RoaD-RuNNer: Collaborative DNN partitioning and offloading on heterogeneous edge systems

Andreas Kosmas Kakolyris*, Manolis Katsaragakis*, Dimosthenis Masouros*, Dimitrios Soudris*
*Microprocessors and Digital Systems Laboratory, ECE , National Technical University of Athens, Greece
*{akakolyris, mkatsaragakis, dmasouros, dsoudris}@microlab.ntua.gr

Abstract*—Deep Neural Networks (DNNs) are becoming extremely popular for many modern applications deployed at the edge of the computing continuum. Despite their effectiveness, DNNs are typically resource intensive, making it prohibitive to be deployed on resource- and/or energy-constrained devices found in such environments. To overcome this limitation, partitioning and offloading part of the DNN execution from edge devices to more powerful servers has been introduced as a prominent solution. While previous works have proposed resource management schemes to tackle this problem, they usually neglect the high dynamicity found in such environments, both regarding the diversity of the deployed DNN models, as well as the heterogeneity of the underlying hardware infrastructure. In this paper, we present RoaD-RuNNer, a framework for DNN partitioning and offloading for edge computing systems. RoaD-RuNNer relies on its prior knowledge and leverages collaborative filtering techniques to quickly estimate performance and energy requirements of individual layers over heterogeneous devices. By aggregating this information, it specifies a set of Pareto optimal DNN partitioning schemes that trade-off between performance and energy consumption. We evaluate our approach using a set of well-known DNN architectures and show that our framework i) outperforms existing state-of-the-art approaches by achieving 9.58× speedup on average and up to 88.73% less energy consumption, ii) achieves high prediction accuracy by limiting the prediction error down to 3.19% and 0.18% for latency and energy, respectively and iii) provides lightweight and dynamic performance characteristics.

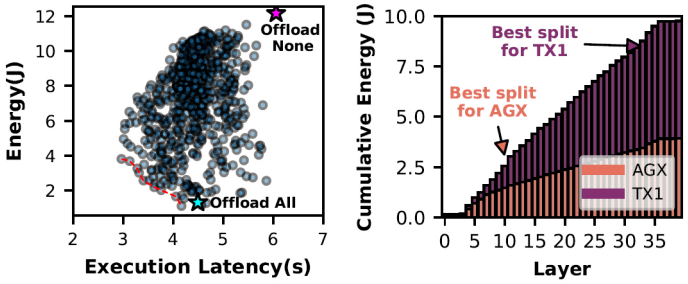
***Index Terms*—Cloud, Edge Computing, Resource Management, Neural Networks, Offloading, Collaborative Filtering, Partitioning**

I. Introduction

Over the last years, the growth of applications that utilize sophisticated Machine Learning (ML) techniques to make value out of complex data is rapidly increasing and is expected to grow further in the future. In this direction, Deep Neural Networks (DNNs) are being widely adopted by many application domains, including, but not limited to, autonomous driving [1], biomedical and healthcare applications [2] and Intelligent Personal Assistants (IPAs) [3], mainly due to their capability in offering high prediction accuracy.

Such types of applications are typically deployed at the edge of the computing continuum, closer to where data are generated, in order to enhance security and minimizing the data transfer latency to the cloud [4]. Even though DNNs provide extremely accurate results, their computational and memory requirements can skyrocket, thus introducing several barriers on how to be efficiently deployed on resource-contrained edge computing devices [5]. Considering also that DNNs are gradually becoming deeper to support more discriminative results and more levels of hierarchy are integrated in the learned representation [6], the aforementioned problem becomes even more intense.

This work has been partially funded by EU Horizon program under grant agreement No 101096110 (https://www.privateer-project.eu/).



(a) ResNet101 energy and latency for different partitioning schemes

(b) ResNet101 energy's optimal partitioning for different devices

Fig. 1: DNN optimal partitioning schemes w.r.t. *a)* different optimization goals and *b)* device heterogeneity.

