CONTINUER: Maintaining Distributed DNN Services During Edge Failures

Ayesha Abdul Majeed*, Peter Kilpatrick*, Ivor Spence*, and Blesson Varghese*†

*School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, UK
E-mail: {aabdulmajeed01, p.kilpatrick, i.spence}@qub.ac.uk
†School of Computer Science, University of St Andrews, UK
E-mail: bv6@st-andrews.ac.uk

Abstract—**Partitioning and deploying Deep Neural Networks (DNNs) across edge nodes may be used to meet performance objectives of applications. However, the failure of a single node may result in cascading failures that will adversely impact the delivery of the service and will result in failure to meet specific objectives. The impact of these failures needs to be minimised at runtime. Three techniques are explored in this paper, namely repartitioning, early-exit and skip-connection. When an edge node fails, the repartitioning technique will repartition and redeploy the DNN thus avoiding the failed nodes. The early-exit technique makes provision for a request to exit (early) before the failed node. The skip connection technique dynamically routes the request by skipping the failed nodes. This paper will leverage trade-offs in accuracy, end-to-end latency and downtime for selecting the best technique given user-defined objectives (accuracy, latency and downtime thresholds) when an edge node fails. To this end, CONTINUER is developed. Two key activities of the framework are estimating the accuracy and latency when using the techniques for distributed DNNs and selecting the best technique. It is demonstrated on a lab-based experimental testbed that CONTINUER estimates accuracy and latency when using the techniques with no more than an average error of 0.28% and 13.06%, respectively and selects the suitable technique with a low overhead of no more than 16.82 milliseconds and an accuracy of up to 99.86%.**

Index Terms—**Distributed DNNs, Edge computing, Failures**

I. INTRODUCTION

Distributed Deep Neural Network (DNN) services that underpin many modern applications are known to benefit from edge computing [1], [2]. The responsiveness of applications can be improved because edge computing offers compute resources to process data closer to the source where it is generated. Pretrained DNN models are adopted by real-time image recognition applications such as virtual reality gaming or autonomous vehicles, where devices continuously transmit the data to the cloud for inference. To achieve distributed DNN inference, DNNs are partitioned by the granularity of neural network layers. As a result, DNN layers that demand a substantial amount of processing power can be offloaded to the cloud or to edge servers.

The failure of one resource, such as a compute node in an edge computing environment, can result in cascading failures. This adversely impacts the DNN service and may violate application requirements, such as latency thresholds [3]. Traditionally, techniques, such as checkpointing, container migration, and server replication have been adopted to reduce the

impact of failures [4], [5]. However, they are not best suited for edge environments in which resources may be compute/storage limited and are highly decentralised.

One technique to minimise the impact of failures specific to distributed DNN services is repartitioning [6]. A monolithic DNN can be partitioned and its layers are distributed across multiple resources to meet performance objectives of the application [7]. If a node fails, then the DNN can be repartitioned such that a new distribution of the DNN layers is generated so that the layers originally mapped on to the failed edge node can be executed elsewhere. While DNN repartitioning will maintain the original accuracy, it is expensive in that it has both computational and communication overheads making it unsuitable as a general technique for mitigating failures in any edge environment.

Alternate techniques, namely early-exit [8] and skip connection [9] that exploits certain characteristics of the DNN to maintain the service of a distributed DNN when an edge node fails are explored in this paper. While these techniques have been adopted in the literature to meet runtime requirements of applications (considered further in Section VI), but have been minimally considered in the context of edge failures. In the early-exit technique, a classification request can be terminated early before it reaches a failed edge node. While early-exit reduces the inference latency, it impacts the accuracy of the DNN. In the skip connection technique, the route of a classification request can be dynamically adapted to skip failed nodes. However, adopting skip connection reduces the latency but requires extra resources and increases bandwidth use.

This paper develops a novel framework, namely CONTINUER that maintains the service of distributed DNNs when edge outages occur. Failure recovery strategies for edge services have been relatively less studied compared to failure detection [10], [11]. Therefore, CONTINUER focuses on service recovery from failures by presenting a mechanism to recover from a service failure. CONTINUER selects one of the three techniques (repartitioning, early-exit and skip connection) by accounting for: (i) trade-offs in accuracy and performance characteristics (such as latency and downtime[1] incurred) for each technique, and (ii) user-defined objectives,

---

[1]Downtime in this paper refers to the time taken to recover from edge failure.

such as accuracy, latency and downtime thresholds.

CONTINUER operates in two phases. The first is the profiler phase in which the accuracy and latency of a DNN can be estimated for the three techniques with low overhead to respond quickly to failures. The second is the runtime phase in which a scheduler selects the best technique for an edge failure given the estimated accuracy and latency of a DNN for the three techniques and user-defined objectives. Using a lab-based testbed comprising two processor platforms and two production DNNs, namely ResNet-32 and MobileNetV2 it is demonstrated that CONTINUER (i) estimates the accuracy and latency of DNNs for the three techniques with an average percentage error of no more than 0.28% and 13.06%, respectively, and (ii) selects a suitable technique when an edge node fails with a low overhead of no more than 16.82 milliseconds and high accuracy of up to 99.86%.

The remainder of this paper is organised as follows: Section II presents the techniques used for maintaining the service of a DNN when there is an edge failure. Section III presents the CONTINUER framework, including the design considerations and the operating phases. Section IV presents both the profiler and runtime phases of CONTINUER. Section V presents the experimental results obtained from a lab-based testbed to confirm the feasibility of CONTINUER. Section VI presents the related work. Section VII concludes the paper by considering the limitations of the current work and future directions of this research.

II. BACKGROUND

This section presents the context of our work together with two DNNs that are used as case studies.

A. Edge Failures

Service outages at the edge can occur as a result of intermittent network connectivity, and link or node failure [12]. To ensure the continued availability of services, mechanisms such as check-pointing [5], replication [4] and rescheduling of services [11] have been used. Since a distributed DNN is deployed over different edge nodes to provide collaborative inference [13], the failure of a single node may have a cascading effect and cause the failure of other nodes.

When an edge node fails, DNN applications deployed on the edge will need to be redeployed. Redeploying the failed services by repartitioning the DNN incurs a downtime making it unsuitable for latency-critical applications. Alternative approaches to ensure the resilience of a DNN service in the face of edge outages exploit certain characteristics of the DNN. For example, dynamically adapting to routes within the DNN such that failed nodes are skipped (referred to as skip-connection) or terminating requests before reaching a failed node.

