Analyzing Distributed Deep Neural Network Deployment on Edge and Cloud Nodes in IoT Systems

Majid Ashouri, Fabian Lorig, Paul Davidsson, Romina Spalazzese
Internet of Things and People Research Center
Malmö University
Malmö, Sweden
e-mail: {firstname.lastname}@mau.se

Sergej Svorobej
School of Computer Science and Statistics
Trinity College Dublin
Dublin 2, Ireland
e-mail: sergej.svorobej@tcd.ie

Abstract— **For the efficient execution of Deep Neural Networks (DNN) in the Internet of Things, computation tasks can be distributed and deployed on edge nodes. In contrast to deploying all computation to the cloud, the use of Distributed DNN (DDNN) often results in a reduced amount of data that is sent through the network and thus might increase the overall performance of the system. However, finding an appropriate deployment scenario is often a complex task and requires considering several criteria. In this paper, we introduce a multi-criteria decision-making method based on the Analytical Hierarchy Process for the comparison and selection of deployment alternatives. We use the RECAP simulation framework to model and simulate DDNN deployments on different scales to provide a comprehensive assessment of deployments to system designers. In a case study, we apply the method to a smart city scenario where different distributions and deployments of a DNN are analyzed and compared.**

Keywords-Edge Computing, Internet of Things, Distributed Deep Neural Networks, Simulation, Smart Cities

I. INTRODUCTION

In recent years, Deep Neural Networks (DNN) have evolved rapidly and achieved great success in various Machine Learning applications [1]. DNNs are a specific type of artificial neural networks with multiple layers of feature extraction. They are well suited for image recognition, speech recognition, and other tasks where the identification of useful features in the data is difficult. Advancements in hardware capabilities have provided the opportunity for researchers to implement complex DNNs with high classification accuracy [2]. However, when DNNs become deeper, they demand more processing capability to have acceptable latency for both training and inference [3].

Using DNNs for image recognition tasks is common in IoT systems, e.g., for monitoring traffic or crowds. Such computationally heavy tasks are traditionally executed in the cloud due to the lack of processing capacity on IoT edge nodes. In the case of DNNs, this is not always feasible because of the large amount of data that has to be sent through the network, which causes latency and high bandwidth usage. Edge computing has been introduced recently to overcome this problem. When executing DNNs in IoT systems, computational tasks can be distributed and deployed on both edge and cloud nodes, often referred to as Distributed Deep Neural Networks (DDNN) [4]. In contrast to deploying all computation to the cloud, DDNNs often result in a reduced amount of data that is sent through the network, and might decrease latency as well as increase privacy of the system [4].

However, we believe there are some gaps in the current approaches, e.g.:

- The optimal deployment of the DNN layers (components) depends on several different factors, such as accuracy, energy consumption, response time of task execution, computational capacity, network capacity, and privacy. However, current approaches consider only one or a few factors when assessing deployment scenarios [5].
- The current approaches usually consider only a single or a few sources of data when assessing DNN deployment alternatives [4][6][7]. Thus, the impact of scaling the number of IoT edge nodes producing data is neglected, which might affect the general suitability of a specific deployment.
- The measurements of the required metrics do not replace the need for human judgment [8]. Each application has its own requirements which should be specified by the system designer. Moreover, the importance of the metrics are not equal for the different applications. This is often not considered in the literature.

To address these gaps, we propose a multi-criteria decision-making framework based on Analytical Hierarchy Process (AHP) for the comparison and selection of appropriate deployment alternatives. Moreover, we use simulation to analyze the performance with respect to the relevant metrics and the scalability of DDNN deployment alternatives. In this way, we link human judgement to the simulation measurements, integrate several metrics, and provide a ranking of deployment alternatives for different system sizes.

We extend the existing RECAP simulation framework [9] to model and assess DDNN deployment alternatives while varying the size of the system. We implement a DDNN application behavior model and additional performance metrics, such as resource capacity and drop rate, to provide a more comprehensive assessment of deployment scenarios to support system designers. The usefulness of the approach is shown in a case study concerning a smart city scenario where road intersections are equipped with cameras for the estimation of the number of vehicles using DDNN. We simulate AlexNet lyers and analyze and compare different deployment scenarios of this DNN.

The remainder of this paper is organized as follows. The related work is discussed in Section II. In Section III, the relevant metrics for evaluating DDNNs and our ranking method for deployment scenarios is explained. The description of the smart city use case, AlexNet, simulation setup, and experiments analysis are presented in Section IV. In Section V, we conclude and provide pointers to future work.

II. RELATED WORK

To improve the efficiency of IoT systems, processing and storage tasks can be executed closer to the edge of the network, where data is generated by sensors [10]. In contrast to traditional cloud computing approaches, where all data is first sent to the cloud, the processing of data closer to source allows for instance for reduced response times and bandwidth savings. Depending on where the nodes that execute the tasks are located, some works refer to the term "fog computing". However, in this work, we define edge computing as the use of computing and storage resources that are located in the network between the cloud and the sensors/actors.