Traditionally, to overcome this limitation, the DNN inference of edge devices was offloaded to high-end servers hosted on cloud premises (e.g., Amazon Elastic Inference, Azure Machine Learning). However, this approach leads to huge amount of data travelling back and forth in the continuum resulting in high latency and energy consumption. Moreover, all the computational effort is being concentrated in the cloud, rendering it incapable to cope with the ever-increasing demand for resources [7]. Aiming to deliver more efficient and energy proportional computing systems, hardware vendors (e.g., Nvidia, Xilinx) are introducing more powerful edge devices, which often integrate conventional CPUs, hardware accelerators and specialized units for DNN computations (e.g., Nvidia's Tensor cores) as a system-on-chip (SoC). While such hardware devices deliver increased compute density, they are also extremely power hungry, thus, becoming restrictive for applications that operate under certain energy or battery constraints.

Aiming to identify the "golden ratio" between performance and energy efficiency for edge computing devices, while also limiting the network latency bottleneck, *DNN partitioning and offloading* has been identified as a promising solution [8], [9]. The aim of DNN partitioning is to provide a fine-grained layer-level split of the DNN, where a portion of the computation is performed locally on the edge device and the rest on the cloud. Still, identifying an optimal partitioning scheme is not trivial and depends on a triptych of user-defined

978-3-9819263-7-8/DATE23/© 2023 EDAA

requirements and hardware-oriented criteria, i.e., *i)* the architecture of the deployed DNN model (e.g., computational/energy requirements and data offloading overhead per layer), *ii)* the inherent nature of the application itself (e.g., latency critical applications would sacrifice energy for performance) and *iii)* the underlying hardware characteristics. As a motivational example, Fig. 1a shows the performance and energy of all the possible partitioning schemes for a ResNet101 architecture. We see that model partitioning can provide significantly faster execution and less energy consumption compared to on-board and offloaded execution, while also each optimization objective is attained through different partitioning schemes. Moreover, Fig. 1b further reveals, that the optimal partitioning scheme also relies on the underlying hardware, with different edge devices providing different splits.

To tackle these challenges, several prior research approaches have examined the problem of DNN partitioning and offloading [9], [10], [8]. These solutions assume that the architecture of the deployed model, as well as the underlying hardware are known a priori and apply the offloading scheme based on extensive profiling of the DNN. However, edge computing environments are extremely dynamic, both with respect to the devices arriving in the network and the alternative applications deployed. Moreover, novel DNN architectures are emerging [4], leading to more and more complex techniques, such as skip-layer connections, early exits and others.

In order to adapt to the challenges that new DNN architectures impose and to the dynamic nature of recent edge computing systems, we design an efficient resource management framework to dynamically allocate, partition and offload DNN layers among edge and cloud resources, aiming to provide performance and/or energy consumption optimizations and trade-offs. **The novel contributions of this work are:**

- **We present `RoaD-RuNNer`**, a novel resource management framework consisting of a set of *Offline* and *Online Decision Making Mechanisms*.
- **We implement a collaborative filtering based prediction mechanism** in order to provide per layer predictions regarding execution time and energy consumption.
- **We design and integrate a dynamic partition mechanism**, for efficiently splitting and offloading DNN layers.
- **We conduct an extensive experimental evaluation of our proposed framework**. We compare our framework with a set of baseline algorithms and state-of-the-art DNN offloading approaches over real hardware and networking, showing that it outperforms existing state-of-the-art approaches by achieving 9.58× speedup on average and up to 88.73% less energy consumption average.

The remainder of this paper is organized as follows: Section II presents related work. In Section III we analyze our proposed prediction and offloading mechanism. Section IV provides our experimental and comparative evaluation of the proposed resource management scheme, while Section V concludes this work.

II. Related Work

Several works have been conducted in order to address the resource management challenges of DNNs deployed on edge computing systems. Deep learning model partitioning and offloading is becoming a rising trend in recent edge computing systems. Research conducted by [11] considers various present and future challenges for efficiently deploying deep learning models on the edge. In a similar perspective, authors of [12] provide an overview of the overarching architectures, frameworks, and emerging key technologies towards training/inference for deep learning models at the network edge.

