# CEED: Collaborative Early Exit Neural Network Inference at the Edge

Yichong Chen
*Imperial College London*
London, United Kingdom
yichong.chen119@imperial.ac.uk

Zifeng Niu
*Imperial College London*
London, United Kingdom
zifeng.niu19@imperial.ac.uk

Manuel Roveri
*Politecnico di Milano*
Milano, Italy
manuel.roveri@polimi.it

Giuliano Casale
*Imperial College London*
London, United Kingdom
g.casale@imperial.ac.uk

***Abstract*—Collaborative inference at the edge has gained traction in recent years as one of the main trends within edge computing. The early exit neural network (EENN) architecture supports this by balancing inference time and accuracy with configurable early exit thresholds within the neural network. Such thresholds enable the dynamic tuning of the processing latency of a job based on confidence scores. However, most distributed EENN setups use a preset confidence threshold and assume constant data arrivals. This assumption exposes the system to potential data loss due to finite memory capacity in the edge devices. To address these issues, we propose CEED, an AI-based optimization framework to enable collaborative EENN inference on a multilayer edge infrastructure. CEED integrates an EENN predictor and a Loss ratio predictor to rapidly evaluate confidence threshold configurations and job assignment to devices. Experiments conducted on a physical testbed show that CEED significantly improves existing EENN inference methods by striking a better balance between end-to-end system loss ratio and EENN inference accuracy.**

***Index Terms*—Edge computing, Early-Exit Neural Networks, Quality of Service.**

## I. Introduction

Deep neural networks (DNNs) are becoming increasingly deeper to achieve more accurate predictions [1], [2]. However, this increase in complexity can result in significant inference times, especially on edge devices constrained by limited hardware resources. Moreover, when edge devices exhaust their memory, they may be unable to accommodate new data, potentially resulting in data loss, which reduces reliability [3].

To reduce the incidence of these issues, techniques like model distillation, quantization, and pruning [4] have thus been proposed to shorten the inference times. However, these methods often result in reduced accuracy compared to the original model. The early exit neural network (EENN) architecture has been introduced to address this limitation [5], [6], [7]. EENNs integrate internal classifiers (ICs) in the processing pipeline that allow the output at intermediate stages, depending on confidence scores computed by the ICs. Such scores reflect the likelihood of accurate classification at early stages. Early exits thus allow EENNs to trade inference time with accuracy.

Collaborative inference over a distributed edge system with EENNs is a new trend, which can further enhance the computational and memory advantages of these models for inference serving [8], [9], [10]. However, splitting EENN layers among a set of edge devices exacerbates the difficulty in choosing the right early exit point, as this becomes also affected by the heterogeneous device processing speeds, the available network bandwidth, and the computational cost of transferring data [11]. Moreover, choosing appropriate IC confidence thresholds to activate early exits is a difficult problem even on a single device due to the complex dependencies between ICs, loss ratios, and job arrival rates [12], [13], [8]. For example, the probability of input data reaching an IC is contingent upon the thresholds of the previous ICs and their decision not to force an earlier exit, which also depends on the job data content.

To address this complexity, we present CEED, an AI-based optimization framework leveraging two performance predictors, the EENN predictor and the Loss ratio predictor, addressing two complementary goals. The EENN predictor tackles the challenge of configuring confidence thresholds of ICs by predicting, using a transformer, the confidence score and exit probability for incoming jobs. The Loss ratio predictor, instead, uses a graph neural network (GNN) to evaluate the system loss ratio, avoiding the cost of simulation [14], [15]. Leveraging these AI models, CEED strives to identify the optimal balance between high classification accuracy and a predefined target loss ratio.

We examine the performance of CEED on real-world EENNs deployed on edge devices and show that it can substantially improve Quality-of-Service (QoS) metrics, such as accuracy, latency and system loss ratio, compared to existing EENN configuration methods. To the best of our knowledge, CEED is the first method to jointly optimize accuracy, latency and loss ratios to deliver QoS in collaborative EENN-based inference at the edge.

In summary, the key contributions of this paper are:

- We propose an EENN predictor, based on a transformer model, to forecast the mean confidence score and the exit probabilities of ICs based on a given confidence threshold configuration to enable reasoning on the best EENN configuration for a given inference task.
- A Loss ratio predictor based on a GNN is also introduced to forecast the system loss ratio based on the system configuration speedily and accurately.
- We present the CEED optimization framework developed to optimize EENN-based collaborative inference at the edge. Using real EENN measurements and a physical testbed, we demonstrate that CEED can surpass existing methods in achieving better QoS for inference tasks.

This paper is organized as follows: Section II presents background on edge inference and EENNs. Section III introduces the problem tackled in this paper alongside an illustrative example. Section IV and V respectively detail the AI models and

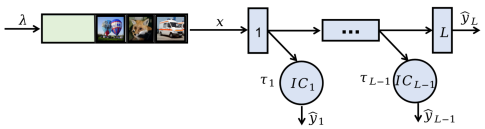



Fig. 1: **An EENN with $L$ layers and $L-1$ ICs, along with a finite input buffer.** For a device $i$, at the $l$th IC, the confidence score $\delta_{i,l}$ is calculated and compared with the confidence threshold $\tau_{i,l}$. The job may exit early and produce a classification $\widehat{y}_l$ if $\delta_{i,l} \geq \tau_{i,l}$. If the job proceeds to the final layer $L$, the output will be the final classification $\widehat{y}_L$. Additionally, the jobs arrive in the buffer at rate $\lambda_i$ and are lost if the image size exceeds the residual memory capacity.

optimization method that define CEED. Finally, experimental results are given in Section VI, followed by conclusions in Section VIII.

## II. Background

### A. Edge-based Collaborative Inference

This paper considers collaborative inference at the network edge based on convolution neural networks (CNNs) implemented as EENNs. Fig. 1 illustrates the processing pipeline of a typical EENN when inference is run on a single edge device. The EENN consists of a sequence of $L$ layers and $L-1$ intermediate classifiers (ICs) to control early exits. We regard these layers as *logical* abstractions, i.e., any sequence of consecutive physical layers that do not allow early exit is seen here as a single logical layer. This occurs frequently for EENNs with residual connections, where the connected layers typically do not feature ICs in-between [2]. The device running the EENN is assumed to have a finite memory space to store incoming images (i.e., *jobs*) that arrive stochastically to the device, sitting in a first-come first-served (FCFS) waiting buffer until they begin service[1]. Three main types of data can consume memory on the device: the data size of the resident jobs, the memory required for hosting the layers of the EENN, and the system data (e.g., firmware, O/S, ...). Among these, we focus on the most complex component, namely the memory occupancy due to the resident jobs, which fluctuate stochastically with new arrivals. An arriving job is lost (i.e., dropped) at a device whenever its data size exceeds the available memory capacity at the device.