In recent years, DNNs have evolved and achieved great success in various Machine Learning applications. Advancements in hardware capabilities have provided the opportunity for researchers to propose very deep networks with high accuracy in classification of input data [11]. However, such networks require powerful CPUs/GPUs to guarantee completion of the tasks in a low latency bound [3].

Similar to other applications, using DNNs in IoT is becoming more frequent [12]. However, there are specific circumstances in IoT which make the deployment of DNNs a complex task. The source of data is usually close to the edge of the communication network, which causes communication overhead and latency. On the one hand, a massive amount of data might be generated by IoT devices, which may overload the centralized cloud services. On the other hand, edge devices are usually powered with limited resources, which restricts the execution of very deep DNNs on edge devices. An alternative solution is to split and distribute the DNNs in inference phase; from devices with limited resources in the edge to more powerful resources on cloud, to reduce the load on edge nodes (see Figure 1). In this case, the computational-intensive layers of the DNN are expected to be executed on the cloud side and communicational-intensive layers close to the edge.

Several approaches have been proposed to achieve this distributed deployment. Neurosurgeon [6] is a pioneering study for splitting and distributing DNNs. By considering energy consumption and latency, they propose a model to find the optimal point to split the layers of the DNNs between mobile devices and cloud. However, the approach is limited to a specific case of edge computing by considering only two deployment scenarios (mobile device or cloud) and with one source of data generation. To generalize this idea on more devices, the MeDNN method [13] proposes an adaptive splitting of DNNs onto multiple mobile devices. Although a reduced computation and communication time has been achieved for DNN model inference in this approach, the focus is still limited for mobile devices and on latency metric.

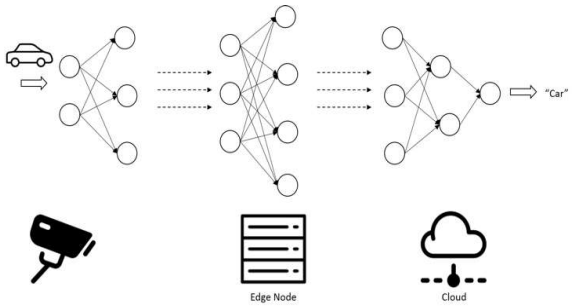

Figure 1. Distributed Deep Neural Networks

Early-exit of DNN layers is another category of distributing DNNs proposed in [4][14]. The goal of this approach is to reduce unnecessary data transmission and processing in upper layers by considering several exit points at certain layers and early classification of input data without the need for processing all DNN layers. DDNN [4] is an example of this approach to provide a model for early exit on cloud, edge, and end node devices. The authors tune their model by considering the accuracy of classifications and the percentage of early classifications. Edgent [14] is another early-exit based approach which studies the trade-off between accuracy and latency for partitioning DNNs.

To the best of our knowledge, the proposed approaches usually consider a small number of metrics for distributing DNNs, which may limit the system designer for evaluating the usefulness of a deployment scenario. There is also a lack of consideration of the human input for the deployment decisions. Moreover, we argue that scaling data sources and edge nodes is another important aspect that should be also considered in the deployment of DDNNs. For this purpose, simulation can provide a great help in analyzing deployments of high scale scenarios by providing relevant system behavior metrics.

Computer simulation can be used to facilitate the design, analysis, and optimization of complex interconnected systems [15]. Based on a model of a real-world system, simulation experiments can be conducted to evaluate *what-if* questions and to provide insights in the system's behavior and performance under different operational conditions. In contrast to real-world experiments, simulation studies allow for more cost- and time-efficient investigations. For the investigation of IoT systems with edge computing, multiple simulators exist that support modeling and assessing of different aspects of such systems, e.g., load balancing or network utilization [16]. One of them is the RECAP simulator [9], that allows for the simulation of large-scale edge computing scenarios.

III. DEEPDEP METHOD

In this section, we introduce the DeepDep method which supports the decision of deploying DDNNs over a continuum of devices from the edge to cloud. In DeepDep, we rank the available alternatives for the deployment based on the quality requirements of the application. In other words, the purpose of this method is to prioritize a list of alternatives, which meet the user requirements. To achieve this, we initially need to

define a set of metrics for quantifying different user requirements, and then rank the deployment alternatives based on the measurements. Following, we present the quality metrics used in our method and the ranking model which is based on Analytical Hierarchy Process.