Aiming to provide DNN resource management schemes, authors of [10] propose a partitioning-based DNN offloading technique for edge computing, by dividing the DNN model into partitions and uploading them to the edge server. In a similar perspective, authors of [13] present a distributed progressive inference engine that addresses the challenge of partitioning CNN inference across device-server setups. In [14] a framework that dynamically partitions a DNN model that adapts to the changes of computational resources and network condition is proposed. Authors of [8] design a workload partitioning algorithm to decide efficient DNN partitioning policy in real-time, while in [15] authors utilize several IoT devices by creating a local collaborative network for a subset of deep learning models, mainly focusing on the impact of convolutional layers. Last but not least, authors of [9] present the efficiency of DNN processing on the cloud based on pre-loaded layers.

Although research has illuminated the resource management and offloading of neural network based applications on edge computing systems, no study to date, according to our knowledge, has incorporated collaborative filtering in order to produce an efficient, decentralized resource management solution for Neural Network offloading at single layer granularity, while targeting heterogeneous CPU/GPU architectures. *Neurosurgeon* [9], is the most similar approach to ours, however there exist several fundamental differences. We target a fully dynamic DNN offloading framework, in contrast to *Neurosurgeon*, where DNN weights are loaded a priori to the cloud server. Moreover, *Neurosurgeon* is limited to operate over models without skip-layer connections, in contrast to `RoaD-RuNNer`, which is designed to operate over all types of DNN/CNN.

III. `RoaD-RuNNer`: A collaborative DNN partitioning & offloading framework

`RoaD-RuNNer` tackles the problem of DNN partitioning and offloading over heterogeneous CPU/GPU edge computing systems, where a portion of the computational burden can be offloaded from the edge to the cloud for remote execution. Each individual edge node accommodates a set of DNN inference tasks which need to be executed. `RoaD-RuNNer`'s goal is to identify optimal DNN splittings, down to the granularity of a single layer and offload computationally heavy slices, aiming to optimize the total inference execution latency and/or the energy consumption. Figure 2 depicts an overview of `RoaD-RuNNer`. The major components of our proposed framework are split in two major phases: (i) *Offline Phase* and (ii) *Online Phase*. The former is composed of *DNN-Profiler* and *Network Profiler*, while the latter consists of the *Predictor* and *Offloader* mechanisms. The core functionality of these components is described in the rest of this section.

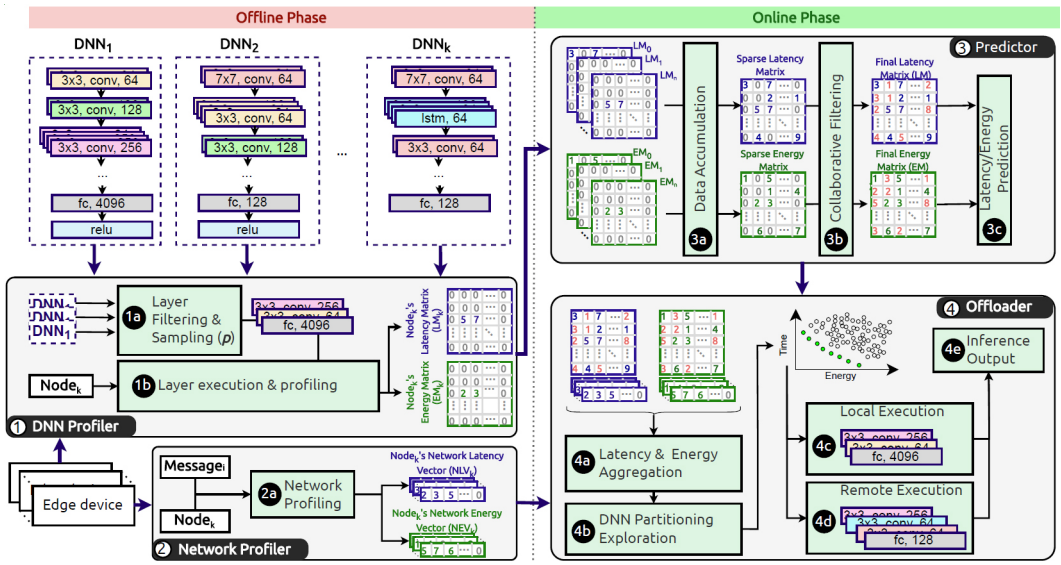


Fig. 2: Overview of Online and Offline RoaD RuNNer Architecture.