B. Underpinning Techniques

The techniques used in CONTINUER to ensure continued availability of services are repartitioning, early-exit and skip connections. While the repartitioning technique has been employed in the literature to adapt a distributed DNN for performance efficiency (layers of the DNN are distributed across multiple resources, such as the edge and the cloud), it has not been employed in the context of edge node failures. Similarly, early-exit and skip connection are two techniques that are employed in the literature for reducing delays in inference to effectively use limited compute resources that may be available [14]. For the first time, these techniques will be utilised to ensure that a distributed DNN can carry on providing services when an edge outage occurs.

1) Technique 1 – Repartitioning: A DNN model comprising a sequence of layers can be partitioned and distributed over the edge and the cloud to meet privacy and performance objectives (such as end-to-end inference latency) of an application. Inference requests generated from a device may be processed by the first partition (the initial sequence of layers of the DNN) on the edge. The intermediate results are transferred via the network to the second partition (the remaining sequence of layers of the DNN) on the cloud. The performance of the partition executing on the edge may vary due to the system load (such as CPU or memory utilization) and the overall performance can be affected due to an unstable network between the edge and the cloud. The system must adapt to these changes that occur at runtime to maintain the required performance of the DNN application. This is achieved by repartitioning (finding a different layer at which the DNN should be partitioned) and redeploying the partitions on the edge and the cloud [1], [15]. In the repartitioning technique, another edge-cloud pipeline can be employed to deploy the distributed DNN to reduce the service downtime on the edge [6].

2) Technique 2 – Early-exit: The second technique is early-exit, which as the name implies makes provision for an inference request to exit (early) before the last layer of the DNN. This allows for accelerating inference albeit there may be accuracy losses. To this end, the base DNN architecture is required to be modified and intermediate classifiers are added to the layers after which inference requests can exit. The classifiers added to the base model allow an input sample to be classified in the intermediate layers [8], [13].

Production DNNs comprising many layers will be large and have substantial computational requirements. This makes it challenging to deploy such DNNs on the edge where there may be resource limitations, and if deployed the end-to-end latency will increase. Dynamic DNNs that make use of early-exit can reduce the end-to-end latency and can be adapted to suit the computational resources available during inference and the input characteristics provided to the DNN [14].

3) Technique 3 – Skip connection: This technique facilitates the skipping of one or more layers of the DNN model.

A skip connection is achieved within a DNN by using gating networks that are inserted between layers in the DNN model. The gating networks map the output of a previous layer to a binary decision whether to enable or skip the next layer [16]. Skip connections were designed to accelerate the training speed and improve the accuracy of large DNNs [9]. Skip connections have been used to provide input-aware dynamic

inference and reduce the computational cost of using large DNNs [17].

C. DNN Selection

Two DNNs, namely ResNet-32 and MobileNetV2 are selected for investigation in the CONTINUER framework.

ResNet-32: The Residual Network (ResNet) [9] is designed with skip connections to speed up the training process and to achieve a high accuracy. The ResNet model consists of residual blocks, made up of two or more convolutional layers, and skip-connections, which allow direct paths between any two residual blocks. A residual block is defined as

$$y = F(x, \{Wi\}) + x \tag{1}$$

where x denotes the input and y denotes the output vector, $F(x, Wi)$ is the function for the residual mapping to be learned and $F + x$ is the operation performed by a shortcut connection and element-wise addition.

Residual blocks have been found not to have a strong relationship with each other [18] and it is also noted that the classification errors increase when more residual blocks are skipped from the model during inference.

MobileNetV2: This is a DNN model for compact, low-latency, low-power mobile devices [19]. The architecture consists of 17 residual blocks followed by a 1×1 convolution, a global average pooling layer, and a classification layer.

D. Suitability of Underpinning Techniques for Edge Failures

ResNet and MobileNetV2 are selected as the base DNN network because of their architectural design, which includes pre-defined skip connections. Assume a DNN is distributed over edge nodes $N = \{n_1, n_2, n_3, n_4, n_5\}$ in a network. If a service outage occurs at n_3, then the path to n_4 and n_5 is disconnected, thereby preventing any further inference requests from being processed.

To ensure resilience when nodes fail (n_3 in the example), the DNN can be redeployed over the first two nodes (n_1 and n_2). For this the first technique presented above, namely repartitioning can be employed. When the DNN is repartitioned, a new partitioning layer at which the DNN can be partitioned and deployed on n_1 and n_2 will be identified. This technique will achieve the same accuracy as the original DNN, but may incur a downtime for repartitioning [6].

Alternatively, the failed node (n_3) may either be bypassed by using the skip connection technique or the inference request can be terminated at n_2 by using the early-exit technique. Skip connection will address the edge failure problem by bypassing the node that failed. The early-exit technique addresses the edge failure problem by terminating incoming requests before the failed node. The skip connection technique may have a higher accuracy than early-exit, however, the relative end-to-end latency will be higher.

III. THE CONTINUER FRAMEWORK

In this section, the design of CONTINUER is presented. The design considerations are: it should be agnostic to the
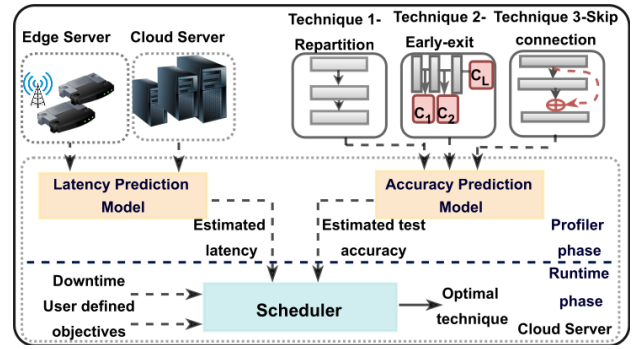

Fig. 1: Overview of the CONTINUER framework

infrastructure; decision making should be rapid with low overhead; and user-defined objectives should be accounted for.

A. Assumption

A DNN is represented as a directed acyclic graph with a set of layers $L = \{l_1, l_2, l_3, ...\}$ and a group of layers is termed as a block. The set of blocks is denoted by $B = \{b_1, b_2, b_3, ...\}$. An edge computing system is assumed that consists of a set of nodes $N = \{n_1, n_2, n_3, ...\}$ over which a DNN model is distributed. It is also assumed that each block is placed on a different node as shown in Figure 3.

B. Technique Selection

Figure 1 illustrates the proposed CONTINUER framework which operates in two phases.