In the collaborative edge setting [9], [16], the model shown in Fig. 1 generalizes as shown in Fig. 2. The system consists of several edge devices organized in $K+1$ layers that collaborate with each other to support EENN inference tasks. Each device can host a subset of the $L$ layers, and any given layer can be replicated multiple times and placed across multiple devices. Without loss of generality, this model also allows an edge system to run different EENNs simultaneously across the edge

[1]While our EENN is demonstrated for the sake of simplicity using image classification tasks, the proposed method is general and may be applicable to other tasks.

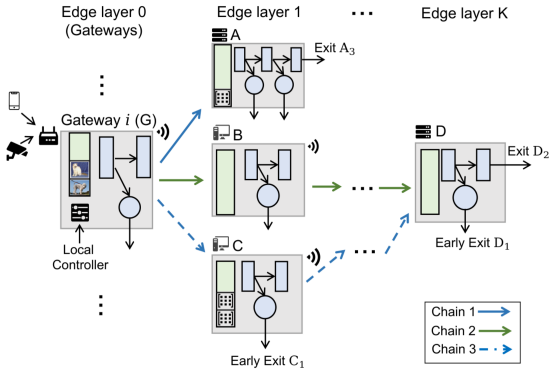



Fig. 2: **Edge-based collaborative inference architecture.** Each device hosts a number of EENN layers (blue rectangles) and their associated ICs (blue circles). Green rectangles show the input buffers. Jobs may exit at any IC within the device or finish when they reach the last EENN layer.

infrastructure. However, to simplify notation, we assume from now on a system running a single distributed EENN.

In the edge setting, jobs first arrive at the devices at the initial layer (Edge Layer 0), referred to as *gateways* [17]. Throughout, the arrival process of jobs to the gateways is assumed to be modeled by a Poisson process with a known and fixed arrival rate $\lambda_i$ at gateway $i$, either representing an image stream from a single source or the superposition of multiple independent Poisson streams. Upon completing processing at a device, if the job has not exited early at any of the ICs within that device, the intermediate activations are transferred over the network to the device that hosts the next EENN layer.

Due to layer replication and the various exit points for a job, the edge system effectively offers various possible processing paths, each referred to as a *chain*, to serve an arriving job. That is, a chain is a sequence of layer-device pairs that the job is scheduled to traverse for its processing. Because a chain always includes the gateway $i$, this implies that different gateways will tag incoming jobs using a different subset of chains, although these may partially overlap on some of their subpaths. For example, Fig. 2 shows three possible chains for an incoming job at the gateway G shown on the left. A chain may be, in general, shorter than the entire length $L$ of the EENN, but could additionally also be partially traversed by the job if any IC along the chain reaches sufficient confidence to force its early exit. Instead, if the job completes at the last layer of its chain, then the inference is completed without early exit. Thus, a chain effectively specifies the *maximal path* a job is allowed to traverse across the system.

### B. EENN thresholds and confidence

In this section, we briefly introduce the notation and internal working of an EENN, so as to be able to describe the interplay between processing latency and EENN classification accuracy, which is considered by CEED to optimize the QoS of the inference process.

Returning to Fig. 1, consider a job arriving into the system at a device $i$. Upon completion of processing at layer $l$, if an IC is positioned after layer $l$, it receives the output from layer $l$ and computes the confidence score $\delta_{i,l}$. For example, a particular EENN implementation may consider the entropy-type confidence score $\delta_{i,l}$ defined in [5][2], which we also used in our tests.

The decision to allow an early exit is controlled by a confidence threshold $\tau_{i,l} \in [0, 1]$ for the IC for layer $l$. If $\delta_{i,l} \geq \tau_{i,l}$, the job exits early and gives classification $\widehat{y}_l$. If not, the job continues processing through the EENN pipeline. If the confidence scores do not meet the threshold conditions at any of the ICs, the EENN model delivers the predicted classification at the final activation following the last layer $L$ as having a (logical) IC with confidence threshold $\tau_{i,L} = 0$.

In the next sections, we regard confidence thresholds as a key control knob available to the CEED system to tune inference. That is, for an inference job that first arrives at gateway $i$, we assume that the job is paired with a confidence threshold configuration, denoted as $\boldsymbol{\tau}_i = \{\tau_{i,l} | l = 1, \ldots, L\}$, that assigns the threshold values for all the logical layers (up to layer $L$ at maximum) along the chain traversed by the job. If a chain terminates earlier than layer $L$, say at a layer $l' < L$, then all the thresholds $\tau_{i,h}$, $h \geq l'$, are set to 0 so as to force exit at $l'$. Such information can be carried by the job via metadata carried with the intermediate activations exchanged by the devices. Because confidence thresholds determine the likelihood that a job will exit early, assigning a given vector $\boldsymbol{\tau}_i$ to a job effectively provides a way for a controller to influence inference latency. Indirectly, this can, therefore, influence downhill memory usage for the devices across the chain and their loss ratios. We quantify the latter through a system loss ratio metric $D$, defined as the average ratio between the number of jobs lost at any device to the total number of arrivals to the EENN.

However, since the interplay between $\boldsymbol{\tau}_i$ and these metrics is indirect, given that the ICs operate based on data content, a sophisticated reasoning method would be needed to characterize the relationships among the QoS metrics. Our proposal, detailed later in the paper, is to rely on transformer models and GNNs to capture this complex interplay.

To better see the dependence between a configuration $\boldsymbol{\tau}_i$ and EENN classification accuracy, we give a few definitions. A summary of the main notation introduced in this and the next section is available in Table I. Let $\ell(y, \widehat{y}) = \mathbb{1}(y = \widehat{y})$ be a function computing the discrepancy between the label $y$ and the predicted output $\widehat{y}$. The accuracy of the EENN given a threshold configuration $\boldsymbol{\tau}_i$ can be denoted as $A(\boldsymbol{\tau}_i) = \mathbb{E}_{\mathcal{U}}[\ell(y, \widehat{y})]$, where $\mathcal{U}$ is the image dataset. We further define the mean confidence score $\Delta_i$ of the EENN given a threshold configuration $\boldsymbol{\tau}_i$ as follows: given a sequence of input jobs $\mathcal{X}$ from $\mathcal{U}$ and a threshold configuration $\boldsymbol{\tau}_i$, the exit probability of each IC can be estimated as $q_{i,l}(\boldsymbol{\tau}) = |\mathcal{X}_l|/|\mathcal{X}|$ and $\sum_{l=1}^{L} q_{i,l}(\boldsymbol{\tau}_i) = 1$, where $\mathcal{X}_l$ are jobs that exit at the $l$th IC. The mean confidence score of each IC can also be computed by $\Delta_{i,l}(\boldsymbol{\tau}_i) = \sum_{x \in \mathcal{X}_l} \delta_{i,l}(x)/|\mathcal{X}_l|$. Based on the above definitions, for jobs entering the system at gateway $i$, we can summarize its mean confidence score by $\Delta_i(\boldsymbol{\tau}_i) = \sum_{l=1}^{L} q_{i,l}(\boldsymbol{\tau}_i)\Delta_{i,l}(\boldsymbol{\tau}_i)$.

