Cost-Aware Neural Network Splitting and Dynamic Rescheduling for Edge Intelligence

Daniel Luger
University of Vienna
Vienna, Austria
d.luger@univie.ac.at

Atakan Aral
University of Vienna, Austria
Umeå University, Sweden
atakan.aral@umu.se

Ivona Brandic
Vienna University of Technology
Vienna, Austria
ivona.brandic@tuwien.ac.at

ABSTRACT

With the rise of IoT devices and the necessity of intelligent applications, inference tasks are often offloaded to the cloud due to the computation limitation of the end devices. Yet, requests to the cloud are costly in terms of latency, and therefore a shift of the computation from the cloud to the network's edge is unavoidable. This shift is called edge intelligence and promises lower latency, among other advantages. However, some algorithms, like deep neural networks, are computationally intensive, even for local edge servers (ES). To keep latency low, such DNNs can be split into two parts and distributed between the ES and the cloud. We present a dynamic scheduling algorithm that takes real-time parameters like the clock speed of the ES, bandwidth, and latency into account and predicts the optimal splitting point regarding latency. Furthermore, we estimate the overall costs for the ES and cloud during run-time and integrate them into our prediction and decision models. We present a cost-aware prediction of the splitting point, which can be tuned with a parameter toward faster response or lower costs.

CCS CONCEPTS

• **Computer systems organization** → *Distributed architectures*; • **Computing methodologies** → *Distributed artificial intelligence*.

KEYWORDS

Edge Intelligence, Edge Computing, DNN Splitting, Cost-Awareness

ACM Reference Format:
Daniel Luger, Atakan Aral, and Ivona Brandic. 2023. Cost-Aware Neural Network Splitting and Dynamic Rescheduling for Edge Intelligence. In *6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578354.3592871

1 INTRODUCTION

In the past several years, the number of IoT devices has increased enormously. According to Cisco, 50% of the worldwide network devices will be IoT devices by 2023, reaching 14.7 billion devices [5]. At the same time, more and more IoT devices require AI/ML to complete their tasks [9]. Deep learning (DL) is a subcategory of ML and consists of multiple different connected layers [11].

EdgeSys '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0082-8/23/05.
https://doi.org/10.1145/3578354.3592871

Deep learning uses deep neural networks (DNN) to solve problems similar to the neurons in our brains [4]. The major advantage of DNN is that some problems are of such complexity that it would be impossible to describe them entirely in mathematical notation. DNNs use a strategy to iterate to a solution without knowing a complete function. Such networks run primarily in cloud data centers with nearly unlimited computational resources. Therefore, IoT devices offload their tasks to data centers for inference, and the outcome is returned to the IoT device. This is feasible only for non-time-sensitive applications because sending a task to the cloud and back costs substantial time. However, many applications have stringent time requirements, such as autonomous driving, distance surgery, and game streaming. Therefore, the task is shifted to a server in the user's proximity, called an edge server (ES). This shift of AI into an ES is called edge intelligence (EI) [7, 15], which can reduce latency [13] and improve privacy [1] significantly.

However, the edge might not provide sufficient computation resources, so large DNNs could not be computed at an acceptable time at the edge, rendering the shift ineffective. There exist different methods, such as model pruning or quantization, that target the deployment of DNNs on edge resources. Pruning is a compression technique that removes less important weights or filters, whereas quantization deals with mapping the model parameters and activations into low-precision quantized levels to avoid costly FLOPs. However, those two approaches can suffer from accuracy loss [14]. Another possible solution, which preserves accuracy, is to split the DNN into two parts, the first running at the edge and the second in the cloud shown in Figure 1. By doing this, the edge has to compute less, and only a small amount of inter-layer data is sent to the cloud compared to the raw input data. However, environments change over time, and therefore, the best point of the DNN is not always the same [10]. Therefore, the splitting point might have to change during runtime, especially for long-standing DNNs (e.g., for days).