Profiler phase: The profiler phase collects the values for the metrics on which CONTINUER relies and operates in offline mode. These metrics are the accuracy and the end-to-end latency of the DNN model and the downtime incurred when selecting a suitable technique when a node fails. There are two components within the profiler phase, namely the Accuracy Prediction Model and the Latency Prediction Model. In the profiler phase, the accuracy and latency of the DNN model are profiled for training the accuracy and latency prediction models. The Accuracy Prediction Model estimates the accuracy of the DNN at runtime if any of the three techniques (partitioning, early-exit and skip connection) were to be selected given the pretrained weights of the DNN can be provided as input.

The purpose of the Latency Prediction Model is to estimate the layer latency of the techniques given the layer hyperparameters of the DNN. It is not feasible to measure accuracy and end-to-end latency at runtime using a profiling technique due to timing constraints and the speed required in selecting a technique to mitigate the impact of node failure and maintain the service of a DNN. Therefore, prediction models are employed that estimate accuracy and end-to-end latency during runtime. The profiler phase is discussed in Section IV.

Runtime phase: During the runtime phase the Scheduler determines the suitable technique that needs to be used given a node failure. The Scheduler takes as input the estimated accuracy, estimated end-to-end latency, and downtime (empirical) and selects the optimal technique for mitigating the impact of

3

the node failure. During runtime, the value for the downtime metric is calculated as the time taken to predict and retrieve the estimated accuracy and end-to-end latency parameters. The runtime phase is further discussed in Section IV.

IV. THE PROFILER AND RUNTIME PHASES

To determine a suitable technique to maintain the service of a distributed DNN when a node failure occurs, the CONTINUER framework makes use of three metrics, namely accuracy, latency, and downtime associated with each technique.

A. Profiler Phase

Resource-independent and resource-specific values are gathered in the profiler phase. The latency metric is a resource-specific value that depends on the underlying hardware. Accuracy and downtime are resource-independent, depending only on the DNN model and the technique (repartitioning, early-exit and skip connection), respectively. The values of accuracy and latency of the techniques are estimated in the profiler phase, which is carried out offline. Downtime values are however not estimated, instead empirical values of downtime of each technique are used.

In this section, first the partition points defined within the repartitioning technique and the modification of the DNN models for defining early exit points and skip connections are presented. Then the data collection approach and the prediction models used for estimating the latency and accuracy of each technique for different exit points and skip connections that can be used when a node fails are considered. Finally, the empirical values obtained for downtime metric are discussed.

1) Repartitioning: In the repartitioning technique, the DNN is partitioned after each residual block represented by grey bars in Figure 2. The partitioned points are defined under the assumption that each residual block of DNN is placed on different nodes. The architecture of ResNet-32 consists of an initial convolutional layer, batch normalisation and activation layers, 15 residual blocks, followed by a global average pool layer and dense layer. The architecture of MobileNetV2 consists of 17 residual blocks, followed by a 1×1 convolution, a global average pooling layer, and a dense layer. ResNet-32 can be distributed on up to fourteen nodes (Figure 2a) and MobileNetV2 on up to eleven nodes (Figure 2b).

Model training parameters for Repartitioning: For repartitioning, the ResNet-32 model is trained with a learning rate of $1e-4$ and MobileNetV2 is trained with a learning rate of $1e-3$. Both models are trained using loss function categorical cross-entropy on the CIFAR-10 dataset[2] that contains 50000 training and 10000 test images of resolution 32×32 consisting of 10 classes, with a batch size of 64 and epoch size of 500. The epoch size is set to 500 to generate a dataset of 500 instances for the accuracy prediction model for predicting accuracy through pretrained weights. The accuracy obtained for ResNet-32 is 82.52% and for MobileNetV2 85.54%.

2) Early-exit: CONTINUER adds exit points on ResNet-32 and MobileNetV2 under the assumption that each block of layers is placed across different nodes in the edge. Given that ResNet-32 can be distributed across 13 nodes, up to 13 different exit points can be added; one after each node $(n_1 - n_{13})$ as shown in Figure 3a. The green bars represent individual layers, whereas the grey bars represent blocks distributed on thirteen nodes for ResNet-32. For MobileNetV2 there are 10 different positions where exit points can be added which are placed on up to 10 nodes $(n_1 - n_{10})$ (Figure 3b).

In ResNet-32, an exit point is defined after each residual block. Each exit point comprises a convolutional layer with $filter\ size = 32, kernel\_size = 3, strides = 2$, followed by a classifier that has a max pool layer, batch normalisation layer, and two dense layers of $units = 64$, and $units = 10$ respectively. The early exit points have the aforementioned layers so that prediction accuracy can be improved. The convolution layers specified at the exits points are fine-tuned based on experience to extract coarse level features of an input images that will be used by the classifers at the exit points.

For MobileNetV2, the exit points are defined after the residual blocks represented as grey bars and numbered 2, 4, 5, 7, 8, 9, 11, 12, 14, and 15 in Figure 3. The structure of the exit point defined for MobileNetV2 residual Block 2, includes a batch normalisation layer and then a convolutional layer with $filter\ size = 96, kernel\_size = 3, strides = 1$, followed by the classifier that has a global max pool layer and two dense layers of $units = 64$ and $units = 10$. A batch normalisation layer, followed by two convolutional layers with filter sizes of 160 and 80 are defined for residual blocks 4 and 5, followed by a classifier that has a global max pool layer and two dense layers of $units = 64$ and $units = 10$. For residual blocks 7, 8, 9, 11, and 12, a batch normalisation layer is defined followed by a convolutional layer with a filter size of 320, and a classifier that has a global maxing pool layer, and two dense layers of $units = 64$ and $units = 10$. For blocks 14, and 15, a batch normalisation layer, followed by a convolutional layer with $filter\ size = 160, kernel\_size = 3, stride = 1$ is defined, followed by a classifier layer that has a global max pool layer and two dense layers of $units = 64$ and $units = 10$. A convolutional layer with a stride of $size = 1$ and a global max-pooling layer are employed based on experience and trial-and-error to improve the prediction performance of MobileNetV2. The batch normalisation layer is utilised to improve the training performance.

Model training parameters for Early-exit: For training the DNN models that can make use of the early exit technique, a cross entropy loss function L_i is employed for each of the early exit points $i = 1, ..., N$, and a total loss function L_T is generated that is the weighted sum of these loss functions. The ResNet-32 model along with the intermediate exit points that are added is trained with a learning rate of $1e-3$ and for MobileNetV2, the model with exit points defined is with a learning rate of $1e-4$. Both models are trained with 500 epochs and a batch size of 64, on the CIFAR-10 dataset.

