2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC) | 978-1-6654-6087-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/UCC56403.2022.00012

Edge-MultiAI: Multi-Tenancy of Latency-Sensitive Deep Learning Applications on Edge

SM Zobaed*, Ali Mokhtari*, Jaya Prakash Champati†, Mathieu Kourouma‡, Mohsen Amini Salehi*

*{sm.zobaed1, ali.mokhtari1, amini}@louisiana.edu
HPCC Lab, School of Computing and Informatics
University of Louisiana at Lafayette, LA, USA

†jaya.champati@imdea.org
IMDEA Networks Institute
Madrid, Spain

‡mathieu_kourouma@subr.edu
Southern University and A&M College
LA, USA

***Abstract*—Smart IoT-based systems often desire continuous execution of multiple latency-sensitive Deep Learning (DL) applications. The edge servers serve as the cornerstone of such IoT-based systems, however, their resource limitations hamper the continuous execution of multiple (multi-tenant) DL applications. The challenge is that, DL applications function based on bulky "neural network (NN) models" that cannot be simultaneously maintained in the limited memory space of the edge. Accordingly, the main contribution of this research is to overcome the memory contention challenge, thereby, meeting the latency constraints of the DL applications without compromising their inference accuracy. We propose an efficient NN model management framework, called Edge-MultiAI, that ushers the NN models of the DL applications into the edge memory such that the degree of multi-tenancy and the number of warm-starts are maximized. Edge-MultiAI leverages NN model compression techniques, such as model quantization, and dynamically loads NN models for DL applications to stimulate multi-tenancy on the edge server. We also devise a model management heuristic for Edge-MultiAI, called *iWS-BFE*, that functions based on the Bayesian theory to predict the inference requests for multi-tenant applications, and uses it to choose the appropriate NN models for loading, hence, increasing the number of warm-start inferences. We evaluate the efficacy and robustness of Edge-MultiAI under various configurations. The results reveal that Edge-MultiAI can stimulate the degree of multi-tenancy on the edge by at least $2\times$ and increase the number of warm-starts by $\approx 60\%$ without any major loss on the inference accuracy of the applications.**

***Index Terms*—Edge computing, Multi-tenancy, Deep learning applications, Memory contention.**

I. Introduction

A. Motivation and Overview

With the expeditious advances of smart IoT-based systems, they are becoming an indispensable part of our day-to-day life. Such systems often provide their users with multiple latency-sensitive Deep Learning (DL) services, such as object detection, face recognition, and motion capture, that can collectively unlock use cases to improve the human's quality of life. An exemplar use case of such IoT-based systems is SmartSight [1], illustrated in Figure 1, that aims at providing ambient perception for the blind and visually impaired people. The system operates based on a smartglass (IoT device) and a companion edge server (e.g., smartphone). The smartglass continuously captures the inputs via its sensors (e.g., camera and microphone) and requests the edge server to process DL-based applications, such as object detection to identify obstacles; face recognition to identify acquainted people; speech recognition, and NLP to understand and react to the user's commands. To make SmartSight usable, the edge server has to continuously execute multiple (a.k.a. *multi-tenant*) DL application to process incoming requests with low-latency and high accuracy. It is noteworthy that, although cloud datacenters can mitigate the inherent resource limitations of the edge, due to the network latency overhead and data confidentiality [2], [3], [4], offloading the latency-sensitive service requests to the cloud is not a tractable approach in many use cases.

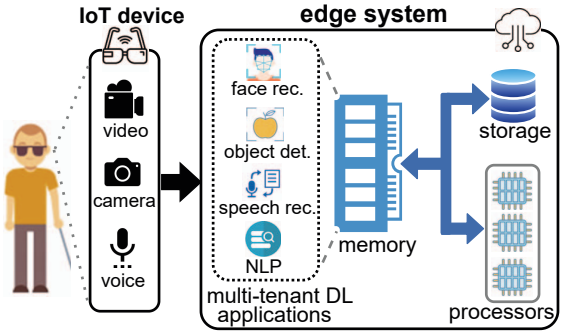

Fig. 1: Bird-eye view of SmartSight, an IoT-based system that continuously receives various inputs from the smartglass (IoT device) sensors, and processes them via multi-tenant DL applications running on the edge server.

DL applications utilize bulky Neural Network (NN) models at their kernel to infer on the inputs received from the sensors. The NN models have to be kept in memory to enable low-latency (a.k.a. *warm-start* [5]) inference operations. Otherwise, because the NN model size is often huge, loading it into the memory in an on-demand manner (a.k.a. *cold-start*) is counterproductive and affects the latency constraint of the DL applications. As the edge servers naturally have a limited memory size (e.g., 4 GB in the case of Jetson Nano [6]), multi-tenant execution of DL applications on them leads to a memory contention challenge across the processes [2], [7]. Accordingly, the main challenge of this study is to resolve the memory contention across multi-tenant DL applications without compromising their latency and accuracy constraints.

In the deep learning context, there are techniques based on the idea of approximate computing, such as quantization [8], that make the model edge-friendly via compressing its NN model, hence, reducing its inference time and accuracy. To

978-1-6654-6087-3/22/$31.00 ©2022 IEEE
DOI 10.1109/UCC56403.2022.00012

TABLE I: Load time, inference time, and accuracy of popular NN models individually running on Samsung Galaxy S20+ as the edge server.

NN Models	Bit Width	Size (MB)	Loading Time (ms)	Inference Time (ms)	Accuracy (%)
InceptionV3	FP32	105	650	100	78.50
	INT8	24	380	80	77.20
VGG16	FP32	528	820	52	71.30
	INT8	132	185	40	70.18
MobileNetV1	FP32	89	600	15	70.56
	INT8	23	192	8	65.70
MobileNetV2	FP32	26	110	10	72.08
	INT8	9	65	7.5	63.70
MobileNetV3	FP32	14	80.3	7.80	74.04
	INT8	8	47.45	6.21	71.32
MobileBERT	FP32	96	1100	62	81.23
	INT8	26	890	40	77.08

understand the impact of such approximations, we conducted a preliminary experiment using a Samsung Galaxy S20+ as the edge server; and five popular DNN models, namely InceptionV3, VGG16, MobileNetV1, MobileNetV2, MobileNetV3, MobileBERT, each one at two quantization (precision) levels, namely FP32 and INT8 bit widths. In Table I, we report the average loading time, inference time, and accuracy for their individual executions. We observe that: (A) for all the models, the loading time is 8—17$\times$ more than its inference time; (B) Loading the high-precision model (FP32 bit width) occupies $\approx$3.5$\times$ more memory than the low-precision (INT8 bit width) one; and (C) Loading a low-precision model can reduce the inference accuracy by around 3—6%. These results demonstrate that the model compression has a considerable potential to mitigate the memory footprint of the DL applications. Moreover, the model loading time invariably dominates the inference time [9]. Accordingly, our hypothesis is that *the efficient use of model compression and the edge memory can enhance the multi-tenancy and inference time of DL applications without any major loss on their inference accuracy*.

