E3: Early Exiting with Explainable AI for Real-Time and Accurate DNN Inference in Edge-Cloud Systems

Changyao Lin
lincy@stu.hit.edu.cn
Harbin Institute of Technology
Harbin, China

Zhenming Chen
chenzm@cscec.com
China Construction Steel Structure Engineering Co., LTD
Shenzhen, China

Ziyang Zhang
ziyang.zhang@polimi.it
Politecnico di Milano
Milano, Italy

Jie Liu
jieliu@hit.edu.cn
Harbin Institute of Technology
Shenzhen, China

ABSTRACT

Edge intelligence applications frequently generate deep learning inference tasks with varying Service Level Objectives (SLO, such as accuracy and real-time requirements). For such tasks, recent progressive inference modes support early exit from different branches to satisfy inference requirements. However, existing edge-cloud progressive neural architectures cannot simultaneously achieve high accuracy and real-time performance for different data features. Therefore, we utilize explainable AI technique to construct and train a novel progressive neural architecture E3. E3 can progressively extract the most important features for inference, ensuring higher accuracy at early-exit points. While the less important features in the later stage are highly compressible, thereby reducing edge-cloud transmission overhead. Furthermore, E3 cooperates with online execution control to launch tasks and decide the exit point for each task, ensuring resource utilization and real-time performance, and adapting to bandwidths and deadlines. Experimental results on various edge-cloud platforms, datasets, and reference models demonstrate that E3 is more lightweight, efficient, energy-saving, and incurs almost no additional runtime overhead compared to traditional architectures. Under stringent deadlines, the average accuracy of tasks increases by $> 50\%$, and the deadline satisfaction rate approaches 100%.

CCS CONCEPTS

• **Computer systems organization** → **Real-time system architecture**; • **Computing methodologies** → **Neural networks**; **Computer vision tasks**.

KEYWORDS

Edge Intelligence, Progressive Inference, Explainable AI

SenSys '25, May 6–9, 2025, Irvine, CA, USA

ACM ISBN 979-8-4007-1479-5/2025/05
https://doi.org/10.1145/3715014.3722076

ACM Reference Format:
Changyao Lin, Zhenming Chen, Ziyang Zhang, and Jie Liu. 2025. E3: Early Exiting with Explainable AI for Real-Time and Accurate DNN Inference in Edge-Cloud Systems. In *The 23rd ACM Conference on Embedded Networked Sensor Systems (SenSys '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3715014.3722076

1 INTRODUCTION

With the development of edge computing [40] and hardware [28, 30, 31], an increasing number of intelligent applications, such as autonomous driving [44], virtual reality [45], and smart manufacturing [55], are being deployed at the edge to provide users with more real-time services [57]. These applications generate different deep learning inference tasks (data), which usually vary in terms of difficulty, resource overhead, and service level objective (SLO, such as real-time and accuracy requirements) [52].

Existing edge-cloud collaborative progressive inference architectures [23, 24, 54] can adjust the exit point of inference according to different SLOs, striving to satisfy various accuracy and latency requirements. However, their accuracy at early-exit points is not high enough. Especially in some difficult scenarios, they require a very late exit or even frequent offloading to the cloud to meet accuracy requirements, which sacrifices real-time performance, or force an early exit to pursue real-time performance, resulting in low accuracy. Current work still lacks an efficient architecture or training framework that can ensure higher accuracy at early-exit points for different data features, thereby meeting both real-time and high-accuracy requirements for different tasks. In addition to introducing new offline training methods, it is also necessary to cooperate with online execution control to ensure that different tasks can exit from appropriate branches, so as to release resource occupation, and save energy consumption at the edge.

To compensate for the shortcomings of existing architectures, we propose a novel edge-cloud collaborative progressive inference framework E3, which combines eXplainable AI (XAI) techniques [16, 39, 41] with early-exit techniques [43] to construct an efficient multi-branch neural network (NN). We shift the rationale of early-exit network construction from fixed to agile and data-centric. The goal of our design is to progressively extract the most important features for inference so that high accuracy can be achieved at early-exit points. The basic idea is to incorporate the knowledge about the heterogeneity of different input data into training so that

the computation required to extract important features can be migrated from online inference to offline training. More specifically, we interpret such heterogeneity as the contribution or importance of different data features for NN inference, and employ XAI to explicitly evaluate and differentiate the importance during training. In this way, the most important features can be assigned to the early branches in the inference phase, making the early results more accurate. Besides, the later branches can further infer on the less important features and combine the results of early branches, thereby making the inference accuracy as high as possible when exiting, i.e., satisfying real-time and accuracy requirements simultaneously.

However, the major challenge of above process is that the data features in different channels may have similar importance for NN inference, and they may all be either important or unimportant. In this case, the most important features are difficult to concentrate for extraction, leading to a decrease in accuracy. This necessitates the execution of more later branches to extract the important features to improve accuracy, thus causing additional latency. To address this challenge, E3 intentionally manipulates the importance of data features through non-linear transformation in the high-dimensional feature space to ensure that such importance distribution is skewed on different data. In other words, we enforce that after transformation, only the features in the specific k channels contribute significantly to the branch NN inference, so that these important features can be extracted and inferred earlier. We achieve such a skewness manipulation by a highly lightweight feature extractor, and jointly train the feature extractor and branch NNs in each E3 block to ensure accuracy. Since only a few of the most important features are extracted for each branch, a lightweight branch NN can achieve high accuracy, facilitating efficient deployment and execution at the edge. E3 progressively improves inference accuracy by stacking multiple E3 blocks in the edge-cloud system, which is more efficient and lightweight than just inserting branches on the original model in traditional methods [23, 24, 43, 54]. Moreover, E3 can highly compress features that are unimportant in the later block, reducing edge-cloud transmission overhead while hardly affecting end-to-end accuracy.

In addition to improving early-exit accuracy by training a novel progressive architecture, different real-time requirements as well as resource and bandwidth fluctuations at runtime also provide scheduling opportunities for online execution control. E3 maintains a priority-driven queue and a task launcher at the edge for concurrency control, thereby improving resource utilization and alleviating contention. When executing tasks concurrently, E3 profiles the execution time of each block and the edge-cloud transmission overhead, so as to control the exit point for each task to ensure that it does not exceed the deadline. The high accuracy in the early stage also enables the tasks to exit earlier to free up resources.

In summary, the main contributions of this paper are as follows:

- We propose E3, to our knowledge, which is the first to apply XAI technique to construct efficient progressive neural architecture with high accuracy at early-exit points.
- E3 cooperates with stable online execution control to launch inference tasks and decide their exit point, ensuring resource utilization and real-time performance.
- We implement E3 on various edge-cloud platforms. Compared to traditional architectures, E3 is more lightweight, efficient, energy-saving, and incurs almost no additional runtime overhead. Under stringent deadlines, the average accuracy of tasks increases by > 50%, and the deadline satisfaction rate approaches 100%.

2 RELATED WORK

Optimization for Deep Neural Network (DNN) Inference at the Edge. The constraints of computing and storage resources at the edge pose challenges in satisfying diverse inference requirements. To achieve SLOs and enhance Quality of Service (QoS), several typical techniques have been developed recently. Model compression [5, 12] and knowledge distillation [13, 46] are some offline optimization techniques aimed at reducing the number of parameters and computational load for edge DNNs, accelerating execution, and striving to ensure accuracy. However, they lack mechanisms for dynamic scheduling according to different tasks and fluctuating resources, leading to low efficiency at runtime. Progressive inference [43, 52, 54] inserts multiple early-exit branches into the original model, allowing inference instances with different SLOs to exit from different branches at an early stage, thus reducing both inference latency and resource occupation. The construction and training for early-exit models are challenging, and they also need to be combined with online scheduling to control when the task exits. Model partition [19, 56] divides a large model into several parts, each deployed on a different device for distributed collaborative inference, effectively addressing the challenge of deploying a large model on resource-constrained devices. However, the distributed inference usually utilizes some data compression techniques [1, 6] to reduce the transmission overhead between devices. Inappropriate data compression may be accompanied by significant accuracy loss. In addition to the above optimization at the edge, edge-cloud collaborative inference [16, 48, 51] is also a typical mode. Edge devices offload part of computational load to the cloud through certain strategies, so as to save edge resources. However, the long-distance transmission and irregular bandwidth fluctuations exacerbate end-to-end latency, which introduces a huge optimization cost.