A. *Offline Phase*

Offline Phase consists of two distinct components: (i) *DNN Profiler* (❶) and (ii) *Network Profiler* (❷), which are responsible for generating the required data and knowledge to be fed as input to the run-time mechanisms later. As input to the framework, we provide alternative neural network configurations to each single node of the composed edge computing network.

DNN Profiler: Aiming to take advantage of the inherent distributed nature of the edge computing paradigm and the heterogeneity in terms of edge devices and deep learning models, we implement a collaborative filtering mechanism [16], [17], in order to train our system to efficiently predict per layer execution time and energy consumption for each device, respectively. As a first step of the offline decision making, we integrate a *Layer Filtering and Sampling* (❶a) component, aiming to filter and sample the layers of the alternative neural networks that are fed as input to our framework. More specifically, a small percentage p of the input layers in each DNN is randomly selected to be accurately profiled on the edge device itself. This ensures that profiling time does not skyrocket when the total number of layers in the DNN increases. The percentage p is defined through experimentation.

Next, the layers that have been selected through the filtering and sampling process are propagated as input to the *Layer Execution & Profiling* (❶b) step. Each of the sampled layers is executed locally on each node, respectively. The execution is profiled, aiming to gather data regarding the execution latency and the energy consumption per layer. Thus, for every single node k, we extract the Node's Latency Matrix (LM_k) and Node's Energy Matrix (EM_k), respectively. The LM_k and EM_k are propagated as input for further processing to the *Predictor* mechanism, as described in Section III-B. This process is triggered once for each node.

Network Profiler: The efficiency of an edge computing system is directly related to its ability to effectively operate over the existing network infrastructure. Moreover, offloading decision making mechanisms should consider the overhead imposed by the underlying network. Thus, for each individual node k we integrate an extended *Network Profiling* mechanism (❷a). As input we provide a set of alternative workloads to be sent/received from each edge device to the cloud server. We profile the transmission latency and power of data sent and received from and to the edge device. The profiled messages range from KB to GB orders of magnitude.

After the profiling is finalized, each node k is characterized by two vectors: (i) The Network Latency Vector (NLV_k) and the Network Energy Vector (NEV_k), which represent the latency and energy requirements for alternative message sizes, respectively. The produced vectors are utilized as dataset to produce polymonial trendlines, in order to predict the latency and energy requirements of a given input message size. Thus, for each individual device, fourth-order polynomial curves are generated in order to be utilized for run-time prediction during the *Online Phase*.

B. *Online Phase*

An efficient dynamic resource management scheme should be able to make decisions in a run-time manner. Thus, we integrate into our framework two key components: (i) *Predictor* (❸) and (ii) *Offloader* (❹). The former is responsible for dynamically predicting latency and energy per layer, while the latter is responsible for network and collaborative filtering data aggregation, dynamic DNN partitioning andRoa offloading.

Predictor: The output matrices produced by the *DNN Profiler* (❶) of each single node k during the offline phase are accumulated to the cloud server (❸a). Two new sparse matrices are produced, i.e. *Sparse Latency Matrix* and *Sparse Energy Matrix*, respectively. Each row of the matrix represents an edge node, while each column denotes a single layer type. The *Collaborative Filtering* mechanism (❸b) is triggered, based on matrix decomposition, i.e. based on the factorization of each matrix into a product of matrices. The unknown values