[2] $\delta_{i,l} = 1 - \frac{1}{\log(|\mathcal{C}|)} \sum_{c=1}^{\mathcal{C}} (-o_{i,l,c} \log(o_{i,l,c}))$, where $\mathcal{C}$ is the class set that the EENN is designed to classify, and $o_{i,l,c}$ is the activation value of the $l$th IC for class $c$

TABLE I: Summary of main notation

| | |
|---|---|
| $I$ | Number of gateways |
| $L$ | Number of EENN layers and ICs |
| $K$ | Number of edge layers except the gateways |
| $A_i$ | Number of chains available from gateway $i$ |
| $\delta_{i,l}$ | The confidence score of a job at the $l$th IC in gateway $i$ |
| $\tau_{i,l}$ | The confidence threshold of the $l$th IC for gateway $i$ |
| $\boldsymbol{\tau}_i$ | The confidence threshold for gateway $i$ |
| $q_{i,l}$ | Exit probability of the $l$th IC for gateway $i$ |
| $\Delta_{i,l}$ | Mean confidence score of the $l$th IC for gateway $i$ |
| $\Delta_i$ | Mean confidence score of the EENN given $\boldsymbol{\tau}_i$ |
| $\langle i, j \rangle$ | Mapping for a job arriving at the $i$th gateway on the $j$th chain |
| $\langle i, j, l \rangle$ | Mapping for a job in $\langle i, j \rangle$ also to exit at the $l$th IC |
| $s_{i,j}$ | The assigned score of the corresponding chain $\langle i, j \rangle$ |
| $\boldsymbol{a}_i$ | Chain assignment policy at gateway $i$, $\boldsymbol{a}_i = \{(\langle i, j \rangle, s_{i,j}), \forall j \in \Omega_i\}$ |
| $\Omega_i$ | Set of chains available to the CEED controller at gateway $i$ |
| $\Pi$ | Control policy, $\Pi = \{(\boldsymbol{\tau}_i, \boldsymbol{a}_i) \| i = 1, \ldots, I\}$ |
| $D$ | System loss ratio |
| $p_{i,j}$ | Probability of selecting chain $j$ as the assigned chain |
| $\lambda_i$ | Arrival rate of jobs at the $i$th gateway |
| $\gamma$ | User-defined penalty term |
| $r$ | Reward |
| $t_{max}$ | Maximum processing time of the EENN at the gateway |
| $\rho$ | Load factor, $\rho = \lambda t_{max}$ |
| $R$ | Mean response time |

## III. Problem Statement

We consider the problem of jointly optimizing the IC confidence thresholds and chain assignment for incoming jobs to the multilayer edge system. In order to do so, we assume that a controller continuously runs alongside the distributed EENN. This controller is statically parameterized at the initial deployment of the network with a control policy obtained from CEED, which is subsequently applied during system operation. The control policy knows: i) offline profiling of the processing times at each of the EENN layers; ii) the maximum network bandwidth profiled offline at each device; iii) the arrival rate of jobs estimated at each gateway (reciprocal of the estimated mean inter-arrival time). Models would require periodic re-training in case the system parameters change significantly over time. As we reported in Section VI, this is inexpensive since CEED runs in our tests in 2-4 minutes.

The role of the control policy is to assign confidence thresholds $\boldsymbol{\tau}_i$ and chain to a job based on its entry point into the system, the gateway $i$. To further reduce the overhead at the gateways, we assume that the gateway assigns to each arriving job the same IC confidence threshold vector $\boldsymbol{\tau}_i$, which, however, must be decided offline. This decision is extremely challenging due to the aforementioned impact on several metrics, but also due to threshold dependencies

TABLE II: Results of the motivating example for different EENN threshold configurations of the EENN derived from ResNet50 ($L = 9$; $K = 2$; $i = 1$).

| Chain | Threshold | Accuracy | Latency (s) | Median EENN exit layer |
|---|---|---|---|---|
| G→ $A_3$ | $\boldsymbol{\tau}_i$ | 0.8824 | 0.5520 | $l = 8$ |
| G→ $A_3$ | $\boldsymbol{\tau}'_i$ | 0.7225 | 0.4917 | $l = 5$ |
| G→B→ $D_2$ | $\boldsymbol{\tau}_i$ | 0.9455 | 0.3331 | $l = 9$ |
| G→B→ $D_2$ | $\boldsymbol{\tau}'_i$ | 0.8157 | 0.3007 | $l = 6$ |
| G→C→ $D_2$ | $\boldsymbol{\tau}_i$ | 0.9028 | 0.5041 | $l = 9$ |
| G→C→ $D_2$ | $\boldsymbol{\tau}'_i$ | 0.7613 | 0.4116 | $l = 5$ |

between ICs[3], heterogeneous device processing speed and memory capacities, network bandwidth availability, and the CPU overheads on the device associated with data transfer.

Formally, to select an optimal trade-off among these metrics, CEED seeks to determine an optimal control policy $\Pi = \{(\boldsymbol{\tau}_i, \boldsymbol{a}_i)|i = 1, \ldots, I\}$, where $I$ is the number of gateways. As we discuss later in section V, we seek to determine $\Pi$ so that it maximizes the overall accuracy of the EENN while simultaneously minimizing the system loss ratio. The $\Pi$ control policy involves setting the confidence threshold $\boldsymbol{\tau}_i$ and determining the chain assignment policy $\boldsymbol{a}_i$ for each gateway $i$. In CEED, the chain assignment policy is specified by action-score set $\boldsymbol{a}_i = \{(\langle i, j\rangle, s_{i,j})|j = 1, \ldots, A_i\}$ where $\langle i, j\rangle$ represents the action for gateway $i$ that maps a job to the $j$th chain, $A_i$ is the number of chains available to the controller to select from, and $s_{i,j}$ is a score assigned to this chain mapping, which is the primary driver of the control policy. The optical control policy is then deployed to the local controllers at the gateways and executed during runtime.

In more detail, during the offline design of the policy, we assume that $A_i$ equals all possible paths from gateway $i$. A CEED local controller shipped on the gateway $i$ at runtime maps incoming jobs to chain $j$ according to a probability $p_{i,j}$ proportional to $s_{i,j}$, computed by the sparse-softmax

$$p_{i,j} = \begin{cases} \dfrac{\exp s_{i,j}}{\sum_{k\in\Omega_i} \exp s_{i,k}} & j \in \Omega_i, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

where $\Omega_i$ is the set of chains associated to the top-$k$ values of $s_{i,j}$. In the case of ties, these are broken at random. Thus, at runtime, the local controller will only have at its disposal $k$ actions, namely the mapping to the $k$ chains with the largest scores $s_{i,j}$. Hence, $\Omega_i$ reduces the computational cost of enacting (and possibly re-training) CEED at runtime.

### A. Illustrative example

Fig. 2 illustrates the control problem for an architecture consisting of $K$ layers. From left to right, three chains (Chain 1/2/3) are available to process jobs that arrive at the gateway $G$. Each chain features a different (maximal) path across the following edge layers. As the job routes through the system along its assigned chain, which is decided by a local controller on the gateway, it may exit early at any intermediate edge device, depending on the ICs on that device.

[3]For example, an early IC that filters out data that is simple to classify, will affect the data distribution seen by later ICs.