Progressive Inference in Edge-Cloud Systems. Recent work has explored the integration of progressive inference (early-exit) architecture [43] with edge-cloud model partition [19] to fully utilize the advantages of both. Edgent [24] integrates workload offloading and progressive neural architecture. However, it only allows a single early-exit branch for all inputs, which cannot fully exploit the benefits of progressive inference. SPINN [23] and EdgeML [54] also adopt such combination approach and consider different early-exit branches for different inputs to allow flexible inference and computation offloading. SPINN uses the same early-exit threshold for all branches, while EdgeML assigns an independent threshold to each branch, resulting in a significant performance improvement. SPINN adopts a multi-objective optimization algorithm to generate execution strategies, which may be inefficient due to the huge solution space caused by continuous thresholds and partition points. EdgeML optimizes the execution strategy dynamically through a reinforcement-learning-based framework, which can not only adapt more flexibly to the dynamic environment between edge and

cloud but also accommodate new platforms through knowledge transfer. However, the learning-based method is usually unstable and costly to train in practice. In addition, these progressive architectures simply insert branch NNs containing convolutional and fully-connected layers into the original model, then retrain it (the unified loss function is the weighted sum of each branch's loss) [43]. This incurs a lot of additional runtime overhead, and leads to two outcomes for difficult instances: late exit to ensure accuracy at the expense of real-time performance, or early exit to satisfy real-time requirements at the expense of accuracy. There still lacks an efficient architecture and training method to encourage high accuracy at early-exit points for any data features, so as to simultaneously ensure real-time performance and accuracy.

AI Attribution. Traditional attribution methods apply zero masks [36] or random permutation [10] to specific input variables, and then empirically evaluate the importance of inputs via the induced output variation. Attention-based methods [47, 53] embed a learnable weighting layer into the original NN to evaluate feature importance. However, these techniques are sensitive to the NN structure and cannot ensure accurate evaluation. Recent XAI tools [39, 41] are more accurate and robust. They derive importance using gradients of NN outputs with respect to the inputs, which is fine-grained and can clearly show how much each input feature contributes to the output in percentage terms. Such XAI techniques have been widely used for analyzing data features and understanding the NN's behavior, but existing work rarely explores their application in improving NN inference efficiency. The way of evaluating feature importance with XAI techniques guides us to design more efficient progressive NN that can extract important features earlier to promote both high accuracy and early exit.

3 MOTIVATION

3.1 Deficiencies of Existing Architectures

According to the characteristics of Convolutional Neural Network (CNN) [22], the deeper the layer, the larger the receptive field, and the more abstract (high-level) the features. The typical image classification task requires features to contain more high-level information for fine classification. In order to perform global classification, it also requires NN to have the ability to perceive the entire image, which heavily relies on the inference capability of deep CNNs. Therefore, although existing progressive architectures [23, 24, 43, 54] achieve early exit by inserting multiple branch NNs into the original model, the branch NN at shallow needs to be configured with more layers to ensure accuracy, which cannot significantly reduce FLOPs, contradicting the original design intention of progressive architectures.

In addition, existing progressive neural architectures simply insert branch NNs consisting of convolutional and fully-connected layers into the original model, then retrain it, leading to two outcomes for difficult instances: late exit to ensure accuracy at the expense of real-time performance, or early exit to satisfy real-time requirements at the expense of accuracy. To validate this, we apply the advanced edge-cloud progressive neural architecture EdgeML [54] to EfficientNetV2 [42] pre-trained on ImageNet [37], insert three branches (i.e., four exit points in total including the exit point

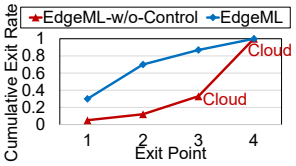


Figure 1: Early-exit CDF of EdgeML [54] without and with online execution control.

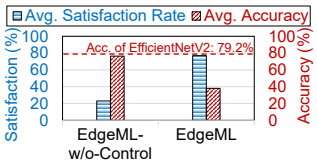


Figure 2: The average task accuracy and deadline satisfaction rate of EdgeML without and with online execution control.

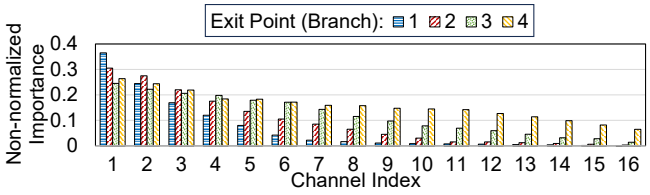


Figure 3: We utilize XAI tool [41] to explore the importance distribution of features in each branch of EdgeML.

of original model) at equidistant points of FLOPs across the original model layers, and retrain such a branchy model [43]. Then, using the *imgaug* [18] library, we randomly add noise such as rain, snow, fog, occlusion, strong light, blurring, etc. to the ImageNet ILSVRC2012 test set, which is common in some scenarios like autonomous driving in bad weather and welding workbench under strong light. We input these difficult instances into the framework for classification, with the relative deadline set to 50 ms, and test it on an edge-cloud platform composed of an edge device NVIDIA Jetson Nano [32] and a cloud server equipped with NVIDIA GeForce RTX 3080 [29]. The edge-cloud bandwidth is fixed at 10 Mbps.

Fig. 1 shows the results of EdgeML without online execution control, where we set the confidence threshold for all exit points to 0.5, and deploy the first two branches at the edge, the other two in the cloud. The inference can only exit when the threshold is satisfied at a certain exit point (except for the last one), which is accuracy-first, so these difficult instances are prone to late exit or even be offloaded to the cloud, severely sacrificing real-time performance and resulting in a low deadline satisfaction rate, despite the average accuracy being close to that of the original model (Fig. 2). Fig. 1 also shows the results of EdgeML with learning-based online execution control, where it can dynamically decide the confidence threshold for each exit point and the edge-cloud partition point, ensuring as late the inference exits as possible under the premise of satisfying the deadline requirement, which is real-time-first. Therefore, early exit results in low accuracy for the difficult instances (Fig. 2).

3.2 Application of Explainable AI (XAI)

Recent work on eXplainable AI (XAI) evaluates feature importance by some attribution tools. Typical XAI tool such as Integrated Gradients (IG) [41] takes multiple linear interpolations between the input features and a naive baseline, and feeds them to a reference NN. Then it computes the gradient of each interpolation with respect to

the output (e.g. confidence score) of reference NN, and accumulates the gradients to measure the importance of features.

To further explain the fundamental reason why real-time and accuracy requirements cannot be satisfied simultaneously under the existing architectures, we utilize such XAI tool to explore the importance distribution of features in each branch of EdgeML. Specifically, we consider each branch NN and the backbone NN before it as a reference NN, then input the same data to evaluate the importance distribution of features in 16 channels after the input layer. As shown in Fig. 3, in most channels, the importance of features in the earlier branches is lower than that in the later ones, i.e., the critical features of difficult instances are usually extracted in the late stage, resulting in diminished early accuracy. These critical features are difficult to highly compress, causing much edge-cloud transmission overhead, and further diminishing real-time performance. This prompts us to pursue a new architecture and training method that can enhance the importance of early features, so as to satisfy real-time performance and high accuracy simultaneously.

In reverse, this way of evaluating feature importance using XAI tools guides us to design more efficient progressive neural architecture. To achieve high accuracy for early exit, only a few of the most important features must be extracted for inference at each early branch. Therefore, the importance distribution must be skewed on early features. The higher such skewness is, the fewer features dominate the NN inference, and just lightweight branch NNs are needed to achieve high accuracy. However, according to the results in Fig. 3, the importance skewness does not consistently exist in all branches. This prompts us to design a new feature extractor that intentionally manipulates and enhances the skewness for each branch. In addition, to eliminate the additional runtime overhead brought by online importance evaluation, such knowledge about importance must be incorporated into offline training.

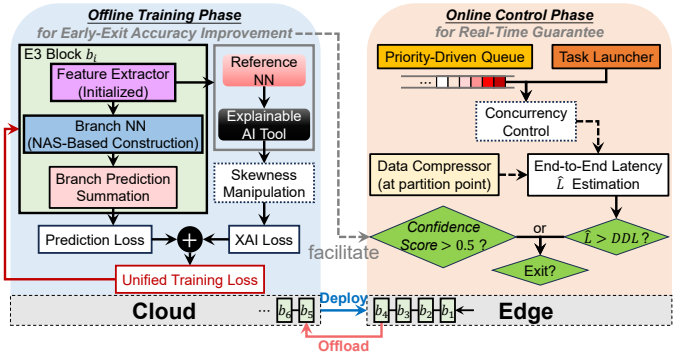


Figure 4: E3 overview.

