Scheduling Inputs in Early Exit Neural Networks

Giuliano Casale, Manuel Roveri

***Abstract*—Early exit neural networks (EENs) reduce the processing times of deep convolutional neural networks by means of internal classifiers (ICs) that allow jobs, being the input of the EEN, to exit early from the processing pipeline. However, the current designs used in pervasive systems ignore variability in data arrival rates, exposing EEN-based services to potential loss of the incoming jobs, due to finite input buffer capacity. Motivated by this issue, we introduce and study the *early exit* scheduling problem, which aims at dynamically configuring IC thresholds at runtime to achieve effective trade-offs between job classification accuracy, processing time, and job loss ratio. We argue that deciding the EEN exit layer for a job at the start of its processing makes the problem mathematically tractable, allowing us to develop policies to control buffer backlog, classification accuracy, and processing time across the EEN layers. The main contribution of the paper is the introduction of single-exit IC threshold configurations as a mechanism to allow the scheduling policy to reliably predict the best EEN exit layer of each input job. Three scheduling policies that leverage this idea are proposed to dynamically schedule job arrivals to an EEN-based service. The proposed solution, here tailored to EENs based on convolutional neural networks (CNNs), is fairly general and can be applied to different use cases. The two application scenarios considered in this paper focus on image classification and intrusion detection. Experiments on some popular CNNs for the two aforementioned application scenarios indicate that the proposed policies can achieve significant savings in processing times and improve job loss ratio compared to both ordinary EENs and CNNs while still providing high mean classification accuracy.**

***Index Terms*—Neural network, early exit, configuration, threshold, scheduling, dependability.**

I. Introduction

In recent years Convolutional Neural Networks (CNNs) [1] have emerged as the state-of-the-art solution in many classification, recognition, and detection applications [2]. The key strength of CNNs resides in their ability to process inputs through the processing pipeline by progressively extracting features characterized by increasing complexity and meaning [3].

However, the standard CNN architecture, which typically comprises many heterogeneous processing layers, has been recently questioned [4], [5]. First, recent studies show that most of the input data can be correctly classified in CNNs by resorting only to a sub-sequence of the whole processing pipeline, hence potentially saving on processing time and energy consumption, two relevant issues in edge devices (e.g., Internet-of-Things units, embedded devices or edge computing units) that are technologically constrained in memory, computation, and energy [4], [5]. Next, CNNs are known to suffer overthinking effects, whereby a correct prediction is changed into a misclassification during the pipeline processing [6]. In such cases, for subsets of inputs, shallower processing pipelines can be more accurate.

Early exit neural networks (EENs) have been recently introduced to address these limitations [4], [5], [7], [8]. EENs incrementally process the input data and can provide the output either at the end of the processing pipeline or at an earlier stage using *internal classifiers* (ICs). ICs autonomously decide whether an input should progress further in the processing pipeline based on a confidence indicator of output reliability, e.g., estimated posterior probabilities [6], [8], normalized entropy [5], or distance metrics [4]. Thanks to their flexibility, ICs are already present in some of the most popular CNNs, e.g., the Inception architecture [9], where they are primarily used to increase classification accuracy. Moreover, EENs are widely seen as a promising algorithmic and technological solution for a wide range of application scenarios including pervasive image classification and pattern-recognition, natural language processing, on-device recognition of user activities or status, and intrusion/anomaly detection in pervasive networks [5], [10].

While research on EENs is growing steadily in the last few years [4]–[8], [11], EEN designs are not yet engineered for dependability. Current approaches normally assume that the arrival rate of input data is fixed (e.g., periodic) and that the classification output is provided before the appearance of the next input. Designers typically address this need through resource oversizing to ensure that the time between the arrival of two consecutive inputs is always larger than the end-to-end processing time of the entire pipeline. Thus, whenever data arrival rates exceed the design assumptions, losses may be observed if the capacity of the input buffer is exceeded. Depending on the specific target application, such losses may have various impacts, ranging from performance deterioration [12] to reduced dependability [13].

To address these issues, we introduce and study the *early exit scheduling problem*. The problem considers EENs where input jobs (representing the input data to be processed by the EEN) arrive stochastically and wait in an input buffer until processing at the EEN starts. Whenever the input buffer is full, newly arrived jobs are lost. The problem is defining a scheduling policy to dynamically configure the IC thresholds at runtime to strike the desired balance between the job loss ratio and EEN performance metrics such as classification accuracy, job completion times, or buffer backlog size.

A key challenge in addressing this problem is the ability to handle the complex inter-dependencies between IC threshold values, which arise due to the fact that early exit at an IC changes the data characteristics seen at later ICs. For

G. Casale is with the Department of Computing, Imperial College London, London, UK, SW7 2AZ.
E-mail: g.casale@imperial.ac.uk
M. Roveri is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 27100 Milan, Italy.
E-mail: manuel.roveri@polimi.it. This paper is partially funded by the Luxottica Smart Eyewear Lab (SEL) Project and supported by the PNRR-PE-AI FAIR project of the NextGeneration EU program.

example, accurate early classification implies that only hard-to-classify inputs reach later ICs. However, such a model is often impractical to develop, being input data dependent. To decouple scheduling from data characteristics, we propose a class of non-preemptive schedulers that decide the exit layer of jobs at their instant when the job starts its processing. With this approach, IC threshold settings can be more efficiently decided, making the EEN configuration simple to maintain at runtime, while retaining the ability to deploy advanced scheduling policies. Moreover, the mathematical problem underpinning scheduling is more tractable than under general threshold configurations by virtue of independence from data content.

It should be noted that deciding the early exit at the start of processing contrasts with the idea of dynamically evaluating, by means of the ICs, the accuracy of jobs as they progress through the EEN layers. We resolve this tension by showing that, if the classification accuracy for a job *conditional* on its exit layer can be profiled offline, which applies for example to independent and identically distributed (i.i.d) inputs, then under broad assumptions non-preemptive scheduling does not restrict in principle the achievable mean classification accuracy.

The ability to profile the conditional classification accuracy at the IC layers offline also relies on the assumption that the probability distribution of the data does not change over time. This is a quite general assumption in machine learning and consistent with our assumption that the process generating the data is i.i.d. over time.

Leveraging the above results, we propose reactive and proactive policies that can trade at runtime the expected classification accuracy of the EEN in return for lower processing times and job loss ratios. Our policies rely on different techniques and include: (i) a reactive list-based policy that continuously updates the exit point of the incoming job using a simple rule, (ii) a model-based policy that schedules jobs leveraging queueing theory and computational optimization, and (iii) a proactive rate-based policy that models the problem as a k-item knapsack problem [14] to choose the exit layer for a job based on an estimated time budget. Our policies differ for the information available for the scheduling algorithm and their implementation complexity. Using real data collected from actual EENs, we study the different characteristics of these policies and highlight their benefits and shortcomings for EEN scheduling.

In particular, we have successfully applied the proposed policies to a custom CNN and to two state-of-the-art CNNs used in image classification, i.e., the VGG16 and VGG19. Our validation also illustrates the applicability of our policies to EEN-based intrusion detection in network traffic. The proposed solution can be applied to newly-trained and pre-trained EENs, both tuned with transfer learning. We also show that unlike solutions in the literature that consider a limited number of ICs, typically in the order of six to ten, our proposal can be applied even to EENs endowed with several tens of ICs.

The main contributions of the paper are as follows:

- We introduce a novel job scheduling problem in EENs where input jobs arrive stochastically and a decision must be taken on how to best configure the IC thresholds for each incoming job to mitigate the adverse effects of finite buffer capacity;
- We propose a new family of EENs, called single-exit EENs, that under mild assumptions can achieve the same mean accuracy as traditional (multi-exit) EENs, while being much easier to model and control in the context of the proposed scheduling problem.
- Using the proposed single-exit EENs, we introduce multiple reactive and proactive policies to dynamically schedule jobs in order to balance job loss and accuracy of the completed jobs.

The rest of the paper is organized as follows. Section II presents related work. The problem statement and key definitions are introduced in Section III. Section IV follows with an introduction to the considered class of EEN configurations. Scheduling policies based are presented in Sections V, VI , and VII, respectively. Finally, experimental results are given in Section VIII, followed by conclusions in Section IX. The online supplement presents additional materials, including details on experimental parameters and additional sensitivity analyses.

II. Related Work

A. EEN-based systems

A relatively wide literature exists in the field of EENs. The advantages and weaknesses of the long pipeline processing in deep neural networks are explored in [6], discussing, in particular, the aforementioned overthinking phenomenon. To address this issue off-the-shelf CNNs are modified by introducing confidence-based ICs, i.e., the early exits. The works in [4] and [8] propose a joint training of neural networks and thresholding mechanisms to support adaptive neural networks. In both works, the arrival rates of the input are neglected and few ICs are manually allocated in the EEN by the CNN designer. [7] provides an overview of the main learning approaches for EENs, i.e., layerwise training and separate training. In addition, this work explores early exit placement techniques based on energy consumption or efficiency constraints. In this direction, a technological integration between EENs and Fog computing is proposed in [5], where distributed training and inference of neural networks with early exits is implemented in a multi-tier Fog platform, i.e., from IoT to Cloud. Similarly, [15] introduces a platform for the minimum-energy and delay-constrained execution of EENs in a stacked Fog computing ecosystem. Compared to that work, we consider continuous-time scheduling decisions for each job, whereas [15] assumes batched inputs. Also, we assume that the scheduler manages a single device and operates on the device itself. Thus, our simple policies unlike this prior work can only rely on algorithms that are both simple and inexpensive to run. Finally, [16] introduces a dynamic partitioning of the neural network pipeline between the edge device and the Cloud, based on traffic network congestion and edge device load. The authors also propose dynamic adaptation of the threshold of the ICs but the work neither models input arrival times nor input buffer capacity available to store incoming data. In addition, a common threshold is considered for all the early exits of the EENs.

In summary, while a wide literature about EENs exits, none of the available solutions considers inter-arrival times of jobs nor the buffer capacity of the input jobs. In addition, differently

from what here proposed, the EEN solutions present in the literature rely on multi-exit configurations, hence requiring the processing of all the ICs up the exit layer in the EEN.