We propose each DL application to be equipped with multiple NN models with different precision levels. The low-precision models have a small memory footprint, hence, allowing for a higher multi-tenancy of DL applications with their models loaded into the memory (i.e., warm-start inference) that enhances the service latency. However, loading overly low-precision (over-quantized) models to maximize multi-tenancy and warm-start inference is not viable, because it reduces the inference accuracy and renders the multi-tenant DL applications to be futile. On the contrary, loading high-precision (large) NN models on a memory-limited edge system for an indefinite time period unnecessarily occupies an excessive memory space that is detrimental for the multi-tenancy and warm-start inference of other tenants. That is, other tenants face a significant slow down (as noted in Table I), because they cannot keep their NN model in memory and have to load it from the storage (i.e., cold-start) to perform the inference operation. Therefore, an ideal solution for a multi-tenant edge system should be able to dynamically load a suitable model from the set of models available to the application (a.k.a. model zoo), such that it neither interrupts the execution of other applications, nor causes a cold-start inference for them.

B. Problem Statement

The research question that we investigate is: *how to maximize the number of warm-start inferences for multi-tenant DL applications on edge without compromising the inference accuracy?* The question indicates a trade-off between two objectives: fulfilling the latency constraint of DL applications and maintaining their inference accuracy. The former objective entails having the NN models of DL applications loaded into the memory (i.e., warm-start inference), whereas, the latter entails retaining high-precision NN models in the memory.

For application $A_i \in A$ with $M_i = \{m_i^k \mid 1 \leq k \leq q_i\}$ as its model zoo, let $r_i(t)$ be a Boolean function that represents an inference request for A_i at time t with value 1. Also, let $m_i^* \subseteq M_i$ be an NN model of A_i with size of s_i^* that is currently loaded in the memory. This means that, for application A_j that does not have any of its NN models currently in the memory, we have $m_j^* = \varnothing$ and $s_j^* = 0$. Then, $M^* = \bigcup_{i=1}^{n} m_i^*$ represents the set of currently loaded NN models that occupy $S^* = \sum_{i=1}^{n} s_i^*$ of the memory space. A cold start event for the request arrives at time t for A_i, denoted $C_i(M^*, t)$ and shown in Equation (1), occurs when there is no NN model in memory for A_i (i.e., $M_i \cap M^* = \varnothing$).

$$C_i(M^*, t) = \begin{cases} r_i(t) & M_i \cap M^* = \varnothing \\ 0 & otherwise \end{cases} \tag{1}$$

Assume that utilizing $m_i^* \in M_i$ results in an inference accuracy that we denote it as χ_i^*. Then, based on Equation (2), for n multi-tenant DL applications, we can formally state the objective function as minimizing the total number of cold-start inferences, while maximizing the accuracy of the inferences. In this case, the total memory size available for the NN models (denoted S) serves as the constraint.

$$\min\left(\int_t^{\infty} \sum_{i=1}^{n} C_i(M^*, t)\ dt\right), \quad \max\left(\int_t^{\infty} \sum_{i=1}^{n} \chi_i^*(t)\ dt\right)$$
$$\text{subject to:} \qquad \forall t, \ \sum_{i=1}^{n} s_i^* \leq S \tag{2}$$

Note that optimal NN model management decisions do not have a greedy nature. That is, minimizing the number of cold-start inferences at a given time t does not necessarily lead to the minimum total number of cold-starts with maximum accuracy during the entire applications' lifetime. In other words, the system may experience a cold-start at time t to prevent multiple ones at a later time. That is why, the objective function of Equation 2 includes integrals over t to the ∞ to encompass the impacts of the decisions at t on the future cold-starts and accuracy levels. In the objectives, the NN models of application A_i are only chosen from its model zoo (M_i), thus, the accuracy ($\mu_i(t)$) and size functions (s_i) are discrete functions. It is needless to say that *minimizing the number of cold-start inferences* is equivalent to *maximizing the number of warm-start events* [10]. In the rest of this study, we use these two interchangeably.

C. Solution Statement and Contributions

To stimulate multi-tenancy on the limited edge memory, we develop a framework, called Edge-MultiAI, that takes advantage of a model zoo for each DL application and can dynamically swap the NN models of the applications. To maximize the number of warm-starts with high inference accuracy across multi-tenant DL applications, our approach is to proactively load the high-precision NN models for the applications that are expected to receive inference requests, while loading low-precision models for the others. We utilize the recent memory usage information to predict the memory availability for the next executions while not interrupting other active applications. We develop model management heuristic policies that make use of the expected memory availability and the usage pattern of multi-tenant DL applications to choose a suitable NN model for the requester application right before the inference operation, thereby, both the latency and inference accuracy of the application are fulfilled.

In summary, the contributions of this work are as follows:

- We develop an NN model management framework for multi-tenant DL applications on the edge server, called Edge-MultiAI, that efficiently utilizes the memory such that the multi-tenancy degree and number of warm-start inferences are maximized without any major compromise on the inference accuracy. Edge-MultiAI dynamically loads the high-precision NN model for the requester application, while loading low-precision ones for others.
- We develop iWS-BFE policy along with three other baseline heuristic policies within Edge-MultiAI to choose the suitable model for the application performing inference, and to decide how to allocate memory for it.
- We evaluate and analyze the efficacy of the proposed heuristics in terms of their effectiveness and robustness against uncertainties in the inference request prediction.

The rest of this paper is organized as follows. Section II discusses background study and related prior works. We explain the overview of Edge-MultiAI architecture in Section III. Next, we discuss experimental evaluation and performance analysis in Section IV. Finally, Section V concludes the paper and provides a few avenues for the future studies.

II. Background and Related Work

A. Edge AI

1) The Scope for Edge Intelligence: Numerous research have been undertaken to explore the applications, scopes, and benefits of edge-based AI for the seamless execution of latency-sensitive smart applications [11], [12], [2], [13]. Murshed et al. discussed different DNN-based practical applications such as video analytics and image recognition for enabling edge AI [12]. Zhou et al. surveyed on various training and inference techniques for NN models on edge devices [13]. Chen and Ran discussed different techniques that can help to accelerate the DL training and inference on the edge-based systems [11]. Han et al. explored the ways to accelerate the training convergence for the edge-based architectures [2]. Wang et al. surveyed the development of DL applications on edge from the latency and bandwidth perspectives [14]. Zhou et al. [13] claimed that although higher edge intelligence reduces data offloading and improves the privacy, the latency and energy consumption overhead can increase.