Figure 4 shows the accuracy of ResNet-32 and Mo-

(a) Partition points defined for ResNet-32
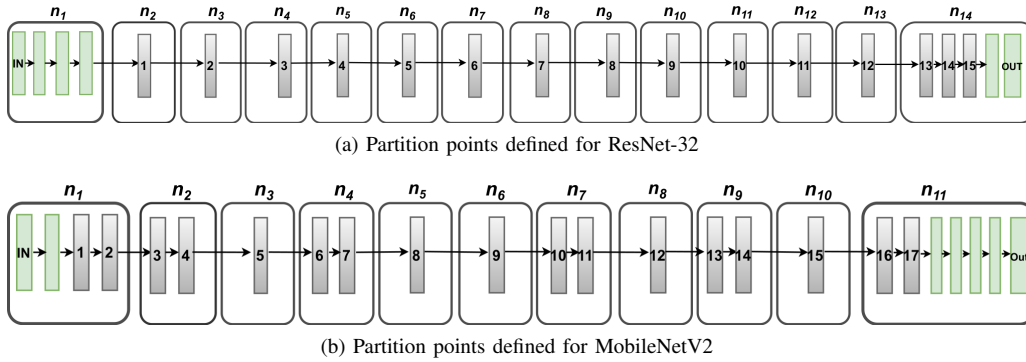

(b) Partition points defined for MobileNetV2

Fig. 2: Distribution of partition points of ResNet-32 and MobileNetV2 when layers and block of layers are distributed across different compute nodes.

(a) Early exit points ($E_1 - E_{13}$) defined for ResNet-32

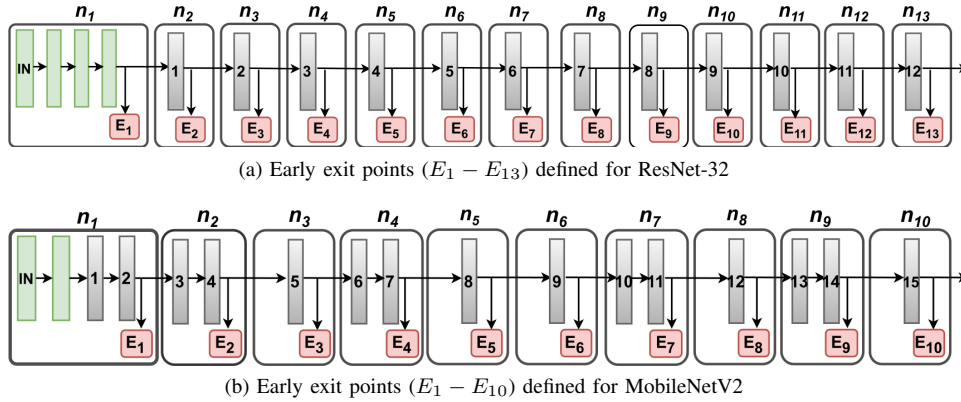(b) Early exit points ($E_1 - E_{10}$) defined for MobileNetV2

Fig. 3: Distribution of exit points when layers and block of layers are distributed across different compute nodes. Red blocks represent the new exit points defined for the DNN models in CONTINUER.
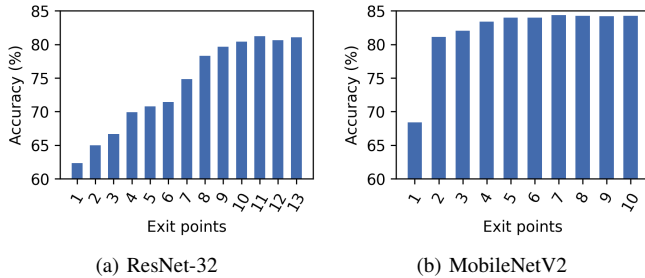

(a) ResNet-32   (b) MobileNetV2

Fig. 4: Accuracy of early exit points defined in ResNet-32 and MobileNetV2.

bileNetV2 for the different exit points. The x-axis shows the exit points and the y-axis shows the accuracy of the DNN models. As shown in the Figure 4a, an accuracy lower than 70% is noted for the initial early exit points (E_1 to E_4) ranging from 62.33% to 69.92%. Similarly, for MobileNetV2, E_1 has an accuracy of 68.39%, this is to be expected. However, there is a trade-off between accuracy and the end-to-end latency of the DNN models when using different exit points, which will be presented in Section V.

3) Skip connection: CONTINUER uses the default skip connections available in ResNet-32 and MobileNetV2. Figure 5 shows the skipping policy for skip connections for

ResNet-32 and MobileNetV2. The grey bar represents a block of layers and the green bar represents a single layer. The red dotted line represents the skip connections added to the base model resulting in a total of 10 skip connections for ResNet-32 and 9 skip connections for MobileNetV2.

Model training parameters for Skip connection: The learning rate is set to $1e - 4$ for ResNet-32 and $1e - 3$ for MobileNetV2. Both models are trained with a batch size of 64 and epoch size 500 using the categorical cross entropy as loss function. Residual blocks which have layers in the path of the skip connection are ignored and these blocks are represented by a red star as shown in Figure 6. Figure 6 shows the accuracy of ResNet-32 and MobileNetV2 for the skip connections defined at different positions. The x-axis shows the number of skip connections whereas the y-axis shows the accuracy. For ResNet-32, highest accuracy of 84.98% is obtained for skip connection 12 whereas for MobileNetV2 an accuracy of 86.91% is obtained for skip connection 8. The accuracy obtained for the skip connections defined in the ResNet-32 and MobileNetV2 indicates that skipping layers at runtime has a low impact on prediction accuracy. The settings for training hyperparameters defined for early-exit and skip connections are determined by trial and error.