A. Metrics

To comprehensively study a system, a system designer should investigate and measure various and relevant quality characteristics. A quality can be measured by a set of metrics. Zhou et al. [5]introduced several key metrics for the evaluation of DDNNs in inference part: *latency*, *accuracy*, *energy consumption*, *privacy*, *communication overhead,* and *memory footprint* [5]. We consider these metrics in our framework with some modifications. In addition, we believe *processor usage* and *capacity* are other important metrics that should also be considered in the measurements. In the following, relevant metrics for evaluating DeepDep are described:

Response Time: The demand for latency varies in IoT applications. Real-time applications such as red light violation detection requires responses in a short timeframe, while for some other applications such as smart temperature controlling of buildings latency can be tolerated. We measure this metric as the duration from when an IoT device generates data as input for a DNN over the execution of all DNN layers in inference process to responding to the next corresponding component.

Energy consumption: In IoT, energy consumption is considered as a key metric especially for battery-powered devices. The execution of DNNs usually imposes computation and network load which may drain the battery easily.

Processor usage: DNNs are often computational-intensive tasks, which demands for powerful processors, especially when scaling the number of data generation nodes. This metric can also indicate the cost of the deployment approximately. We measure this metric based on the amount of CPU capacities allocated for processing all DNN layers.

Memory Usage: In IoT, data inputs for DNNs can be generated in a high frequency and large in size (e.g., images) which demands relatively large memory space for processing them. Moreover, very deep networks containing millions of parameters should be kept in memory.

Capacity: This metric represents the scalability of the system in terms of adding new data sources. We measure capacity as the maximum number of requests that can be processed in a unit of time in addition to the current load. This metric relies on the different resource capacities such as processing, memory, and bandwidth. Here, we consider processing capacity in our experiments.

Communication overhead: the communication overhead impacts both latency and energy consumption. Having large input data and generating even larger outputs between DNN layers may cause problems for data transmission between distributed nodes.

Accuracy: Zhou et al. defined accuracy as the input samples that get the correct predictions from inference to the total number of input samples [5]. However, the rate of data capturing and processing them can also impact the accuracy of the application. There is usually a trade-off between the rate of data inputs for DNNs, response time, and available resources. Here, we measure accuracy as the average number of the processed requests generated by each data source.

Privacy: Extending a centralized cloud to decentralized edge nodes may provide both opportunities and challenges for user and data privacy [17]. Distributed data processing limits accessibility of data and increases the overall privacy, but at the same time heterogeneous and distributed devices are less resilient against security attacks and more prone to data leakage.

It should be noted that in our use case implementation for the simulation, we are not modelling (estimating) *energy consumption* and *memory usage*, since they are not relevant for the use case introduced in Section IV-A, but they can be easily added for future cases. Moreover, we are also not modeling privacy and accuracy in terms of classification accuracy for another reason, since simulation is not a suitable method for measuring these metrics. However, they can still be measured through other methods such as prototype implementation or surveying experts. Therefore, because of their importance and the feasibility of integrating these metrics in our AHP based ranking method, we consider them as the relevant criteria in our model.

B. Ranking deployment scenarios

Various criteria (metrics) can influence the decision of finding an optimal deployment for distributed DNNs, which makes this process a complicated task. This type of problem is usually identified as multiple criteria decision making (MCDM) [18]. To rank the deployment scenarios, we utilize the Analytical Hierarchy Process (AHP) that is one of the most used MCMD methods. AHP is a method for relative measurement [19]. In relative measurement, the proportion between the quantities is the interested outcome, and not the exact measurement of them. Relative measurement in general and AHP in particular suit well for the problems where the best alternative among different decision choices should be selected. Formally, in AHP, there is a goal and a finite set of alternatives, $X = \{x_1, \ldots, x_n\}$, from which the best alternative should be chosen. In addition, there are a set of criteria $C = \{c_1, \ldots, c_m\}$, which are relevant to the goal and will make one alternative preferable to another.

One important aspect that is usually neglected in the assessment of the alternatives is human judgment on evaluation of priorities. Obviously, the metric used for assessments are not identical in terms of importance and priority for each application, and the use of metrics and quantitative measures does not stop the need for human judgment [8]. One of the main features of AHP is linking human inputs with the quantitative measurements provided by the metrics [20]. To achieve this, AHP uses pairwise comparisons of decision criteria. In pairwise comparison, the user compares each criterion with other criteria in terms of its importance for the application. AHP is also very flexible for adding or removing criteria and comparing them in multi-layer hierarchical structure. Our AHP based ranking method is based on several steps described in the following.