Previous proposals, such as Neurosurgeon [10], focus on finding the optimal splitting point. By building prediction models to reduce energy consumption and latency with layer-specific parameters,

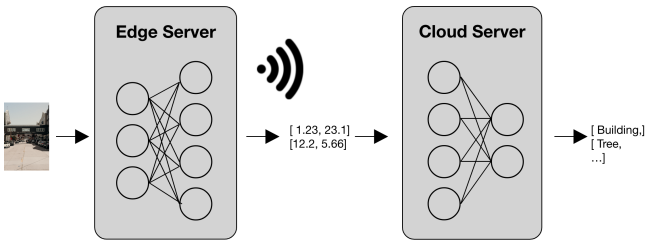


Figure 1: Splitting of a DNN over Edge and Cloud Servers.

Table 1: Comparison of Our Approach to Related Literature on DNN Splitting.

	Kang et al. [10]	Lin et al. [12]	Gao et al. [8]	This work
Dynamic	✓		✓	✓
Energy Optimization	PREDICTED		PREDICTED	
Cost Optimization		✓	COULD-ONLY	✓
Latency Optimization	LAYER-BASED	TIME-CONSTRAINT	LAYER-BASED	LINEAR, CLOCK-SPEED
Mobile Devices		✓	✓	
Edge Nodes		✓	✓	✓
Cloud Nodes		✓		✓

it is possible to infer the optimal splitting point. However, this approach does not take costs into account. Lin et al. [12] consider the cost between an end device, ES, and cloud. Yet, this algorithm neither co-optimizes cost and latency nor is dynamic. Gao et al. [8], on the other hand, apply multi-optimization for cost, energy, and latency and is dynamic but focuses on a mobile device rather than an edge server; hence, no cost is calculated at the edge. Table 1 summarizes the previous work and compares it to our work.

This work utilizes prediction models that take the current clock speed, bandwidth, network latency, ES cost, and cloud cost into account. We present a dynamic rescheduler that optimizes the system's cost and latency. The next section describes the proposed system model, whereas Section 3 explains the prediction models. Section 4 provides experimental results in a real test bed, and Section 5 concludes the paper.

2 SYSTEM MODEL

2.1 Edge Server

The ES is the central part of the proposed system. It hosts the scheduling algorithm, the prediction models, and the DNN, as shown in Figure 2. The ES is located at the network's edge, where the data is generated. It can be the user equipment (UE) itself, as in voice assistance, or a server close to the UE, like a private hospital server that processes local data. This server, however, should not be more than about one hop away from the UE. The ESs often have limitations, such as energy, bandwidth, and computational power. This leads to the necessity of offloading some of the decision-making to the cloud. In our work, we have two objectives that have to be optimized. *i)* Cost. ESs can be rented or bought. When renting, ESs have a higher fee per hour than the cloud due to the optimized location of the ESs [2]. However, ESs can be owned by the application provided and used extensively at no further cost. In this case, computing locally on the ES can be cheaper than offloading to the cloud. *ii)* Latency. Low bandwidths and long latencies are the results of the utilized communication mediums due to the location of the servers. This directly affects communication latency. Those limitations also vary over time through environmental influences like the weather and flash crowds. Furthermore, the computational capability of an ES impacts the computational latency, represented by clock speed.

2.2 Stateless Cloud

The cloud data center consists of servers with virtually unlimited resources regarding energy, storage, and computational power. However, they are often geographically far from the UE, which

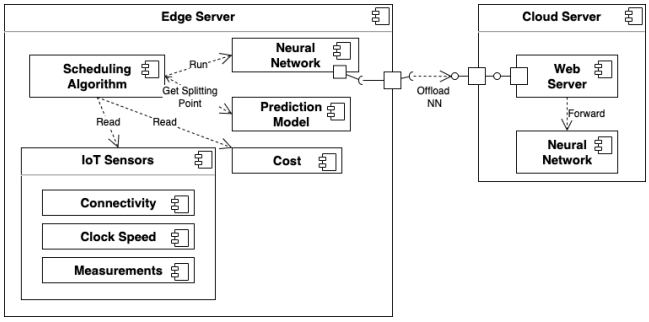


Figure 2: UML Component Diagram of the Proposed System.