2) Multi-tenant Execution on Edge: Prior studies investigated AI multi-tenancy on the edge servers. Mao et al. proposed a mobile computing framework, MoDNN, to execute DL applications simultaneously on resource-constrained devices [15]. MoDNN can partition pre-trained DNN models across several mobile devices to accelerate tensor processing with reduced device-level computing cost and memory usage while achieving $2.17\times$—$4.28\times$ speedup.

Multi-tenant execution across edge servers can lead to undesirable latency in application execution. Ko et al. proposed DisCo, a multi-tenant DL application execution offloading framework that enables execution of both the compute- and data-intensive parts of applications either on the device or on the edge [16]. Hadidi et al. discussed that complex DNN models are sensitive to data loss as they depend more on the nuances in the data [17]. They mentioned losing one layer of the Inception V3 model can deteriorate the accuracy by more than 50%. They utilized distributed DNN models on IoT systems to reduce the processing and the memory footprints.

The aforementioned research works addressed the problem of accelerating multi-tenant applications without considering the memory constraint of the edge servers. The only exception, to the best of our knowledge is [18], in which the authors explored the executing the obstacle detection application in an autonomous vehicle with ultra low-latency constraint upon compromising with other executing applications. They proposed a reinforcement learning-based technique to scavenge memory from a non-priority application, hence, executing the obstacle detection application immediately and avoid accidents. Although their technique is effective to serve the latency-sensitive task, multi-tenant executions is out of their scope [18]. In contrast to these works, we investigate the problem of memory management to increase the degree of multi-tenancy and the number of warm-start inferences, thereby, improving the practical usability of IoT-based systems.

B. DNN Model Compression

Model compression techniques allow for running a model on different resource-constrained devices. There are mainly two techniques to reduce the complexity of a given DNN model: making use of a fewer bit widths (a.k.a. *quantization*) and using fewer weights (a.k.a. *pruning*). These techniques have been considered individually and together to serve the purpose of model compression.

Quantization. Quantization reduces the computational resource demand at the expense of a diminutive loss in accuracy. By default, the model weights are float32 type variables which means 4 bytes are associated with each model weight with a significant amount of memory requirements. Model weights can be reduced from 32 bits to 8 bits (or even shorter [7]) to accelerate inference operation.

Pruning. The pruning technique is applied to reduce the memory consumption of the model to accelerate the inference operations. An effective pruning technique removes redundant connections and/or reduces the width of a layer while ensuring a slight impact on the inference accuracy. Therefore, the pruned models are retrained to compensates the loss in accuracy. Failure of selecting proper pruning candidates affects inference tasks and make the pruned model futile. Some studies have also been conducted on the selection of appropriate pruning candidates.

For compatibility with the IoT devices, Yao et al. proposed DeepIoT [19], a reinforcement learning pruning technique for DNN models in the IoT devices. However, during pruning the model parameters, they only considered the execution time speed-up, hence, the technique inevitably exhibits inferior inference accuracy performance. As noted above, aggressive pruning often substantially degrades the inference accuracy. Training and inference with high pruning with negligible impact on the accuracy is still an open research problem [7].

Warm-Start vs Cold-Start Inference. Provided the increasing complexity of DNN models, loading even compressed models to the edge memory is a burden. The problem is further complicated in scenarios where the edge server has to continuous maintain multiple applications in its memory (i.e., multi-tenancy) which is cost-prohibitive.

Nonetheless, cold-start inferences should be avoided as they bring about a remarkable inference latency (see *loading time* in Table I for more details). Some research works have been accomplished to avoid cold-start inferences. For instance, to support latency-sensitive applications, in [20], the authors proposed cold-start of a DNN model in the background while the user is browsing a specific web page. By utilizing system resources, their technique tracks the user's browsing activity and loads the task-specific model in parallel during browsing activity to avoid the cold-start.

III. Allocating Multi-tenant Deep Learning Applications on the Edge Systems

A. Architectural Overview & System Design of Edge-MultiAI

Figure 2 illustrates the architectural overview of Edge-MultiAI that facilitates multi-tenancy of DL applications on a resource-limited edge system via enabling the applications to only swap their NN models, instead of the entire application. The framework consists of three tiers: (i) Application tier, (ii) NN model manager, and (iii) Memory tier.

Application Tier. The incoming multi-modal inputs from the connected IoT devices trigger execution of multi-tenant DL applications in the application tier. The *model zoo* for each DL application acts as a repository that contains NN models with different compression levels (sizes) and inference accuracy (a.k.a. various *precision levels*). The *model loader* is responsible for loading the chosen NN model from the model zoo into the edge memory.

NN Model Manager. NN model manager comprises of three components: (i) *application request predictor*, (ii) *memory predictor*, and (iii) *memory optimizer*. "Application request predictor" collects historical requests to each application and trains a lightweight (edge-friendly) many-to-one vanilla recurrent neural network (RNN) time series prediction model, similar to the one in [21], to periodically foresee the inference request arrivals for each application. Upon arrival of each request, "memory predictor" is in charge of predicting the memory availability based on the recent memory allocations in the entire edge system. We leverage the historical memory allocation data and train another many-to-one vanilla RNN time-series prediction model to predict the available memory.

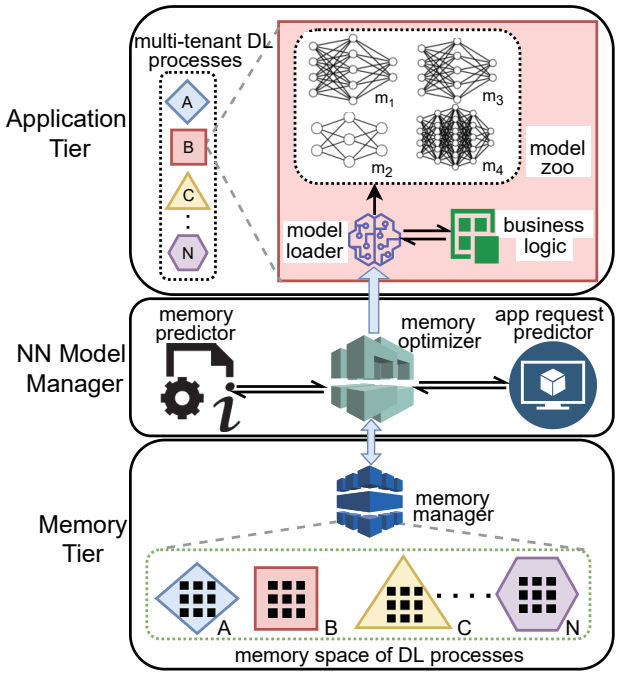

Fig. 2: Architectural overview of the Edge-MultiAI framework with three tiers: Application, NN Model Manager, and Memory.