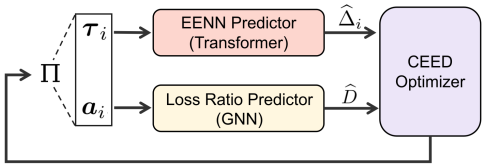



Fig. 3: **CEED overview**. The control policy $\Pi$ consists of confidence threshold configurations $\boldsymbol{\tau}_i$ and chain assignment policy $\boldsymbol{a}_i$, for all entry gateways $i = 1, \ldots, I$, evaluated by the EENN predictor and by the Loss ratio predictor. The policy is updated iteratively by the optimizer with the estimated mean confidence score $\widehat{\Delta}_i$ and system loss ratio $\widehat{D}$.

To explore the dependencies among confidence thresholds and chain assignment in a system of this kind, we have run experiments on the system in Fig. 2, with $K = 2$ edge layers and $L = 9$ EENN layers. We test two different threshold configurations $\tau_i$ and $\tau'_i$ for jobs that enter at gateway device $i$ for inference on a ResNet50 EENN model, described later in Section VI. The results in Table II indicate that it is possible for threshold configurations to significantly influence both the accuracy and the latency of jobs, even when these are assigned to the same chain, i.e., follow the same route through layers and devices. The assignment to chain also has significant effects, for example, on latency. The early exit mechanism introduces a negative correlation between inference accuracy and latency, thus defining a trade-off among these metrics. Thus, for effective EENN operation, a control policy is needed to assign both chains and EENN thresholds to incoming jobs, in order to reach the desired QoS goals in terms of accuracy, latency, while reducing the loss ratio.

## IV. CEED Prediction Models

An overview of the CEED methodology is shown Fig. 3. CEED allows a system administrator to decide a control policy $\Pi$. Such policy is then enacted at runtime by the local controller on each gateway device. The only computation that takes place at runtime is the randomized decision, based on the probability distribution $p_{i,j}$, $j = 1, \ldots, A_i$, of the chain assigned to an incoming job at gateway $i$.

To characterize the complex dependencies between QoS metrics in distributed EENNs, CEED relies on two deep learning models, namely the *EENN predictor*, based on a decoder-only transformer, and the *Loss ratio predictor*, based instead on a GNN model. The EENN predictor estimates the mean confidence scores $\widehat{\Delta}_i$ that the EENN will deliver under the selected threshold configuration $\boldsymbol{\tau}_i$ for each gateway $i = 1, \ldots, I$. Such values are later used to estimate the overall EENN classification accuracy under the policy $\Pi$. Concurrently, the Loss ratio predictor estimates the system loss ratio $D$ resulting from the chain assignment policies $\boldsymbol{a}_i$.

The CEED optimizer uses the mean confidence scores $\widehat{\Delta}_i$ and system loss ratio $\widehat{D}$ estimated by the predictors as inputs

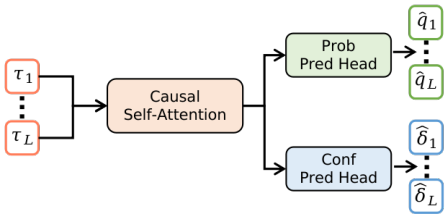



Fig. 4: **EENN predictor architecture**. The EENN predictor takes the threshold configuration $\boldsymbol{\tau}_i$ as input and predicts the corresponding exit probability $\widehat{q}_{i,l}$ and mean confidence score $\widehat{\delta}_{i,l}$ at each IC.

to an optimization program, given later in Eq. 6, which seeks to find the optimal policy $\Pi$ for a given deployment of the EENN over the multilayer edge architecture. Details of the underpinning optimization program are given in Section V.

### A. Transformer-based EENN predictor

To assess the mean confidence score of the EENN given a $\boldsymbol{\tau}_i$, we define the EENN predictor, as shown in Fig. 4, which takes the threshold configuration $\boldsymbol{\tau}_i$, as input and predicts the exit probabilities $\widehat{q}_{i,l}$ and confidence score $\widehat{\delta}_{i,l}$ of each IC and further predicts the mean confidence score $\widehat{\Delta}_i$.

In the execution pipeline of the EENN, confidence thresholds are sequentially applied from the first IC to the last one. Therefore, we model the input thresholds as a sequence. Moreover, predicting the confidence score and exit probability can be viewed as sequence-to-sequence learning, with each threshold $\tau_{i,l}$ associated with a unique confidence score and exit probability. Attention-based transformers have demonstrated remarkable performance in processing time series and natural language tasks [18]. Therefore, we design the EENN predictor as a decoder-only transformer, which is more computationally efficient than a full transformer, while still featuring high accuracy, to leverage these capabilities at runtime.

The threshold configuration $\boldsymbol{\tau}_i$ is first processed with a linear layer to project $\tau_{i,l}$ into an embedding representation. As the self-attention mechanism does not encode positional information, a position encoding derived from sine and cosine functions of different frequencies is added to the embedding representation. The encoded embeddings are processed with a causal self-attention function:

$$\text{Attn}(\boldsymbol{\tau}_i) = \text{softmax}\left(\frac{\Phi_Q(\boldsymbol{\tau}_i)\Phi_K(\boldsymbol{\tau}_i)^T}{\sqrt{d_{model}}} \circ H\right)\Phi_V(\boldsymbol{\tau}_i) \quad (2)$$

where $\Phi_Q$, $\Phi_K$ and $\Phi_V$ represent linear layers, $d_{model}$ denotes the hidden dimension, $\circ$ is the Hadamard product, and $H \in \mathbb{R}^{L\times L}$ is the causal mask with $h_{m,n} = 1$ for $m \leq n$ and 0 otherwise. This causal masking ensures that later thresholds can only access information from earlier thresholds, reflecting the dependency where the confidence score and exit probability of a later IC are contingent upon earlier ICs, but not vice versa. The outputs from the causal self-attention modules are processed by the confidence and probability prediction heads, both formed with linear layers and ReLU activations,

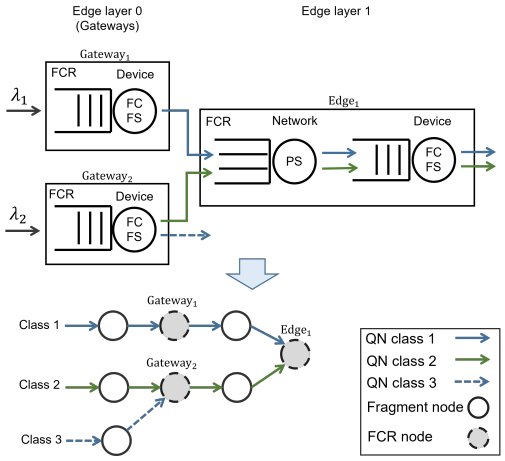



Fig. 5: **Queueing model and its graph representation used to train the GNN-based Loss ratio predictor**. The upper graph presents an example QN model for a system deploying the distributed EENN. FCR models the memory constraints at the three devices. The lower part illustrates the input graph representation of the QN to the GNN.

to determine the final confidence scores and exit probabilities for each IC.