can result in high response times. The time required for data to reach the cloud depends on the network bandwidth, network latency, and data size. In most DNNs, data size varies among the layers. Because speed is crucial for a fast prediction, the cloud is implemented stateless. A stateless cloud is an approach for service-oriented architecture (SOA) for cloud computing environments. In traditional SOA, service providers maintain state information about their clients, which can lead to performance and scalability issues. With the implementation of a stateless cloud, providers do not maintain the state information of the users. This leads to better scalability and performance in cloud environments. This fits perfectly with our approach since multiple servers send huge number of requests, and there is no need for storing client state information.

2.3 Communication

In general, ESs are in close proximity to where data is generated, enabling high-speed communication between UE and ES. ESs are often installed in environments where the connection to the cloud is also fast. However, ESs have to be installed in rough and rural areas, like in environmental use cases or in autonomous vehicles. Especially under rough conditions, efficiency is crucial for fast decision-making. Since the data size significantly impacts communication latency, scheduling the splitting point to a layer where the data size is small might have a huge impact. Figure 3 illustrates the communication sequence from the data generation at the edge (green boxes) to the predicted result in the cloud (yellow boxes).

In this work, we consider three latency types as highlighted in Figure 3 with the red boxes: *i)* edge latency is the time required by the edge to run the first part of the DNN; *ii)* communication latency is the time required to send the data from the edge to the cloud; and *iii)* cloud latency is the time to finish the execution

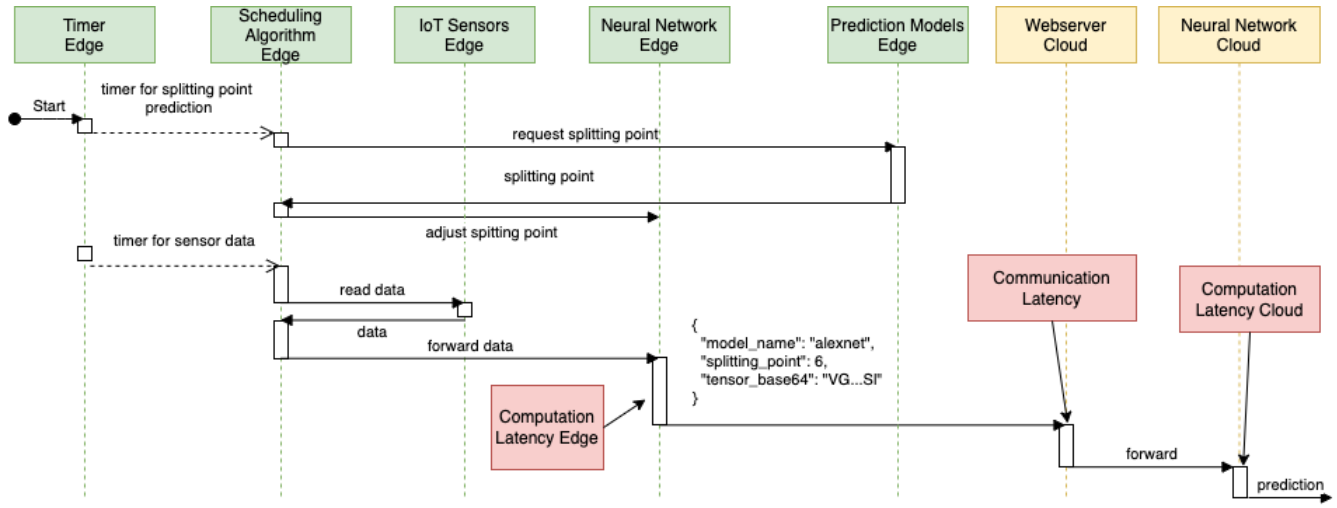


Figure 3: UML Sequence Diagram of the Proposed Cost-Aware Dynamic Scheduler.

of the second part. The ES receives the input data, runs it at the edge for a given amount of layers, and then offloads the data to the cloud, where the DNN inference is finalized. The time spent until offloading on the ES is called edge latency and depends on the computation power of the ES. The overall end-to-end latency is the time between calling the DNN at the edge and the predicted output. The time taken to finish the NN by the cloud is called cloud latency. Finally, the time consumed between offloading and receiving is called communication latency. A *Timer Edge* periodically triggers the prediction of the new splitting point, and the tasks.