Memory optimizer interacts with the application "request predictor" and "memory predictor" to receive: (A) the request arrival time for different applications plus the information of their model zoo; and (B) the memory availability information. Then, the memory optimizer feeds the received information to an NN model management policy that determines the highest possible precision NN model that can be loaded to serve the inference request of a DL application with the minimum impact (in terms of the prediction accuracy or latency) on the execution of other applications. Upon facing memory shortage for an arriving inference request, the memory optimizer scavenges the memory allocated to the NN models of other applications via either loading a lower-precision model or forcing them to cold-start. After procuring adequate memory, the memory optimizer informs the "model loader" to load the appropriate NN model of the requested application.

Memory Tier. The tier includes the "memory spaces" allocated to the applications; and a "memory manager" that keeps track of the currently loaded models, the available memory spaces, and the current status of the applications. The memory manager communicates these information to the NN Model

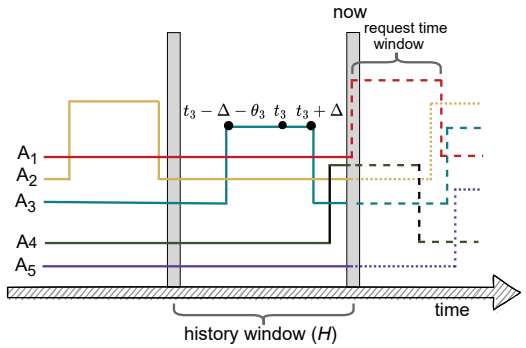

Fig. 3: A sample scenario of inference requests for five multi-tenant applications, namely A_1 to A_5. Each pulse represents the time window within which an inference request is expected. Solid lines expresses the event that has already happened and dashed lines after "now" are the request predictions.

Manager to efficiently allocates them to the arriving requests.

B. Heuristics to Manage Models of Multi-tenant Applications

1) Overview: Recall that the aim of NN model management policy is to minimize the number of cold-start inferences and maximize the inference accuracy for multi-tenant DL applications on the edge servers. To that end, the memory optimizer strives to maximize the time to retain the loaded models in the edge memory. However, due to limitations in the available memory space, it is not possible to retain the highest precision NN model of all applications in the memory. To resolve this memory contention, the NN models of the applications that are unlikely to be requested in the near future should be assigned a lower priority to remain in the memory. Furthermore, Edge-MultiAI makes it possible to dynamically load NN models for the applications. This means that, upon predicting time t as the inference request time for a given DL application, Edge-MultiAI can be instructed to load the high-precision NN model of that application immediately before performing the inference. Similarly, in the face of a memory shortage, for the application(s) that are unlikely to be requested at time t, Edge-MultiAI can be instructed to unload their NN models or, more interestingly, replace them with a lower precision one.

However, we know that the request arrivals are inherently uncertain [1] and no prediction model can precisely capture the exact request time for an application. To capture the uncertainty, we consider a request time window, denoted as Δ, around each predicted request time. The value of Δ is obtained from profiling past request predictions and calculating the mean difference of actual arrival time and the predicted ones across all applications. In addition, there is a time overhead, denoted as θ_i, to load the chosen NN model of an application A_i into the memory. In sum, to prevent a cold-start for A_i that is predicted to perform inference at time t, as shown in Figure 3, the NN model has to be loaded at time $(t_i - \Delta - \theta_i)$ and kept in memory until $(t_i + \Delta)$. Furthermore, there is uncertainty in predictions of "no request" for an application at a given time. That is, at time t, there can be an inference request for an application that was predicted not to have an request at that time. To make the system robust against this type of uncertainty and to avoid cold-start inferences in these circumstances, an ideal policy should load low-precision NN models for these applications. Hence, an unpredicted inference request can be still served as a warm-start by the low-precision model and the latency constraint is maintained.

In this work, the set of applications whose NN models are retained in memory outside of their predicted request time window are called the *minimalist* set, and denoted as A'. Similarly, the set of applications that are in their request time window and we load a high-precision model for them are called *maximalist*, and denoted as A^*. To resolve the memory contention, the policy can be based on scavenging memory from the minimalist applications to procure the required memory space for the maximalist ones. That is, in the event that application A_i is predicted to have an inference at time t_i, it becomes a member of A^* set at time $t_i - \Delta - \theta_i$, and then becomes a member of A' set after $t_i + \Delta$; thus, its model can be evicted from the memory in the event the memory space is needed for another maximalist application. The NN model eviction is only permitted from A' set and we aim at retaining a low-precision model for the applications in this set. However, due to high inference demand, A' have to unload their models (i.e., switch to cold-start) to free space for the model of the applications that are in the maximalist set. In an extreme situation, if A' is empty, or the scavenged memory from A' cannot procure sufficient space to load the suitable model for application A_i, the next (smaller) model for A_i is considered, and the aforementioned steps are repeated. Ultimately, if the scavenged memory space is inadequate for the lowest precision model of A_i, an inference failure occurs.

The memory contention problem can be reduced to the classic binary Knapsack optimization problem [22] where from a collection of items, each one with a weight and a value, we need to select items such that the total value is maximized, while the total weight is bounded to a limit. This problem is known to be NP-Complete,hence, we can rely on the heuristic-based solutions for it [23]. In the next part, we discuss four NN model management (a.k.a. *NN model eviction*) policies to manage the memory for multi-tenant DL applications such that the number of warm-start inferences is maximized without any major impact on the inference accuracy.

2) Policy 1: Largest-First Model Eviction (LFE): In this policy, to allocate memory for the NN model of a maximalist process, we first evict NN models from set (A' that occupy the highest memory space, until there is enough space to allocate the high-precision NN model of A^*. For that purpose, members of A' are sorted based on the size of their currently loaded NN model in the descending order. In the event that evicting all the NN models of A, does not free enough memory space to allocate the NN model of the request, a lower precision NN model (smaller in size) is tried for allocation. This procedure continues until a model from the model zoo can be allocated in the memory; otherwise, the edge system is not able to serve that request at that time.

3) Policy 2: Best-Fit Model Eviction (BFE): The limitation of LFE is to evict the largest NN models of the minimalist applications, irrespective of the exact memory requirement. This means that adopting LFE can free more memory space than the actual requirement. To tackle the issue, we implement the BFE policy where applications in the minimalist set are sorted based on the difference between their model sizes and the actual memory requirement. Then, the NN model with a minimum difference is chosen for eviction. The memory requirement for a maximalist application is first calculated based on its highest precision (largest) NN model to gain the highest inference accuracy. However, in the event that evicting the NN models of all the minimalist applications do not free enough memory space to allocate the desired NN model, BFE iteratively selects the next high-precision model from the model zoo of the requested application.