The model is trained using a combination of a Mean Squared Error (MSE) on the predicted confidence score with a Kullback-Leibler (KL) divergence on the predicted exit probabilities, the latter computed after a softmax function to ensure that the sum of the exit probabilities always equals 1. The training loss function $\ell$ used to train the EENN predictor is expressed as

$$\ell = \mathbb{E}\left(\frac{1}{L}\sum_{l=1}^{L}(\delta_{i,l} - \widehat{\delta}_{i,l})^2 - \sum_{l=1}^{L} q_{i,l}\log\frac{\widehat{q}_{i,l}}{q_{i,l}}\right) \quad (3)$$

where $\delta_{i,l}$ and $q_{i,l}$ are labeled confidence score and exit probabilities, respectively. The labeled data of a threshold configuration are obtained from data that are not employed during the training of the EENN.

### B. GNN-based Loss Ratio Predictor

One difficulty associated with predicting loss ratios is that this metric strongly depends on the stochastic behavior of both the edge system and the network, and it is also affected by the early exit configuration. In alignment with a recent trend of using GNNs for network modeling [19], we define a GNN surrogate that can accurately predict loss ratios for any given set of exit probabilities of the jobs at the EENN layers. The GNN is trained on simulated loss ratios for a class of stochastic queueing networks (QNs) with finite capacity constraints, which is hard to deal with analytically as the underlying Markov chain is not known to have a product-form analytical solution [20].

*1) Simulation model:* Fig. 5 gives an example of QN that is used to train the predictor by means of simulation. This

model characterizes each device within the system as a queueing station surrounded by a Finite Capacity Region (FCR) representing a finite limit on the device memory capacity. The FCR is configured so that it drops jobs that would overflow the remaining memory capacity, resulting in data loss. Moreover, jobs are allowed to have different sizes. Scheduling at the queues representing devices is assumed to be FCFS, similarly to Fig. 1. Network bandwidth is modeled by a Processor Sharing (PS) queue, however, other policies may be adopted in principle to better describe the physical testbed and network characteristics. Using such variations only requires generating a different simulation-based training set for the GNN model. The QN classes are defined by a triplet $\langle i, j, l \rangle$, each mapping to a single chain and early exit point in the EENN.

*2) GNN surrogate model:* To the best of our knowledge, no class of surrogate models exists that is tailored to the multiclass QNs with FCRs of the kind described above. The closest proposal in the literature is the ChainNet GNN [21]. The key mechanism of ChainNet is a customized message-passing mechanism that passes messages along the chain and within each chain step. The ChainNet GNN is, however, less general than the one proposed here. Despite relying on the same message-passing mechanism, our GNN covers missing features that are critical for the EENN problem at hand, namely: 1) FCRs to describe memory capacity; 2) network bandwidth modeling via PS nodes; and 3) a method to describe early exits within the GNN. Thus, CEED generalizes this baseline by modeling all three missing features into the GNN that underpins the CEED loss ratio predictor, as we detail in the next subsections.

In the CEED GNN model, we call *fragment* the subset of layers executed on the same device for a specific QN class $\langle i, j, l \rangle$. This is an integral element of the GNN message-passing algorithm involved in modeling the computational processing. As shown in Fig. 5, the processing of a job fragment involves two phases: network transmission and layer execution, in the simulation model. This two-phase processing is modeled in the input graph passed to the GNN by a node named *fragment node*, which has as features the network transmission time $d_f$, processing time $t_f$, and the input data size for the fragment $f$. These attributes define the behavior of the fragment within the device. Meanwhile, since a device is modeled as an FCR, a corresponding node, named ***FCR node***, is included in the input graph to host the fragment. The features of these nodes are the device memory capacity and the maximum network bandwidth. Directed edges are then annotated in the input graph for each job class, specifying the fragment served by the edge device contained in the FCR. The GNN model predicts the throughput of each fragment and the system loss ratio.

*3) Memory consumption modeling:* To model the memory limit within the FCR, we subtract from the available device memory the component required to deploy the corresponding layers on the device based on the given placement. Consider an arbitrary edge device $k$ in the infrastructure. Assume that the fragment $f$ running on the device $k$ has a memory usage $m_{k,f}$. The FCR of device $k$ enforces at all times that $\sum_f m_{k,f} n_{k,f} \leq M_k$, where $M_k$ is the maximum allowed memory usage on device $k$ and $n_{k,f}$ is the current number of jobs stored in memory at $k$ that need processing with fragment $f$.

This constraint is enforced at all times in the simulation so that the training dataset of the GNN learns the impact of memory limits on latency and loss ratios.

*4) Network modeling:* Our GNN models network impact on inference in two ways. First, CEED continuously profiles the network transmission time of jobs and supplies the reciprocal of this value as the service rate of the PS queues in the input graph passed to the GNN. Next, the GNN models CPU contention for network transfer on the devices running the EENN using a novel fixed-point iteration, where the GNN predicted throughputs are used to dynamically adjust the input estimates of the device process times. At iteration $z+1$, the processing time on a device is computed as

$$t_f^{(z+1)} = \frac{t_f^{(0)}}{1 - \big( \underbrace{\sum_{f' \in \mathcal{F}_{in}} d_{f'} X_{f'}^{(z)}}_{\text{transmit in}} + \underbrace{\sum_{f' \in \mathcal{F}_{out}} d_{f'} X_{f'}^{(z)}}_{\text{transmit out}} \big)} \tag{4}$$

where $t_f^{(z)}$ is the processing time of fragment $f$ at the $z$th iteration, and $t_f^{(0)}$ is the (offline profiled) processing time of the layer. Additionally, $X_f$ is the throughput of fragment $f$ predicted by the GNN. Here, $d_f$ is the profiled transmission time of the data required by the fragment $f$, $\mathcal{F}_{in}$ are instead the fragments for which the device needs to receive data to execute, and $\mathcal{F}_{out}$ are the fragments for which the device sends data to another device for execution. This iteration stops when the adjusted processing time converges, or the number of iterations reaches the predefined maximum limit. The Loss ratio predictor predicts the throughput and system loss ratio using the last updated processing time $t^{(Z)}$.

*5) Modeling early exits in the GNN:* In order to model early exit, the GNN considers all possible chains $\langle i, j \rangle$ by enumerating all feasible split points in the EENN and possible placements across the edge system. This set can be restricted in size at will based on EENN size and edge testbed constraints; here we assume to require that at most a single device is used per edge layer.

The GNN then needs to decide the arrival rate for all of its classes. Each class describes the chain and the exit layer $l$ of the job, thus this is simply

$$\lambda_{\langle i,j,l \rangle} = \lambda_i p_{i,j} q_{i,l}. \tag{5}$$

where $q_{i,l}$ is the exit probability along the $l$th IC, a term that is estimated by the transformer-based EENN predictor. With this parameterization, the QN model is completely parameterized and the GNN is able to produce the required throughput and loss ratio predictions for each QN class. These are then used to obtain aggregate quantities, in particular the overall system loss ratio $\hat{D}$, which is defined as one minus the ratio of the system throughput across all QN classes to the total arrival rate at the gateways.