B. Scheduling with controllable processing

Scheduling with controllable processing times is an area of deterministic scheduling with a large body of work, we point to [17] for a survey. In such a problem, the scheduling policy has control over the duration of the processing time of jobs in order to optimize some performance metrics. Closest to EENs is compressible scheduling with discrete resources (DSCR), which restricts the values of the processing time to a discrete set. In EEN, such values correspond to the processing times accumulated by jobs before leaving at the available ICs. Complexity results for DSCR are available in [18] for various performance metrics and with job release times, showing that the problem is in several settings NP-hard. However, for a problem involving the scheduling of n jobs, the simplest setting that minimizes the sum of total completion time and total processing cost in a given schedule can be solved in $O(n^3)$ time using an assignment problem.

The main difference of DSCR compressible scheduling compared to scheduling in EEN-based systems is that the latter feature an input buffer and uncertainty on the arrival instants (i.e., release times) of jobs. In a queueing setting, compressible scheduling often goes under the name of controllable service. The work in [19] develops an optimal Markovian policy for queues with unrestricted buffer size and Poisson arrivals. In this setting, there exists a finite set of service types and the scheduler accumulates a reward at a given rate until the next scheduling decision. However, unlike the present paper, the work assumes an infinite waiting buffer. Follow-up works such as [20] obtain similar results in a more general decision process, with applications to queues with general arrival or service distributions and multi-server queueing stations (e.g., $M/G/1$, $G/M/m$). Also in this case, unlike the present work, the analysis does not touch upon finite capacity systems.

Lastly, we note that several works have examined control of queues in an online setting, including situation where the arrival rate is unknown [21]. Such policies rely on the online estimation of the unknown quantities, by using for instance the maximum-likelihood estimators for single-server queues introduced in [22]. However, such techniques are typically used to parameter queues with i.i.d. inter-arrival times (e.g., $M/G/1$), whereas arrival processes in real EEN deployment are often non-i.i.d., as in the case of bursty or periodic arrivals. Some works have considered extended queuing models that address this difficulty, e.g., based on Markov-modulated arrival processes [23], however existing work in this area for finite capacity systems focuses on determining the target queue sizes across job types (classes) without allowing control of processing times [24]. A similar shortcoming applies to related work in the networking literature, for example in ATM and DiffServ management [25], [26], which typically assumes job classes (i.e., service types) to be immutable, instead of controllable as in the present paper.

Lastly, we mention the works in [27], [28], which although not falling into the realm of controllable scheduling, study a problem with some commonalities to it. In the papers, the optimal ordering of a set of classifiers needs to be chosen to minimize the processing times to a successful classification. It is shown that the optimal solution needs to satisfy a certain ordering among the ratios of classifier processing times to the probability of a successful classification. Unlike our work, the works in [27], [28] focus on ordering classifiers for a *single* job, whereas we consider a stream of input jobs arriving into the EEN-based system, thus exhibiting queueing and loss phenomena. Moreover, in our setting, the jobs are forced to progress sequentially through the classifiers, which cannot be reordered.

In summary, existing works consider different theoretical aspects relevant to scheduling in EEN-based systems such as discretely compressible processing times, buffer admission control, and online learning of arrival characteristics. However, there is a lack of a solution directly applicable to EEN-based systems that encompass all these characteristics altogether. This paper investigates scheduling policies to address this need, validating them on measured EEN system parameters.

TABLE I
SUMMARY OF THE MAIN NOTATION

Symbol	*Description*
L	Number of layers
K	System capacity (buffer length + job being processed)
k_i	Total jobs in the system at start of the ith service period
q_l	Exit probability for a job at layer l, $\sum_{l=1}^{L} q_l = 1$
p_l	Single-exit processing time required to exit at layer l
p	Mean processing time, $p = \sum_{l=1}^{L} q_l p_l$
p_{max}	Maximum job processing time $\max_{1\leq l\leq L} p_l$
s^2	Squared coefficient of variation (SCV) of processing time
$\tau_{i,l}$	Early exit classifier threshold at layer l for the ith job
τ_i	Configuration vector $\tau_i = (\tau_{i,1}, \ldots, \tau_{i,L-1})$
$\mathcal{T}$	Sequence of configuration vectors $\{\tau_1, \tau_2, \ldots, \tau_N\}$
$\gamma^{(l)}$	Single-exit configuration vector $\gamma_{i,l} = 0 \wedge \gamma_{i,j} = 1$, $j \neq l$
$\mathcal{G}$	Sequence of single-exit configurations $\{\gamma_1, \gamma_2, \ldots, \gamma_N\}$
G^L	Single-exit configuration vectors for a L-layer EEN
λ	Arrival rate of jobs to the edge service
ρ	Load factor $\rho = \lambda p$
R	Mean response time for processed jobs
D	Loss ratio, i.e., fraction of incoming jobs lost due to full buffer
$A^{(l)}$	Conditional accuracy for exit at layer l under $\gamma^{(l)}$
A	Mean EEN accuracy for the processed jobs
A_{norm}	Normalized mean accuracy in [0,1]

III. Problem Statement

A. Convolutional neural networks

We consider a CNN-based service, e.g., for image classification or intrusion detection. The CNN consists of $L-1$ hidden layers, indexed by l, $1 \leq l \leq L-1$, and a final classification layer ($l = L$). Hidden layers may include, for instance, *convolutional* layers, *activation* layers (e.g., ReLu) or *pooling* layers (e.g., max or average pooling) and fully connected layers. The final classification layer is typically a *softmax* layer whose output is a class label in $\Omega_C = \{\omega_1, \ldots, \omega_C\}$.

Each CNN input x_i is sequentially made available to the CNN according to an unknown, arbitrary, exogenous arrival process, and processed until producing a classification output $\hat{y}_i \in \Omega_C$. We shall equivalently refer to x_i as an input, or a job. We assume $x_i \in U \subseteq \mathcal{Z}^{h\times w\times c}$ to have size $h \times w$ and c

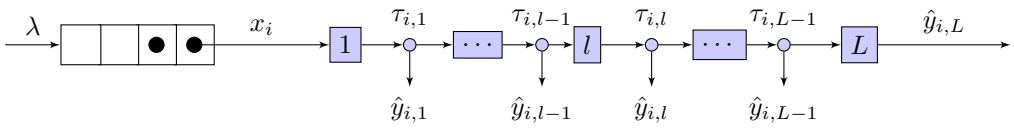


Fig. 1. An EEN-based service consisting of an early exit neural network and its input buffer. White boxes represent input buffer slots; black circles are input jobs; shaded boxes represent neural network layers that process the job admitted for processing; small circles represent early exit classifiers Δ_l. In this example, jobs arrive into the input buffer at rate λ and are lost if the buffer is full. The input job x_i at the head of the input buffer is scheduled for processing. The job will traverse the network until either its early exit, producing a classification $\hat{y}_{i,l}$ at some layer $l < L$, or generating its final layer classification $\hat{y}_{i,L}$. At layer l with $l \in \{1, \ldots, L-1\}$, the input can exit early if the classification confidence $\delta_{i,l}$ exceeds the configured early exit threshold $\tau_{i,l}$, that is, $\delta_{i,l} \geq \tau_{i,l}$. If the job arrives to layer L, the output is always the final classification $\hat{y}_{i,L}$. We emphasize that the values of $\delta_{i,l}$ are computed at runtime when the job is processed by the corresponding ICs Δ_l.

channels that result in a vector of activations $\xi_{i,l}$, $l \leq L-1$, at the output of hidden layer l. The CNN is allowed to process a single input at any given time, thus an arrival x_i may reside for some time in a buffer of size $K-1 \geq 0$ before processing starts. If a job arrives when the buffer is full, it is lost.

We assume processing to be both work-conserving[1] and non-preemptive within each layer and that the edge device has sufficient capacity to run a scheduling policy to decide, at the instant in which the CNN completes processing for an input x_i, which job to process next among those available in the buffer. If no input is available, the CNN service remains idle.

B. Early exit neural networks: the architecture

From now on, we assume the CNN to be implemented as an EEN, i.e., equipped with a set of $L-1$ ICs to allow early exits after hidden layers, as shown in Figure 1 in a configuration that also includes an input buffer for incoming jobs. In the figure, the processing pipeline after the buffer illustrates the general structure of the EENs we assume. Each early exit is associated with an IC Δ_l, placed after the hidden layer l. For ease of presentation, we say in the following that an input exits at layer l if it exits at the associated IC Δ_l. The decision on whether an input should exit or not at an IC depends on a measure of confidence for the classification accuracy produced by the IC itself.

In more detail, upon processing job x_i, the IC Δ_l receives in input the activation vector $\xi_{i,l-1}$ from the hidden layer $l-1$. The early exit is controlled by a threshold $\tau_{i,l} \in [0,1]$ such that the larger $\tau_{i,l}$, the higher the probability for an input x_i to proceed to the next layer. The last layer of the CNN, i.e., layer L, may be considered, without any loss of generality, a virtual IC Δ_L that always provides a classification output, i.e., with threshold $\tau_{i,L}$ always set to 0, thus we omit throughout this threshold from our notation.

The IC Δ_l produces a pair $(\hat{y}_{i,l}, \delta_{i,l})$ where

- $\hat{y}_{i,l} \in \Omega_C$ is the output classification of the early exit, i.e., $\hat{y}_{i,l} = \arg\max_{1 \leq m \leq C} \{u_{i,l}(m)\}$ where $u_{i,l}(m)$ is the m-th output value of the softmax layer of Δ_l;
- $\delta_{i,l} \in [0,1]$ is the output confidence, i.e., the estimated a-posteriori probability corresponding to class $\hat{y}_{i,l}$ given by $\delta_{i,l} = \max_{1 \leq m \leq C} \{u_{i,l}(m)\}$.

[1]A server is work-conserving if it never idles whenever work is available for processing.

We assume[2] that *every* hidden layer has an associated IC. Then, the output $\hat{y}_i = f(x_i, \tau_i)$ of the EEN for input x_i is computed as follows

$$f(x_i, \tau_i) = \begin{cases} \hat{y}_1 & \delta_{i,1} \geq \tau_{i,1} \\ \hat{y}_l & \delta_{i,k} < \tau_{i,k},\ \forall k < l \text{ and } \delta_{i,l} \geq \tau_{i,l} \\ \hat{y}_L & \delta_{i,k} < \tau_{i,k},\ \forall k < L \end{cases} \quad (1)$$

being $\tau_i = (\tau_{i,1}, \ldots, \tau_{i,L-1})$ the configuration vector applied to the EEN for processing the ith job.