The above three techniques can be extended to other DNN models. For DNN models that do not have default skip
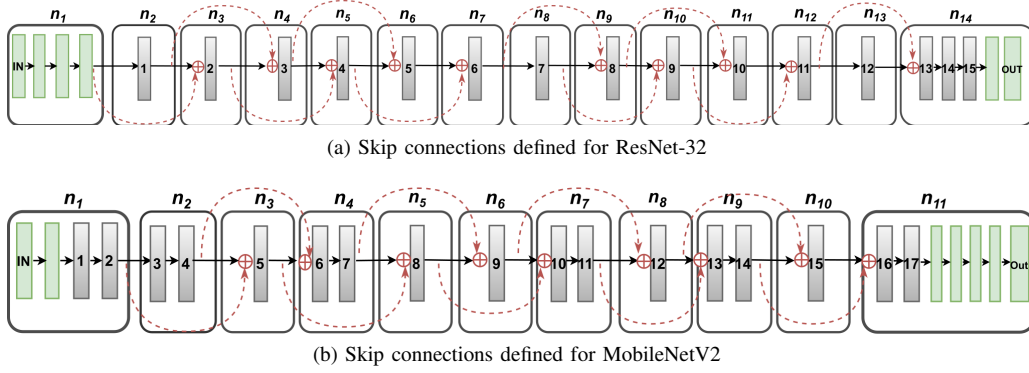
(a) Skip connections defined for ResNet-32

(b) Skip connections defined for MobileNetV2

Fig. 5: Skip connections for ResNet and MobileNetV2 models. The pink dotted lines are the skip connections added to the base model in CONTINUER.
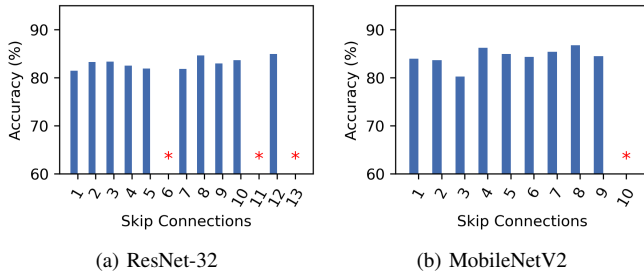

(a) ResNet-32

(b) MobileNetV2

Fig. 6: Accuracy of skip connections of ResNet and MobileNetV2 when using different skip connections. The red star indicates that skip connections are not possible at these points.

TABLE I: Layer hyperparameters of the DNN model.

Layer Type	Hyperparameters
Batch Normalisation	input shape, input channel
Convolution	input shape, input channel, kernel size, stride, filter
ReLu	input shape, input channel
Dense	input shape, input channel
Add	input shape, input channel
Dropout	input shape, input channel
Depthwise Convolution	input shape, input channel, kernel size, stride
Global Pool Average	input shape, input channel

TABLE II: Accuracy of predicting the latency of different layer types

Layer Type	MSE	R^2
Batch Normalistaion	0.045	0.957
Convolution	0.040	0.980
ReLu	0.021	0.993
Dense	0.021	0.854
Add	0.009	0.995
Dropout	0.044	0.941
Depthwise Convolution	0.008	0.994
Global Pool Average	0.023	0.992

connections, skip connections need to be defined at specific locations in the DNN model and model retraining is required and similarly for early-exit technique.

B. Metrics

The objective of the CONTINUER framework is to select the most suitable technique when a node fails while considering metrics such as end-to-end inference latency, accuracy and downtime. This section presents the methods adopted for obtaining the values of these metrics.

i) End-to-end Latency: CONTINUER adopts a layer-wise approach that has been adopted in the literature to profile the inference latency of each layer of the DNN [2], [20]. Based on the values of profiled end-to-end latency, a Latency Prediction Model (as shown in Figure 1) is developed for each type of layer by varying the layer's hyperparameters as shown in Table I. The Keras layers API[3] is used for extracting the execution time of each layer type on two different processors that will be further discussed in Section V. The advantage of using a layer-wise approach is that it profiles the inference latency of each type of layer instead of the whole DNN model, thus making it DNN-independent with a low profiling cost.

To predict the end-to-end latency of the DNN, the Latency Prediction Model is trained using XGBoost [21]. The hyperparameters of XGBoost are optimised using the Optuna[4] optimisation library. The following best performing hyperparameters are identified by the optimisation framework for XGBoost $learning\ rate = 0.1, n\_estimators = 1000, max\ depth = 10, colsample\ bytree = 1, minchild\ weight = 1, seed = 123$. The histogram-based algorithm is chosen as the XGBoost tree method. The quality of the predictions of the Latency Prediction Model is evaluated by quantifying the Mean Squared Error (MSE) and Coefficient of determination (denoted as R^2). Table II shows the accuracy of each layer latency prediction models. The R_2 values except that of the dense layer, are close to 1 and a very low MSE is noted. This indicates that the Latency Prediction Model is fit for estimating layer latencies.

ii) Accuracy: Two approaches have been used in the literature to estimate the DNN model accuracy without profiling the input data. The first uses the model training parameters and hyperparameters [22]. The hyperparameters include the characteristics of the DNN architecture, such as number of layers, and the training parameters include learning rate and

---

[3]https://keras.io/api/layers/

[4]https://optuna.org/

loss function. The second approach involves predicting the accuracy of the DNN model by providing the pre-trained weights of the DNN model as input [23]. The training/test data or the meta-data of the DNN architecture may not be available in the production setting or when an edge node fails. Hence, estimating DNN accuracy using the characteristics of the DNN is impractical. The second approach is thus adopted in CONTINUER by the Accuracy Prediction Model.

TABLE III: Parameters for predicting accuracy

Parameters	Description
Activation	Activation function of the DNN model
B_init	Initialisation used for bias weights
DNN_architecture	DNN model (ResNet32, MobileNetV2)
Epochs	Total number of epochs (500)
Learning_rate	Learning rate
num_layers	Number of layers
Optimiser	Optimiser used (Adam optimiser)
Train_fraction	Fraction of the total number of training samples
Train_accuracy	Accuracy on the training set
Train_loss	Loss on the training

Table III shows the parameters used when training the Accuracy Prediction Model for ResNet-32 and MobileNetV2. The parameters are extracted by implementing a custom Keras callback function that is invoked at the end of each epoch during training.

The Accuracy Prediction Model is based on the Light Gradient Boosting Machine (LightGBM) [24] that predicts accuracy by providing the pretrained weights of the DNN model. The settings defined for LightGBM hyperparameters are the following: $learning\ rate = 0.1, n\_estimators = 100, max\ depth = -1, colsample\ bytree = 1.0, minchild\ weight = 0.001$. A split ratio of 80:20 is used to split the data into training and testing data. The pre-trained weights provided to LightGBM are pre-processed by using the mean, variance, and q^{th} percentiles for $q\epsilon\{0, 25, 50, 75, 100\}$ for each layer of the DNN model as presented in the literature [23]. The ResNet-32 model is trained with a learning rate set to $1e-3$ for all the three techniques. For the repartitioning and skip connection techniques, the learning rate is set to $1e-3$ for MobileNetV2 and to $1e-4$ in early-exit. Both models are trained using the loss function categorical cross-entropy on the CIFAR-10 dataset with a batch size of 64 and epoch size of 500. Model training parameters are determined by trial and error. The MSE obtained is 0.223 (low indicates that the estimation has a high accuracy) and R^2 is 98.01% (high indicates that there is a high correlation between the input parameters to the prediction model and the output).

iii) Downtime: The downtime is the time taken to retrieve the estimated accuracy and latency from the Accuracy Prediction Model and Latency Prediction Model, respectively and for the Scheduler (discussed further in Section IV-C to select one of the three techniques, namely repartitioning, early-exit and skip connection when an edge node fails. The repartitioning and skip connection techniques have an additional 0.99ms downtime to reinstate connections [6].