Algorithm 1: DNN Partitioning Algorithm

```
Partition(network):
    for layer in (0,...,N-1) do
        /* predict local for layers 0 to layer */
        l1=predictLocal(0, layer)
        for j in (layer+1,...,N-1) do
            /* predict network,cloud for layers i + 1 to j */
            n=predictNetwork(layer+1, j)
            c=predictCloud(layer+1, j)
            /* predict network,cloud for layers j + 1 to N − 1 */
            l2=predictLocal(j+1, N-1)
            totalPredictions=accumulatePredictions(l1,n,c,l2)
    /* find Pareto Optimal */
    paretoPoints = DesignSpaceExploration(totalPredictions)
    return paretoPoints
```

are filled, and the *Final Latency Matrix(LM)* and the *Final Energy Matrix(EM)* are produced. After the predictions are finalized, each device retrieves its corresponding results from the cloud server, which are utilized for the final *Latency/Energy Prediction* (3c). Opposed to prior works, which mostly rely on extended profiling for each new deep learning model [13], [10], our framework is adaptable to dynamic scenarios. New nodes that dynamically join the network are integrated in the existing latency and energy matrices and the collaborative filtering matrices are updated. New incoming DNNs can benefit from the existing layers in the collaborative filtering matrices.

Offloader: The last component of the *Online Phase* is responsible for the aggregation of network and collaborative filtering data and the dynamic DNN partitioning and offloading. For each single node k, the network profiling vectors (NLV_k, NEV_k), and the collaborative filtering matrices (LM, EM) for latency and energy are aggregated (4a), in order to provide the final prediction per layer, including computation and transmission overhead. Next, we proceed to the *DNN Partitioning Exploration* (4b), in order to provide a set of Pareto optimal solutions, aiming to optimize performance and/or energy consumption. The DNN partitioning exploration algorithm is illustrated in Algorithm 1. A set of Pareto optimal solutions in generated. Each Pareto point corresponds to an alternative partition, on which a subset of DNN layers is executed locally (4c) and the rest are offloaded for remote execution on the cloud server (4d). Each partition is identified by two indexes i, j where layers 0 to i are executed locally, layers $i + 1$ to j are executed on the cloud and the layers from $j + 1$ and up to the end of the network are executed locally. Fully local and fully remote executions remain valid alternatives, in case they belong to the Pareto optimal solutions. Moreover, in contrast to existing approaches [9] our framework is designed to handle shortcut dependencies, i.e. skip-layer connections. Such kind of dependencies are resolved by encapsulating the layers that create the dependency in an larger atomic block that has a single input and output and are offloaded as an individual component.

IV. Experimental Evaluation

A. Experimental Setup

Hardware Infrastructure: We deploy an in-house system setup consisting of heterogeneous CPU/GPU devices, the specifications of which are shown in Table I. As edge devices, we utilize a set of NVIDIA GPU-SoCs, to exploit power/performance trade-offs. As offloading machine, we employ a powerfull x86 server equipped with an NVIDIA V100 GPU, which forms a typical setup both in edge and cloud premises [18].

Technical Implementation: *RoaD-RuNNer* is implemented in Python programming language. All devices are interconnected through 80MB/s wireless network. We utilize the ZeroMQ messaging protocol for control and the FTP protocol for transmission of the actual data and layers. In order to overcome the architectural differences of heterogeneous CPUs all applications are integrated inside docker containers, while the corresponding GPU implementations are developed in CUDA. The collaborative filtering component used for estimating the performance and energy impact of different layers is accelerated through the use of C++ for increased performance.

Examined DNN models: We examine famous DNN architectures and known variations, i.e. AlexNet (alex), MobileNetV2 (mv2), Resnet18 (res18), Resnet34 (res34), Resnet50 (res50), Resnet101 (res101), Resnet152 (res152), VGG11, VGG13 and VGG16, which are widely used for performing object detection and image classification tasks at the edge [19], [20], over alternative input image sizes (224×224, 512×512 and 768×768). The input models are derived from PyTorch [21] and are integrated to `RoaD-RuNNer`.