That is, the input job traverses the EEN by computing, hidden layer by hidden layer, the activations $\xi_{i,l}$ and the corresponding output confidence provided by IC Δ_l. When $\delta_{i,l} \geq \tau_{i,l}$, $\hat{y}_l$ becomes the output of the EEN and the processing is terminated, otherwise $\xi_{i,l}$ is fed to the layer $l+1$.

C. Early exit neural networks: the training

Our methodology assumes the EEN to be trained following the Knowledge Distillation approach [7] whose goal is to initially train the original L-layer CNN introduced in Section III-A and subsequently train the ICs Δ_l with $l \in \{1, \ldots, L-1\}$. The following assumptions apply to the training of EENs:

- $x_i \in U$ and $y_i \in \Omega_C$;
- (x_i, y_i) are independent and identically distributed from an unknown but stationary probability density function $P(x, y)$;
- a training set $Z_M = \{(x_1, y_1), \ldots, (x_M, y_M)\}$ of finite dimension is extracted according to $P(x, y)$.

Note that the second assumption implies that the average accuracies conditional on exit at a specific IC Δ_j of the ICs estimated during the training phase remain valid even during the operational life of the system. That is, we assume that the distribution of data $P(x, y)$ on which the EEN is trained matches the one at test time, an assumption that holds with good approximation in application scenarios where the environment in which the system is operating does not vary significantly over time (e.g., a smart device recognizing vocal commands of a given set of users or classifying scenes/persons in a given house or building).

While in ordinary EENs the threshold $\tau_{i,l}$ is assumed to be a constant for every job, in this work we allow it to depend on the ith served job. In more detail, the $\tau_{i,l}$ values are decided by

[2]We emphasize that the proposed solution does not require to have ICs after each layer of the EENs. The positioning of the ICs are left to the EEN designer. In case of skip connections, the ICs mys be placed before or after the residual block.

the scheduling policy, which changes the EEN configuration over time based on processing times, accuracies of the layers, and observed system metrics at the instant of service start of the ith job, e.g., buffer occupancy, recent arrival rates.

D. Processing time model

An effective estimate of the processing time p_l for all the layers $l = \{1, \ldots, L\}$ is crucial for the scheduling policy of the jobs in the EENs. Throughout, we assume processing times to be deterministic so that any two inputs processed by the same layers and ICs will incur identical processing times. With the goal of being independent of a specific technological implementation, we model processing time p_l as the number of Multiply-and-Accumulates (MACs) operations to compute the output $\hat{y}_l$ with $l = \{1, \ldots, L\}$. Considering MAC operations is a quite common approach for measuring the computational demand of CNNs since most of the operations are carried out in *convolutional* and *fully connected* layers [29] [30], while other types of layers, such as Max Pooling or ReLu, typically introduce a negligible computational overhead. The corresponding number of FLOPs is typically assumed to be two times the number of MAC operations, which may be translated into absolute processing times based on technology-specific benchmarking data[3] [29]. Under a given configuration τ_i, we may then estimate the processing time of input as

$$p_l(\tau_i) = \sum_{j=1}^{l} (p_j^{\Phi} + \mathbb{1}(\tau_{i,j} > 0) p_j^{\Delta}) \tag{2}$$

for $1 \leq l \leq L$, where p_j^{Φ} and p_j^{Δ} correspond to the MACs to compute the j-th processing layer and the j-th IC of the EEN, respectively, and where $\mathbb{1}(X)$ is 1 if predicate X is true or 0 otherwise. Note that the virtual IC at layer L is implicitly assumed to have $p_L^{\Delta} = 0$ and that the term $\mathbb{1}(\tau_{i,j} > 0)$ is introduced because disabling one or more ICs using their threshold configuration leads to excluding the corresponding overheads p_j^{Δ} from (2). Further details about how to compute p_j^{Φ} and p_j^{Δ} for convolutional and fully connected layers may be found in Appendix D the online supplement.

We emphasize that (2) does not guarantee the monotonicity of the processing times since the ICs share the general architectures (i.e., the number and type of processing layers), but not the implementation (i.e., the number of input neurons) that depends on the size of the input activation.

E. Scheduling problem

The problem we focus on consists of devising a *scheduling policy* π that decides the sequence of early exit configurations $\mathcal{T} = \{\tau_i | i = 1, 2, \ldots\}$ to be used for processing the jobs as they arrive to the system. The arrival process is in general uncertain and the instants of job arrival become known to the scheduler only when the job arrives at the buffer. Incoming jobs arrive asynchronously according to a stochastic arrival process with an average rate λ, which may be estimated by the scheduling policy using an online algorithm. The scheduling policy needs to determine a sequence $\mathcal{T}$ that controls the trade-off between steady-state performance (e.g., response time, loss ratio) and classification accuracy.

The scheduler is the software component shipped with the EEN that enacts a scheduling policy π, dynamically reconfiguring the EEN thresholds as instructed by the policy. We assume that the scheduler can dynamically apply configuration τ_i to the EEN at the start of service of the ith job without incurring appreciable setup times or other costs. This assumption easily holds in real-world conditions since threshold values τ_i are stored as variables in data structures present in the firmware running in edge devices. After processing N inputs, we call n_l the number of jobs completed at layer l, either due to early exit or due to final classification. The fraction of jobs that completes at l is then $q_l(N) = n_l/N$, $\sum_{l=1}^{L} q_l = 1$. In what follows, we consider an infinite sequence of input jobs ($N \to \infty$) and assume that the partial sums of $q_l(N)$ converge to a limit q_l that we regard as the exit probabilities of the l-th layer, i.e., the probability that a processed job eventually exits at layer l.

As the system evolves under a sequence of configurations $\mathcal{T}$ imposed by the scheduling policy π, a variable number of ICs may be activated before an input reaches layer l based on the chosen thresholds, resulting in a corresponding sequence of processing times $p_l(\tau_i)$, $1 \leq l \leq L$, $\tau_i \in \mathcal{T}$. We assume $p_l = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} p_l(\tau_i)$ to be the limiting value of the processing time for an exit at a given layer l, so that $p = \sum_{l=1}^{L} q_l p_l$ represents the observed mean processing time of the EEN for output completion at the final layer L.

Our focus is at heart on a scheduling problem for a stochastic queueing system. As standard in queueing theory and stochastic scheduling [31], we then use expected values to average performance across different sample paths that are possible due to stochasticity. Specifically, to compare the performance of different policies, we consider the following metrics:

- R: *mean response time*. For an incoming job, this is the average time elapsed from arriving into the EEN buffer until producing the output classification decision. Thus, R is the sum of the mean waiting time spent in the buffer and the mean processing time p required for the inference.
- D: *loss ratio*. The steady-state fraction of jobs lost out of total arrivals to the EEN.
- A: *mean classification accuracy* for the processed jobs[4].

For a given sequence of configuration vectors $\mathcal{T}$, let $\ell(y_i, f(x_i, \tau_i))$, $\tau_i \in \mathcal{T}$, be a function computing the discrepancy between the expected output y_i and the output $f(x_i, \tau_i)$ for the ith job. We obtain the mean classification accuracy as $A \equiv A(\mathcal{T}) = \mathbb{E}_U [\ell(y_i, f(x_i, \tau_i))]$.

We may also introduce a notion of *conditional accuracy* A_l for jobs that exit at layer l as follows. The sequence of configuration vectors $\mathcal{T}$ induces a partitioning $\{U_1, \ldots, U_L\}$ of the input space U based on the exit layers of the jobs, where U_l represent the subspace of U comprising the input x_i exiting at the l-th layer. The conditional accuracy at the l-th layer may

[3]For instance, considering a 200Mhz ARM Cortex M chipset, a MAC takes about $0.05\mu s$ and $0.006\mu J$ for its execution. Futher details about the energy-consumption evaluation are given in Appendix B of the online supplement.

[4]We emphasize that we the distribution of the data $P(x, y)$ and the distribution of the inter-arrival times are assumed to be independent of each other.

Algorithm 1 EEN scheduler

Input: Scheduling policy π

1: **while** true **do**
2: Pick a job i at the head of the queue or, if none is available, await until one joins the queue.
3: Compute the exit layer probability distribution: $q_i = \pi$(policy-specific input parameters)
4: Choose exit-layer l sampling from q_i
5: Enact the EEN configuration that forces the exit of job i at layer l
6: Sleep until job i exits
7: **end while**

also be computed as $A_l \equiv A_l(\mathcal{T}) = \mathbb{E}_{U_l}\left[\ell(y_i, f(x_i, \tau_i))\right]$. It further holds that $A = \sum_{l=1}^{L} q_l A_l$.

IV. EEN Scheduler

The policies we propose for the early exit scheduling problem assume a specific scheduling architecture to enact at runtime the chosen configuration sequence $\mathcal{T}$. A pseudocode that summarizes the scheduling algorithm is given in Algorithm 1. The algorithm is activated. Upon starting to serve the ith job, the scheduler selects at random an early exit for the job according to some probability vector[5] $q_i = (q_{i,1}, \ldots, q_{1,L})$ $\sum_{l=1}^{L} q_{i,l} = 1$, that specifies a probability $q_{i,l} \geq 0$ for the job to exit at layer l of the EEN. This vector is chosen, statically or dynamically, by the scheduling policy, for example based on the buffer state, the instantaneous arrival rate, or other criteria. Using q_i, the scheduler enforces the decision on the edge device using an appropriate threshold configuration for the ICs. The job then executes non-preemptively until completion. While preemption may be implementable in some edge devices, the class of policies considered in this paper does not resort to it due to the overheads for the edge device of storing intermediate activations and stopping or resuming execution[6]. Note that our notion of non-preemptive scheduling also excludes the possibility of altering exit layer decisions while the job is executing. The policies we consider are also non-anticipative and work-conserving. Arrivals are assumed indistinguishable from each other and thus assumed to have the same priority. We therefore we do not consider scenarios where jobs have a different weight, priority or class due to some application-specific metric (e.g., data freshness).

As noted in the introduction, the concept of this architecture differs fundamentally from standard EEN usage, where system designers delegate to the ICs the exit decision. Nevertheless, we observe in Section IV-B that, under the assumptions detailed in Section III-C, this scheduling model does not restrict achievable classification accuracy.