TABLE IV: Processor platforms used in the experimental studies.

Platform	CPU	Clock Freq.	Memory
Platform 1	Intel(R) Core (TM) i7-8700	3.20GHz	16GB
Platform 2	Intel(R) Core (TM) i5-8250U	1.60GHz	16GB

C. Runtime phase

The Scheduler is a key component used in the runtime phase. The Scheduler takes as input the estimated accuracy and the estimated latency of the DNN for all three techniques and the downtime (empirically obtained) that will be incurred by each technique and determines the suitable technique for node failure. The accuracy is obtained from the Accuracy Prediction Model, the end-to-end latency is obtained from the Latency Prediction Model. The goal of the Scheduler is to select the technique that best satisfies any user provided objectives, such as thresholds for end-to-end latency, accuracy and downtime.

To minimise the cost of selecting a suitable technique subject to user-defined objectives, a classic additive weighting method is employed [25]. Different weights are assigned to each objective and then the weighted sum of the normalised values of the user-defined objectives is minimised. If the accuracy, end-to-end latency and downtime objectives are denoted as A, L and D, respectively, then the normalised objectives are denoted as A', L' and D' (normalised to a value between 0 and 1 using the Linear Max-Min technique).

The suitable selection of technique considering user requirement is formulated in Equation 2.

$$min \sum \omega_1 A' - \omega_2 L' - \omega_3 D' \qquad (2)$$

where $\omega_1, \omega_2, \omega_3$ are the dynamic weights of accuracy, end-to-end latency, and downtime. The value of each weight factor, represents the level of importance, and is set by the user. For instance, if the user has not specified a latency threshold, a weight factor of 0 is assigned to latency objective.

V. EXPERIMENTAL STUDIES

This section presents the experimental setup used to assess the CONTINUER framework and the results obtained.

A. Experimental Setup

Experimental studies are carried out on two 64-bit x86 processor platforms as shown in Table IV; similar processors have been used as representative of edge environments in the literature [26], [27]. Since latency is resource dependent, the results of the Latency Prediction Model are obtained from both platforms. Accuracy is not impacted by the platforms. The repartitioning, early-exit, and skip connection techniques are examined in the context of two DNNs in CONTINUER, namely ResNet-32, and MobileNetV2. The DNNs are implemented using the Tensorflow library and the Keras API.

B. Results

The performance of the three key components in the CONTINUER framework, namely the Latency Prediction Model, the Accuracy Prediction Model and the Scheduler will be evaluated in this section. It will be demonstrated that the

TABLE V: Average percentage error when estimating latency using the repartitiong, early-exit and skip connection techniques for ResNet-32 and MobileNetV2.

Technique	Platform 1		Platform 2	
	ResNet-32	MobileNetV2	ResNet-32	MobileNetV2
Repartitioning	3.48%	2.44%	2.33%	0.51%
Early-exit	10.10%	3.22%	13.06%	5.07%
Skip connection	2.92%	2.92%	3.06%	0.73%

latency of the DNNs (resource dependent) can be estimated on both platforms with a high accuracy. Similarly, accuracy (resource independent) of the DNN model can be estimated with a high accuracy without the need for profiling the DNNs when a node fails. The results will also highlight that the Scheduler that is based on estimated data will select a suitable technique given user-defined objectives with a high accuracy.

1) Quality of estimating latency and accuracy metrics: The quality of the results estimated by the Latency Prediction Model and Accuracy Prediction Model is evaluated by comparing the estimated latency and accuracy with the measured latency and accuracy. Measured accuracy is obtained from the trained ResNet-32 and MobileNetV2 on the CIFAR-10 dataset for the three techniques. Measured latency is obtained by profiling the trained ResNet-32 and MobileNetV2 DNNs for the three techniques.

Figure 7 shows the measured and predicted latency on different platforms for ResNet-32 and MobileNetV2. The x-axis shows the node number on which a block of DNN model is deployed and the y-axis shows the latency of each of the techniques (repartitioning, early-exit, and skip connection). For repartitioning the latency is a constant for all nodes since the entire DNN is repartitioned and redeployed once an edge node fails. For early-exit, the inference request will be completed before a failed node through an exit point. Hence, latency for early-exit is the execution time of the request, which increases when the inference request passes through a larger number of nodes. For skip connection, the latency metric is the end-to-end latency of entire DNN deployed on edge nodes excluding the failed node (the node that was bypassed by the skip connection).

For the skip connection, red stars indicate nodes on which a skip connection is not possible. Figure 7 shows the measured and predicted latency of each technique. Table V shows the average percentage error of each technique for ResNet-32 and MobileNetV2. The percentage error is the difference between the measured and actual values of latency and accuracy of the three techniques. It is noted that the average percentage error for the three methods is relatively low; the maximum is 13.06% for the early-exit technique on ResNet-32.

Figure 8 shows the measured and predicted accuracy of ResNet-32 and MobileNetV2. The x-axis shows the node number on which a block of DNN model is deployed, whereas the y-axis shows the measured and estimated percentage accuracy of the DNN. The accuracy of ResNet-32 and MobileNetV2 remains the same on each node for repartitioning. For early-exit the accuracy of ResNet-32 increases for higher node number, whereas for MobileNetV2 the accuracy is 68.39% for

TABLE VI: Average percentage error when estimating accuracy using the repartitiong, early-exit and skip connection techniques for ResNet-32 and MobileNetV2.

Technique	ResNet-32	MobileNetV2
Repartitioning	0%	0.12%
Early-exit	0.03%	0.03%
Skip connection	0.06%	0.28%

TABLE VII: Quality of the selection made by the CONTINUER Scheduler when selecting a suitable technique.

DNN Model	Platform 1	Platform 2
ResNet-32	99.86%	99.86%
MobileNetV2	86.12%	99.83%

the first exit point and a higher accuracy is noted at subsequent exit points. The accuracy of ResNet-32 and MobileNetV2 varies slightly for the skip connection.