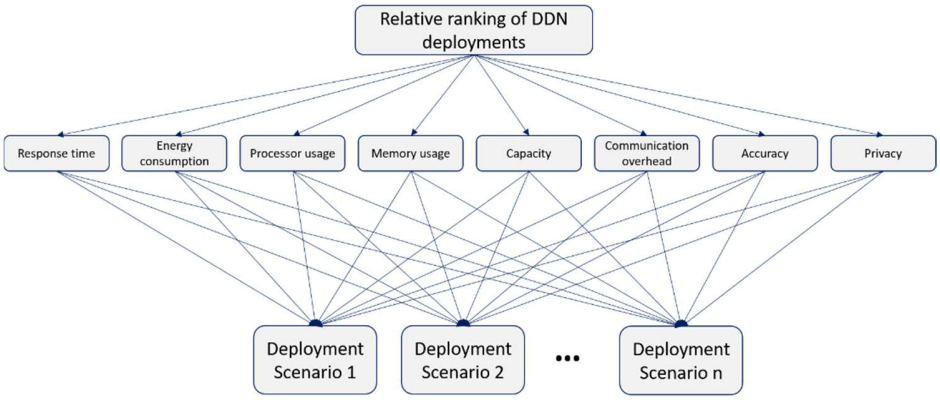

Figure 2. AHP Hierarchy for ranking DDN deployment

1) Forming AHP hierarchy

In this step, the goal, related criteria, and the set of alternatives for the specific application (in our case deployment of DDNNs) should be identified and structured in a hierarchy (see Figure 2). Here, the goal in the first layer is to find relative ranking of possible scenarios for deployment of distributed DNNs. The set of criteria in the second layer are the metrics listed in Section III-A. Finally, the third layer is the set of possible alternatives for the deployment which can vary based on application and infrastructure.

2) User judgement

The next step is to perform pairwise comparison among the criteria. Here, the user compares each criterion to another by assigning a relative importance value. Table I shows the scale suggested by Saaty et al. [20].

TABLE I. THE LIST OF RELATIVE IMPORTANCE VALUES

Equal importance	1
Somewhat more important	3
Definitely more important	5
Much more important	7
Extremely more important	9

Formally, the pairwise comparisons are collected in a pairwise comparison matrix, $\hat{A} = (\hat{a}_{ij})_{m\times m}$ structured as follows:

$$\hat{A} = \begin{pmatrix} \hat{a}_{11} & \cdots & \hat{a}_{1m} \\ \vdots & \ddots & \vdots \\ \hat{a}_{n1} & \cdots & \hat{a}_{mm} \end{pmatrix}$$

With $a_{ij} > 0$ expressing the degree of preference of c_i to c_j.

3) Measuring criteria values for alternatives

This step refers to collecting data by measuring the metrics (criteria) listed in Section III-A for each deployment scenario. We perform this step by simulation. More detailed information is provided in Section IV.

The output of this step for each criterion is *A(k)* which is the matrix of pairwise comparisons between alternatives according to criterion *k*. Note that, to be consistent in the model, for the metrics such as response time which the lower value is desirable, for the lower values we consider higher rates in pairwise comparison.

4) Linking user priorities to the measurements and ranking deployment scenarios

Once a pairwise comparison matrix for the criterion k is completed, it should be normalized and converted to a weighted matrix that is called a priority vector $W(k) = (w_1, \ldots, w_n)^T$. There are several ways of calculating the priority vectors. Here we use the method proposed by Saaty et al. [20] which estimates a priority vector based on the principal eigenvector of *A(k)*.

However, to have the final ranking of the alternatives, we need to combine the priority vectors for all criteria. The user input (matrix $\hat{A}$) should also be considered in the calculation. Hence, before combining the *W(1)…W(m)* vectors, we need to form matrix $\hat{A}$ in form of normalized weights $\widehat{W} = (\widehat{w}_1, \ldots \widehat{w}_m)$. We can use eigenvector-based method again but this time to convert matrix $\hat{A}$ to the priority vector $\widehat{W}$. Finally, we are able to combine all *W(k)* vectors as a single vector. Here, we use weighted arithmetic mean as an averaging function to linearly combine *W(1) … W(m)* vectors.

$$W_{n\times 1} = \widehat{w_1}W(1) + \widehat{w_2}W(2) \ldots \widehat{w_m}W(m) = \sum_{i=1}^{m} \widehat{w_i}W(i)$$

The result is the vector *W* with a score for each alternative. Thus, now we have a final ranking and we can choose the best alternative, which is the one rated the highest.

IV. EXPERIMENTS AND RESULTS

In the next section, we first briefly introduce the use case scenario, AlexNet as a DNN for our use case, and then we explain the infrastructure and deployment scenarios. Finally, the experiment results and examples of ranking method are presented.

A. *Use case scenario:*

To explore the DeepDep ranking method for DNNs deployment, we investigate the deployment of AlexNet [21] for automatic scheduling of traffic lights. Detecting the number of cars behind each lane is the core part of the application. Vision-based camera systems are popular to detect the number of cars for traffic estimation [12][22]. DNN approaches and particularly Convolutional Neural Networks (CNNs) have achieved a great success in object detection in recent years and have also been applied for traffic estimation [23]. In this section, we evaluate the deployment of AlexNet as a well-known CNN for traffic estimation scenarios. One possible deployment scenario is to host AlexNet on powerful smart cameras with sufficient processing and memory resources. However, this scenario will impose high deployment cost, since for each lane at least one smart camera is needed. Another option is to use cameras only as sensing devices without processing and memory resources and upload the captured images to other processing nodes in the network to reduce the deployment costs. In this case study, we consider cheap and resource-constrained cameras to capture images and will solve the problem of finding appropriate AlexNet deployment on distributed resources from edge to cloud.