In addition, we introduce in Section IV-A single-exit configurations of EENs as a means to enhance scheduling tractability. Indeed, even restricting our attention to a finite training dataset Z_M, an analytical formulation of A_l, albeit important to explicit

[5]Note that i indexes only processed jobs, ignoring lost jobs due to finite buffer capacity.

[6]Edge devices typically rely on firmware and not on preemptive operating systems.

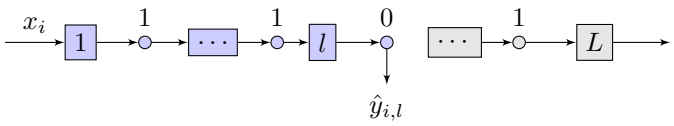


Fig. 2. Single-exit configuration $\gamma_i^{(l)}$ for the ith job. Thresholds $\tau_{i,j}$, $j < l$, are set to 1 disabling the first $l-1$ ICs. The input x_i is then forced to exit at layer l by the threshold setting $\tau_{i,l} = 0$. The remaining layers are not traversed and their ICs are also disabled.

the dependence between accuracy and exit layer, is fairly difficult to obtain. To develop such a model, according to (1), we would need to characterize the filtering effect of the first $l-1$ ICs on the classification confidence $\delta_{i,l}$ of the l-th IC, obtaining some multi-dimensional function of the configuration vector $\mathcal{T}$ and of the data content of the inputs x_i. Single-exit configurations, defined next, overcome this difficulty, as also illustrated later in the running example in Appendix A of the online supplement.

A. Non-preemptive scheduling

At the time t_i when the EEN begins serving the ith job, the scheduling policy π is invoked to decide the IC of exit of the job. After obtaining the probability vector q_i for the ith job returned by π, the EEN scheduler chooses at random its early exit to be at layer l, $1 \leq l \leq L$, with probability $q_{i,l}$. The resulting decision is readily translated into a suitable EEN configuration by setting $\tau_{i,l} = 0$ at that layer, disabling all the preceding ICs by setting $\tau_{i,j} = 1$, $j \neq l$. We assume the convention that also the remaining ICs are disabled, even though the input cannot reach them. The configuration enacted with this approach is illustrated in Figure 2 and call it a *single-exit* configuration $\gamma_i^{(l)} = \{\tau_i : \tau_{i,l} = 0 \wedge \tau_{i,j} = 1, j \neq l\}$. We emphasize that single-exit configurations differ from multi-exit configurations since the latter use thresholds allowing more than one exit in the EENs. Note that, in single-exit EENs, since $\gamma_i^{(l)}$ disables all ICs at layers $j \neq l$, no inter-dependency arises between the layer thresholds. Thus, the IC configurations of the other layers play no role on the resulting mean accuracy at exit layer l. This overcomes the aforementioned issue of representing inter-dependences between IC, as this subset of EEN configurations does not need to explicitly model data characteristics and the resulting IC inter-dependencies. In addition, in single-exit configurations, all the ICs fro which $\tau_{i,l} = 0$ are disabled, hence not executed. This allows to further reduce the computational demand by executing only the IC for which $\tau_{i,l} = 1$.

B. Achievable classification accuracy

The proposed class of non-preemptive schedulers produces a sequence of single-exit configurations, each choosing deterministically the early exit point of each job. We now note that such a sequence may achieve in principle any given *mean* accuracy A falling within a range that can be determined offline during EEN design. To see this, let $G_L = \{\gamma^{(l)} | l = 1, \ldots, L\}$ be the set of all possible single-exit configurations for a L-layer EEN and denote by $A^{(l)}$ the conditional accuracy at exit layer l when jobs are processed with the single-exit

configuration $\gamma^{(l)} \in G_L$, which forces exit at layer l. Further, let $A^{(m)} = \max_{l:A^{(l)} \leq A} A^{(l)}$ and $A^{(M)} = \min_{l:A \leq A^{(l)}} A^{(l)}$, for some pair of layers $m = \arg\max_{l:A^{(l)}<A} A^{(l)}$ and $M = \arg\min_{l:A<A^{(l)}} A^{(l)}$. We have the following statement.

Observation 1. *Suppose that a L-layer EEN has an arbitrary configuration sequence $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$ for the input jobs resulting in a steady-state mean accuracy A. If the set of single-exit configuration G_L is such that $A \in [A^{(m)}, A^{(M)}]$ for some pair of layers m and M, then there exists a configuration sequence $\mathcal{G} = \{\gamma_1, \gamma_2, \ldots\}$ that achieves mean accuracy A using only single-exit configurations $\gamma_i \in G_L$, $i \geq 1$.*

Proof. Since by definition $A^{(m)} \leq A$ and $A \leq A^{(M)}$, by Caratheodory's theorem we may always find a $q_m \in [0, 1]$ such that $A = q_m A^{(m)} + (1-q_m)A^{(M)}$. Consider now a randomized scheduling policy that, for the admitted job, chooses with probability q_m the single-exit configuration $\gamma^{(m)} \in G_L$ forcing an exit at m and with probability $1 - q_m$ the configuration $\gamma^{(M)} \in G_L$ forcing an exit at M. By construction, this policy leads to a steady-state fraction q_m of jobs that exit at layer m with accuracy $A^{(m)}$ and a fraction $1 - q_m$ that exits at layer M with accuracy $A^{(M)}$. Thus, the configuration sequence $\mathcal{G}$ arising from this policy achieves at steady-state mean accuracy $A_{\mathcal{G}} = q_m A^{(m)} + (1 - q_m)A^{(M)} = A$. □

In typical designs, real-world EENs normally satisfy the assumptions of Observation 1, since the target accuracy typically falls between the accuracy at the least and most accurate layers of a single-exit configuration. We also note that the above observation can be readily extended to arbitrary convex combinations of the conditional accuracy of any subset of the L exit layers.

It is also useful to remark that, while Observation 1 shows that there always exists a single-exit configuration sequence $\mathcal{G}$ that achieves the same *mean* accuracy of a given configuration sequence $\mathcal{T}$, the two sequences may differ in terms of the distribution of the confidences $\delta_{i,l}$ for the input jobs i. It is also possible that $\mathcal{T}$ and $\mathcal{G}$ result in differences in other performance metrics since, for example, the processing times p_l under $\mathcal{G}$ will normally be lower than with $\mathcal{T}$. This is because single-exit configurations disable a subset of the ICs. In fact, when IC j is disabled (i.e., $\tau_{i,j} = 1$), there is no overhead for input processing at j. Formally, using the notation of Section III-D, the processing time of an EEN evolving through single-exit configurations reduces to

$$p_l = \sum_{j=1}^{l} p_j^{\Phi} + p_l^{\Delta} \tag{3}$$

where only the IC Δ_l of the l-th processing layer is activated, hence saving the processing time p_j^{Δ} associated with all the ICs Δ_j, $j \neq l$. Note that, unlike the corresponding model in the multi-exit case, p_l remains constant in all configurations in $\mathcal{G}$ that exit at layer l, simplifying the analysis underpinning the scheduling decisions.

Overall, the above properties argue for the benefits in terms of tractability, accuracy, and processing times of implementing EEN scheduling policies by means of single-exit configurations, which motivates the chosen EEN scheduler architecture.

C. Running case

As an example we consider a custom 25-layer CNN endowed with three ICs deployed in layers $\{6, 12, 18\}$, and the final classifier in layer 25, that is, $L = 4$. The three ICs are characterized by the three thresholds $\tau_{i,1}$, $\tau_{i,2}$ and $\tau_{i,3}$ for the ith served job, while $\tau_{i,4}$ is omitted. Details on the architecture of the CNN and its training are given in Section VIII, while the experimental results on single-exit configurations $A^{(i)}$ and multi-exit configurations A_i are provided in Appendix A of the online supplement. In this example, $A^{(i)}, i = 1, \ldots 4$ ranges from 0.5973 to 0.8439. If we consider a multi-exit configuration $\tau^d = (\tau_{i,1}, \tau_{i,2}, \tau_{i,3}) = (0.9, 1.0, 0.8)$, we observe experimentally conditional accuracies leading to $A = 0.7838$ (see Table I in the online supplement for details). Therefore, it is possible to find a single-exit configuration, setting $m = 2$ and $M = 3$, so that $A^{(m)} = 0.7543$ and $A^{(M)} = 0.8121$, respectively, which yields an exit probability $q_m = 0.4896$ and an estimated accuracy equal to $A = 0.7838$ but with a reduction of the corresponding processing time by approximately $12 \cdot 10^6$ MAC.

V. Reactive Scheduling

Here and in the following sections, we present policies that can leverage single-exit configurations to schedule EEN input jobs. The first scheduling policy we present is a reactive scheme that is simple to implement and requires limited parameterization and information on the EEN state.

The proposed policy, denoted as $q = \pi_L(I)$, is parameterized by an ordered list of possible exit points $I = (i_1, i_2, \ldots, i_m)$ indexed by an index $j^* \in \{1, \ldots, m\}$ persisted in memory by the EEN scheduler. Initially, j^* is set to the last element of the list $j^* = m$, which is assumed to maximize mean accuracy. The policy assumes that there is a positive correlation between the index $j^* = m$ and the ability of the system to maximize accuracy under increasing losses. Thus, upon a job loss, the state variable is decreased to $j^* = j^* - 1$, assuming that this can limit future job losses.

In contrast, whenever the input buffer becomes empty ($k_i = 0$), the index variable increases to $j^* = j^* + 1$. For example, if $I = \{1, 2, \ldots, L\}$, then the proposed reactive policy has an elevator-type behavior, sliding the job exit point back-and-forth across the EEN layers in response to losses. The policy design uses the return to the empty level $k_i = 0$ to increase j^*, considering this to indicate the termination of a busy period driven by some input burst.

Conceptually similar schemes are used in other resource management problems, such as in active queue management [32], where the queue length is controlled with a threshold that dynamically changes based on transfer losses and successes, and in cache replacement policies [33], where successive hits to an item in cache produce increasing level of protection of the item from eviction, therefore with a memory mechanism similar to the one presented here to recover and persist high accuracy in periods with few losses. The key benefit of the proposed reactive policy is its ease of understanding and implementation. Yet, the policy is inherently reactive and does not explicitly leverage knowledge of the arrival rate λ, which the edge device can record based on historical data.