2.4 Dynamic Splitting

Splitting a network requires a point at which the neural network is partitioned into two parts. The first part runs at the first server, the edge, and the second in the cloud. The point where the network is partitioned is called the splitting point and is dynamically calculated in this work. However, the existing networks are usually not built for splitting; therefore, the network must be prepared. This is a manual, straightforward process in which particular functions are implemented into an existing DNN. However, it is essential to preserve all the existing layers and functions to ensure the prediction outcome stays precisely the same. Therefore the tensor, which has to be offloaded, is converted into a base 64 encoded string. This string is added to the body of a POST request to the cloud. Furthermore, the splitting point has to be sent along to continue at the right point in the cloud. If multiple models are used with the same cloud server, the model must also be specified and sent along. The base 64 encoded tensor is decoded and transformed back into a tensor when received by the cloud, which can be directly used to call the model. This work focuses on CNNs with chain structure thus, parallel layers, but the idea applies to other DNNs as well.

3 PREDICTION MODELS

We focus on two main objectives: cost and latency. Our goal is to predict the optimal splitting point for minimizing the cost of the servers and the latency based on real-time parameters. We investigate two different strategies i) the application provider owns the ES; therefore, does not have to pay rent for the ES, and ii) they rent the ES as well as the cloud. In the first scenario, it is cheaper to compute at the edge, whereas, in the second, it is cheaper to offload since the ESs are more costly [6]. To find the optimal splitting point, three prediction models are created. The first predicts the influence of a change in the clock speed on the computation time; the second predicts the influence of the bandwidth and network latency change on communication latency; and the third predicts the cost. Based on these, a dynamic splitting point scheduler, which minimizes overall latency and reduces cost at the same time, is proposed.

3.1 Computation Latency

Deploying ESs in unaffiliated environments with limited energy supply and relying on batteries and solar power results in fluctuations and scarcity in computational resources. Therefore a variety of edge nodes with different clock speed settings coexist. We include clock speed in our models to predict edge computation latency.

The first step is the prediction of the computation latency in a specific layer independent of the clock speed. Therefore we measure the final layer latency and assume that the edge latency has a linear behavior. Since the edge latency is zero before the first layer, we can create a linear equation. With this equation, the predicted computation latency in a specific layer can be calculated, albeit only for this specific server, under the specific setting and the specific DNN. The next step is to adapt the prediction model so that the calculation of the edge latency takes the clock speed into account. To that end, we investigate the relationship between the computation latency and the clock speed on the specific server by running the DNN on the server. We benchmark the execution time under various clock speed settings and compute the latency slope, which estimates the latency in this layer when multiplied by the layer number. Because each DNN has completely different layer latencies due to the different numbers and types of layers, the data points cannot be combined directly and must be normalized to

Table 2: Reference Configurations and On-Demand Prices for AWS EC2 [2]

	Instance Name	Hourly Rate	vCPU	Memory	Storage	Type	Location
Edge Server	t3.xlarge	$0.224	4	16 GiB	EBS Only	Wavelength Zone	US East (Verizon) - NY
Cloud Server	t3.xlarge	$0.1664	4	16 GiB	EBS Only	Region	US Rast (Ohio)

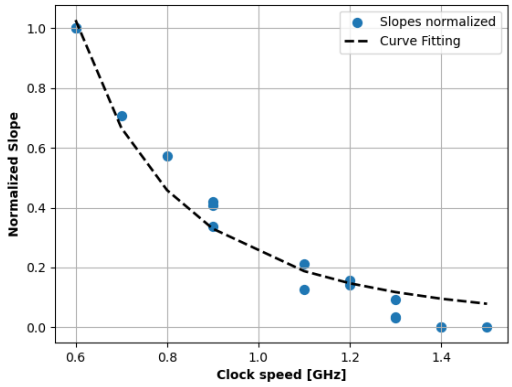


Figure 4: Normalized Slopes with Curve Fitting.