4) Policy 3: Warm-Start-aware Best-Fit Model Eviction (WS-BFE): Let $A_i \in A^*$ an application that is currently in the maximalist set, and $A_j \in A'$ an application that is currently in the minimalist set. It is technically possible that the predicted request time window of A_i overlaps with the one for A_j. In this case, LFE and BFE policies potentially choose to evict the NN model of A_j in favor of the A_i model. This is because both of these policies are backward-looking and ignore the fact that A_j can be requested soon after evicting its NN model. Such an eviction decision increases the likelihood of a cold-start inference and to avoid that, we develop WS-BFE that assigns the lowest eviction priority to those applications in A' that have overlapping time window with A_i.

In our early experiments, we realized that another reason for cold-start inferences is due to uncertain nature of request arrivals. That is, a minimalist application is unexpectedly requested. To minimize the likelihood of cold-start inference in these circumstances, we implement WS-BFE to *replace* the evicted NN model with the lowest-precision (i.e., smallest) NN model of that application. As such, in the event of an unpredicted request the minimalist applications, there is a low-precision model available to carry out a warm-start inference.

5) Policy 4: Intelligent Warm-Start-aware Best-Fit Eviction (iWS-BFE): To make WS-BFE robust against uncertainties in the application request time prediction, we enhance it by applying the Bayesian theory and proposing a new policy, called iWS-BFE. This policy is inspired from the widely-adopted LRU-K cache management policy [24] that considers *the least recently used (i.e., requested) applications are not likely to be requested in the near future*. Similarly, iWS-BFE only considers members of A' as eviction candidates, denoted by E', that are not recently requested. Figure 3, shows a scenario of predicted request times for A_1—A_5. To procure memory for A_1, we have $A' = \{A_2, A_3, A_5\}$. Because A_3 was requested during the "history window" (H), it is likely to be requested in the near future. Hence, iWS-BFE, chooses $E' = \{A_2, A_5\}$ for eviction. The value of H is determined based on the mean request inter-arrival time of all applications.

In addition to considering LRU, iWS-BFE also makes use of the request prediction, provided by Edge-MultiAI. That is, it considers the most appropriate application for eviction as the one that has not been recently requested, and is predicted to be requested the latest in future. However, the request time predictions are uncertain, and the system can receive an unexpected request from members of E' in the current request window. To make iWS-BFE robust against such uncertainty, we calculate the probability of an unexpected request. For application $A_j \in E'$, let r_j denote an unexpected request. Then, the probability of r_j occurring during the current request window (i.e., $[t, t+\Delta]$) is defined as $P(r_j|A_i \in A^*)$. The application that is likely to be requested unexpectedly is not an optimal choice for eviction. Therefore, in Equation 3, to calculate the fitness score of A_j for eviction (denoted $Score(A_j)$), we consider $1 - P(r_j|A_i \in A^*)$. To take the predicted request time of A_j into consideration, we calculate the distance between its predicted request time and the current time (i.e., $t_j - t_i$). To confine the value between [0,1], we normalize the distance based on the latest predicted distance across all k applications.

$$Score(A_j) = \frac{t_j - t_i}{\max_{k \in E'}(t_k - t_i)} \cdot \big[1 - P(r_j|A_i \in A^*)\big] \qquad (3)$$

The pseudo-code of the iWS-BFE policy is provided in Algorithm 1. It begins with an initial set of eviction candidates, called $\tau \subseteq A'$, that is formed based on the applications that were not requested during the history window (H). From τ, in Step 3, a list of eviction candidates (denoted E) whose elements do not overlap with the request window of active application (A_i) is derived. Next, in Step 4, we use Equation 3 to calculate the fitness score for each $E_k \in E$ and then, build a max-heap tree of E based on the fitness scores (Step 5). In Steps 6—10, the policy iteratively retrieves the application with the highest fitness score (i.e., the max-heap root, denoted w) and foresees the amount of memory that can be scavenged upon replacing its loaded model with the lowest-precision one. Once the policy finds enough memory to be scavenged such that the NN model of A_i (denoted m_i) can be loaded, in Step 13, it enacts all the NN model replacement decisions and then loads m_i in Step 14. In the event that the scavenged memory is insufficient, the policy switches to the next NN model for A_i that has a lower size and accuracy (Step 17). In the worst case that even the smallest NN model of A_i cannot fit in the memory, the inference request fails (Step 17) [25].

IV. Performance Evaluation

A. Experimental Setup and Evaluation Metrics

To evaluate the efficacy of Edge-MultiAI and its NN model eviction policies, we benchmarked five different DL applications, namely face recognition, speech recognition, image classification, next sentence prediction, and text classification, and recorded their real characteristics, including the model size, and the inference accuracy (shown in Table II). We have developed the E2C simulator that enables modeling the IoT-based systems with different characteristics and configurations, and is available publicly for the community access through our

Algorithm 1: Pseudo-code for iWS-BFE NN model eviction policy

Function `iWS-BFE(`A', A^*, A_i, H`)`
  $\tau \leftarrow$ Select $\forall A'_j \in A'$ not requested during H
  $E \leftarrow$ Determine $\forall A'_j \in \tau$ non-overlapping with request window of A_i
  $\forall E_k \in E$ calculate fitness score using Equation 3
  Build max-heap tree of E based on fitness score
  while $size(m_i) >$ *available memory* **do**
    $w \leftarrow$ Extract root of the max-heap tree
    If $w = \emptyset$ **then** break the loop
    Measure memory scavenged by replacing model of w with its lowest-precision one
    Add scavenged amount to *available memory*
  end
  if $size(m_i) \leq$ *available memory* **then**
    Enact NN model replacement(s) decisions
    Scavenge the leftover memory to load m_i
  end
  else
    If there is no model left to check **then** the inference request fails
    Repeat Step 6—10 with the next (smaller) model
  end
end

Application	NN Model	Bit Width	Size (MB)	Accuracy (%)
Face recognition	VGG-Face	FP32	535.1	90.2
		FP16	378.8	82.5
		INT8	144.2	71.8
Image classification	VIT-base-patch16	FP32	346.4	94.5
		FP16	242.2	81.3
		INT8	106.7	72.2
Speech recognition	S2T-librisspeech	FP32	285.2	89.7
		FP16	228.0	77.2
		INT8	78.4	68.0
Sentence prediction	Paraphrase-Mini LM-L12-v2	FP32	471.3	88.2
		FP16	377.6	81.7
		INT8	98.9	76.2
Text classification	Roberta-base	FP32	499.0	91.1
		FP16	392.2	82.4
		INT8	132.3	76.6

TABLE II: Application-specific models with different precision variants that are experimented.