Also, the values of conditional accuracies of the EEN layers and processing times are only indirectly considered, as the policy assumes their monotonicity with the list index value. More sophisticated proactive policies that fully leverage this information are introduced in later sections.

An example illustrating the execution of this policy on the running case is given in Section C-A of the online supplement.

VI. Model-Based Proactive Scheduling

In this section, we attempt to obtain larger gains in performance than in our reactive policy by first developing a class of proactive scheduling policies based on queueing theory. Queueing theory is used to model in detail the stochastic arrivals to the edge service and the resulting queueing process. Under Markovian assumptions for the arrival process, the policy leverages *offline* global search to determine an optimal stationary probability vector q^* that can meet a steady-state loss ratio D^*. At runtime, the exit probability vector q^* is then used identically for each incoming job ($q_i = q^*, \forall i$), proactively achieving the desired loss ratio in the long term, if the model assumptions are met with good approximation. Then, we propose a generalization of the policy for online use and a heuristic that largely reduces the computational cost of finding a close approximation to q^* in large EENs.

A. Global optimization program

The optimization program relies on a queueing model to describe the EEN-based service as a $M/GI/1/K$ finite capacity queue with capacity K given by the sum of the buffer space and the job being processed. We briefly review here the mathematical treatment of this queueing model in the state-of-the-art and then focus on its application to EEN-based services. The main contribution of this section is to apply this queueing system to define a model-based approach to EEN scheduling under stochastic arrivals.

The $M/GI/1/K$ is a queueing system with Poisson job inter-arrival times with rate λ, arbitrary i.i.d. processing times, and a total capacity of $K-1$ jobs in the buffer and one in service. We point to [34] for a review of exact and approximate analysis techniques for $M/G/1/K$ queueing systems and we here focus on the exact technique developed in [35]. Extensions to non-Poisson arrivals are discussed in Section VI-C.

The loss ratio D for a $M/GI/1/K$ queues with arrival rate λ can be obtained from the analysis of the buffer length observed at the instant of service start of a new job, which forms a discrete-time Markov chain [35]. The processing time distribution $F(y)$ is assumed arbitrary with mean p. The stationary distribution of the buffer-length process, excluding the job just departed and the one entering service, is indicated with $(\sigma_0, \sigma_1, \dots, \sigma_{K-2})$, $\sum_j \sigma_j = 1$. It can be shown that $\sigma_0 = 1$ if $K < 3$ or otherwise [35]

$$\sigma_0 = \sigma_0(a_0 + a_1) + \sigma_1 a_0 \tag{4}$$

$$\sigma_j = \sum_{i=0}^{j+1} \sigma_i a_{j+1-i} \tag{5}$$

$$\sigma_{K-2} = \sum_{i=0}^{K-2} \sigma_i \hat{a}_{K-1-i} \tag{6}$$

in which $\hat{a}_j = \sum_{k=j}^{\infty} a_k$ and we define the probability of receiving j arrivals after completing the new job in service as

$$a_j = \int_0^{\infty} \frac{(\lambda y)^j}{j!} e^{-\lambda y} dF(y) \tag{7}$$

The loss ratio for the $M/GI/1/K$ queue is obtained as $D = 1 - (\sigma_0 a_0 + \rho)^{-1}$, where $\rho = \lambda p$ is the load factor under a mean processing time $p = \int_0^{\infty} y dF(y)$.

Since the $M/GI/1/K$ queue assumes a constant arrival rate λ and state-independent service, the model-based policies considered in this section are restricted to the form $q_i = \pi_M(D^*, \lambda) \equiv q$, which is independent of the index of the ith job and its start time t_i. Since the exit probability vector q depends only on the arrival rate λ, the effect of the scheduling policy is to effectively assign the processing time distribution[7] of the $M/GI/1/K$ queue. This can be shown to be the finite mixture with a cumulative distribution function

$$F(y) = \sum_{l=1}^{L} q_l \mathbb{1}(y \geq p_l)$$

where $\mathbb{1}(y \geq p_l) = 1$ if $y \geq p_l$, and 0 otherwise. From these, we can compute the queue load factor as $\rho = \lambda p = \lambda \sum_{l=1}^{L} q_l p_l$.

Moreover, the probability a_j of receiving j arrivals after completing the new job in service, which is the critical parameter to solve the $M/GI/1/K$ queue given in (7), reduces to

$$a_j = \sum_{l=1}^{L} q_l \frac{(\lambda p_l)^j}{j!} e^{-\lambda p_l}. \tag{8}$$

Summing up, for all the a_k with $k \geq j$, we find after algebraic manipulations that

$$\hat{a}_j = \sum_{l=1}^{L} q_l \left(1 - \frac{\Gamma(j, \lambda p_l)}{\Gamma(j)}\right) \tag{9}$$

where $\Gamma(j, x) = \int_x^{\infty} t^{j-1} e^{-t} dt$ is the (upper) incomplete gamma function and $\Gamma(j) = \Gamma(j, 0)$. The expressions of a_j and $\hat{a}_j$ are derived based on the assumption that the service process of a single-exit EEN may be modelled as a mixture and and can be used to parameterize the $M/GI/1/K$ queue and estimate the loss ratio for any candidate exit probability vector q. In particular, we define the $\pi_M(D^*, \lambda)$ as the policy that returns a vector q that is a maximizer for the mean accuracy $A = \sum_{l=1}^{L} q_l A_l$, which may be found by solving the following non-linear optimization program

$$\begin{aligned} \pi_M(D^*, \lambda) = \arg\max_{q} \quad & \textstyle\sum_{l=1}^{L} q_l A_l \\ \text{s.t.} \quad & \textstyle\sum_{l=1}^{L} q_l = 1 \\ & D(\lambda, q) = D^* \\ & q_l \geq 0, \ \forall l \end{aligned} \tag{10}$$

where $D(\lambda, q)$ is the $M/GI/1/K$ loss ratio expression under the selected exit vector q and D^* is the maximum acceptable loss ratio chosen by the EEN designer. Objective functions other than the mean accuracy A may be used to obtain variations of the $\pi_M(D^*, \lambda)$ policy.

[7] For uniformity with our EEN terminology, we refer to the service time of the queueing system as its processing time distribution.

B. Heuristic linear programming solution

The optimization program (10) is in general non-convex and thus increasingly difficult to solve as the number of layers L grows. While the optimal vector q can be pre-computed offline, the burden and cost of using a global optimization program may be significant on some instances and incur numerical difficulties. It also typically exceeds the capacity available on an edge device, which excludes the option of deploying the scheduling algorithm on the edge device itself. Thus, in this section, we propose a heuristic solution of (10) that returns a fairly accurate solution, usually in milliseconds, with far lower computational requirements than a global search. Such a solution may also be used as an initial point to supply to a global solver upon evaluating program (10).

Noting that the load factor is bounded $\rho \in [\lambda p_1, \lambda p_L]$, we study the optimization program (10) for m independent assignments of ρ within this range, thus fixing the mean processing time of the EEN. For each ρ value, we use the following tight approximation of the $M/GI/1/K$ loss ratio developed in [34] as a function of ρ and the squared coefficient of variation (SCV)[8] s^2 of the processing times, i.e.,

$$\widehat{D}(\rho, s^2) = \frac{\rho^{(\sqrt{\rho}s^2 - \sqrt{\rho} + 2K)/(2+\sqrt{\rho}s^2-\sqrt{\rho})}(\rho - 1)}{\rho^{2(1+\sqrt{\rho}s^2-\sqrt{\rho}+K)/(2+\sqrt{\rho}s^2-\sqrt{\rho})} - 1}. \quad (11)$$

Using (11), we may numerically estimate the largest value of s^2 that achieves the target loss ratio, i.e., $s^2_{max}(\rho) = \arg\min_{0 \le s^2 \le s^2_+} ||\widehat{D}(\rho, s^2) - D^*||_2$, where s^2_+ should be chosen as an arbitrary, but finite, upper bound on s^2. It can be verified numerically that, for fixed ρ and $K \ge 1$, (11) is non-decreasing in s^2, suggesting the following linear program (LP) to estimate the optimal exit probability vector

$$\begin{aligned} \pi^{heur}_{MB}(\lambda, \rho) = \arg\max_{q} \quad & \textstyle\sum_{l=1}^{L} q_l A_l \\ \text{s.t.} \quad & \textstyle\sum_{l=1}^{L} q_l = 1 \\ & \textstyle\sum_{l=1}^{L} q_l p_l = (\rho/\lambda) \\ & \textstyle\sum_{l=1}^{L} q_l p_l^2 \le (1 + s^2_{max}(\rho))(\rho/\lambda)^2 \\ & q_l \ge 0, \ \forall l \end{aligned} \quad (12)$$

In the above LP, the second equality constraint imposes the value assigned to ρ, while the third constraint is equivalent to requiring that the SCV of the EEN processing times does not exceed $s^2_{max}(\rho)$. Note that $s^2_{max}(\rho)$ can be computed before solving the LP. Compared to (10), the optimization is now reduced to a small LP with L variables, L bounds on the q_l variables, and 3 constraints. Small models of this kind may be efficiently solved, for example, using the simplex method [36].

Lastly, the proposed heuristic, which we denote as $\pi^{heur}_{MB}(\lambda)$, returns the q vector obtained by (10) that produces the maximum objective value while respecting the constraints, that is, the $\pi^{heur}_{MB}(\lambda, \rho)$ vector associated with the maximum mean accuracy A. Note that a solution always exists as long as the trivial policy that requires all jobs to exit at the first IC achieves a loss ratio lower than the target D^*.

The application of this policy to the running case is illustrated in Section C-B of the online supplement.

[8]The SCV of a random variable X is defined as $s^2 = Var(X)/E(X)^2$.

