE: A face detection benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5525–5533.
[22] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *Syst. Man Cybern.*, vol. 42, no. 4, pp. 463–484, 2012.
[23] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794.
[24] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Mach. Learn.*, vol. 42, no. 3, pp. 287–320, 2001.
[25] D. H. Wolpert, "Original contribution: Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.
[26] Z. Yu and C. Zhang, "Image based static facial expression recognition with multiple deep network learning," in *Proc. Int. Conf. Multimodal Interact.*, 2015, pp. 435–442.
[27] Y. Xiao, J. Wu, Z. Lin, and X. Zhao, "A deep learning-based multi-model ensemble method for cancer prediction," *Comput. Methods Programs Biomed.*, vol. 153, pp. 1–9, 2018.
[28] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive selection of deep learning models on embedded systems," 2018, *arXiv: 1805.04252*.
[29] E. Park et al., "Big/little deep neural network for ultra low power inference," in *Proc. IEEE 10th Int. Conf. Hardware/Softw. Codes. Syst. Synth.*, 2015, pp. 124–132.
[30] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. Int. Conf. World Wide Web*, 2010, pp. 661–670.
[31] W. Chu, L. Li, L. Reyzin, and R. E. Schapire, "Contextual bandits with linear payoff functions," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 208–214.
[32] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári, "X-armed bandits," *J. Mach. Learn. Res.*, vol. 12, no. 46, pp. 1655–1695, 2011.
[33] M. Valko, N. Korda, R. Munos, I. Flaounas, and N. Cristianini, "Finite-time analysis of kernelised contextual bandits," in *Proc. 29th Conf. Uncertainty Artif. Intell.*, 2013, pp. 654–663.
[34] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, Third Quarter 2017.
[35] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang, "Exploring fine-grained resource rental planning in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 3, no. 3, pp. 304–317, Third Quarter 2015.
[36] L. Chen and J. Xu, "Budget-constrained edge service provisioning with demand estimation via bandit learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2364–2376, Oct. 2019.
[37] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 762–772, Dec. 2013.
[38] X. Lv, D. Ming, T. Lu, K. Zhou, M. Wang, and H. Bao, "A new method for region-based majority voting CNNs for very high resolution image classification," *Remote Sens.*, vol. 10, no. 12, 2018, Art. no. 1946.
[39] J. Huo et al., "Ensemble segmentation for GBM brain tumors on MR images using confidence-based averaging," *Med. Phys.*, vol. 40, no. 9, 2013, Art. no. 093502.
[40] R. M. Cruz, R. Sabourin, G. D. Cavalcanti, and T. I. Ren, "META-DES: A dynamic ensemble selection framework using meta-learning," *Pattern Recognit.*, vol. 48, no. 5, pp. 1925–1935, 2015.
[41] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[42] L. Chen, J. Xu, and Z. Lu, "Contextual combinatorial multi-armed bandits with volatile arms and submodular reward," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 3247–3256.
[43] J. H. Rolfes and F. Vallentin, "Covering compact metric spaces greedily," *Acta Mathematica Hungarica*, vol. 155, pp. 130–140, 2018.
[44] L. Fei-Fei, J. Deng, and K. Li, "ImageNet: Constructing a large-scale image database," *J. Vis.*, vol. 9, no. 8, pp. 1037–1037, 2009.
[45] PyTorch, "Models and pre-trained weights," 2017. [Online]. Available: https://pytorch.org/vision/master/models.html

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 an 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.



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 UCLA, 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.