B. *Alexnet*

AlexNet is a popular CNN, which achieved a great success on classification of images [21]. AlexNet layers consist of 5 convolutional layers and 3 fully connected (FC) layers. Each convolutional layer contains a convolutional filter (a set of kernels), a non-linear activation function (ReLu), and the max-pooling function in layers one, two and five. Convolution (Conv) layers use kernels to extract features from an input data. By a set of kernels, they convolve the input to a set of feature maps. For example, it can convolve the image *width*×*hight*×*depth* dimensions to feature map *width*×*height*×*channels* dimensions. In a convolutional layer, there are usually many kernels. For example, the first Conv Layer of AlexNet contains 96 kernels of size *11*×*11*×*3*. By using a non-linear function, activation functions convert each input data to an output data, to filter out the values below a certain threshold. AlexNet utilizes ReLU (Rectified Linear Unit) activation function, which simply maps negative values to zero and non-negative values unchanged. Pooling layers are usually used to down sampling feature map produced by the convolution filter. AlexNet uses the max pooling method which summarizes the most activated presence of a feature. Finally, in FC and softmax layers, the main goal is to classify the input data. In a FC layer, all the nodes in the current layer are connected to all the nodes in the previous layer and simply computes the weighted sum of the inputs. FC layers are usually computation-intensive components as shown in Figure 3. Finally, by a softmax function, the input data is classified through calculating the probability of each class.

In our experiments, we consider all functions of a layer (e.g., filter, ReLU, max pooling in convolutional layer) as a single component. The amount of required task-clock and communication overhead for each layer represented in Figure 3. Task-clock shows how much CPU time is required to execute a job. For example, Conv1 requires about 5ms task-clock, which means that having a 4 cores CPU and considering 100% CPU allocation to complete this job, it takes 1.25ms to finish that job. For calculating the amount of task-clock, we executed AlexNet on an Intel Core(TM) i7-7600U CPUs and measured the amount of required task-clock by Process explorer [24] and Perf [25] tools. Then to find the required task-clock for processing each layer we used the proportions provided in [6].

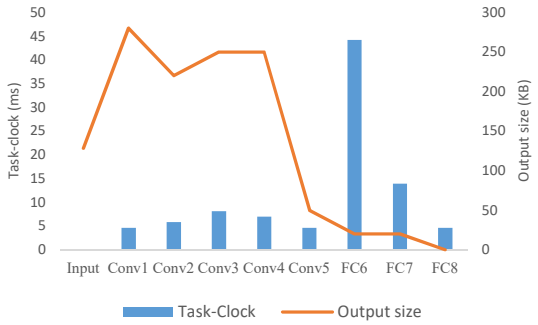

Figure 3. Computation cost and output size of AlexNet layers

C. *Simulation scenarios*

To evaluate different placement options for AlexNet layers, we modelled a distributed cloud-to-edge system by extending the RECAP simulation framework [9]. Our system is comprised of an infrastructure, application, and workload models.

The *infrastructure model* defines the hardware and network configuration and topology. We design a 3-tier system of edge, local server, and cloud based on a proximity to end user devices. Each tier composed of sites each containing virtualized hardware capable of hosting virtual entities such as virtual machines or containers. As shown in Figure 4, the edge tier (marked with letter "*E*") is the closest tier to the user and serves as a first contact point between devices (cameras in our case) and the network. Followed by local server tier (letter "*L*") and cloud tier. To reflect physical distance between the devices and tiers, for each link between tiers we are increasing network delay with lowest latency of 1ms between device and an edge site, 10ms between an edge site and a local server and 100ms between a local server and cloud. However, as a trade-off to proximity we assume that the physical size of a site is increasing higher up the tier stack and capable to host more hardware resources. To reflect this in our model we only have available 2 CPU cores at every edge site, 16 and 32 CPU cores at L_1 and L_2 servers respectively and 128 CPU cores at cloud tier. In a similar way, we have a tree topology network between tiers with 1Gb bandwidth network link between devices and an edge site, 10Gb between an edge site and a local server and 100Gb bandwidth between a local server and a cloud site.

The *application model* is based on the design of AlexNet CNN image recognition layers from Conv1 to FC8 (shown in Figure 3). Every layer is modelled as a standalone component that can be deployed as a container within any tier site of our infrastructure. Data exchange between application

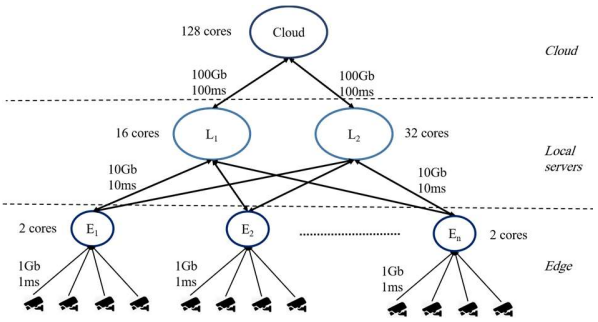

Figure 4. Infrastructure model

components and CPU processing time demand was modeled by using averages of measured values captured from real system benchmarks presented in Section IV-B.