C. Application to online scheduling

A practical limitation of the model-based policy is the assumption of a constant-arrival rate. In certain deployment settings, EEN-based systems may be asynchronously activated by events in the environment, which may result in bursty arrivals to the edge device. In such cases, if the device has sufficient compute capacity, the heuristic proposed in the previous section may be recomputed on the fly, based on the observed arrival rate λ to the device during the burst. When this is too expensive relatively to the compute capacity of the device, we propose that the edge device is equipped with a lookup table that returns the exit probability vector $q = \pi_M(D^*, \widehat{\lambda}_i)$, or $q = \pi^{heur}_{MB}(D^*, \widehat{\lambda}_i)$, based on a suitable estimator of the arrival rate λ_i obtained at the instant t_i of service start of the ith job. The estimator may be obtained as $\widehat{\lambda}_i = 1/I_W(t_i)$, where $I_W(t_i)$ is the average inter-arrival time between the last W jobs that have arrived by time t_i, including those lost due to finite buffer capacity. Similar estimators may be found in works such as [21], [22], and can be inexpensively updated online at the instant of arrival of each job, e.g., using a one-pass update of the average $I_W(t_i)$. The window size W is user-specified. In practice, due to finite storage capacity, the lookup table may be stored only for J possible intervals of the value $\widehat{\lambda}_i$, thus storing only J exit vectors q on the edge device. Any given value $\widehat{\lambda}_i$, including those that fall outside the range stored by the lookup table, is assigned the exit vector of the closest interval.

VII. Rate-Based Proactive Scheduling

We now develop a proactive policy that takes into account the incoming arrival rate, conditional accuracies, and processing times. This policy strikes a balance between a reactive approach and a fully model-based approach. Indeed, the queueing-based scheduler relies on several assumptions, such as that the system is close to steady-state or that the arrival process is Poisson. Such assumptions may not always hold; for example, the arrival rate may vary faster than the time required by the EEN to settle to a stationary behavior. In such settings, online scheduling based on the current arrival rate and buffer length may help the system to adapt at a faster pace than with steady-state models. Therefore, in this section we focus on developing a policy that relies on fewer assumptions than the model-based policy.

A. Description

The policy developed in this section is activated at time t_i of the ith service start when there are $k_i \ge 1$ jobs in the system. The general form of the policy is $q = \pi_R(k_i, \widehat{\lambda}_i)$, in which $\widehat{\lambda}_i$ as before denotes an estimator of the arrival rate at time t_i computed on the last W arrivals.

To account for finite buffer capacity, the policy is given a time budget B_i to ensure the completion of the k_i jobs present in the system. The time budget is set to $B_i = \widehat{\lambda}_i^{-1}(K - k_i)$, which can be seen as an estimate of the time before the buffer is full, assuming a constant arrival rate $\widehat{\lambda}_i$, after which the buffer is at risk of job loss.

The exit probability distribution q_i returned by the policy attempts to complete all the k_i jobs in the buffer within time

B_i. This implies that when the buffer is nearly full, so that B_i becomes small, the policy is forced to process all enqueued jobs as fast as possible.

In order to decide the exit layer of each job, the policy solves an integer-valued k-item knapsack problem (k-KP) [14]. This is a variant of the knapsack problem, where up to a maximum of k items can be added to the knapsack; in our case, this is the maximum number of schedulable jobs, that is, $k = k_i$. Let $x_l \in \{0, \ldots, k_i\}$ be an integer variable that counts the number of jobs required to exit at layer l. Since every job needs at least to exit at the fastest processing layer, let us indicate the latter with $l_{min} = \arg\min_l p_l$, determined breaking ties first by prioritizing higher conditional accuracy and then by lower values of l_{min}. We may then restrict the scheduler to decide only the exits at layers $l \neq l_{min}$, assuming that all other jobs will be scheduled to exit at l_{min}. Then, the k-KP of interest is the integer linear program

$$\max \quad \sum_{l \neq l_{min}} \sum_{j=1}^{k_i} A^{(l)} x_l \tag{13}$$

$$\text{s.t.} \quad \sum_{l \neq l_{min}} \sum_{j=1}^{k_i} p_l x_l \leq B_i - k_i p_{l_{min}} \tag{14}$$

$$\sum_{l \neq l_{min}}^{L} x_l \leq k_i \tag{15}$$

$$x_l \in \{0, \ldots, k_i\} \quad \forall l = 2, \ldots, L, \; j = 1, \ldots, k_i \tag{16}$$

In this formulation, item weights are given by the processing times p_l, and profits are given by the conditional accuracies $A^{(l)}$. Therefore, the objective is to maximize the mean accuracy of the k_i jobs. The summations on index l range in $1, \ldots, L$ with $l \neq l_{min}$. The capacity constraint (14) models the time budget $B_i - k_i p_{l_{min}}$ left available to process all jobs once the minimum possible processing time is factored. The k-item constraint is summarized in (15).

The above integer linear program may be solved using branch and bound, obtaining an optimal vector x^*. The policy returns an exit vector $q_i = (q_{i,l})$ with elements given by the fraction of jobs scheduled to exit at layer l is then computed as

$$q_{i,l} = \begin{cases} k_i^{-1} x_l^* & l \neq l_{min} \\ 1 - \sum_{l \neq l_{min}} k_i^{-1} x_l^* & l = l_{min} \end{cases} \tag{17}$$

Thus, the rate-based policy constructs the exit probability vector based on the mix of early exit points observed in the solution of the k-KP at optimality. Note that this solution depends via B_i on the current estimate $\widehat{\lambda}_i$ of the arrival rate.

An example illustrating this policy on the running case is shown in Section C-C of the online supplement.

B. Application to online scheduling

Although knapsack problems are NP-hard, the optimization program used by the policy has just two constraints and $L - 1$ integer variables with $k_i + 1$ values each, thus it is typically small in real-world EEN instances and can often be solved in millseconds. Yet, to avoid placing a computational burden on the EEN device, it may be preferable to generate the q_i vectors offline and store them in a lookup table as already suggested for the model-based policy in Section VI-C. However, unlike the table used for the model-based policy, the lookup table for the reactive policy requires in input both the rate estimator $\widehat{\lambda}_i$ and the number of jobs $k_i \geq 0$ in the buffer. Thus, it grows in size as $O(JKL)$, which is also the number of variables in the k-KP optimization program.

In a situation where lookup table size grows too large for the memory available in the device, the EEN designer may reduce any of the three parameters J, K, and L to compensate. Reducing J trades dynamisms of the scheduling policy, as the algorithm can adapt less frequently to varying arrival rates. L may be reduced by disregarding some possible exit points. K may instead be replaced by a value $K^* < K$, then applying the solution for $k_i = K^*$ any time $k_i \geq K^*$. For example, in our experiments with $K = 9$ we have observed that setting $K^* = 5$ produced almost identical results. Alternatively, heuristics or approximation algorithms for k-KP may also be considered [14].

VIII. Experimental results

We now validate the effectiveness of the proposed EEN scheduling policies. To achieve this goal, we have implemented EENs based on a custom CNN and two state-of-the-art CNNs, i.e., the VGG16 and VGG19 neural networks. The training datasets are two benchmarks for image classification, i.e., *CIFAR10* and *CIFAR100*, and a public-available dataset for intrusion detection.

A. Evaluation methodology

1) EENs and datasets: The EENs used in the experimental validation are based on three different CNNs:

- *Custom CNN*: a 28-layer CNN neural network comprising seven 3×3 2D Convolutional layers, three Max Pooling layers, and two dense layers, details are given in Appendix A of the online supplement;
- *VGG16*: a CNN [37] comprising 16 trainable layers (i.e., 13 convolutional layers and 3 dense layers) and 43 processing layers (i.e., ReLu, Max Pooling, Batch Normalization, Softmax, etc.);
- *VGG19*: a CNN [37] comprising 19 trainable layers (i.e., 16 convolutional layers and 3 dense layers) and 56 processing layers (i.e., ReLu, Max Pooling, Batch Normalization, Softmax, etc.).

The maximum number L of early exits that can be realized on the three CNNs above is limited by their number of layers and equal to $L \leq 24$ for the *Custom CNN*, $L \leq 32$ for *VGG16*, and $L \leq 64$ for *VGG19*. The L ICs, which are evenly distributed between the input layer and the flatten layer of the CNN, are designed as follows: a 2×2 Max Pooling; a flatten layer; a dense layer with 1024 neurons, and a final dense layer with a number of neurons equal to the number of classes in the classification problem. The considered classification loss function is the categorical cross-entropy, while the CNNs and the ICs have been trained with stochastic gradient descent.

Three different training datasets are considered:

- *CIFAR10* and *CIFAR100*. Both datasets refer to image classification problems. The former consists of 60000 32x32 color images in 10 classes, with 6000 images per class. 50000 images are available for the training and 10000 for the testing. Similarly, the latter consists of 60000 32x32 color images but with 100 classes (hence having 600 images per class). Also in this case, there are 50000 training images and 10000 test images.
- *CICIDS2017* dataset [38] for intrusion detection. The size of the dataset, which originally comprises 3119345 instances referring to 5 days of measurements, has been reduced to 60000 samples, organized into 50000 training samples and 10000 test samples, so as to keep both the same data structure and the same EEN used for *CIFAR10* and *CIFAR100*. This reduction has been carried out by evenly selecting the samples in the five days, while each sample comprises (as in the original dataset) 83 features. The intrusion detection problem has been reformulated as a classification problem comprising the normal activity and the Brute Force FTP, the DoS, and the WebAttack, while the considered EEN is the custom CNN.

2) Performance Metrics: We simulate 10^6 input jobs arriving at an edge device running the EEN. The simulation parameters are set as in Table II. Processing times p_l, $1 \leq l \leq L$, are varied to match the values measured on the EENs described in Section VIII-A1 trained on *CIFAR10* and *CIFAR100*, details are given in the online supplement. In total, and considering that some EENs cannot reach $L = 32$ and $L = 64$ due to their limited depth, this experimental design consists of 432 simulations.

Job arrivals are either based on synthetic arrival processes or on the intrusion detection dataset. Synthetic arrivals follow an i.i.d. process with exponential ($a^2 = 1.0$, a Poisson process), hyper-exponential ($a^2 = 10.0$), or Erlang-10 ($a^2 = 0.1$) inter-arrival time distribution. Since the average processing time cannot be estimated prior to running the simulation, since it is policy dependent, we set $\lambda = \rho/p_{max}$ as an approximation, which only depends on the target load factor and the maximum processing requirement from the EEN. The inter-arrival times for the intrusion detection dataset are based on the *Infiltration* and *WebAttack* traces present in the *CICIDS2017* dataset.