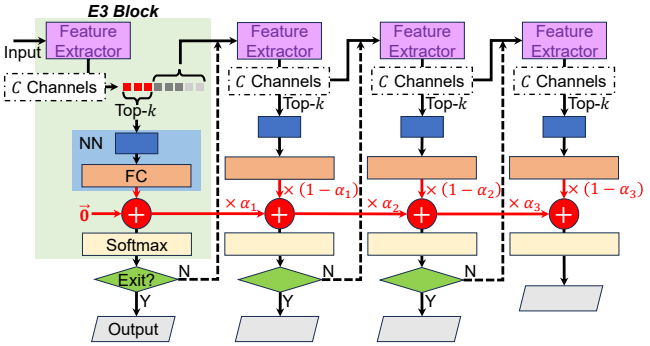


Figure 5: E3-Net architecture. Here is an example of stacking four E3 blocks.

4 E3 OVERVIEW

For inference tasks (data) of varying difficulty, our goal is to construct a neural architecture that can progressively extract the most important features for inference, facilitating high accuracy at early exit points, i.e., satisfying both high-accuracy and real-time requirements. Here we start with the typical image classification task because its NN can be easily extended as a backbone to other tasks such as semantic segmentation and object detection. To achieve the goal, we utilize XAI technique to construct a novel progressive neural architecture E3. As shown in Fig. 4, E3 includes two phases:

Offline Training Phase for Early-Exit Accuracy Improvement. E3 employs the XAI tool to extract feature importance information from the reference NN to train and stack each E3 block, so as to construct our progressive NN. In our training framework, we initialize the feature extractor and design a unified loss for both skewness manipulation and branch prediction to ensure fast convergence of joint training. We also utilize Neural Architecture Search (NAS) technique to assist in constructing efficient and lightweight branch NNs. We train the entire architecture in the cloud and then deploy as many E3 blocks as possible to the edge, ultimately achieving efficient edge-cloud progressive inference.

Online Control Phase for Real-Time Guarantee. E3 maintains a priority-driven queue and a task launcher at the edge to keep the number of instances being concurrently inferred at a certain level, thereby improving resource utilization and alleviating contention. At the same time, E3 profiles the execution time of each block and the edge-cloud transmission overhead when executing tasks concurrently, so as to accurately estimate end-to-end latency and control the exit point for each task to ensure that it does not exceed the deadline. The high accuracy in the early stage facilitates the tasks to exit earlier to free up resources. The data compressor can highly compress features that are unimportant in the later stage of inference, reducing edge-cloud transmission overhead while hardly affecting end-to-end accuracy.

5 OFFLINE TRAINING PHASE

5.1 E3 Neural Network (E3-Net)

Most progressive NNs [23, 24, 43, 54] are constructed by inserting branches on the original model, while we construct our E3-Net with the original model as the reference NN for XAI tool [41]. Specifically, we follow [16] to design the E3 block and then stack the blocks to construct the whole multi-branch network. Fig. 5 shows an example of stacking four E3 blocks (i.e. four exit points).

An E3 block consists of the feature extractor, a branch NN, and branch prediction summation.

5.1.1 Feature Extractor. As described in Section 3.2, the features in different channels may have similar importance for NN inference, and they may all be either important or unimportant. In this case, the most important features are difficult to concentrate for extraction, leading to a decrease in accuracy. This necessitates the execution of more later branches to extract the important features to improve accuracy, thus causing additional latency. To address this challenge, E3 intentionally manipulates the importance of data features through non-linear transformation in the high-dimensional feature space to ensure that such importance distribution is skewed on different data. In other words, we enforce that after transformation, only the features in the specific k channels contribute significantly to the branch NN inference, so that these important features can be extracted and inferred earlier. We achieve this skewness manipulation through a feature extractor, and utilize XAI tool to jointly train the feature extractor and branch NN.

To facilitate deployment at the edge and reduce runtime overhead, the feature extractor must be highly lightweight. In practice, it is constructed with two convolutional layers, each having C output channels. To further save memory consumption and jointly train all branch NNs, all E3 blocks share a feature extractor that can progressively extract important features for different inputs. In each E3 block, the extractor maps the input to the same number of channels C, which is also the number of channels after removing the input layer from the reference NN. The importance of features in these channels will be evaluated by the reference NN during offline training. To eliminate the substantial overhead brought by the temporary evaluation of importance in online inference, the trained extractor concentrates the most important features in the specific k ($k < C$) channels, then directly inputs them into the branch NN for inference. The remaining less important $C - k$ channels' features are input into the next E3 block for further extraction.

5.1.2 Branch NN. Typical progressive neural architectures, such as [23, 24, 43, 54], incorporate convolutional and fully-connected layers on each branch NN, and require that the FLOPs of the current branch should not exceed the FLOPs of exiting from the next branch, which is a basic requirement of early-exit networks. However, at the same time, they require that the previous branch should be configured with more layers to perform more detailed inference on features that are not sufficiently extracted in the early stage, which contradicts the original intention of reducing FLOPs through early exit. The branch NN of E3 also contains convolutional and fully-connected layers, but unlike these traditional progressive architectures. Since only a few of the most important features are extracted for each branch, a lightweight branch NN can achieve high accuracy, facilitating efficient deployment and execution at the edge. Furthermore, since the features extracted in the later blocks are less important, more complex branch NNs are required to ensure accuracy improvement. Otherwise, it is prone to cause "overthinking" [20], i.e., the results in the later stage are inferior. The more complex and less reached late-stage inference can be deployed in the resource-rich cloud to ensure inference efficiency. We further validate this insight and explain how to construct and train branch NNs in Section 5.2.4. Since the important features are concentrated in the k channels and input to the branch NN for inference, the number of input channels of the branch NN is k.

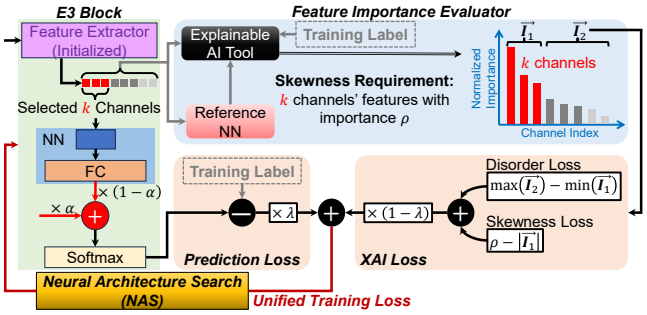


Figure 6: Offline training for each E3 block.

5.1.3 Branch Prediction Summation. The current branch incorporates the results of previous branches with a learnable weight $0 < \alpha_i < 1$, allowing the results of previous branches can be reused. This is essential for establishing connections between E3 blocks and improving accuracy, as the inference on critical features in previous branches is more important, and the later inference serves merely to further enhance confidence or rectify errors. This is similar to the residual connections in CNNs [11] and skip connections in Transformers [8, 35, 47, 50]. Multiple such connections help improve end-to-end accuracy with little increase in parameters and FLOPs [15], which is another advantage of E3-Net compared to the existing early-exit NNs. Moreover, such point-to-point weighted sum has two advantages over NN-based alternatives (e.g. adding additional NN layers for combination). First, it is more lightweight and increases negligible overhead. Second, it maintains the dimension alignment across different branch outputs, while using NN layers (e.g. convolutional or fully-connected layers) to combine the branch outputs may entangle them and disrupt this alignment, adversely affecting the final inference accuracy.

5.2 Offline Training Framework

As illustrated in Fig. 6, E3 utilizes the XAI tool to evaluate feature importance based on the reference NN, then incorporates the skewness of importance into a unified loss function for training, so as to jointly train the feature extractor and branch NN. Meanwhile, E3 jointly trains and stacks the E3 blocks via the shared feature extractor and branch prediction summation. The offline training framework encompasses the following steps.

5.2.1 Initialize the Feature Extractor. The highly lightweight feature extractor may not have enough representation ability to achieve the learning goal at the initial stage of training, leading to instability and difficulty in convergence. Therefore, it is necessary to initialize the feature extractor before joint training. We follow the algorithm in [16] to select k initial channels from the output of feature extractor, wherein the top-k important features (evaluated by the XAI tool) are most likely located. Specifically, we make such selection according to the possibility that the top-k important features are located in a channel, and count this possibility by feeding training data into the reference NN. The features in these k channels will directly serve as the input for the branch NN. In this way, joint training does not need to explore from scratch which channels the important features are located in, but starts from a more certain

stage with less ambiguity, so that the requirement on the feature extractor's initial representation ability can be lower, greatly reducing the learning difficulty and ensuring the training quality.