The *workload model* captures the frequency of image frames submitted by the surveillance cameras to the image recognition application AlexNet. In our model, we assume that a total of 4 cameras are connected by wired link to every edge site. Each image frame sent from a camera constitutes a single request that needs to be processed by the modelled system.

To evaluate system performance, we create multiple simulation scenarios where we vary the AlexNet component deployment locations and workload.

As shown in Table II, we created 7 different placement scenarios where application components are distributed among different tiers and sites from edge to cloud. Each edge site has 4 cameras attached to it, hence, by increasing the number of edges we also vary the workload. In addition, we vary the number of submitted frames (requests) per second (FPS) per camera, which also impacts system resource demand. For all placements we increase the number of edge sites from 1 to 10 to 50 and to 100 and all of the scenarios are run with 1 FPS and 4 FPS. As a result, all the named workload and deployment variations create a total of 56 simulation scenarios.

After each simulation, we collect the data about system performance which is captured in form of request processing time, CPU and network utilization.

Table II. Simulation scenarios placement variations

		Deployment Scenarios						
Tier	**Site**	**E-L1**	**E-L2**	**L1**	**Cloud**	**Edge**	**L1-Cloud**	**L2-Cloud**
Edge	E1…En	Conv1, ... Conv5	Conv1, ... Conv5			Conv1 ... FC8		
Local servers	L1	FC6, ... FC8		Conv1, ... FC8			Conv1, ... Conv5	
	L2		FC6, ... FC8					Conv1, ... Conv5
Cloud	C1				Conv1, ... FC8		FC6, ... FC8	FC6, ... FC8

D. Ranking deployment scenarios

In this section, we will show how our ranking method selects an appropriate scenario for the deployment. The first phase is to form the AHP hierarchy and identify the relevant criteria. Here we consider the following criteria, *response time (RT)*, *processor usage (PU)*, *processor capacity (PC)*, and *accuracy (AC)*. Based on our measurements, the amount of required RAM will not impact the decision (14MB for processing all AlexNet layers for a request). Moreover, since end devices and E nodes are connected by cable power, energy consumption is not a challenge here. Similarly, due to availability of power resources and network links, availability metric has not been considered. In the deployment scenarios, convolutional layers that generate a massive amount of data are located on the same device, which makes communication overhead also trivial in our case.

The second step is user judgment, comparing each criteria to others. It should be noted that the user scores vary form one application to another. Here, based on the importance of each criteria in our scenario we form the comparison matrix $\hat{A}$ shown in below.

$$\hat{A} = \begin{array}{c|cccc} & \text{RT} & \text{PU} & \text{PC} & \text{AC} \\ \text{RT} & 1 & 3 & 5 & 3 \\ \text{PU} & - & 1 & 3 & 1 \\ \text{PC} & - & - & 1 & 1/3 \\ \text{AC} & - & - & - & 1 \end{array}$$

Consequently, the priority vector is $\widehat{W}$= <0.522, 0.2, 0.2, 0.078>, which shows the normalized weights for response time, processor usage, processor capacity and accuracy respectively. The next step is estimating the value of each criterion for all the metrics and forming the pairwise comparison matrices. We used estimated data by RECAP for this purpose.

Figure 5 shows the response time of deployment scenarios for various topologies. The figure shows the impact of different scales on the response time. Please note that the labels indicate the topology based on the number of *E* nodes (intersections with traffic lights). When the processing latency is more than two seconds, the new requests in the processing queue are dropped to control overloading of the system. Therefore, when the number of received requests is more than the maximum capacity, instead of increasing the response time, more request will be dropped. Figure 6 shows the average drop rate of the deployment scenarios for each topology. Clearly when the edge nodes grow in number, more requests are dropped due to lack of available resources in the deployment scenarios. *Edge* is the only scenario which is not overloaded in any topology, but with the cost of allocating more CPU cores. Table III shows the amount of allocated CPU for each deployment scenario. We considered a fixed CPU allocation policy without elasticity for the *L* nodes and *Cloud* in our experiments.