**Algorithm 1** CEED Objective function

**Input** $\boldsymbol{\tau}_i$, $\boldsymbol{a}_i$ for $i = 1, \ldots, I$

1: **function** REWARD
2: $\widehat{\Delta}_i$, $\widehat{q}_{i,l} \leftarrow$ EENN predictor in Section IV-A.
3: Select chains with top-$k$ assigned score
4: $p_{i,j} \leftarrow$ sparse-softmax($s_{i,j}$)
5: Build graph representation $G$ as the input to GNN
6: $\lambda_{\langle i,j,l \rangle} \leftarrow$ Eq. 5 using $p_{i,j}$, $\widehat{q}_{i,l}$
7: $\widehat{D} \leftarrow$ Loss ratio predictor in Section IV-B.
8: $r \leftarrow$ Eq. 6 with $\widehat{\Delta}_i$, $\widehat{D}$
9: **return** $r$

## V. CEED ALGORITHM

We are now ready to leverage the deep learning models proposed in the last section. The threshold configurations $\boldsymbol{\tau}_i$ and the assigned scores $s_{i,l}$ within $\boldsymbol{a}_i$ are randomly initialized from a uniform distribution between 0 and 1. These variables, $\boldsymbol{\tau}_i$ and $\boldsymbol{a}_i$ for every i, are then used as inputs for calculating the objective function in CEED, as detailed in Algorithm 1. The confidence scores $\widehat{\Delta}_i$ and the exit probabilities $\widehat{q}_{i,j}$ of ICs are predicted by the EENN predictor for each $\boldsymbol{\tau}_i$ (line 2).

For each $\boldsymbol{a}_i$, chains with the top-$k$ assigned scores are selected to determine the placement. The probabilities of selecting a chain are computed using Eq. 1 (line 3-4). Job classes for modeling the early exit and the graph representation of the system are generated based on the chains with top-$k$ assigned scores and the placement (line 5). Arrival rates for each service are computed using Eq. 5 (line 6). The system loss ratio is then predicted by the Loss ratio predictor using the constructed graph and arrival rates (line 7). The objective is to balance the trade-off between the system loss ratio and the mean confidence score. Therefore, the objective function, denoted as reward $r$, is defined as follows (line 8):

$$r = \sum_{i=1}^{I} \left( \frac{\lambda_i}{\sum_{j=1}^{I} \lambda_j} \widehat{\Delta}_i \right) - \gamma \max(\widehat{D} - D^*, 0) \qquad (6)$$

where $\lambda_i$ is the arrival rate of jobs at the $i$th gateway. $D^*$ denotes the acceptable loss ratio, and $\gamma$ is a penalty parameter. Both $D^*$ and $\gamma$ are user-defined values determining the trade-off between the mean confidence score and the system loss ratio. The mean confidence score of the system is calculated as the weighted sum of the mean confidence scores of jobs arriving at each gateway, adjusted according to their respective arrival rates.

The objective of the optimization process is to maximize the reward $r$. To achieve this, we apply gradient descent (GD) to iteratively minimize $-r$ by adjusting the threshold configurations $\boldsymbol{\tau}_i$ and the assigned scores $s_{i,j}$. The mean confidence scores $\widehat{\Delta}_i$ are derived from the EENN predictor, a differentiable function with the threshold configuration as its input, allowing for the computation of gradients with respect to $\boldsymbol{\tau}_i$. Similarly, since the assigned scores influence the arrival rates of services as outlined in Eq. 5, and these arrival rates are inputs to the Loss ratio predictor, it is feasible to compute gradients with respect to $s_{i,j}$. The gradients of the reward with respect to $\boldsymbol{\tau}_i$ and $s_{i,j}$ are then calculated using the chain rule. In practice, these gradients are computed via back-propagation, which is similar to training the parameters of a neural network. Therefore, the threshold configurations $\boldsymbol{\tau}_i$ and assigned scores $s_{i,j}$ are updated with momentum-based methods such as Adam [22], which can enhance the optimization process and avoid getting stuck in local optima.

The optimization iteration is terminated when the absolute difference between the consecutive rewards falls below a user-defined threshold or when the iteration count reaches the predetermined maximum iteration.

## VI. EVALUATION

### A. Evaluation Methodology

*1) EENNs:* The EENNs used in the experiment are based on three different CNNs, namely: a) *Custom CNN*, a 27-layer CNN with 3 ICs (comprising two linear layers); b) *ResNet50* [2], with 16 residual blocks and 9 ICs where, due to the residual connections, the ICs can only be placed between the residual blocks; c) *ResNet101* [2], which has similar structure as ResNet50, but with 33 residual blocks and 33 ICs.

*2) Image datasets:* Two image classification datasets are used to train the EENNs: a) *CIFAR10* [23]: comprising 60,000 images in 10 classes. From this dataset, 5,000 images from the test set are used as the workload for the system. The custom CNN and ResNet50 are trained and tested with CIFAR10. b) *ImageNet-1K* [24]: comprises over 1 million images in 1,000 classes. From this dataset, 10,000 images from the validation set are used as the workload for the system. ResNet101 is trained and tested with this dataset.

*3) System setup:* We have built a 3-layer physical testbed with one gateway layer and two edge layers ($K = 2$). Four Raspberry Pi 4B Rev 1.1, each equipped with a 300MB memory capacity and a CPU BogoMIPS index of 108, serve as gateways where images are collected and processed and run the control policy $\Pi$. The middle layer ($k = 1$ in Fig. 2) is referred to as near-edge (NE) and consists of three Raspberry Pi 4B Rev 1.4, each equipped with a 350MB memory capacity and a CPU BogoMIPS index of 108. The last layer ($K = k = 2$ in Fig. 2) is referred to as the far-edge (FE) layer, and includes a MinisForum Z83-F equipped with a 400MB memory capacity and a CPU BogoMIPS of 2880. Gateways and the NE layer are connected via a fast Ethernet switch, providing 100Mbps of dedicated bandwidth. Data is transmitted from the NE layer to the FE layer over WLAN at a speed of 30 Mbps.

*4) Baselines:* Most baselines in the literature consider partitioning an NN across a small subset of parts. We thus consider four EENN deployments: gateway-only, NE-only, FE-only, and split operation using a basic implementation of Neurosurgeon [11], which we denote as NE+FE. The first three partition methods specify that the job will be processed exclusively on a specific layer. In contrast, the NE+FE deployment partitions a neural network into two parts that are processed on NE and FE layers. In our implementation, the

TABLE III: Hyperparameters for generating QN simulations

| Hyperparameter | Value | Hyperparameter | Value |
|---|---|---|---|
| Num. of devices | [1, 20] | Arrival rate | [1, 10] |
| Num. of layers | [1, 5] | Process time | [0.001, 1] |
| Num. of chains | [3, 350] | Transmit time | [0.001, 2] |
| Memory size | [2MB, 128MB] | Job size | [0.1MB, 10MB] |

last method explores all possible EENN splitting points to find the partition that achieves the shortest inference time.