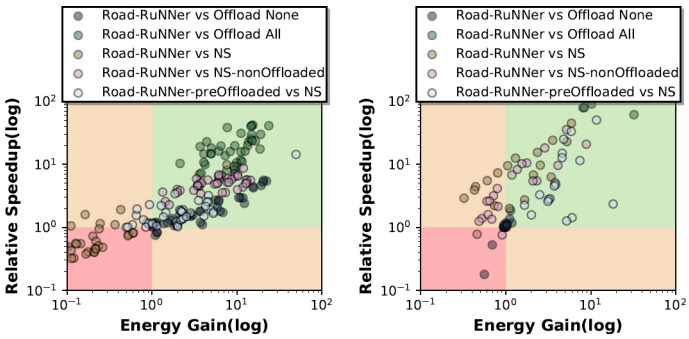
Reference Baselines: We evaluate the impact of our approach based on three key metrics: (i) performance (ii) energy consumption and (iii) prediction accuracy of our collaborative filtering approach. We compare against various partition and offloading mechanisms. First, as a baseline we utilize two naive approaches: (i) *Offload None*, which executes all tasks locally, without offloading anything on the cloud and (ii) *Offload All*, in which all tasks are offloaded to the cloud for remote execution. Moreover, we implement from scratch and compare against a state-of-the-art resource management algorithm for DNN offloading, namely *Neurosurgeon*(NS) [9]. Since *Neurosurgeon* is designed with the assumption to operate with a priori offloaded layers on the cloud infrastructure, we also implement a version of *Neurosurgeon* with online layer offloading, namely *NS-nonOffloaded* and a version of `RoaD-RuNNer`-*preOffloaded*, where the layers are offloaded a priori to the cloud.

B. Evaluation

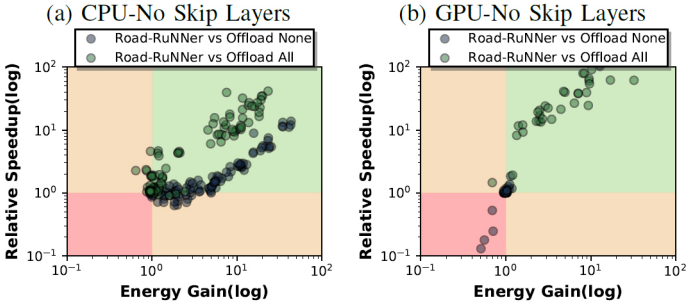
Performance and Energy Evaluation: In the first comparative experiment, we evaluate `RoaD-RuNNer` in terms of performance and energy consumption against the approaches presented in Section IV-A, as illustrated in Fig. 3. Given the fact that *Neurosurgeon* is designed to operate only over non-Residual Neural Networks, we utilize as benchmarks the VGG11,13,16 and AlexNet models. X axis indicates the corresponding energy gain, while Y axis denotes the relative speedup of our framework compared to other approaches for CPU (Fig.3a) and GPU (Fig.3b) execution, respectively. The output is divided in four distinct quadrants, on which `RoaD-RuNNer` does not achieve any speedup or energy gain (red), achieves either speedup only or energy gain only (orange) and achieves both speedup and energy optimization (green). We observe that *Road-RuNNer* clearly outperforms the *Offload All* and *Offload None* approaches, by leading up to 45.41× speedup, 95.84% energy reduction compared to the former and up to

TABLE I: Technical characteristics of heterogeneous Edge nodes and Cloud Server

Device	CPU	L2	L3	DRAM	GPU
Jetson Nano	4×Cortex-A57@1.4GHz ARMv8 64-bit	2MB	-	4GB	128×Maxwell@0.9GHz
Tegra X1	4×Cortex-A57@1.4GHz ARMv8 64-bit	2.5MB	-	4GB	256×Maxwell@1.0GHz
Xavier NX	6×Carmel@1.4GHz ARMv8.2 64-bit	6MB	4MB	8GB	384×Volta@1.1GHz
Xavier AGX	8×Carmel@2.2GHz ARMv8.2 64-bit	8MB	4MB	32GB	512×Volta@1.4GHz
Cloud Server	2×20 core Intel Xeon Gold 5218R@2.1GHz	1MB	28MB	128GB	5120xVolta-V100@1.2GHz



(a) CPU-No Skip Layers (b) GPU-No Skip Layers

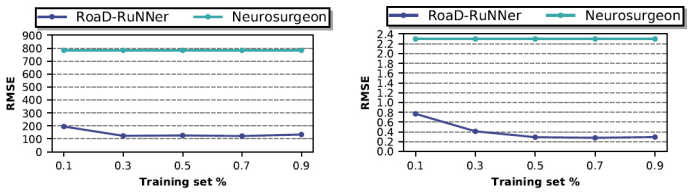


(c) CPU-Skip Layers (d) GPU-Skip Layers

Fig. 3: Performance and Energy Comparison of RoaD-RuNNer framework against other approaches for CPU and GPU nodes for alternative DNN workloads.