the range $[0, 1]$. Figure 4 illustrates normalized slopes of the edge latency from the three different DNNs (Alexnet, Mnist, VGG16). Here, we use 60% of the data points and 40% is for validation. After normalizing the data points, Power Law can be used to generate a fitting curve through all measurements. The resulting model to predict the edge latency has to be de-normalized before being used for a specific DNN. This can only be done with the upper and lower bounds. The resulting prediction function for the latency based on the clock speed is given in Equations 1 and 2.

$$f_{edge}(CS, SP, M) = f_{denorm}(a * CS^b, up\_b(M), l\_b(M)) * SP \quad (1)$$

$$f_{denorm} = slope_{norm} * (up\_b(M) - l\_b(M)) + l\_b(M) \quad (2)$$

a, b = Parameters from the curve fitting
CS = Clock speed
up_b = NN specific upper bound
l_b = NN specific lower bound
M = NN model
SP = Splitting Point

The prediction of the cloud computation latency works the same way as the prediction for the edge computation latency but without the dependency on the clock speed. This is because the cloud runs on virtual CPUs with a fixed clock speed.

3.2 Communication Latency

ESs are usually installed in close proximity to the data generation to achieve low latencies. However, when the ES sends data to the cloud, the communication latency can be high, depending on where the ES and cloud data center are located. In this work, we assume the ES is in a region with volatile internet connectivity. This can result from an external condition such as weather or increased internet traffic. Furthermore, intermittent connectivity can also appear when the ES is mobile, such as a car driving through regions with good or bad connectivity. Therefore, the connectivity must be considered when offloading a DNN since the data size can be massive. Therefore, the splitting point needs to be dynamically recalculated. To achieve this, we create a prediction model for the splitting point based on the current bandwidth and network latency. Since we know the current bandwidth and network latency and the data size in each layer, the communication latency can be estimated. Equation 3 takes the bandwidth, network latency, and splitting point as input and calculates the communication latency.

$$L_{comm}(BW, PL, SP, OS) = \frac{OS(SP) * 8}{1000 * BW} + \frac{PL}{2 * 1000} \quad (3)$$

OS = Array of layers with offload sizes [Byte]
BW = Bandwidth [kbit/s]
PL = Ping Latency [ms]

3.3 Cost

To predict the costs arising during the computation of a task, we use the two strategies presented at the beginning of this section, namely owned and rented ES. The two strategies are given in Equation 4. We use real-world AWS prices for our calculation. AWS offers ESs with ultra-low latencies for 5G services at a higher price than the cloud resources, as shown in Table 2 [2]. We choose the corresponding price for the partitions that run on edge and cloud.

$$f_{cost}(RT) = \begin{cases} 0 + C_{cloud}(RT_{cloud}), & \text{if ES is owned.} \\ C_{edge}(RT_{edge}) + C_{cloud}(RT), & \text{if ES is rent.} \end{cases} \quad (4)$$

RT = Runtime

3.4 Combined Prediction Model

To combine the prediction models to optimize the overall latency as well as cost based on clock speed, bandwidth, and network latency, an optimization function is presented in Algorithm 1. The optimization function can be weighted to optimize lower latency or cost. The weight parameter w can take values from 0 to 1, with 0 to optimize only the latency and 1 to optimize only the cost. First, the cost and the latency are calculated for each layer, assuming it is the splitting point. The overall latency, including edge computation, communication, and cloud computation, and overall cost, including edge and cloud, are considered. The results are stored in two arrays and are sorted in decreasing order of preference. Then, one array entry is compared to the corresponding array entry plus a calculated amount of additional entries based on the weight w. This happens in the function $add\_visited()$. If the weight is 0.5, there is exactly one entry of both arrays added to the $visited$ arrays. Through this, the first match of the two weighted arrays can be found.