Three baselines are compared to CEED's $\Pi$ policy:

*Rate-based* single-exit policy [8] adapts job exit points based on arrival rate and buffer length, which is proportional to the remaining number of jobs that can be stored in the memory.

*AdaEE*, [25]. This is a recent reinforcement learning (RL) algorithm that adapts dynamically confidence thresholds with the highest reward, defined from the difference in confidence between ICs. We generate 10,000 threshold configurations as the action space for RL, pre-trained on the same image set used to train our EENN predictor.

*Last-based*. The no early exit policy, where all jobs exit at the last layer, serves as another baseline. This approach processes images using the CNN model without any ICs, and classifications are made only at the final classifier, thus typically with the largest accuracy.

*5) CEED model training:* To train the EENN predictor, 10,000 thresholds are randomly generated for each type of EENN. About 5,000 images from CIFAR10 and 40,000 images from ImageNet-1K are used to evaluate these thresholds. $70\%$ of the generated thresholds is used for training, and $30\%$ for testing. The final model size of the EENN predictor is 864KB.

The Loss ratio predictor is trained from about 70,000 QN simulations generated with parameters chosen uniformly at random in Table III. We use JSIMgraph [15] as the simulator. The maximum iteration number for the fix-point iteration is 10 and the convergence threshold is set to $1 \times 10^{-4}$. The final model size is 2.7MB.

To obtain the policy $\Pi$, we use the CEED optimization parameterized as follows. The loss ratio constraint is set to 0, aiming to prevent any job losses. The penalty parameter $\gamma$ for the system loss ratio is 2. Each update step uses a learning rate of 0.1. In the optimization procedure, the convergence threshold for the reward is $1 \times 10^{-4}$, with a maximum of 500 iterations. We consider the top $k = 10$ assigned scores in the sparse-softmax function. The time for obtaining the control policy using CEED for custom CNN, ResNet50 and ResNet101 are 65, 79 and 247 seconds, respectively, running on an AMD Ryzen 7 PRO 3700 PC.

*6) Experimental protocol:* We inject images into the testbed spaced by exponential inter-arrival times forming a Poisson process with rate $\lambda = \rho/t_{max}$, where $\rho$ is the load factor, and $t_{max}$ is the maximum processing time of the EENN at the gateway, which in our test is the slowest device.

We evaluate the performance of CEED by comparing it with the three baseline methods in terms of classification accuracy, system loss ratio, and mean slowdown, defined as $R/t_{max}$ where $R$ is the mean response time of the jobs. Slowdown is preferred to response time to better account for the difference in baseline execution speed of the three different EENNs we consider. Each experiment runs for 30 minutes to ensure it reaches a steady state, with measurements recorded over an additional 30-minute period.

### B. Results

*1) Predictors Evaluation:* We first discuss the performance of the CEED transformer and GNN, using the test set of threshold data we generated. MSE is used as the performance metric. For the EENN predictor, the MSE on the confidence scores for custom CNN, ResNet50, and ResNet101 are all low: $0.0099$, $0.0101$, and $0.01905$, respectively. The MSE on the predicted exit probabilities is also low, at $0.00030$, $0.00015$, and $0.00025$, respectively. To assess the performance of the Loss ratio predictor in a real system, we conduct 100 experiments using ResNet50 with various system topologies, arrival rates, and memory. The system loss ratio for each configuration is measured as labeled test data. ChainNet and our CEED GNN are then used to predict the system loss ratio. The Mean Absolute Error (MAE) for the loss ratio is $0.0950$ for ChainNet but reduces to $0.0718$ for CEED's Loss ratio predictor, thus obtaining a $24\%$ improvement.

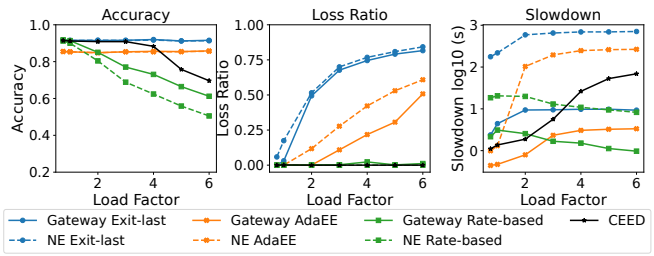
Summarizing, the prediction validations indicate that both CEED predictors generally feature high predictive accuracy on their test sets.

*2) Synthetic Arrival Processes:* First, we show the performance of CEED under various target load factors $\rho \in \{0.75, 1, \ldots, 6\}$, from stable to highly overloaded, leading to increasing losses. This experiment uses the CIFAR10 dataset with ResNet50. Fig. 6a compares systems where baselines are running on the gateway and NE, while the baselines in Fig. 6b are distributed across all testbed layers.

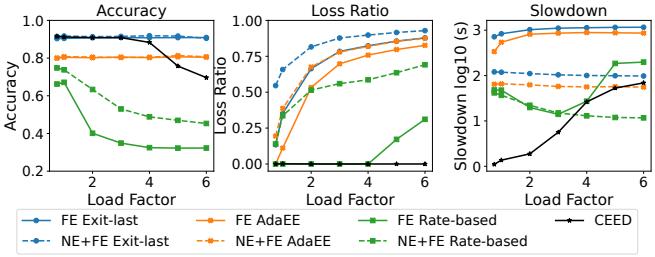
When the gateway layer is stable with a load factor of 0.75, CEED and the baselines distributed on gateway-only and NE can all achieve high accuracy and low loss ratios. However, the system loss ratio and slowdown increase when the EENN runs only on the FE or on NE+FE. This is because the arrival rate at the FE device is four times higher than at the gateway, while the execution speed at the FE device does not sufficiently compensate for this increase.

As the load factor increases, the gateway layer becomes overloaded. This leads to an increase in the system loss ratio for baselines using AdaEE and the exit-last strategy, though their accuracy remains at the same level. This is because these methods do not have a mechanism to prevent job losses. In contrast, the gateway-only and NE rate-based methods effectively prevent job losses but reduce accuracy to 0.77 and 0.68, respectively, at a load factor of 3.

CEED's methodology, instead, thanks to the dynamic selection of the chains, is not forced to serve jobs in a static partitioning configuration, as different input jobs may follow different paths in the EENN. This ultimately reduces job losses and achieves a higher accuracy of 0.91, better than the rate-based methods, while significantly reducing slowdown compared to exit-last methods.