Table IV shows the maximum number of requests each deployment scenario is able to process. The available capacity for future scaling of the system is calculated by subtracting the

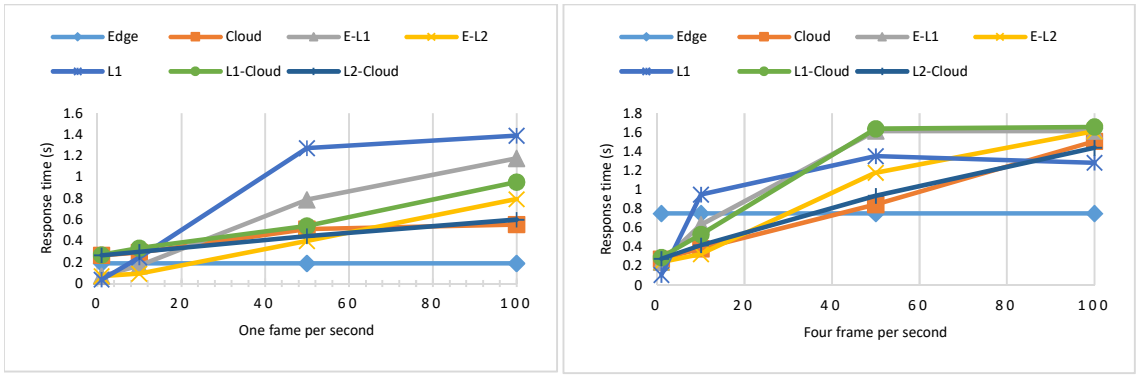

Figure 5. Average response time for one (left) and four (right) frame per second

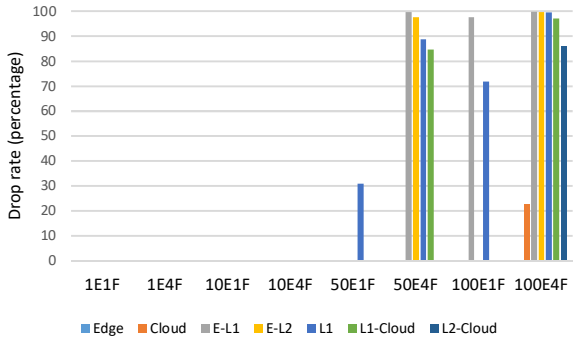

Figure 6. Average request drop rate

average number of completed request from maximum capacity.

Having the criteria values, we are able to form the AHP priority vectors *W(k)* for each criterion and then rank the deployment scenarios. As shown in Figure 5 and Figure 6, the response time and drop rate increases rapidly when the number of sites is more than 50. To show how DeepDep ranks the scenarios, we have considered two cases: *C1*) one edge site and 4 frames per second, *C2*) 50 edge sites and 4 frames per second. The final *W* vector for the cases is shown in Figure 7.

For the case *C1*, based on the user requirements, scenario *L1* where all components are executed on *L1* node has acquired the highest rate. *L1* has the lowest response time and the second lowest processor usage, which result in an increased the final rate. Since there is no dropped request in all deployment scenarios, accuracy is the same for all of them. Although, in *L1*, the capacity is the lowest among deployment

TABLE III. PROCESSOR USAGE OF THE DEPLOYMENT SCENARIOS

	E-L1	E-L2	L1	Cloud	Edge	L1-Cloud	L2-Cloud
1 site	2	128	18	34	16	144	160
10 sites	20	128	36	52	16	144	160
50 sites	100	128	116	132	16	144	160
100 sites	200	128	216	232	16	144	160

TABLE IV. MAX PROCESSOR CAPACITY OF THE DEPLOYMENT SCENARIOS

E-L1	E-L2	L1	Cloud	Edge	L1-Cloud	L2-Cloud
254	510	172	1376	No limit	529	1058

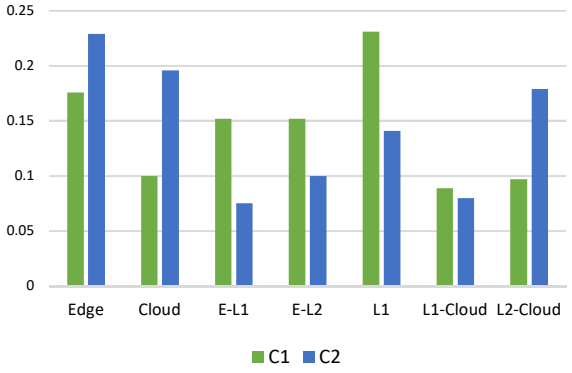

Figure 7. Rankings for C1 and C2 cases

scenarios, based on priority vector $\widehat{W}$ the impact of capacity is also lowest among the considered criteria.

For *C2*, deployment scenario *Edge* in which all components are executed on *E* nodes has acquired the highest rate. *Edge* has the lowest response time, highest capacity and no dropped requests, which distinguish this scenario from others, although processor usage is not the lowest. Cloud-based scenarios *Cloud* and *L2-Cloud* have also received high rates for this case, mainly due to lower response time and higher accuracy (no dropped request) compared to other scenarios.