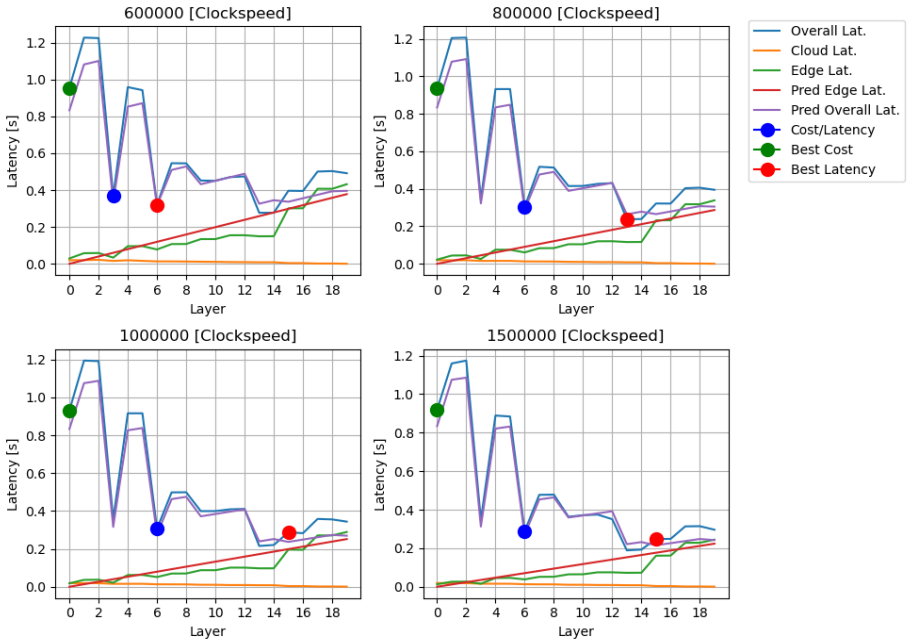


Figure 5: Dynamic Rescheduling of Splitting Points Under Different Clock Speeds.

Algorithm 1 Weighted cost latency optimization algorithm

Require: w
Require: $L = [pred\_overall\_lat]$
Require: $C = [pred\_best\_cost]$
  sort[C] ▷ Cheapest first
  sort[L] ▷ Fastest first
  $visited\_cost = []$
  $visited\_lat = []$
  for `i in range(0, layer_length)` **do**
    $visited\_cost \leftarrow add\_visited(w, C, i)$
    $visited\_lat \leftarrow add\_visited(w, L, i)$
    if `visited_cost in visited_lat ||`
  `visited_lat in visited_cost` **then** ▷ Check if a match
      return `match`
    end if
  end for

4 EVALUATION

4.1 Experimental Setup

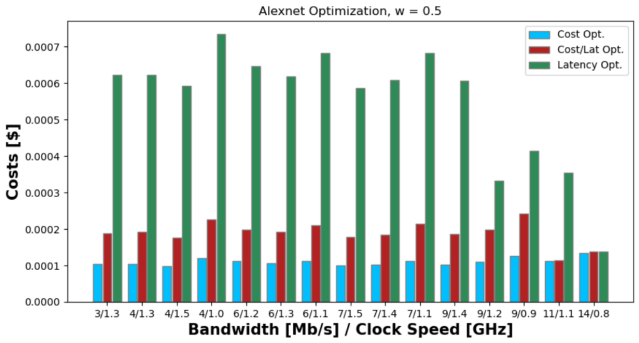
In our test bed, the ES is a Raspberry Pi 4 with 4 GB of RAM, installed directly where the data is generated. The ES hosts three software components; the scheduling algorithm, the prediction models, and the neural networks themselves. Each part is implemented in Python, and the DNNs use PyTorch as the framework. We evaluated our approach with the AlexNet CNN, a CNN for the MNIST dataset, and VGG16 CNN. The training datasets are ImageNet for AlexNet and VGG16 and the MNIST dataset. For brevity and since the results are similar, we report only AlexNet results and the rented edge scenario in full detail.