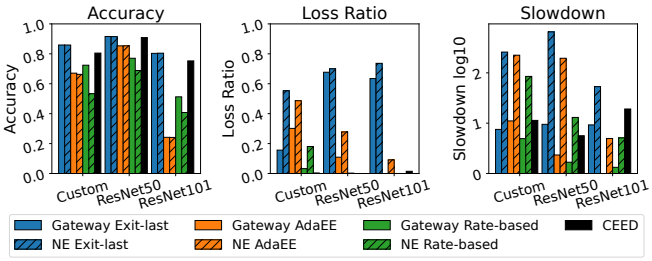



(a) Gateway-based and NE-based methods

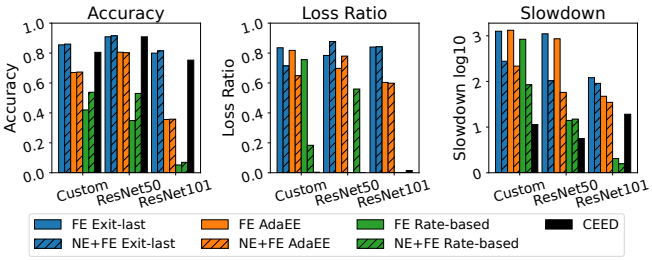



(b) FE-based and NE+FE-based methods

Fig. 6: Performance metric with different load factors




(a) Gateway-based and NE-based methods




(b) FE-based and NE+FE-based methods

Fig. 7: Performance metric with different sizes of EENNs

*3) Scalability Evaluation:* Lastly, we assess the scalability of CEED by evaluating EENNs of varying size in terms of the number $L$ of layers and ICs. The custom EENN is the smallest configuration, ResNet50 is larger, and ResNet101 is the largest. This experiment focuses on a load factor of $\rho = 3$. Fig. 7 shows that CEED consistently achieves higher accuracy than AdaEE and rate-based methods while minimizing job losses across all EENNs. CEED also has a significantly lower slowdown compared to last-based methods. Although the last-based method is more accurate, it has the highest loss ratio. Notably, the accuracy of the AdaEE-based method approaches that of the last-based and CEED in the smaller custom EENN and ResNet50 configurations. However, their performance degrades in the larger-size ResNet101 EENN due to the scalability limitations of the AdaEE action space.

## VII. Related Work

*EENNs on Single Devices*. Several studies have focused on techniques to leverage early exists when running on single devices. For instance, BranchyNet [5] and SPINN [9] accelerate inference by adding side branches into the DNN architecture with early exits controlled by predefined confidence thresholds. Additionally, JEI-DNN [13] and EPNet [26] incorporate additional neural network modules that autonomously make exit decisions, but this approach requires extra training.

Recent works also propose methods for dynamically finding the optimal threshold configuration of an EENN. [8] transforms an EENN with multiple ICs to a single-exit EENN for analytical tractability and applies a scheduling policy before execution. A reinforcement learning (RL) approach named AdaEE [25] selects the confidence threshold from a predefined threshold set based on the reward, which is defined as the confidence gain between the exit IC and the preceding ICs. In contrast to the above works, CEED explores and leverages deep learning models to optimize IC threshold configurations.

*Distributed EENNs*. Prior works on collaborative inference at the edge include Neurosurgeon [11], ANS [27] and the binary-search-based algorithm in [28], which employ brute-force search and heuristics to find the optimal partitioning of a DNN, so as to minimize inference times. However, these methods are not specifically designed to address EENNs. Works such as SPINN [9], Edgent [10], and FIN [29] address this issue. Recently, [30] presents an RL approach to find an optimal EENN partition point based on system state. Compared to these works, CEED extends partitioning across a multilayer edge system not necessarily operating on just the 2/3 layers considered in the above works, which is useful to support the ongoing trend of deploying DNNs on collections of several small devices [31]. Moreover, existing studies often focus on individual performance metrics and neglect loss ratios. Instead, CEED addresses all these aspects.

## VIII. Conclusion

This paper tackles collaborative edge inference using EENNs. CEED controls threshold configurations of the ICs within the EENN and runtime assignment of jobs to paths inside the EENNs (chains). The policy is obtained through an optimization that strives to maximize the accuracy of inference while minimizing the loss ratio, relying on the predictions of AI models based on a decoder-only transformer and a GNN. The evaluation results show that CEED outperforms existing methods for EENNs, achieving better QoS. Future work may explore the impact of concept drift in jobs and explore chains that extend up to the cloud datacenter.

## REFERENCES


[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," vol. 521, no. 7553, pp. 436–444.

[2] 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.

[3] J. Liu and Q. Zhang, "To improve service reliability for AI-powered time-critical services using imperfect transmission in MEC: An experimental study," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9357–9371, 2020.

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[5] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[6] D. Stamoulis, T.-W. R. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu, "Designing adaptive neural networks for energy-constrained image classification," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[7] S. Disabato and M. Roveri, "Reducing the computation load of convolutional neural networks through gate classification," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[8] G. Casale and M. Roveri, "Scheduling Inputs in Early Exit Neural Networks," *IEEE Transactions on Computers*, 2023.

[9] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[10] 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 Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[11] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," vol. 45, no. 1. ACM New York, NY, USA, 2017, pp. 615–629.

[12] E. Baccarelli, S. Scardapane, M. Scarpiniti, A. Momenzadeh, and A. Uncini, "Optimized training and scalable implementation of Conditional Deep Neural Networks with early exits for Fog-supported IoT applications," *Information Sciences*, vol. 521, pp. 107–143, 2020.

[13] J. Chataoui, M. Coates *et al.*, "Jointly-Learned Exit and Inference for a Dynamic Neural Network," in *The Twelfth International Conference on Learning Representations*, 2023.

[14] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[15] M. Bertoli, G. Casale, and G. Serazzi, "JMT: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.

[16] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, and R. S. Couto, "Early-exit deep neural network-a comprehensive survey," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–37, 2024.

[17] J.-W. Kim and H.-S. Lee, "Early exiting-aware joint resource allocation and dnn splitting for multi-sensor digital twin in edge-cloud collaborative system," *IEEE Internet of Things Journal*, 2024.

[18] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[19] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-fermi: Network modeling with graph neural networks (2022)," *arXiv preprint arXiv:2212.12070*.

[20] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[21] Z. Niu, M. Roveri, and G. Casale, "ChainNet: A Customized Graph Neural Network Model for Loss-aware Edge AI Service Deployment," in *DSN 2024 - IEEE International Conference on Dependable Systems and Networks*. IEEE.

[22] D. P. Kingma and J. Ba. (2014) Adam: A method for stochastic optimization.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[25] R. G. Pacheco, M. Shifrin, R. S. Couto, D. S. Menasché, M. K. Hanawal, and M. E. M. Campista, "AdaEE: Adaptive early-exit DNN inference through multi-armed bandits," pp. 3726–3731, 2023.

[26] X. Dai, X. Kong, and T. Guo, "EPNet: Learning to exit with flexible multi-branch network," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 235–244.

[27] L. Zhang, L. Chen, and J. Xu, "Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning," in *Proceedings of the Web Conference 2021*, 2021, pp. 3111–3123.

[28] Y. Duan and J. Wu, "Joint optimization of DNN partition and scheduling for mobile cloud computing," in *Proceedings of the 50th International Conference on Parallel Processing*, 2021, pp. 1–10.

[29] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini. (2024) Resource-aware Deployment of Dynamic DNNs over Multi-tiered Interconnected Systems.

[30] 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.

[31] S. Disabato, M. Roveri, and C. Alippi, "Distributed deep convolutional neural networks for the internet-of-things," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1239–1252, 2021.