5.2.2 Skewness Manipulation. We enforce the feature extractor to skew the importance of various data features towards the selected k channels, i.e., to generate new features with importance concentrated in the selected k channels. In the online inference phase, the feature extractor directly inputs the features in the k channels into branch NN, without re-evaluating the importance, thereby eliminating additional runtime overhead.

We incorporate this skewness into a unified loss function for training. The *disorder loss* (Eq. 1) is used to enforce the most important features to concentrate in the selected k channels after transformation, and the *skewness loss* (Eq. 2) is used to enforce the importance skewness of the transformed features to satisfy the threshold ρ. The loss functions are theoretically differentiable [27].

$$\mathcal{L}_{disorder} = \max\left(\max(\vec{I}_2) - \min(\vec{I}_1), 0\right), \tag{1}$$

$$\mathcal{L}_{skew} = \max\left(\rho - |\vec{I}_1|, 0\right), \tag{2}$$

where vector $\vec{I}_1$ contains the importance values of the selected k channels' features, while vector $\vec{I}_2$ contains the importance values of the remaining channels' features. The importance values are normalized. $|\cdot|$ represents the 1-norm of a vector.

The skewness of feature importance can be adjusted by changing k and ρ, which enables a flexible trade-off between the accuracy and overhead for branch NN inference on devices with different resources. In practice (Section 7.8), we explore setting appropriate values for k and ρ.

During training, the current output of the feature extractor is fed to the XAI tool to evaluate the importance of features in each channel for reference NN inference. In experiments, we choose the Integrated Gradients (IG)-based XAI tool [41] (as described in Section 3.2), since it performs better and can be applied to any NN without additional modification, although the evaluation cost is higher, which only happens during offline training. Besides, this tool requires high accuracy of the reference NN to prevent misjudging importance, so here we use the pre-trained original model as the reference NN. To further avoid possible errors of the reference NN, we also compare the output of reference NN with the training labels, and use it for importance evaluation only when the reference NN predicts correctly.

5.2.3 Unified Training Loss for Each E3 Block. We define *XAI loss* $\mathcal{L}_{XAI} = \mathcal{L}_{disorder} + \mathcal{L}_{skew}$. By combining $\mathcal{L}_{XAI}$ and *branch prediction loss* $\mathcal{L}_{pred}$, we obtain the unified training loss $\mathcal{L}_{unified}$ (Eq. 3) to jointly train the feature extractor and branch NN within each E3 block.

$$\mathcal{L}_{unified} = \lambda \cdot \mathcal{L}_{pred} + (1 - \lambda) \cdot \mathcal{L}_{XAI}, \tag{3}$$

where $0 < \lambda < 1$ controls the weight of the two parts, we explore the appropriate λ through preliminary experiments. Fig. 7 shows the training results of a certain block under different λ. A smaller λ makes the training focus more on the importance skewness, leading to lower prediction accuracy. We observe that a moderate λ between 0.3 and 0.6 can approximately satisfy the importance threshold requirement with almost no impact on the accuracy of branch NN.

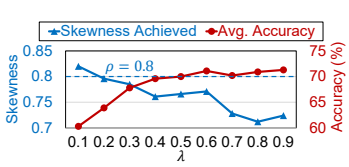


Figure 7: Impact of λ on ImageNet dataset. The reference NN is EfficientNetV2. $C = 24$, $k = 5$, $\rho = 0.8$.

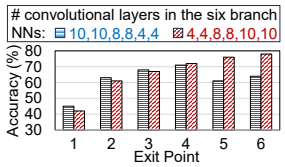


Figure 8: Impact of branch NN structures on ImageNet dataset. $\lambda = 0.5$.

5.2.4 NAS-Based Construction for Branch NN. According to the results in [43], the end-to-end accuracy does not always increase with the number of convolutional layers in the branch NN, indicating the existence of an optimal configuration. Therefore, we utilize Neural Architecture Search (NAS) technique [25] to search for the convolutional layer configuration that optimizes the current branch's accuracy. During the search process, the number of layers in the branch NN can be restricted by minimizing the unified loss. Thus, the search objective for each block is min $\mathcal{L}_{unified}$. This prevents the occurrence of insufficient or redundant layers, and ensures that there are multiple branches under memory constraint, i.e., the NN layers will not all be concentrated in a few branches. Furthermore, NAS can further optimize the structure of each layer, and we can customize candidate operators to be hardware-friendly to reduce execution latency.

Specifically, we utilize the Differentiable Architecture Search (DARTS) [25], one of the state-of-the-art NAS algorithms, to automatically design efficient branch NN structures for each E3 block on specific hardware. DARTS enables efficient search via gradient descent.

However, in practice, it is costly to search in such a large space, so we prune the search space for NAS. The insight in Section 5.1.2 provides an opportunity for pruning. The experimental results in Fig. 8 validate this insight. That is, in our E3-Net, the later branch requires configuring with more NN layers than the earlier one. Otherwise, there will be redundant layers in the early branches, or the layers in the later branches are insufficient to perform more careful inference on the unimportant features. This indicates that when constructing each branch NN, we only need to require NAS to gradually add layers on the basis of the previous branch NN structure until the training accuracy stops improving, instead of searching from scratch.

The training framework iteratively searches for the structure of each branch NN and retrain the corresponding E3 block until convergence. Initially, we set the first branch NN to consist of only one global-average pooling layer and one fully-connected layer. As shown in Fig. 9, each subsequent branch NN is constructed based on the structure of previous one, either keeping unchanged or gradually adding more convolutional layers before the pooling layer. We fix the number of input channels for the branch NN to k, and the number of output channels for each intermediate convolutional layer varies from 16 to 512 exponentially. Since we mainly conduct experiments on the GPU platform, here we choose dense convolutional layers that are more friendly to GPU. If customization is targeted at other processors such as CPU, the depthwise

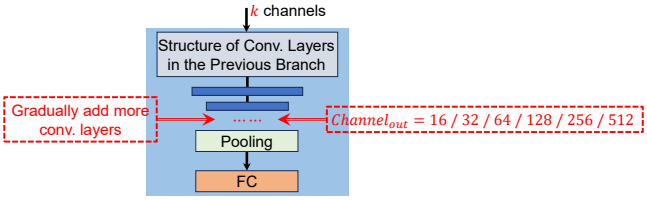


Figure 9: NAS-based construction for branch NN.

separable convolution [14] can be employed. Due to the substantial computation and memory overhead of fully-connected layers, each branch NN contains only one fully-connected layer subsequent to all the layers for classification, with its structure determined by the adjacent layer and branch outputs.

5.2.5 Block Stacking in Edge-Cloud Systems. Since the inference exits from different branches dynamically, we stack and train each E3 block progressively from front to back, so that the accuracy of all branches is high and improves progressively, rather than training all blocks simultaneously to solely pursue high accuracy for the last branch. During the search process for each block, we fix the parameters of previously trained blocks when calculating gradients, and only retrain the parameters of the currently searched block. Meanwhile, we require the average accuracy improvement on the training dataset after adding a block to be greater than 2%, otherwise stop stacking blocks.

To perform the edge-cloud model partition and accelerate training, we first stack and train all E3 blocks on the powerful cloud server. Then, the early blocks are deployed to the edge one by one until the edge device is out of memory, while the remaining blocks are retained in the cloud. A shared feature extractor must also be replicated for the edge. The abundant computation and memory resources in the cloud are sufficient to handle the complex inference in the later branches.

Such a design fixes the edge-cloud partition point and deploys as many early blocks as possible at the edge. If the edge memory is sufficient to deploy all E3 blocks, there is no need for the cloud, thereby eliminating the overhead of edge-cloud transmission. This is motivated by three factors. Firstly, in most environments, especially with the increasingly powerful edge devices, the bandwidth between edge and cloud is insufficient to ensure that the computation latency saved by the cloud can offset the transmission latency, even if the data to be transmitted are compressed. Today's powerful edge devices are striving to complete tasks locally, without offloading to the cloud. Secondly, in E3, since the important features have been extracted multiple times in the early blocks through XAI technique, the features extracted in the later blocks are less important and contribute less to improving end-to-end accuracy, i.e., they are highly compressible (validated in Section 7.6). Therefore, the partition point should be located in the later stage to reduce the feature transmission overhead. Thirdly, compared to potential early offloading in dynamic partition [23, 24, 54], since the execution in the cloud is similar to that at the edge, E3 saves the overhead caused by early offloading to execute more blocks at the edge by fixing the partition point to later stage.