Table VI shows the average percentage error of accuracy metric for ResNet-32 and MobileNet-V2 for each technique. The accuracy metric is more precisely estimated than the latency metric. The maximum average percentage error of 0.28% is noted for skip connection.

2) Quality of selection by the Scheduler: To determine the quality of the technique selected by the Scheduler a parameter sweeping analysis is performed. There are three parameters to sweep, these are the weights for the accuracy, latency, and downtime $\omega = \omega_A, \omega_L, \omega_D$. The constraint for all three parameters is defined as $0.1 \geq \omega \leq 0.9$ (in increments of 0.1). Different combinations of weights are applied on each instance (dataset generated using the normalised values for estimated accuracy, estimated latency and downtime for all techniques and for all nodes) for ResNet-32 and MobileNetV2. Min–Max normalisation is applied on the accuracy, latency and downtime metric to ensure the accuracy of the selection of the Scheduler. Then the objective function defined in Equation 2 takes as input the estimated accuracy, estimated latency and downtime of each technique to determine which technique is suitable when an edge node fails. Parameter sweeping is applied on the normalised dataset which results in 767 instances for ResNet-32 and 590 instances for MobileNetV2. The quality of the selection made is based on the classification accuracy metric, which is the total number of correctly identified techniques divided by the total number of instances.

Table VII shows that the appropriate technique is selected by the Scheduler with up to an accuracy of 99.86%. The results demonstrate that the Scheduler is effective in determining the appropriate technique by considering user-defined objectives.

3) Overhead: Table VIII shows the overhead (maximum downtime) incurred for the repartitioning, early-exit and skip connection in milliseconds (ms). Downtime is the sum of the time taken to retrieve the estimated accuracy and end-to-end latency metrics and to select a suitable technique based on the user-defined objectives. CONTINUER selects a suitable technique within 16.82 ms following a node failure.

Summary: The above results highlight that the Accuracy Prediction Model and Latency Prediction Model of CONTINUER estimate the accuracy and latency metric with a relatively low average percentage error. An accuracy of up
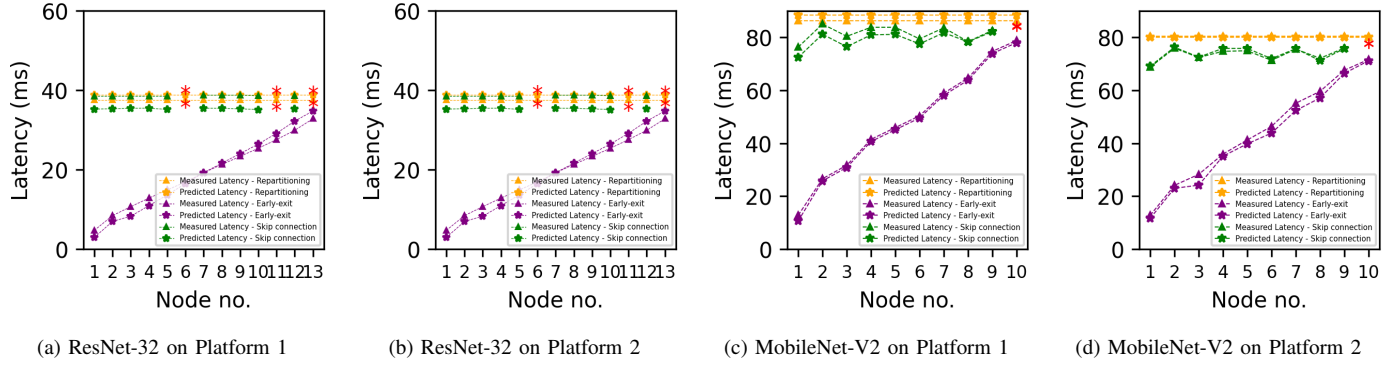
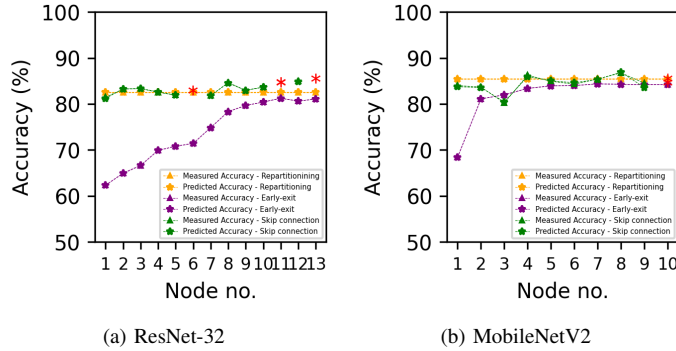Fig. 7: Measured and predicted latency for ResNet-32 and MobileNetV2.

Fig. 8: Measured and predicted accuracy for ResNet-32 and MobileNetV2.

TABLE VIII: Downtime incurred when selecting a technique.

Technique	ResNet-32	MobileNetV2
Repartitioning	3.56ms	16.16ms
Early-exit	1.83ms	9.28ms
Skip connection	3.32ms	16.82ms

to 99.86% is obtained in selecting a suitable technique when an edge failure occurs with a relatively low overhead. These confirm the feasibility of CONTINUER.

VI. RELATED WORK

Two aspects relevant to the research reported in this paper are considered in this section. The first is approaches that address runtime concerns when distributed DNNs are deployed; for example, adapting to variable network speeds and resource availability of compute resources. This is relevant as maintaining services when an edge failure occurs is a runtime concern. The second is consideration of edge failures.

Repartitioning is used to address runtime concerns, such as change in network speed and resource variability, by repartitioning and redeploying DNNs to suit the given operational conditions in NEUKONFIG [6]. Approaches are incorporated to reduce the service downtime when repartitioning.

Early-exit has been used in the context of addressing the runtime concern of varying network speeds and availability for distributed DNNs (e.g, Edgent [28]). Similarly, the latency is reduced in the context of industrial IoT environments using the early-exit technique in Boomerang [29].

Skip connection is used for reducing the inference latency by introducing a layer-wise skipping policy in SkipNet [16]. Supervised and reinforcement learning approaches are used to improve the policy by conditioning it against the input sample. Similar approaches such as BlockDrop [17] skip residual blocks at runtime to reduce inference delay.