Github page[1]. The simulator has implemented all of the NN model eviction policies, and the user can quickly deploy and examine any one of them.

The simulator also enables us to generate workload traces that include the request arrival times for each application during the simulation time. We configure the actual workload to include an equal number of requests for the five applications, and the inter-arrival times between requests for each application are distributed exponentially within the workload. To study the uncertainty exists in the inference request predictions, in the evaluations, we generate two sets of workloads, one includes the predicted arrival times for the multi-tenant applications, and the other one includes the actual arrival times of the applications. The distribution of request arrivals in the actual workload deviates from the distribution of requests in the predicted workload. The degree of deviation between the two is measured based on the Kullback-Leibler (KL) [26] divergence. We explore the impact of this deviation in the experiments of next subsections.

Our evaluation metrics are: (A) The *degree of multi-tenancy* under different request arrival intensity; (B) The *inference latency*; (C) the *inference accuracy*; and (D) The *robustness* metric to measure the tolerance of different eviction policies against the uncertainty exists in the request predictions.

B. Impact of Edge-MultiAI on the Degree of Multi-tenancy

This experiment is to examine the efficacy of Edge-MultiAI in satisfying the incoming requests to the edge server. To that end, as shown in Figure 4, we increased the workload intensity, via the mean number of concurrent requests issued, and in each case measured the *multi-tenancy satisfaction rate*, which is the percentage of warm-start inferences out of the total incoming requests during the simulation time. We examined two cases: (A) without any solution to stimulate multi-tenancy (called, no policy); and (B) with Edge-MultiAI and its iWS-BFE policy in place. The experiment was repeated 10 times and the average rate and 95% confidence intervals for each data point is reported.

The experiment shows that the degree of multi-tenancy achieved by adopting Edge-MultiAI and its iWS-BFE is remarkably higher than the situation where Edge-MultiAI is not in place. The smaller graphs show that this superiority occurs consistently during the simulation time. We also notice that the impact of employing Edge-MultiAI is more effective for higher degrees of multi-tenancy. In particular, we can see that with the mean degree of multi-tenancy is 5, using Edge-MultiAI and its iWS-BFE policy achieves ≈130% higher satisfaction rate than no policy when mean requested degree of multi-tenancy is larger than 2. This experiment justifies the efficacy of Edge-MultiAI and the NN model management in stimulating multi-tenancy of DL applications.

C. Impact of the Eviction Policies on the Cold-Start Inference

The purpose of this experiment is to analyze the impact of different NN model eviction policies on the number of cold-start inferences. For that purpose, we measure percentage of cold-start inferences caused by employing different eviction policies, particularly, upon varying the deviation of request prediction from the actual requests.

The results, illustrated in Figure 5, show that LFE and BFE perform poorly and cause a remarkable number of cold-start inferences, whereas, WS-BFE and iWS-BFE mitigate the cold-srart inferences by at least 65%. This is because, in LFE and BFE, upon evicting an NN model, its corresponding application suffers from a cold-start inference in the event of an unpredicted request. In contrast, in WS-BFE and iWS-BFE, the evicted model is replaced with a low-precision one, hence, unpredicted calls to the corresponding application do not lead

[1]Github page of the E2C simulator: https://github.com/hpcclab/E2C-Sim.git

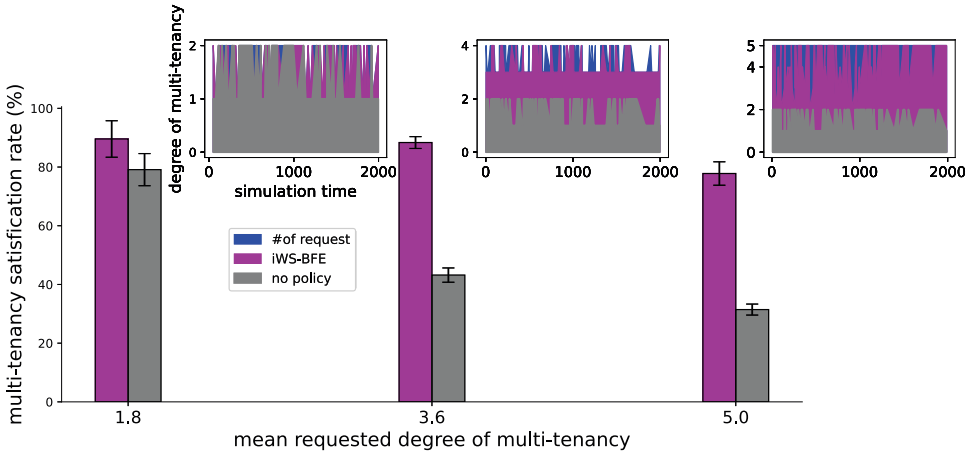

Fig. 4: The impact of Edge-MultiAI and its iWS-BFE eviction policy on satisfying the requested multi-tenancy. The large graph represents the summative analysis via increasing the mean of multi-tenancy requested in the horizontal axis, and showing the percentage of requests that were satisfied in the vertical axis. For each case, the smaller graph more granularly represents the number of concurrent requests issued and fulfilled during the simulation time.

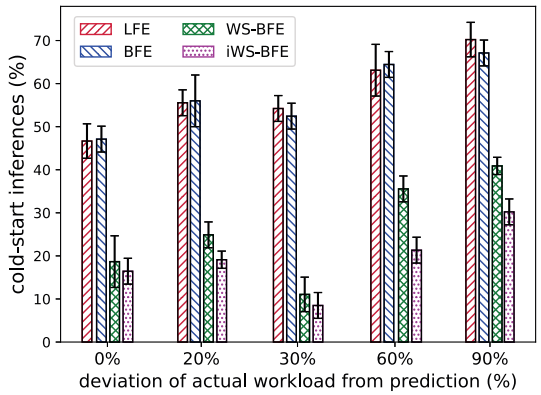

Fig. 5: Measuring the percentage of cold-start inferences of multi-tenant applications resulted from the proposed eviction policies. The horizontal axis shows the deviation between predicted and actual inference request times.