6 ONLINE CONTROL PHASE

Online execution control is to determine the exit point for each inference task according to current hardware resources, available edge-cloud bandwidth, and the task's SLO, ensuring the accuracy while satisfying the deadline requirement. This problem is highly complex, and the corresponding decision-making algorithms may result in significant runtime overhead, affecting model execution. Although the learning-based algorithms [54] can achieve near-optimal results with relatively low overhead, they are unstable and costly to train in practice. In contrast, our method has negligible runtime overhead and is more stable.

Given that the offline training phase has been focused on improving early-exit accuracy by utilizing the XAI technique to construct and train the E3-Net, we perform dynamic execution control in the online phase to focus on ensuring the real-time performance for each inference task. Therefore, a real-time-first exit mechanism is adopted, which makes the inference exit as late as possible under the premise of satisfying the deadline requirement. The inference will exit earlier if the confidence threshold is satisfied before the deadline, so the high accuracy in the early stage can facilitate early exit to free up resources.

6.1 Priority-Driven Queue and Task Launcher

Most modern hardware platforms support the concurrent execution of multiple deep learning tasks [28, 30, 31]. However, due to constrained resources, these tasks may encounter severe resource contention, leading to mutual blocking. As shown in Fig. 10, we explore the average execution latency of a certain E3 block and the GPU utilization under different numbers of concurrent tasks on two edge GPU platforms with different resources. The results indicate that insufficient concurrent tasks will lead to low GPU utilization. Conversely, when the number of concurrent tasks exceeds a certain threshold, although the GPU utilization is high, excessive tasks induce severe resource contention. This is particularly evident on the resource-constrained Nano, where each task experiences varying degrees of blocking, leading to an increased latency.

Therefore, in order to ensure resource utilization, mitigate blocking caused by resource contention among concurrent tasks, reduce task processing latency, and manage the priority for task dequeueing, we introduce a priority-driven queue and a task launcher at the edge to control the number of concurrent tasks in E3.

Specifically, E3 receives inference tasks at the edge and temporarily stores them in the priority-driven queue. We adopt the Earliest-Deadline-First (EDF) strategy, i.e., the task with earlier absolute deadline will have higher priority and be launched first by the task launcher. The task launcher monitors the number of tasks currently being inferred on the edge device, launching a new task whenever one exits, so as to keep the number of concurrent tasks at M. At the optimal number M, the edge GPU utilization approaches 100% and the average latency of tasks is minimized. On the specific device, M is a fixed value that can be measured offline. As shown in Fig. 10, $M_{Nano} = 2$, $M_{NX} = 7$. Moreover, since there is less contention among tasks when the number of concurrent tasks is less than M, even if tasks arrive too late to replenish M, it will only lead to a reduction in the utilization, with almost no impact

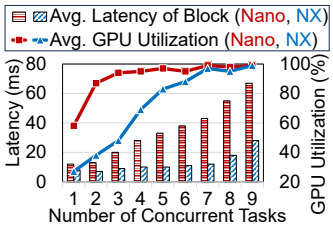


Figure 10: Block latency and GPU utilization on NVIDIA Jetson Nano [32] and Xavier NX [33] under different numbers of concurrent tasks.

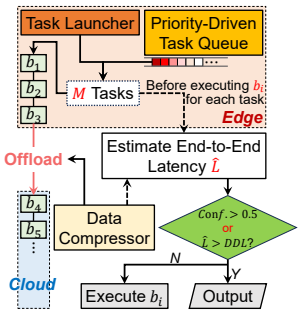


Figure 11: Online execution control flow.

on the inference latency, ensuring that the task does not exceed the deadline due to this.

Since the cloud resources are far more abundant than the edge, and only a fraction of tasks from the edge will be offloaded to the cloud, we only constrain the concurrency at the edge while ignoring the contention in the cloud.

6.2 End-to-End Latency Estimation

The priority-driven queue and task launcher can sustain a stable execution environment, making the execution time of each E3 block at the edge more stable and easier to estimate. While exploring M offline, we can profile the worst-case execution time (WCET) [49] $\hat{l}_{b_i}$ for each E3 block b_i under M concurrent tasks. Considering the concurrency capability of cloud, and only a fraction of tasks from the edge will execute the blocks in the cloud, for E3 blocks in the cloud, $\hat{l}_{b_i}$ is approximated as the WCET when that block is executed alone on the cloud server.

During online execution, the currently available edge-cloud bandwidth $avail\_bw$ can be monitored to estimate the edge-cloud transmission latency, i.e., $\hat{l}_{trans} = \frac{comp\_data}{avail\_bw}$, where $comp\_data$ represents the size of the compressed data to be transmitted. Our data compressor employs learning-based quantization [1] and LZW compression [6], dynamically adjusting the compression rate according to different features, ensuring maximum compression while preserving accuracy. The overhead of the compressor is very low, with the total time consumed by compression at the edge and decompression in the cloud kept within 10 ms. Therefore, we set the estimated time consumption of the compressor $\hat{l}_{comp}$ to 10 ms. The data compression and edge-cloud transmission are only considered when the task execution reaches the partition point, otherwise we set $\hat{l}_{trans} = 0$ and $\hat{l}_{comp} = 0$.

Then, the estimated end-to-end latency of a task after executing block b_i is $\hat{L} = L + \hat{l}_{b_i} + \hat{l}_{trans} + \hat{l}_{comp}$, which is used to decide whether to execute block b_i. L is the actual time consumed before executing b_i, including the queuing time. We use the worst-case time to estimate $\hat{L}$ to minimize the possibility of exceeding the deadline.

6.3 Online Execution Control Flow

As illustrated in Fig. 11, our decision-making logic is intuitive. For each task being executed in E3, we calculate $\hat{L}$ before executing the next block b_i. If $\hat{L}$ does not exceed the relative deadline of the task, and the confidence (i.e., probability value) of the current block prediction does not exceed **0.5**, then execute b_i, otherwise, it exits at the current block. If $\hat{L}$ has exceeded the relative deadline before executing the first block due to queuing, then the task is directly removed from the queue.

When the inference reaches the partition point and decides to continue execution, the data compressor compresses the $C - k$ channels' features outputted by the E3 block before the partition point, then the compressed features are transmitted to the cloud for decompression and execution of the next block. Note that the branch results are transmitted as well, as they are important and the data size is very small, so they are not compressed. The execution in the cloud is similar to that at the edge.

After executing the last E3 block, the inference exits directly.

7 EVALUATION

7.1 Implementation and Setup

Testbed. We implement E3 and reference NNs based on PyTorch [34], and conduct experiments on edge-cloud platforms equipped with different GPUs. For the edge, we use two NVIDIA embedded platforms with different resources. For the cloud, we use a server equipped with NVIDIA GeForce RTX 3080. The specific information of the devices is shown in Table 1. Considering the shared memory and other occupations, their available memory is reduced by 2GB. We use *Wondershaper* [17], a lightweight bandwidth control script, to adjust the available edge-cloud bandwidth to the default 10 Mbps.

Table 1: Devices Information

Device	GPU	Memory	Computility
Jetson Nano	128-core Maxwell	4GB	0.5TFLOPS
Xavier NX	384-core Volta	8GB	6.8TFLOPS
Cloud Server	8704-core GeForce RTX 3080	10GB	29.8TFLOPS

Baselines. We compare E3 with three advanced edge-cloud progressive neural architectures (Edgent [24], SPINN [23], and EdgeML [54]) introduced in Section 2. Since they all follow BranchyNet [43] to construct branchy models, here we take the reference NN in E3 as the original model, insert branches at equidistant points of FLOPs across its layers, and let the number of branches be the same as that of E3 blocks for fair comparison.

We also compare with two non-early-exit architectures: AgileNN [16] and PNC [48], which are primarily designed for data compression and computation offloading, but the strategy is partially similar to E3. This can be regarded as an ablation study to highlight the flexibility and benefits of progressive inference.

In addition, we set a naive baseline, which uses the original model without early-exit branches, and iteratively selects the optimal edge-cloud partition point at runtime according to the method in Neurosurgeon [19].