V. CONCLUSION

Distributing DNN layers in edge nodes and cloud is a deployment approach to overcome the problems such as high latency, privacy issues, and lack of resource for processing them. But finding the appropriate deployment scenario is still challenging. In this paper, we proposed a ranking system to support the decision makers regarding the deployment of

DDNNs. Instead of considering specific and limited metrics in the decision, we proposed a multi-criteria decision-making method based on Analytical Hierarchy Process (AHP) for the selection of appropriate deployment alternatives. Moreover, the user judgement which is usually neglected in deployment suggestion models was integrated to the method. To analyze the decision especially for high scale IoT systems, we used and extended the RECAP simulator, to estimate the value of the relevant criteria. Finally, to show the applicability of the method we analyzed the distribution of AlexNet for automatic scheduling of traffic lights where intersections are equipped with cameras. In the use case, the deployment suggestion for two different cases based on the user inputs and the estimated values form the simulation were explained. The output shows how the proposed method is able to suggest the appropriate deployment solutions based on the user preferences and different scales of the system.

For future work, we plan to investigate the ranking method with more use cases and different DNNs. Considering sensitivity analysis in the decision and extending the simulation tool by supporting more measurement models are also other potential works for future.

ACKNOWLEDGMENT

This work is partially financed by the Knowledge Foundation through the Internet of Things and People research profile (Malmö University, Sweden) and is also partially funded by Science Foundation Ireland (SFI) under the Enable Grant No. 16/SP/3804.

REFERENCES

[1] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[3] Y. Ren, S. Yoo and A. Hoisie, "Performance Analysis of Deep Learning Workloads on Leading-edge Systems," *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Denver, CO, USA, 2019, pp. 103-113, doi: 10.1109/PMBS49563.2019.00017.

[4] S. Teerapittayanon, B. McDanel, and H. T. Kung, “Distributed Deep Neural Networks over the Cloud, the Edge and End Devices,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 328–339, 2017.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing,” *Proc. IEEE*, 2019.

[6] Y. Kang *et al.*, “Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge,” in *ASPLOS ’17 Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 615–629.

[7] S. Teerapittayanon, B. McDanel, and H. T. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” *Proc. - Int. Conf. Pattern Recognit.*, pp. 2464–2469, 2017.

[8] IEEE Standard for a Software Quality Metrics Methodology," in *IEEE Std 1061-1992* , vol., no., pp.1-96, 12 March 1993, doi: 10.1109/IEEESTD.1993.115124.

[9] P.-O. Ostberg *et al.*, “Reliable capacity provisioning for distributed cloud/edge/fog computing applications,” in *2017 European Conference on Networks and Communications (EuCNC)*, Jun. 2017, pp. 1–6, doi: 10.1109/EuCNC.2017.7980667.

[10] Shi, W.; Dustdar, S. The promise of edge computing. *Computer* **2016**, *49*, 78-81.

[11] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, vol. 9, pp. 249–256.

[12] X. Ma *et al.*, "A Survey on Deep Learning Empowered IoT Applications," in *IEEE Access*, vol. 7, pp. 181721-181732, 2019, doi: 10.1109/ACCESS.2019.2958962.

[13] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, “Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns,” in *Proceedings of the 36th International Conference on Computer-Aided Design. IEEE Press,* 2017, pp. 751–756.

[14] E. Li, Z. Zhou, and X. Chen, “Edge intelligence: On-demand deep learning model co-inference with device-edge synergy,” in Proc. of the Workshop on Mobile Edge Communications (MECOMM), 2018, pp. 31–36.

[15] Law, A.M. Simulation modeling and analysis; McGraw-Hill Education: Dubuque, 2013.

[16] S. Svorobej *et al.*, “Simulating Fog and Edge Computing Scenarios: An Overview and Research Challenges,” *Futur. Internet*, vol. 11, no. 3, p. 55, 2019.

[17] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, “Fog Computing for the Internet of Things: Security and Privacy Issues,” *IEEE Internet Comput.*, vol. 21, no. 2, pp. 34–42, 2017.

[18] M. Zeleny, “Multiple Criteria Decision Making”, McGraw-Hill, vol. 25, 1982.

[19] M. Brunelli, Introduction to the Analytic Hierarchy Process., New York, NY, USA:Springer, 2015.

[20] T. L. Saaty and L. G. Vargas, Models, Methods, Concepts & Applications of the Analytic Hierarchy Process. Springer, 2012.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. of NIPS, 2012.

[22] P. Choudekar, S. Banerjee and M. K. Muju, "Implementation of image processing in real time traffic light control," *2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, 2011, pp. 94-98, doi: 10.1109/ICECTECH.2011.5941662.

[23] P. R. Iyer, S. R. Iyer, R. Ramesh, M. R. Anala, and K. N. Subramanya, “Adaptive real time traffic prediction using deep neural networks,” *IAES Int. J. Artif. Intell.*, vol. 8, no. 2, pp. 107–119, 2019.

[24] https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer

[25] https://perf.wiki.kernel.org/index.php/Tutorial