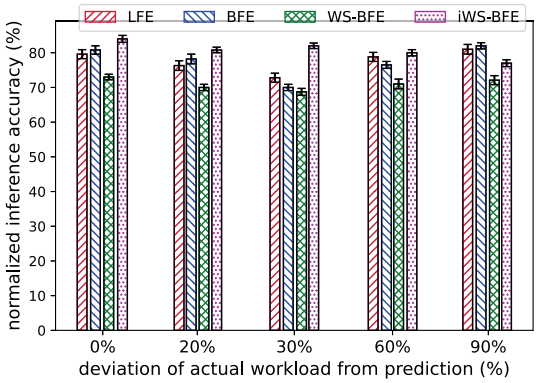

Fig. 6: Measuring the normalized inference accuracy of applications resulted from employing the different eviction policies.

to cold-start inferences. It is noteworthy that, regardless of the employed policy, the percentage of cold-start inferences rises upon increasing the deviation between predicted and actual request times. Nonetheless, we see that even under 90% deviation, iWS-BFE still substantially outperforms other policies. On average, it yields 102% less cold-start in compare to LFE and BFE, and 40% less than WS-BFE.

D. Impact of the Eviction Policies on the Inference Accuracy

In this experiment, we analyze the average inference accuracy caused by employing different model eviction policies. Because the accuracy largely varies across different applications, we perform min-max normalization on the accuracy values. Also, for the cold-start inferences, in the accuracy measurements, we consider the accuracy provided by the NN model after it is loaded into the memory.

Figure 6 shows the normalized mean inference accuracy obtained from employing different NN model eviction policies upon changing the deviation between predicted and actual request times. According to the figure, LFE and BFE policies outperform WS-BFE. This is because, these two policies do not retain the low-precision models in the memory. Therefore, their inference requests either lead to a cold-start (that was explored in the previous experiment), or they load high-precision models that provide a high inference accuracy. Nonetheless, we observe that iWS-BFE outperforms LFE and BFE in most of the cases, except the one with 90% deviation. The reason for the higher inference accuracy of iWS-BFE is that, it nominates cold-start candidates intelligently, based on their probability of future invocations. This results indicate the importance of the scoring (described in Equation 3) on efficiently nominating cold-start candidates. It is noteworthy that the higher inference accuracy of LFE and BFE at 90% deviation comes with the cost of substantially higher cold-start inferences that are detrimental to the "usability" of the IoT-based systems.

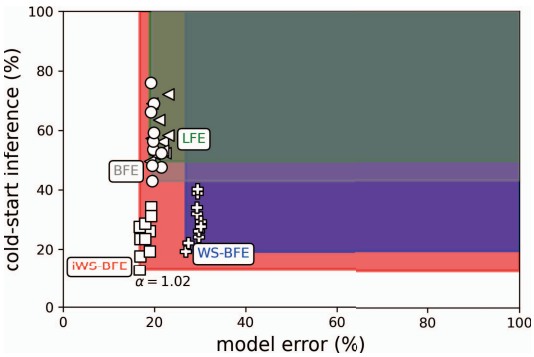

Fig. 7: Bi-objective analysis of the different model selection policies.

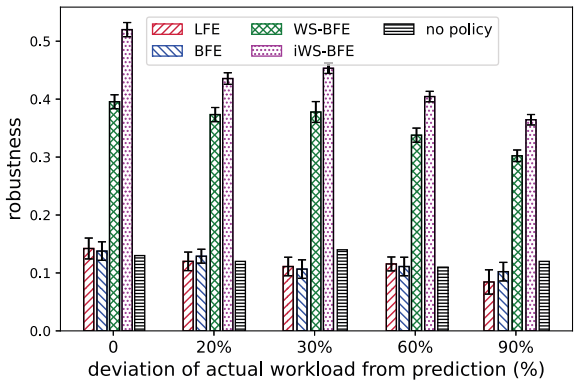

Fig. 8: Robustness of the system against uncertainty in the prediction of inference requests.

E. Bi-Objective Analysis of NN Model Eviction Policies

Recall that the NN model management for multi-tenant applications in a resource-limited edge system is a bi-objective optimization problem that aims at minimizing the number of cold-start inferences and maximizing the inference accuracy. However, these two are generally conflicting objectives and there is not a single optimal solution that can satisfy both objectives. Instead, there could be a range of solutions that dominate other solutions. To analyze which one of the studied policies dominate others, in Figure 7, we plot the percentage of cold-start inferences versus the model error (defined as 100-accuracy) for different policies and Δ values. Let D and σ be the mean and standard deviation of residuals of predicted versus actual request times. Then, $\Delta = D \pm \alpha \cdot \sigma$ ranges by changing the value of $0 \leq \alpha \leq 2$. The deviation of actual versus predicted workload in this experiment is 30%.

For each policy, the colored area shows the cold-start inferences and model error rate that are dominated by that policy. An ideal policy should approach the graph origin (i.e., resulting in zero cold-start and zero model error). In Figure 7, we observe that Edge-MultiAI dominates other policies and form the Pareto-front, particularly with $\alpha = 1.02$. We can conclude that the iWS-BFE policy can significantly improve the usability of the systems via causing fewer cold-start inferences and offering a higher inference accuracy.

F. Analyzing Robustness against Uncertainties

The goal of this experiment is to study how the eviction policies of Edge-MultiAI make the IoT-based system robust against the uncertainty exists between the predicted and actual application request predictor. We define the *robustness metric*, shown in Equation 4, to encompass the ratio of warm-start inferences (denoted ϖ_i) to the total number of requests (denoted γ_i), and the mean prediction accuracy (ψ_i) of each application i throughout the simulation period.

$$R = \frac{1}{n} \cdot \sum_{i=1}^{n} \left[\frac{\varpi_i}{\gamma_i} \cdot \psi_i \right] \tag{4}$$

Figure 8 represents the robustness score achieved by adopting the proposed policies and no policy (a.k.a. baseline) against uncertainties in the inference request prediction. We observe that deploying Edge-MultiAI with any policy provides more robustness than the circumstance where Edge-MultiAI is not in place (no policy). We also notice that the robustness value consistently drops because the rate of inference failure and cold-starts rise for higher deviations. We observe that WS-BFE and iWS-BFE are more robust against deviation than the LFE and BFE. This is because, LFE and BFE do not replace their NN models with a lower-precision one upon eviction, which leads to cold-start inferences for the applications.