Reference NN. E3 is primarily composed of convolutional layers that are more lightweight compared to Transformer blocks [9]. The features focused on by CNNs and Transformers are different, and the features that are important to a Transformer may not hold

the same importance for a CNN [4, 26, 35]. Besides, the baselines consistently lean towards CNN-based models. Therefore, we set up three typical CNN-based reference NNs with different accuracy, FLOPs, and number of parameters: ResNet50 [11], MobileNetV2 [38], EfficientNetV2 [42]. These models have already been verified to achieve considerable performance on multiple open-source datasets, thus they are suitable as reference NNs. Here they are all pre-trained on ImageNet [37].

We have also considered using Transformer blocks to construct the feature extractor and branch NNs, but stacking multiple E3 blocks makes it difficult to be lightweight to deploy on edge platforms. This is a limitation of the proposed E3 architecture.

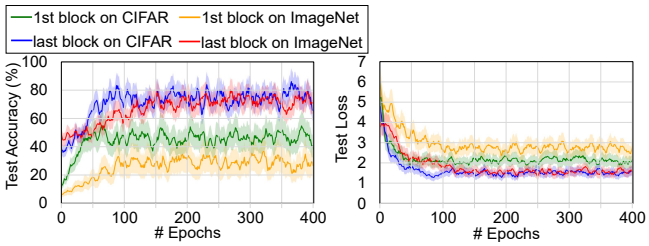
Dataset. We conduct experiments on two image classification datasets of varying difficulty, namely CIFAR-100 [21] and ImageNet ILSVRC2012 [37], where the training set is used to supervise training, and the test set is used to generate inference tasks. In order to test the robustness to difficult instances at the same time, we randomly select 30% of samples in the test set and add noise such as rain, snow, fog, occlusion, strong light, blurring, etc. to them using the *imgaug* [18] library. All the images are reshaped to 224x224.

Parameter Settings. Unless otherwise specified, the following settings are used by default in the experiments. The feature extractor consists of two convolutional layers, each with C output channels, where C is the number of channels after removing the input layer from the reference NN. $\lambda = 0.5$, $\rho = 0.8$, $k = \lceil 20\% \times C \rceil$. $M_{Nano} = 2$, $M_{NX} = 7$. The Stochastic Gradient Descent (SGD) optimizer [3] is used for training, with a learning rate set to 0.01 and the standard weight decay set to 5×10^{-4}. The batch size in training is 64. The batch size for inference is set to 1 to simulate tasks arriving frame-by-frame. The task arrival rate is set to the frame rate of a typical camera, i.e. 30 fps. The relative deadline for each task is set to the same to ensure a throughput close to 100%, which is defaulted to 50 ms. For some parameters specific to the baselines, we employ settings that maximize their respective performance.

Metrics. All the results are averaged over five runs on the test set. We mainly focus on four metrics: (*i*) deadline satisfaction rate, i.e. the ratio of tasks completed before the deadline to the total number; (*ii*) accuracy, i.e. top-1 accuracy in the image classification task; (*iii*) edge energy consumption per task. During execution, since it is difficult to accurately measure the energy consumption for a single task, we use *jetson-stats* [2], an open-source monitoring toolkit, to periodically profile the average power of edge devices, subtract the idle power, then multiply by the total processing time for the entire test set, and finally average the resulting energy consumption to each task; (*iv*) GPU utilization (warps/s), i.e. the number of active warps between two timestamps during execution [7].

7.2 System Overhead

Although the Integrated Gradients (IG)-based XAI tool [41] requires 20 ∼ 100 times of gradient calculations to derive each importance measurement, such evaluation is only conducted during offline training. As shown in Fig. 12, we test the convergence overhead of offline training for E3-Net. Since E3-Net is stacked and trained block by block, here we take the training of the first and last blocks as examples. The earlier block converges faster because it is more



(a) Accuracy convergence (b) Loss convergence

Figure 12: The convergence overhead under different training datasets. We take the first and last blocks as examples here. The reference NN is EfficientNetV2.

Table 2: The FLOPs ratio (left value) and memory occupation ratio (right value) relative to the original model under traditional BranchyNet and our E3-Net on ImageNet ILSVRC2012.

Arch.-Ref. NN	ResNet50	MobileNetV2	EfficientNetV2
BranchyNet	132%, 139%	141%, 147%	128%, 133%
E3-Net	78%, 80%	83%, 84%	70%, 77%

lightweight. Although the later blocks are more complex, they are not trained from scratch due to the summation of important results from earlier blocks, so their convergence is not much slower and can further improve accuracy. These results indicate that despite the increased learning complexity induced by skewness manipulation, fast convergence can still be achieved through appropriate loss function design and initialization for the feature extractor.

Compared to the overhead of offline training, the overhead of online execution control is more concerning. The reinforcement-learning-based algorithm in EdgeML exhibits a high convergence cost (> 200 epochs), and its average execution time reaches 20 ms. Moreover, the additional NNs it introduces lead to more contention for runtime memory and computation resources. In contrast, our online control only involves simple calculations and judgments, so its runtime overhead can almost be ignored.

In addition, as shown in Table 2, since the baselines construct a progressive neural network, BranchyNet, by inserting multiple branches on the original model, it inevitably increases the FLOPs and memory occupation, reaching approximately 1.5 times that of the original model. In contrast, we use the original model as the reference NN for the XAI tool, gradually stacking lightweight E3 blocks to construct E3-Net. The total FLOPs and memory occupation are at most 84% of the reference NN.

7.3 E3-Net Structure

As shown in Fig. 13, we test the E3-Net structure under different training datasets, edge devices, and reference NNs. The results indicate that difficult datasets (e.g. ImageNet) require stacking more blocks and branch NN layers to perform more detailed classification. Due to insufficient memory, Nano is compelled to offload some blocks to the cloud. Employing a reference NN with superior accuracy (e.g. EfficientNetV2) helps to extract important features more

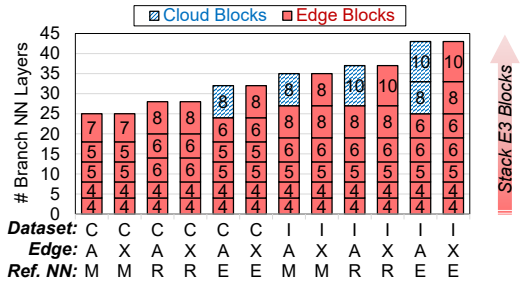
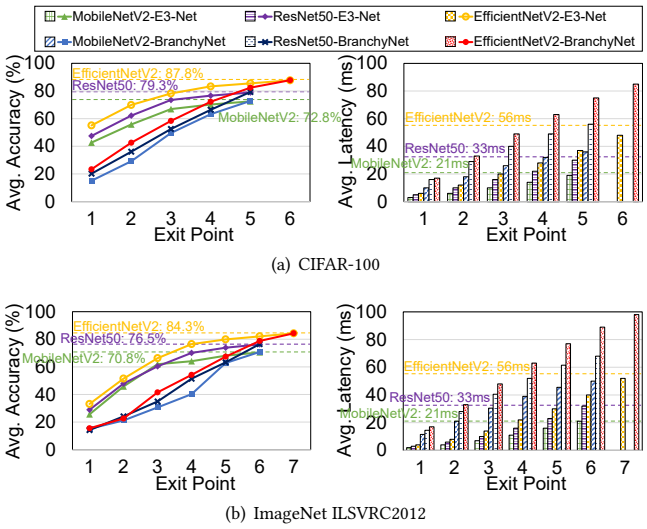


Figure 13: E3-Net structure under different datasets (C: CIFAR-100, I: ImageNet), edge devices (A: Nano, X: NX), and ref. NNs (M: MobileNetV2, R: ResNet50, E: EfficientNetV2).



(a) CIFAR-100

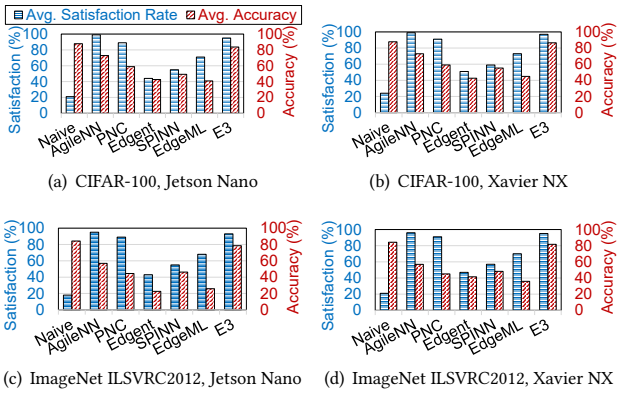
(b) ImageNet ILSVRC2012

Figure 14: The average accuracy and latency at all exit points under different network architectures (tested on the server).

accurately, which elevates the upper accuracy limit that E3-Net can reach, but this requires stacking more blocks. The branch NN layers consistently follow a structure of fewer layers in the earlier branch. Since only a few of the most important features are extracted for each branch, a lightweight branch NN can achieve high accuracy, facilitating efficient deployment and execution at the edge.