Let $A_{min} = \min_{1 \leq l \leq L} A^{(l)}$ and $A_{max} = \max_{1 \leq l \leq L} A^{(l)}$, where $A^{(l)}$ is the conditional accuracy of the single-exit configuration $\gamma^{(l)}$. We compare scheduling policies by average across the simulations the loss ratio D, the mean slowdown R/p_{max} introduced in Section C-A, and the mean accuracy. Detailed results are given in the online supplement, where we also report the mean after normalizing each simulation accuracy result in [0,1], which we call the normalized accuracy

$$A_{norm} = \frac{A - A_{min}}{A_{max} - A_{min}} \tag{18}$$

The latter is used to reduce the dependence of the results on the conditional accuracy values of the individual EENs.

3) Compared Policies: The experiments use the following implementations of the policies described in the paper:

- The reactive policy of Section V parameterized with a list I defined by sequentially adding exit layers, starting from layer 1 and skipping any exit layer l that degrades accuracy, i.e., such that $A^{(l)} < A^{(k)}$, $k < l$.
- The rate-based proactive policy as described in Section VII.
- The model-based heuristic in (12) uses the online parameterization described in Section VI-C.
- We have also run the model-based policy with global optimization program (10), but the overall results are very close to (12) and thus omitted. We point to the online supplement for detailed data.

TABLE II
SIMULATION PARAMETERS

Simulation parameter	Assignments
System capacity	$K \in \{2, 5, 9\}$
Load factor	$\rho \in \{0.25, 0.75, 1.25\}$
Number of layers	$L \in \{2, 4, 8, 16, 24, 32, 64\}$
SCV of inter-arrival times	$a^2 \in \{0.1, 1.0, 10\}$
Moving window size	$W = K - 1$
Lookup table range	$\widehat{\lambda}_i \in [\lambda/2, 2\lambda]$
Lookup table intervals	$J = 3$ with width $\lambda/2$

For ease of comparison with the rate-based policy, the model-based heuristic is parameterized to obtain the same loss ratio D^* as the one we obtain with the rate-based policy simulation, however this parameter is in generally controlled by the system designer and can be assigned other values as needed. We also compare the results of our policies with the exit-first and exit-last policies introduced in Section C-A, which correspond to using static single-exit configurations $\gamma^{(1)}$ and $\gamma^{(L)}$. Note that $\max_l A^{(l)}$ and $\min_l A^{(l)}$ in (18) may not always be associated with the last or first layer, thus the exit-last and exit-first policy do not always result in $A_{norm} = 1$ or $A_{norm} = 0$.

B. Experimental results

1) Synthetic arrival processes: Figure 4(a) presents the overall simulation results. All the metrics shown in the figure are steady-state simulation averages. As the confidence intervals have small error width, errors bars are not shown in the figures. Confidence interval ranges are available in the online supplement. In terms of accuracy, the reactive and rate-based policies provide the best results, with the reactive policy incurring higher loss ratio than the rate-based policy in return for higher accuracy. Conversely, the model-based policy offers visibly better slowdowns than the other policies while retaining an accuracy not too dissimilar to that of the rate-based policy.

Figures 4(b) and 4(c) demonstrate that the results remain consistent also when varying the number of layers. We show $L = 24$ as this is the largest exit layer available in all considered EENs, but similar trends are also observed for $L = 32$ and $L = 64$ in the traces that reach these depths, we point to the online supplement for additional figures that show this. It is noteworthy that the accuracy of both the rate-based and model-based policies increases with more layers, approaching the accuracy of the reactive and exit-last policies.

Figures 3(d), 3(e), and 3(f) illustrate the performance under varying arrival processes. The results for SCV $a^2 = 1$ and $a^2 = 10$ are consistent with the previous results, mainly showing a marginal decrease in the accuracy of the reactive

policy under higher variability. However, the results for low variability show a different picture, suggesting that the rate-based policy improves its accuracy In contrast, the model-based policy becomes much less accurate. The latter results from the Poisson assumption for arrivals in the $M/GI/1/K$ and may be addressed using a more sophisticated description of the arrivals, such as an Erlang or hypo-exponential process, that however would yield a more complex queueing analysis.

Lastly, we remark that sensitivity with respect to other parameters, such as simulated EEN, buffer capacity, load factor, and other layer depths indicate a very similar relative performance of the proposed policiess as shown in Figure 3. We point to the online supplement for more details.

2) Comparison with multi-exit configuration: We now consider again the EEN used in the running case ($L = 4$) and compare the single-exit policies with the multi-exit configuration $\tau^d = (0.9, 1.0, 0.8)$. This multi-exit configuration has conditional accuracies and exit probabilities that have been measured experimentally on a real EEN with 25 processing layers and 4 exits, see Appendix A in the online supplement for further details. Our goal is to demonstrate with this case study the benefit of input scheduling compared to handling arrivals using a static multi-exit configuration, which simply processes jobs in arrival order and without changing the EEN thresholds. In this experiment, we consider a wide range of load factors $\rho \in \{0.25, 0.50, 0.75, \ldots, 3.00\}$ to evaluate performance between situations where exit-last is best ($\rho = 0.25$) and where exit-first is best ($\rho = 3.00$). Inter-arrival times are assumed to be Erlang-10 distributed ($a^2 = 0.1$). The model-based heuristic is configured to target a loss ratio of $D^* = 0.01$ and using a lookup table configured as in Table II.

The results for the slowdown, loss ratio, and mean accuracy collected in these experiments are shown in Figure 4. The multi-exit configuration has flat trends like the exit-last and exit-first policies due to the lack of a dynamic scheduling policy. Note that, under an increasing load factor, the multi-exit configuration cannot react to the increasing loss ratios. The rate-based and model-based policies appear in this scenario to provide the best results, as both deliver the lowest loss ratios and slowdowns. Loss ratio in particular decrease nearly linearly until the load factor exceeds a critical load threshold (in this example around $\rho = 2.30$) after which no policy (including exit-first) can contrast loss. However, the model-based policy does so at the cost of higher accuracy degradation than the rate-based policy. The reactive policy also has the desirable property of interpolating the performance of the exit-last and exit-first case, but compared to the model-based and rate-based policies it does so less rapidly as the load factor increases. Besides its ease of implementation, the main benefit of the reactive policy appears to be the higher accuracy that it offers while adapting compared to model-based and rate-based, but at the cost of increased loss. Overall, the experiments confirm the properties observed in the simulations with synthetic arrivals and argue that the proposed policies offer diverse trade-offs to contrast increasing loss ratios.

3) Trace-driven simulation: Figure 6 shows a set of metrics similarly to the ones shown in the previous tables; we point to Section C-A for a description of the meaning of the columns. The data reported therein illustrates the performance of the proposed policies when jobs follow the inter-arrival times of the *Infiltration* and *WebAttacks* traces within the *CICIDS2017* dataset, which respectively consists of 288602 and 170366 arrivals. Inter-arrival times in both traces are non-i.i.d., as shown by the non-zero autocorrelation function coefficients in Figure 5, *which indicate that the traces are affected by burstiness*. In the trace-driven simulation, the load factor is $\rho = 0.75$ and the system capacity is $K = 2500$, thus much larger than in the simulations to describe settings where devices may have more resources to cope with more frequent job arrivals, e.g. within an ad-hoc network. The results indicate findings consistent with those discussed for the synthetic arrival processes, with a similar relative ranking of the three policies with respect to the performance metrics and with a display of significant benefits over the exit-last and exit-first policies.

In both traces, it is notable in particular that the loss ratio of the rate-based and model-based policies remain very close to the exit-first policy, despite the higher accuracy. Note that maximal and minimal accuracies are identical in both the *Infiltration* and *WebAttacks* experiments as the underpinning EEN is identical in the two simulations.

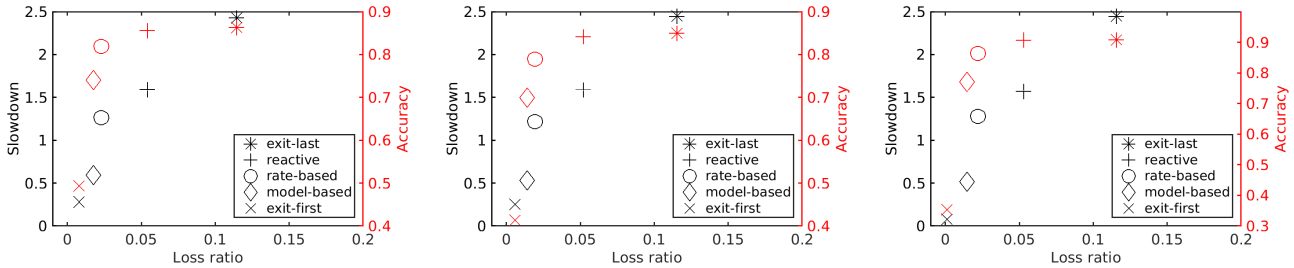
4) Summary of experimental results: The experimental results indicate that each of the proposed scheduling policies have distinctive advantages. The reactive policy is best suited for cases where a degree of reduction in loss ratio is sought, but accuracy should remain as close as possible to a full-pipeline EEN processing. Conversely, the model-based proactive policy allows finer control of the loss ratios and, in the simulations, provides the lowest slowdowns among the proposed policies. However, it can occasionally incur degraded accuracy when model assumptions are violated, as in the case of inter-arrival times with low variability or non-i.i.d. arrivals, as observed in real-world trace experiments. Lastly, the rate-based policy delivers consistently good accuracy and loss ratios across all the experimental settings, offering a robust general-purpose EEN scheduling policy. However, unlike model-based scheduling, both this policy and the reactive policy do not expose a control knob to tune loss ratios.

IX. Conclusion

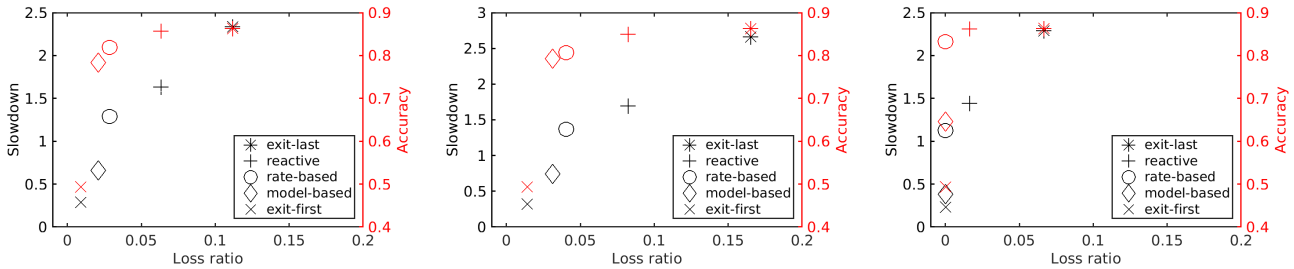
In this paper, we have introduced the EEN scheduling problems, arguing that single-exit configurations provide an effective means to manage EEN configuration at runtime. Based on this observation, we have defined three policies (list-based reactive, rate-based proactive, and model-based proactive) to dynamically schedule job arrivals to an EEN-based service.