The cloud server runs in an Amazon EC2 instance using a Linux Server, with four virtual CPUs at a clock speed of 1,5 GHz and with 32 GB of RAM. The server is implemented as a web server using Python with a REST interface. For simulating bandwidth limitations, we use Wondershaper [3].To measure the execution times, Python function `time.time()` is used, which performs on Unix systems with a precision of 1 microsecond. Since the counter is only valid for one system, we calculate the communication latency by subtracting computation latencies from the total end-to-end latency.

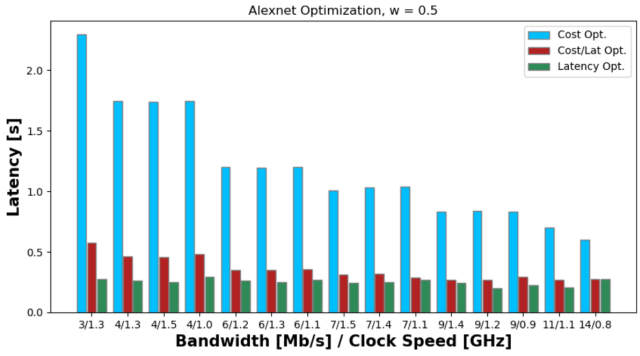
4.2 Numerical Results

Figure 5 illustrates the different offloading strategies when latency, cost, or both ($w = 0.5$) are optimized under various clock speeds and fixed bandwidth. It can be seen that latency optimization and the proposed cost/latency optimization are dynamic, whereas the cost is static. In this scenario, the ES is rented; therefore, the cheapest solution is always to offload immediately. The proposed approach identifies a good trade-off solution with 13.73% higher latency or 35.28% more cost than the two single-objective solutions on average.

For the measurements shown in Figure 6, we chose 20 samples from a pool with Poisson-distributed bandwidths and normally distributed clock speeds. Each measurement is run three times, and the median of the runs has been taken. We also choose a weight w of 0.5, meaning cost and latency are taken into account equally. Figure 6a compares the costs of the different offloading strategies, best cost, best latency, or the proposed combination. In this scenario, the ES is rented, so offloading early is cheaper. Our proposed approach is 65.53% cheaper than the latency-optimization approach on average, while at the same time, 41.76% more costly than the cost-optimization. Figure 6b shows the same scenario with latency



(a) Cost Measurement with Different Offloading Strategies.



(b) Latency Measurement with Different Offloading Strategies.

Figure 6: Alexnet Cost and Latency Optimization (w=0.5).

measurements. The proposed trade-off solution reduces latency by 70.47% on average in comparison to cost-optimization and incurs 29.09% additional latency on average in comparison to the latency-optimization. For the MNIST model, we have a decrease in the cost by 27.13% and a slight increase in latency of 3.35%. For the owning strategy of the ES with the VGG16 model, we achieve a cost reduction of 98.01%, although with an increase of 42.16% latency. To investigate the impact of the weight on cost and latency, we compute solutions with different splitting points by changing the weight from 0 to 1 with a step size of 0.01 as shown in Figure 7.

5 CONCLUSION

In this work, we created a dynamic rescheduler that takes the real-time clock speed, bandwidth, and network latency into account and predicts, through prediction models, the best splitting point for offloading regarding cost and latency. This is crucial for edge servers, which run DNNs in a volatile environment such as environmental monitoring or autonomous driving. The results show that our algorithm combines cost and latency optimization and can reduce cost as well as latency. The trade-off is presented, which allows a user of this rescheduler the focus on either cost or latency or a weighted combination. The next open challenge will be to improve this rescheduling algorithm to include other parameters like CPU Type, RAM, or kernels, which might enable this scheduling algorithm to work on different types of edge resources.