7.4 Accuracy and Deadline Satisfaction

As shown in Fig. 14, we first validate the advantages of E3-Net over BranchyNet on the server. Since important features are extracted for inference, the accuracy of E3-Net is significantly higher than that of BranchyNet in early branches. Besides, the accuracy of E3-Net can approximate that of EfficientNetV2 when only executed to the fourth exit point, which saves more than 60% time compared with one complete inference of EfficientNetV2. This benefits from the lightweight early branch NNs. It means that a sufficient number of blocks can be executed within the deadline to ensure accuracy. It also means that even on the resource-constrained Nano, the accuracy of E3-Net can approximate that of the reference NN at the



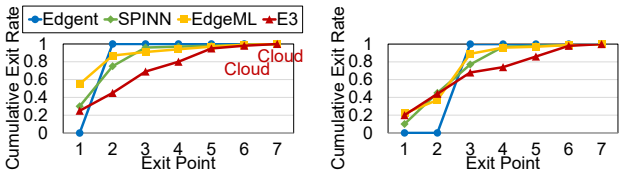
(a) CIFAR-100, Jetson Nano

(b) CIFAR-100, Xavier NX

(c) ImageNet ILSVRC2012, Jetson Nano

(d) ImageNet ILSVRC2012, Xavier NX

Figure 15: The average deadline satisfaction rate and accuracy of task processing under different edge devices and datasets. The reference NN is EfficientNetV2.



(a) Jetson Nano

(b) Xavier NX

Figure 16: Early-exit CDF on different edge devices (tested on ImageNet ILSVRC2012, the reference NN is EfficientNetV2).

edge. The later exit points have less accuracy improvement because the later features are less important and thus the error correction ability of corresponding branches is reduced, contributing little to accuracy. In contrast, the accuracy of BranchyNet in the early stage is not high. BranchyNet inserts branches on the reference NN, and the early branch NNs are more complex, resulting in more computation latency. Notably, the latency when executing to the fourth exit point exceeds that when executing the original EfficientNetV2.

As shown in Fig. 15, E3 outperforms the three progressive architectures in both accuracy and deadline satisfaction rate when processing tasks, with an accuracy increase of >50%, close to the naive baseline, and a deadline satisfaction rate close to 100%. While AgileNN effectively compresses the features for transmission and retains minimal local computation, it lacks multiple extractions for important features. This results in a lower accuracy compared to E3. PNC only progressively compresses features for offloading according to deadline, lacking a high-accuracy early-exit mechanism. Under stringent deadlines, PNC may drop some important features to reduce transmission overhead, thus its accuracy is inferior to E3.

Guided by XAI, E3 provides accuracy guarantee for early exit during the offline training phase. This also enables E3's online control to focus on real-time performance, thereby adapting to requirements with simpler calculations and judgments. Moreover, as shown in Fig. 16, under the default relative deadline and bandwidth, the lightweight branch NNs in E3 enable tasks to exit from later exit

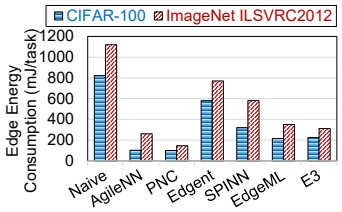


Figure 17: Average edge energy consumption per task on the Nano-Server platform.

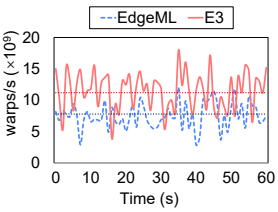


Figure 18: GPU utilization on Nano during task execution.

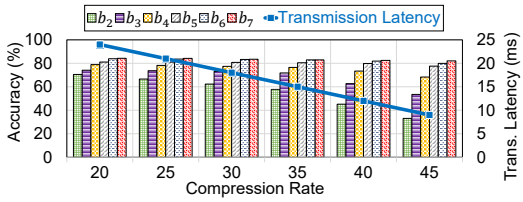
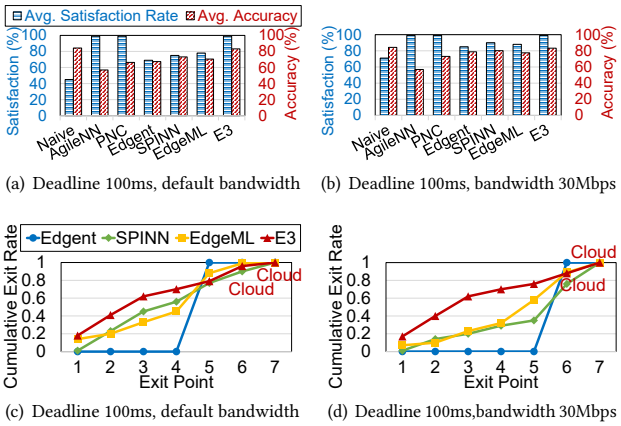


Figure 19: The average end-to-end accuracy and edge-cloud transmission latency of the tasks when compressing the $C-k$ channels' features before block b_i with different compression rates (tested on ImageNet ILSVRC2012, the reference NN is EfficientNetV2).

points, allowing for more inference to further improve accuracy. E3 also reduces additional transmission overhead by decreasing the number of tasks offloaded to the cloud, that is, most tasks exit before the fixed edge-cloud partition point. Especially on the resource-rich NX platform, all E3 blocks are deployed at the edge, which eliminates transmission overhead and enables more blocks to be executed within the deadline to improve accuracy. In contrast, the dynamic edge-cloud partition of the baselines has to balance the benefits and additional overheads brought by early offloading.

7.5 Edge Energy Consumption and Resource Utilization

For resource-constrained edge devices, enhancing energy efficiency is essential. As shown in Fig. 17, we observe the energy consumption per task at the edge. The edge energy consumption in edge-cloud progressive inference (e.g. E3 and EdgeML) is naturally higher than that of compression+offloading mode (e.g. AgileNN and PNC). Both E3 and EdgeML ensure that inference exits as late as possible under the premise of guaranteeing real-time performance. EdgeML also makes a trade-off between task latency and energy, but its learning-based online control algorithm is more complex and has higher runtime energy consumption. The lightweight NNs and online control algorithm in E3 help free up resources and reduce energy consumption at the edge. Consequently, although E3 does not explicitly optimize for energy saving, its energy efficiency is comparable to that of EdgeML. Moreover, as shown in Fig. 18, due to the lack of concurrency control in EdgeML, the edge GPU utilization during execution is 30% lower than E3.



(a) Deadline 100ms, default bandwidth (b) Deadline 100ms, bandwidth 30Mbps

(c) Deadline 100ms, default bandwidth (d) Deadline 100ms,bandwidth 30Mbps

Figure 20: The average deadline satisfaction rate, accuracy, and early-exit CDF of tasks under different relative deadlines and edge-cloud bandwidths (tested on the Nano-Server platform, the reference NN is EfficientNetV2, and the dataset is ImageNet ILSVRC2012).

7.6 Compression and Transmission for Less Important Features

As shown in Fig. 19, the features extracted later in E3 contribute less to improving end-to-end accuracy, i.e., they are less important, and the features in the $C-k$ channel are highly compressible, resulting in lower edge-cloud transmission overhead. This is one of the reasons for deploying as many blocks as possible at the edge. Besides, the overhead of our data compressor is very low, with the total time consumed by compression at the edge and decompression in the cloud kept within 10 ms, and it is only invoked when cloud offloading occurs.

7.7 Adaptability to Deadlines and Bandwidths

As shown in Figs. 20(a) and 20(c), when we extend the relative deadline to 100 ms, the deadline satisfaction rate under all architectures naturally increases. Most tasks can be executed until later branches and even offloaded to the cloud, thereby improving accuracy. The execution mechanism in E3 enables most tasks to exit earlier once satisfying the confidence threshold. The advantages in compression and transmission for unimportant features make E3 further superior to the baselines.