Future work may explore further policies for managing EEN-based service, for example assigning different importance weights, classes or priorities to incoming jobs. Moreover, it would be interesting to explore whether more sophisticated inter-arrival time representations, for example using Markov-modulated point processes, can make model-based policies more applicable under situations far from modeling assumption.

Moreover, future work may consider situations when the i.i.d. assumption on the input data does not hold (e.g., due to seasonality or periodicity effect, trends or faults in

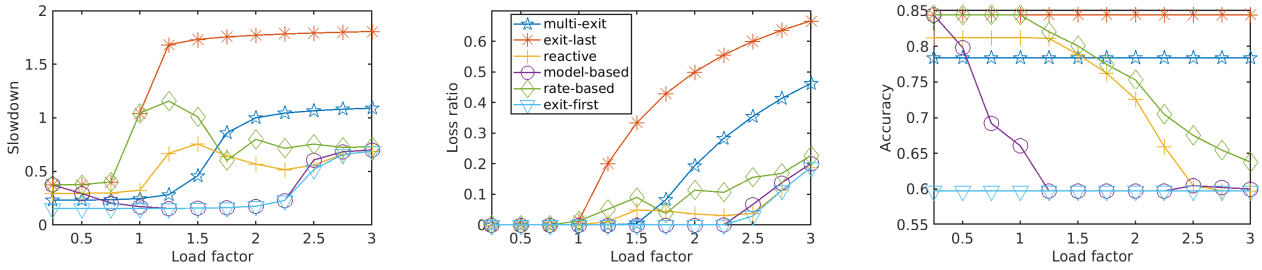


(a) Overall results (b) $L = 8$ Layers (c) $L = 24$ layers



(d) Poisson arrivals ($a^2 = 1$) (e) High variability arrivals ($a^2 = 10$) (f) Low variability arrivals ($a^2 = 0.1$)

Fig. 3. Simulation results comparing slowdown (i.e., the normalizing response time R/p_{max}), accuracy A, and loss ratio D. The metrics are averaged over 648 simulations, we point to the online supplement for additional sensitivity analyses and error bars.



(a) Slowdown R/p_{max} (b) loss ratio D (c) Mean accuracy A

Fig. 4. Simulation results comparing single-exit input scheduling policies with a static multi-exit policy on the running case for different load factors ρ. Unlike the multi-exit configuration, the single-exit policies are able to contrast the increasing loss ratios. The legend shown in (b) applies to all three figures.

sensors/actuators), adaptive mechanisms based on learning in the presence of concept drift [39] may be employed to retrain the model and re-estimate the conditional accuracy of the ICs.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[4] 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 *Proc. of IEEE/ACM ICCAD*. ACM, 2018, pp. 1–8.

[5] 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.

[6] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *International conference on machine learning*. PMLR, 2019, pp. 3301–3310.

[7] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?" *Cognitive Computation*, vol. 12, no. 5, pp. 954–966, 2020.

[8] 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.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[10] C. Xu and J. McAuley, "A survey on dynamic neural networks for natural language processing," *arXiv:2202.07101*, 2022.

[11] 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.

[12] M. M. Mohamed and B. W. Schuller, "I have vxxx bxx connexxxn!: Facing packet loss in deep speech emotion recognition," *arXiv:2005.07757*, 2020.

[13] 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 J.*, vol. 7, no. 10, pp. 9357–9371, 2020.

[14] W. Li, J. Lee, and N. Shroff, "A faster FPTAS for knapsack problem with cardinality constraint," *Discrete Applied Mathematics*, vol. 315, pp. 71–85, 2022.

[15] E. Baccarelli, M. Scarpiniti, A. Momenzadeh, and S. S. Ahrabi, "Learning-

Fig. 5. Autocorrelation function of inter-arrival times in the *Infiltration* and *WebAttacks* traces

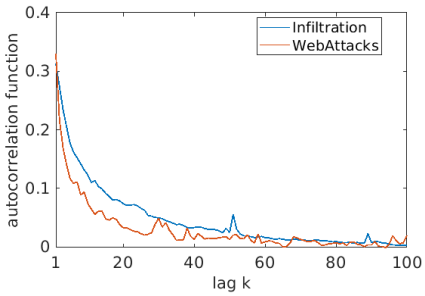


Fig. 6. Scheduling policy performance on the *Infiltration* and *WebAttacks* traces

Policy	*Trace*	*Slowdown* R/p_{max}	*Loss ratio* D	*Mean acc.* A	*Norm. acc.* A_{norm}
exit-last	Infiltration	920.7	0.1613	0.9745	1.0000
reactive	Infiltration	617.7	0.0840	0.9414	0.7717
rate-based	Infiltration	482.4	0.0750	0.9411	0.7697
model-based	Infiltration	371.8	0.0664	0.8986	0.4179
exit-first	Infiltration	270.3	0.0663	0.8295	0.0000
exit-last	WebAttacks	1087.0	0.0636	0.9745	1.0000
reactive	WebAttacks	727.7	0.0112	0.9533	0.8538
rate-based	WebAttacks	467.2	0.0006	0.9515	0.8414
model-based	WebAttacks	231.9	0.0000	0.8797	0.3462
exit-first	WebAttacks	173.4	0.0000	0.8295	0.0000

in-the-fog (LiFo): Deep learning meets fog computing for the minimum-energy distributed early-exit of inference in delay-critical iot realms," *IEEE Access*, vol. 9, pp. 25 716–25 757, 2021.

[16] 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 *Proc. of Mobicom*, 2020, pp. 1–15.

[17] D. Shabtay and G. Steiner, "A survey of scheduling with controllable processing times," *Discrete Applied Mathematics*, vol. 155, no. 13, pp. 1643–1666, 2007.

[18] Z.-L. Chen, Q. Lu, and G. Tang, "Single machine scheduling with discretely controllable processing times," *Operations Research Lett.*, vol. 21, no. 2, pp. 69–76, 1997.

[19] E. Gallisch, "On monotone optimal policies in a queueing model of m/g/1 type with controllable service time distribution," *Advances in Applied Probability*, vol. 11, no. 4, pp. 870–887, 1979.

[20] L. I. Sennott, "Average cost semi-markov decision processes and the control of queueing systems," *Probability in the Engineering and Informational Sciences*, vol. 3, no. 2, pp. 247–272, 1989.

[21] O. Hernández-Lerma and S. I. Marcus, "Adaptive control of service in queueing systems," *Systems & Control Lett.*, vol. 3, no. 5, pp. 283–289, 1983.

[22] I. Basawa and N. Prabhu, "Estimation in single server queues," *Naval Research Logistics Quarterly*, vol. 28, no. 3, pp. 475–487, 1981.

[23] M. F. Neuts, "A versatile markovian point process," *Journal of Applied Probability*, vol. 16, no. 4, pp. 764–779, 1979.

[24] B. F. Lamond, "Optimal admission policies for a finite queue with bursty arrivals," *Annals of Oper. Research*, vol. 28, no. 1, pp. 243–260, 1991.

[25] Z. Lotker and B. Patt-Shamir, "Nearly optimal fifo buffer management for diffserv," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, 2002, pp. 134–143.

[26] A. Pitsillides, P. Ioannou, M. Lestas, and L. Rossides, "Adaptive nonlinear congestion controller for a differentiated-services framework," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 94–107, 2005.

[27] S. Baruah, A. Burns, R. I. Davis, and Y. Wu, "Optimally ordering idk classifiers subject to deadlines," *Real-Time Systems*, vol. 59, no. 1, pp. 1–34, 2023.

[28] T. Abdelzaher, K. Agrawal, S. Baruah, A. Burns, R. I. Davis, Z. Guo, and Y. Hu, "Scheduling idk classifiers with arbitrary dependences to minimize the expected time to successful classification," *Real-Time Systems*, pp. 1–60, 2023.

[29] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[30] M. Roveri, "Is tiny deep learning the new deep learning?" in *Computational Intelligence and Data Analytics*. Springer, 2023, pp. 23–39.

[31] M. L. Pinedo, *Scheduling: Theory, Algorithms, and System*. Springer, 2016.

[32] W.-C. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The BLUE active queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, 2002.

[33] N. Gast and B. Van Houdt, "Transient and steady-state regime of a family of list-based cache replacement algorithms," in *Proc. of SIGMETRICS*, 2015, pp. 123–136.

[34] J. M. Smith, "Optimal design and performance modelling of m/g/1/k queueing systems," *Mathematical and computer modelling*, vol. 39, no. 9-10, pp. 1049–1081, 2004.

[35] S.-C. Niu and R. B. Cooper, "Transform-free analysis of M/G/1/K and related queues," *Mathematics of Operations Research*, vol. 18, no. 2, pp. 486–510, 1993.

[36] G. B. Dantzig, "Linear programming," *Operations research*, vol. 50, no. 1, pp. 42–47, 2002.

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

[38] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[39] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

Giuliano Casale joined the Department of Computing at Imperial College London in 2010, where he is currently a Reader. He does research in performance engineering and cloud computing, topics on which he has published more than 150 refereed papers. He has served on the technical program committee of several conferences in the area of performance and dependability. His research work has received best paper awards at ACM SIGMETRICS, IEEE/IFIP DSN, and IEEE INFOCOM. He is a past chair of ACM SIGMETRICS and serves on the editorial board of ACM TOMPECS and as Editor-in-Chief of Elsevier Performance Evaluation.

Manuel Roveri (Senior Member, IEEE) is a Full Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. He published more than 120 articles in international journals and conference proceedings. His research interests include Tiny Machine Learning and privacy-preserving machine learning. Prof. Roveri is the recipient of multiple recognition including best paper awards at IEEE TNNLS and IEEE CIM. He served as the chair and a member in several IEEE committees and subcommittees and as an Associated Editor for IEEE TNNLS, IEEE TAI, IEEE TETCI, IEEE CIM and Neural Networks.