G. Evaluating the Fairness of NN Model Eviction Policies

In this experiment, our goal is to examine whether the achievements of Edge-MultiAI and its policies, explored in the previous experiments, is fairly distributed across all applications, or some applications benefit more than the others. To that end, we analyze the distribution of cold-start inference and accuracy across different DL applications. The name and the NN model characteristics of the examined DL applications are listed in Table II. Figures 9 and 10, respectively, express the percentage of cold-start inferences and inference accuracy for each application upon using various NN model eviction policies. It is noteworthy that in Figure 9, "no policy" indicates the situation where Edge-MultiAI is not in place, and in Figure 10, "maximum" serve as the benchmark, by showing the use of highest-precision NN model for each application. While Figure 9 shows that WS-BFE and iWS-BFE remarkably outperform the other policies across all the applications, Figure 10 illustrates that, particularly for iWS-BFE, the outperformance does not come with the cost of lower inference accuracy for the applications. More importantly, in both figures, we observe that, for each policy, the percentage of cold-start inferences and accuracy do not fluctuate significantly from one application to the other. This shows that policies are not biased to any particular DL application. Specifically, the rate of cold-start inferences and the accuracy are fairly distributed across different applications.

V. Conclusion and Future Work

Continuous execution of latency-sensitive DL applications is a pressing need of the memory-limited edge systems. The

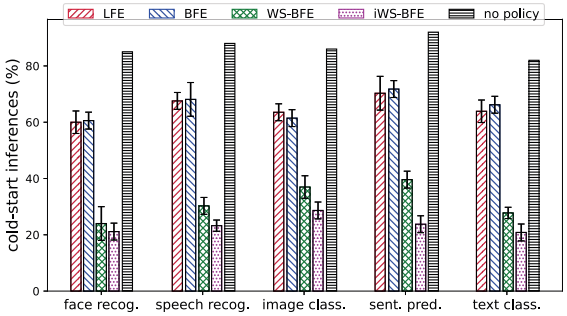

Fig. 9: The percentage of cold-start inferences using different NN model eviction policies versus no policy.

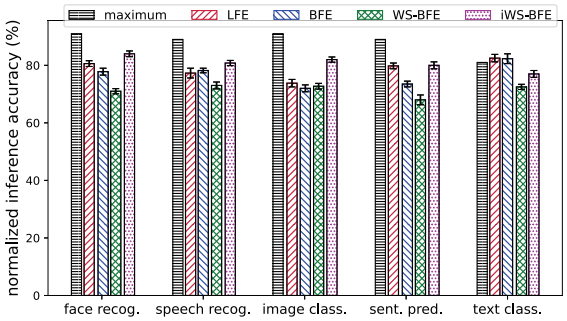

Fig. 10: The inference accuracy obtained from the different policies. The "maximum" is the benchmark, showing the accuracy of the highest-precision model for each application.

research aims to stimulate the degree of multi-tenancy of such applications without compromising their latency and accuracy objectives. We developed a framework, called Edge-MultiAI, to facilitate multi-tenancy of DL applications via enabling swapping only their NN models. The framework was also equipped with model management policies, particularly iWS-BFE, to choose suitable models for eviction and loading to edge memory, such that the percentage of warm-start inferences is maximized without any major loss in the inference accuracy of the applications. Evaluation results indicate that Edge-MultiAI can improve the degree of multi-tenancy by 2×, and iWS-BFE can increase warm-start inferences by 60%. They also show how different policies are robust against uncertainty in the inference request predictions. Last but not the least, the experiments show that the policies are not biased to a certain application in their decisions. There are several avenues to improve Edge-MultiAI. One interesting avenue is to partition the models across edge-cloud continuum to reduce their memory footprint on the edge, hence, further improving the degree of multi-tenancy. Another avenue is to use reinforcement learning to enhance the performance of the policies.

VI. Acknowledgement

This research is supported by the National Science Foundation (NSF) under awards# CNS-2007209 and CNS-2117785.

References

[1] A. Mokhtari, M. A. Hossen, P. Jamshidi, and M. A. Salehi, "FELARE: fair scheduling of machine learning applications on heterogeneous edge systems," in *Proceedings of International Conference On Cloud Computing*, July 2022.

[2] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *Journal of IEEE Internet of Things*, vol. 7, no. 8, pp. 7457–7469, 2020.

[3] S. Zobaed, M. Salehi, and R. Buyya, "SAED: Edge-Based Intelligence for Privacy-Preserving Enterprise Search on the Cloud," in *proceedings of 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 366–375.

[4] R. F. Hussain, A. Pakravan, and M. A. Salehi, "Analyzing the performance of smart industry 4.0 applications on cloud computing systems," in *proceeginds of 22nd IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2020, pp. 11–18.

[5] Q.-V. Dang and C.-L. Ignat, "dtrust: a simple deep learning approach for social recommendation," in *proceedings of 3rd International Conference on Collaboration and Internet Computing (CIC)*, 2017, pp. 209–218.

[6] "Jetson nano developer kit," http://developer.nvidia.com/embedded/jetson-nano-developer-kit, accessed July,2022.

[7] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Journal of Low-Power Computer Vision*. Chapman and Hall/CRC, pp. 291–326.

[8] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[9] D. G. Samani and M. A. Salehi, "Exploring the impact of virtualization on the usability of deep learning applications," in *Proceedings of 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 442–451.

[10] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold start influencing factors in function as a service," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC)*, 2018, pp. 181–188.

[11] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[12] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *Journal of ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Journal of the IEEE Access*, vol. 107, no. 8, pp. 1738–1762, 2019.

[14] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[15] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1396–1401.

[16] K. Ko, Y. Son, S. Kim, and Y. Lee, "Disco: A distributed and concurrent offloading framework for mobile edge cloud computing," in *Proceedings of 9th international conference on ubiquitous and future networks (ICUFN)*, 2017, pp. 763–766.

[17] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3709–3716, 2018.

[18] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," *arXiv preprint arXiv:1706.09829*, 2017.

[19] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017, pp. 1–14.

[20] Y. Ma, D. Xiang, S. Zheng, D. Tian, and X. Liu, "Moving deep learning into web browser: How far can we go?" in *Proceedings of the 19th World Wide Web Conference (WWW)*, May 2019, pp. 1234–1244.

[21] X. Pang, Y. Zhou, P. Wang, W. Lin, and V. Chang, "An innovative neural network approach for stock market prediction," *Journal of Supercomputing*, vol. 76, no. 3, pp. 2098–2118, 2020.

[22] X. Chen, Y. Wu, and Y. Han, "Fepim: Contention-free in-memory computing based on ferroelectric field-effect transistors," in *Proceedings of 26th IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 114–119.

[23] T. V. Christensen, "Heuristic algorithms for np-complete problems," *Project report, Institute of Informatics and mathematical Modelling, Technical University of Denmark*, 2007.

[24] E. J. O'neil, P. E. O'neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *Journal of Acm Sigmod Record*, vol. 22, no. 2, pp. 297–306, 1993.

[25] A. Mokhtari, C. Denninnart, and M. A. Salehi, "Autonomous task dropping mechanism to achieve robustness in heterogeneous computing systems," in *Proceedings of International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 17–26.

[26] T. Van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.