Existing research that addresses edge failures of DNN services has considered the early-exit approach. One such example is SEE [30] in which it is assumed that the duration of the failure is known apriori. This is impractical for a dynamically changing environment such as the edge. LEE fills this gap of SEE by knowing the duration of failure [31]. However, SEE and LEE make the decision for individual video frames and uses buffers to hold pending inference requests. The decision made for a current frame may negatively impact future frames. Another example is DeepFogGuard [32] in which skip connection is used to skip failed physical nodes on which a distributed DNN is deployed. However, the approach used requires retraining the DNN which makes it less suitable for responding to runtime changes, such as the failure of an edge node. In addition, the work does not quantify the downtime incurred by skipping connections during failure. Generally, the above research does not account for the resource limited nature of the edge environment in response to a service outage. Further, existing research does not consider the rapid response to an edge failure that may be required for latency-

critical applications relying on DNNs. They do not quantify the overhead that will be incurred and consider user-defined objectives when an edge node fails.

VII. CONCLUSIONS

This paper presents the CONTINUER framework to maintain the service of distributed DNNs when an edge outage occurs. CONTINUER is underpinned by three techniques, namely repartitioning, early-exit and skip connection, one of which is selected by the framework to continue delivering DNN services when an edge node fails by accounting for trade-offs in accuracy, end-to-end latency and downtime and user-defined objectives. Experiments on a lab-based testbed show that CONTINUER selects the appropriate technique with high accuracy and low overhead, thus confirming the feasibility of the framework.

Limitations and Future Work: Despite the challenge addressed by CONTINUER, it has certain limitations. Firstly, for estimating the end-to-end latency of the DNN, the approach for profiling DNN layers is CPU oriented. Future work will include GPU-aware estimation of latency. Secondly, CONTINUER considers ResNet-32 and MobileNetV2, which by default have skip connections defined in them. The research can be expanded to consider other DNNs. Thirdly, it is assumed in this paper that each DNN block is deployed on a different edge node and only a single node fails. Efforts will be made to address these limitation in the future.

REFERENCES

[1] L. Lockhart, P. Harvey, P. Imai, P. Willis, and B. Varghese, "Scission: Performance-Driven and Context-Aware Cloud-Edge Distribution of Deep Neural Networks," in *IEEE/ACM 13th International Conference on Utility and Cloud Computing*, 2020, pp. 257–268.

[2] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence between the Cloud and Mobile Edge," *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 615–629, 2017.

[3] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent Delay-Aware Partial Computing Task Offloading for Multi-User Industrial Internet of Things through Edge Computing," *IEEE Internet of Things Journal*, 2021.

[4] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[5] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container Migration in the Fog: A Performance Evaluation," *Sensors*, vol. 19, p. 1488, 2019.

[6] A. A. Majeed, P. Kilpatrick, I. Spence, and B. Varghese, "NEUKONFIG: Reducing Edge Service Downtime When Repartitioning DNNs," in *IEEE Int. Conf. on Cloud Engg.*, 2021, pp. 118–125.

[7] H. Wang, G. Cai, Z. Huang, and F. Dong, "ADDA: Adaptive Distributed DNN Inference Acceleration in Edge Computing Environment," in *IEEE 25th Int. Conf. on Parallel and Distributed Sys.*, 2019, pp. 438–445.

[8] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast Inference via Early Exiting from Deep Neural Networks," in *IEEE 23rd International Conference on Pattern Recognition*, 2016, pp. 2464–2469.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[10] M. Soualhia, C. Fu, and F. Khomh, "Infrastructure Fault Detection and Prediction in Edge Cloud Environments," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 222–235.

[11] A. Aral and I. Brandic, "Dependency Mining for Service Resilience at the Edge," in *2018 IEEE/ACM Symposium on Edge Computing*, 2018, pp. 228–242.

[12] A. Aral and I. Brandić, "Learning Spatiotemporal Failure Dependencies for Resilient Edge Computing Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2020.

[13] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed Deep Neural Networks over the Cloud, the Edge and End Devices," in *IEEE 37th Intl. Conf. on Distributed Computing Systems*, 2017, pp. 328–339.

[14] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *arXiv preprint arXiv:2103.04505*, 2021.

[15] F. McNamee, S. Dustdar, P. Kilpatrick, W. Shi, I. Spence, and B. Varghese, "The Case for Adaptive Deep Neural Networks in Edge Computing," in *2021 IEEE 14th International Conference on Cloud Computing*, 2021, pp. 43–52.

[16] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 409–424.

[17] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "BlockDrop: Dynamic Inference Paths in Residual Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[18] A. Veit, M. J. Wilber, and S. Belongie, "Residual Networks Behave like Ensembles of Relatively Shallow Networks," *Advances in Neural Information Processing Systems*, vol. 29, pp. 550–558, 2016.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[20] C.-C. Wang, Y.-C. Liao, M.-C. Kao, W.-Y. Liang, and S.-H. Hung, "PerfNet: Platform-Aware Performance Modeling for Deep Neural Networks," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, 2020, pp. 90–95.

[21] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[22] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, "Predicting the Generalization Gap in Deep Networks with Margin Distributions," in *International Conference on Learning Representations*, 2019.

[23] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. Tolstikhin, "Predicting Neural Network Accuracy from Weights," *arXiv preprint arXiv:2002.11448*, 2020.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, vol. 30, pp. 3146–3154, 2017.

[25] K. P. Yoon and C.-L. Hwang, *Multiple Attribute Decision Making: An Introduction.* Sage publications, 1995.

[26] A. Abouaomar, S. Cherkaoui, Z. Mlika, and A. Kobbane, "Resource Provisioning in Edge Computing for Latency Sensitive Applications," *IEEE Internet of Things Journal*, 2021.

[27] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog Computing for the Internet of Things: A Survey," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–41, 2019.

[28] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand Accelerating Deep Neural Network Inference via Edge Computing," *IEEE Transactions on Wireless Comms.*, vol. 19, no. 1, pp. 447–457, 2019.

[29] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand Cooperative Deep Neural Network Inference for Edge Intelligence on the Industrial Internet of Things," *IEEE Network*, vol. 33, no. 5, pp. 96–103, 2019.

[30] Z. Wang, W. Bao, D. Yuan, L. Ge, N. H. Tran, and A. Y. Zomaya, "SEE: Scheduling Early Exit for Mobile DNN Inference during Service Outage," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019, pp. 279–288.

[31] W. Ju, W. Bao, D. Yuan, L. Ge, and B. B. Zhou, "Learning Early Exit for Deep Neural Network Inference on Mobile Devices through Multi-Armed Bandits," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing*, pp. 11–20.

[32] A. Yousefpour, S. Devic, B. Q. Nguyen, A. Kreidieh, A. Liao, A. M. Bayen, and J. P. Jue, "Guardians of the Deep Fog: Failure-Resilient DNN Inference from Edge to Cloud," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, 2019, pp. 25–31.