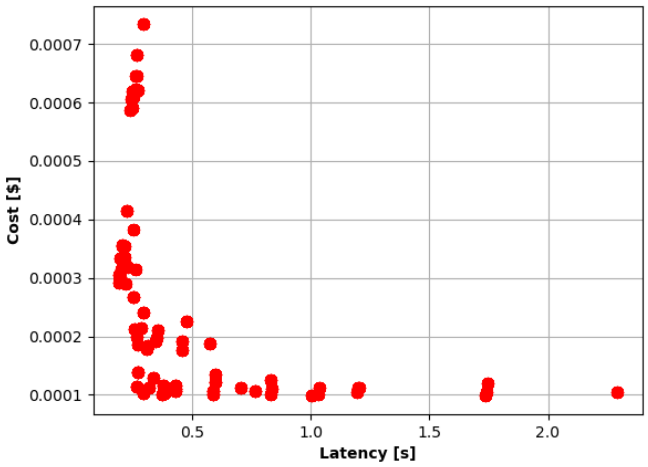


Figure 7: Trade-Off Between Latency and Cost.

ACKNOWLEDGMENTS

This work was supported by the CHIST-ERA grant CHIST-ERA-19-CES-005, by the Austrian Science Fund (FWF): I 5201-N, and by the FFG Flagship Project High Performance Integrated Quantum Computing (HPQC): #45285029.

REFERENCES

[1] Sabtain Ahmad and Atakan Aral. 2022. FedCD: Personalized Federated Learning via Collaborative Distillation. In *Workshop on Distributed Machine Learning for the Intelligent Computing Continuum (DML-ICC)*. IEEE, Vancouver, WA, 6 pages.
[2] AWS. 2023. On-Demand Plans for Amazon EC2. Retrieved 2023-02-22 from https://aws.amazon.com/ec2/pricing/on-demand/
[3] Simon Séhier Bert Hubert, Jacco Geul. 2021. The Wonder Shaper. Retrieved 2023-02-22 from https://github.com/magnific0/wondershaper
[4] Thierry Bouwmans, Sajid Javed, Maryam Sultana, and Soon Ki Jung. 2019. Deep neural network concepts for background subtraction: A systematic review and comparative evaluation. *Neural Networks* 117 (2019), 8–66.
[5] Cisco. 2020. Cisco Annual Internet Report (2018–2023) White Paper. Retrieved 2022-12-21 from https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
[6] Vincenzo De Maio and Ivona Brandic. 2018. First Hop Mobile Offloading of DAG Computations. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Washington, DC, USA, 83–92.
[7] Aaron Yi Ding, Ella Peltonen, Tobias Meuser, Atakan Aral, Christian Becker, et al. 2022. Roadmap for edge AI: a Dagstuhl perspective. *ACM SIGCOMM Computer Communication Review* 52, 1 (2022), 28–33.
[8] Mingjin Gao, Rujing Shen, Long Shi, Wen Qi, Jun Li, and Yonghui Li. 2021. Task Partitioning and Offloading in DNN-Task Enabled Mobile Edge Computing Networks. *IEEE Transactions on Mobile Computing* 22, 4 (2021), 2435–2445.
[9] Sukhpal Singh Gill, Minxian Xu, Carlo Ottaviani, Panos Patros, Rami Bahsoon, et al. 2022. AI for next generation computing: Emerging trends and future directions. *Internet of Things* 19 (2022), 100514.
[10] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, et al. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.
[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.
[12] Bing Lin, Yinhao Huang, Jianshan Zhang, Junqin Hu, Xing Chen, and Jun Li. 2020. Cost-Driven Off-Loading for DNN-Based Applications Over Cloud, Edge, and End Devices. *IEEE Transactions on Industrial Informatics* 16, 8 (2020), 5456–5466.
[13] Ella Peltonen, Ijaz Ahmad, Atakan Aral, Michele Capobianco, Aaron Yi Ding, et al. 2022. The Many Faces of Edge Intelligence. *IEEE Access* 10 (2022), 104769–104782.
[14] Md. Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. 2023. Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review. *Proc. IEEE* 111, 1 (2023), 42–91.
[15] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.