On this basis, we adjust the edge-cloud bandwidth to 30 Mbps. As shown in Figs. 20(b) and 20(d), compared to the default bandwidth of 10 Mbps, a higher bandwidth enables tasks to utilize the cloud resources at a lower cost. The baselines accelerate inference by dynamically deciding the partition point to offload more computation to the cloud. Although E3 employs a fixed partition point, it also offloads more tasks to the cloud to execute more blocks within the deadline, so as to improve accuracy.

7.8 Effect of Skewness Manipulation

The skewness manipulation is the basis for constructing E3. To study its effectiveness, we observe the performance and structure

Table 3: The effect of skewness manipulation on achieved skewness ($|\vec{I}_1|$), average accuracy of tasks (Acc.), total number of layers in all branch NNs (# Layers), and average edge-cloud transmission latency (Trans.). Tested on the Nano-Server platform, the reference NN is EfficientNetV2.

Dataset	CIFAR-100									ImageNet ILSVRC2012								
k	$\lceil 10\% \times C \rceil = 3$			$\lceil 20\% \times C \rceil = 5$			$\lceil 30\% \times C \rceil = 7$			$\lceil 10\% \times C \rceil = 3$			$\lceil 20\% \times C \rceil = 5$			$\lceil 30\% \times C \rceil = 7$		
ρ	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9	0.7	0.8	0.9
$\lvert\vec{I}_1\rvert$	0.67	0.75	0.83	0.69	0.8	0.86	0.71	0.81	0.87	0.66	0.73	0.81	0.71	0.77	0.86	0.7	0.78	0.86
Acc. (%)	80.5	80.4	79.3	82.5	83.7	81.1	81.4	83.4	81.3	75.1	76.3	75.6	78.4	78.5	79.2	78.3	78.9	79.1
# Layers	31	29	29	34	32	31	36	36	34	41	40	40	45	43	42	47	47	46
Trans. (ms)	15.2	13.5	11.9	14.1	12.1	11.2	13.5	12.1	11.3	14.7	13.1	10.9	13.5	11.6	10.3	13.5	11.3	10.5

of E3-Net under different k and ρ. As shown in Table 3, when k is smaller or ρ is larger, there is a higher requirement on the importance skewness. The branch NN can achieve high accuracy with lower resource overheads, making it more suitable for edge deployment. However, the lightweight feature extractor usually cannot meet excessively high skewness requirements. Moreover, higher skewness makes the branch NN more sensitive to the input and prone to wrong perception, even though E3 can alleviate such accuracy decline through progressive inference by stacking multiple blocks. In addition, higher skewness can increase the sparsity of the $C - k$ channels' features, resulting in higher compressibility and thus reducing edge-cloud transmission overhead. In practice, we usually set $\rho = 0.8$ and $k = \lceil 20\% \times C \rceil$, which is a trade-off.

8 CONCLUSION

In this paper, we incorporate explainable AI technique to construct a progressive neural architecture E3, which is more lightweight, efficient, energy-saving, and incurs almost no additional runtime overhead. With stable online execution control, E3 can simultaneously satisfy real-time and high accuracy for different task data, and ensure resource utilization. We validate its superiority and scalability on various edge-cloud platforms, datasets, and reference models. In the future, we are also interested in extending the ideas in E3 to efficient edge inference for other scenarios.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62350710797 and in part by the Science and Technology Plan Project of Shenzhen through Project Number JSGG20220831110002004.

REFERENCES

[1] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. 2017. Soft-to-hard vector quantization for end-to-end learning compressible representations. *Advances in neural information processing systems* 30 (2017).

[2] Raffaello Bonghi. 2024. jetson-stats. https://rnext.it/jetson_stats/.

[3] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 421–436.

[4] Shengcao Cao, Mengtian Li, James Hays, Deva Ramanan, Yu-Xiong Wang, and Liangyan Gui. 2023. Learning lightweight object detectors via multi-teacher progressive distillation. In *International Conference on Machine Learning*. PMLR, 3577–3598.

[5] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. 2021. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in Neural Information Processing Systems* 34 (2021), 19974–19988.

[6] HN Dheemanth. 2014. LZW data compression. *American journal of engineering research* 3, 2 (2014), 22–26.

[7] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2021. Ios: Inter-operator scheduler for cnn acceleration. *Proceedings of Machine Learning and Systems* 3 (2021), 167–180.

[8] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. 2021. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*. PMLR, 2793–2803.

[9] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Trevor Hastie, Robert Tibshirani, Jerome Friedman, Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. Random forests. *The elements of statistical learning: Data mining, inference, and prediction* (2009), 587–604.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4340–4349.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[16] Kai Huang and Wei Gao. 2022. Real-time neural network inference on extremely weak devices: agile offloading with explainable AI. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 200–213.

[17] Bert Hubert, Jacco Geul, and Simon Séhier. 2020. The Wonder Shaper 1.4.1. https://github.com/magnific0/wondershaper.

[18] Alexander Jung. 2020. Overview of Augmenters. https://imgaug.readthedocs.io/en/latest/source/overview_of_augmenters.html.

[19] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[20] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*. PMLR, 3301–3310.

[21] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[23] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–15.

[24] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[26] Yufan Liu, Jiajiong Cao, Bing Li, Weiming Hu, Jingting Ding, and Liang Li. 2022. Cross-architecture knowledge distillation. In *Proceedings of the Asian conference on computer vision*. 3396–3411.

[27] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[28] NVIDIA. 2015. CUDA Multi-Streams. https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/.

[29] NVIDIA. 2020. GeForce RTX 3080 Family. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3080-3080ti/.

[30] NVIDIA. 2020. Multi-Instance GPU. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[31] NVIDIA. 2020. Multi-Process Service. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf.

[32] NVIDIA. 2022. NVIDIA Jetson Nano. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/.

[33] NVIDIA. 2022. NVIDIA Jetson Xavier. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[35] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. 2021. Do vision transformers see like convolutional neural networks? *Advances in neural information processing systems* 34 (2021), 12116–12128.

[36] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.

[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[39] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*. 618–626.

[40] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[41] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. PMLR, 3319–3328.

[42] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*. PMLR, 10096–10106.

[43] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2464–2469.

[44] Siyu Teng, Peng Deng, Yuchen Li, Bai Li, Xuemin Hu, Zhe Xuanyuan, Long Chen, Yunfeng Ai, Lingxi Li, and Fei-Yue Wang. 2023. Path planning for autonomous driving: The state of the art and perspectives. *arXiv preprint arXiv:2303.09824* (2023).

[45] Zhihong Tian, Wei Shi, Yuhang Wang, Chunsheng Zhu, Xiaojiang Du, Shen Su, Yanbin Sun, and Nadra Guizani. 2019. Real-time lateral movement detection based on evidence reasoning network for edge computing environment. *IEEE Transactions on Industrial Informatics* 15, 7 (2019), 4285–4294.

[46] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[48] Ruiqi Wang, Hanyang Liu, Jiaming Qiu, Moran Xu, Roch Guérin, and Chenyang Lu. 2023. Progressive Neural Compression for Adaptive Image Offloading under Timing Constraints. In *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 118–130.

[49] Wikipedia. 2024. Worst-Case Execution Time. https://en.wikipedia.org/wiki/Worst-case_execution_time.

[50] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. 2022. Unified visual transformer compression. *arXiv preprint arXiv:2203.08243* (2022).

[51] Ziyang Zhang, Yang Zhao, Huan Li, Changyao Lin, and Jie Liu. 2024. DVFO: Learning-Based DVFS for Energy-Efficient Edge-Cloud Collaborative Inference. *IEEE Transactions on Mobile Computing* (2024).

[52] Ziyang Zhang, Yang Zhao, and Jie Liu. 2023. Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In *International Conference on Service-Oriented Computing*. Springer, 242–258.

[53] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. 2020. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10076–10085.

[54] Zhihe Zhao, Kai Wang, Neiwen Ling, and Guoliang Xing. 2021. Edgeml: An automl framework for real-time deep learning on the edge. In *Proceedings of the international conference on internet-of-things design and implementation*. 133–144.

[55] Ray Y Zhong, Xun Xu, Eberhard Klotz, and Stephen T Newman. 2017. Intelligent manufacturing in the context of industry 4.0: a review. *Engineering* 3, 5 (2017), 616–630.

[56] Li Zhou, Hao Wen, Radu Teodorescu, and David HC Du. 2019. Distributing deep neural networks with containerized partitions at the edge. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*.

[57] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.