6.61× performance optimization and 95.87% energy reduction compared to the latter, for CPU execution. Similar observations are derived for GPU execution. The initial version of *Neurosurgeon*, on which all layer weights are offloaded a priori, performs better in terms of performance and energy consumption, as the network overhead is a dominant factor. However, compared to the *NS-nonOffloaded*, where layer weights are offloaded dynamically, we observe that `RoaD-RuNNer` provides on average 4.97× optimized performance and 81.85% less energy consumption for CPU, and up to 35.74× optimized performance and 88.73% less energy consumption for GPU, respectively. `RoaD-RuNNer` is designed to provide up to two partition points, in contrast to *Neurosurgeon*, on which there exists only a single breakpoint. Therefore, the latter pays the penalty of either executing locally all the resource intensive layers or offloading their weights, thus paying the network penalty. Similarly, `RoaD-RuNNer`-*preoffloaded*, where all model layers are offloaded offline, our approach outperforms *Neurosurgeon* by up to 54.09× in terms of performance and up to 58.06× in terms of energy consumption.

In contrast to *Neurosurgeon*, `RoaD-RuNNer` is designed to operate efficiently over workloads consisting of DNNs with skip layer connections. Thus, we add to the existing



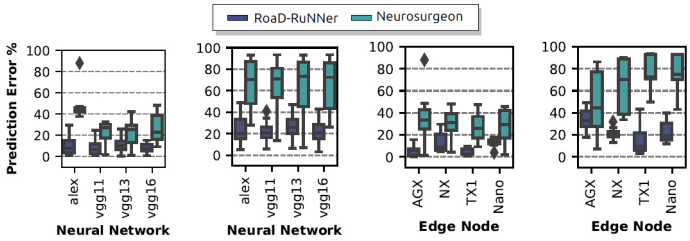
(a) Execution Time RMSE (b) Energy Consumption RMSE

Fig. 4: Root Mean Square Error(RMSE) of execution time and energy consumption over alternative Collaborative Filtering fill percentages.

benchmarks the Resnet18, Resnet34, Resnet50, Resnet101, Resnet152 and MobileNetV2 models and evaluate our approach. The evaluation is depicted in Fig. 3c and Fig. 3d for CPU and GPU executions, respectively. For CPU execution, our framework outperforms *Offload None*, by achieving up to 13.92× optimized performance and 46.07× less energy consumption and *Offload All* by achieving up to 45.41× optimized performance and 24.05× less energy consumption, respectively. Similarly, for GPU executions, `RoaD-RuNNer` achieves up to 1.85× speedup and up to 1.34× less energy consumption compared to *Offload None* and 112.98× speedup and 16.59× less energy consumption on average compared to *Offload All*.

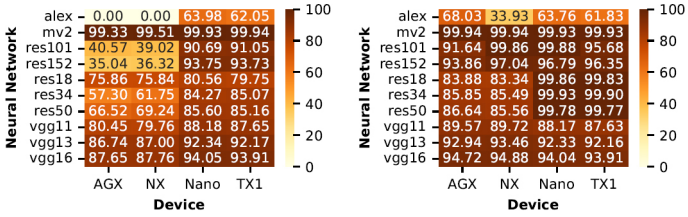
Prediction Accuracy: The efficacy of our framework is directly related to the accuracy of the collaborative filtering mechanism. First, we evaluate the learning phase of the prediction mechanism in terms of Root Mean Square Error (RMSE) related to the size (percentage) of training set and compare with the *Neurosurgeon's* prediction approach. We observe that providing the 30% of our training set, the execution time and energy RMSE has converged. Thus, we set our training set size to 0.3. Compared to *Neurosurgeon*, our proposed mechanism achieves up to 6.45× and 8.48× less RMSE, for execution time and energy, respectively. *Neurosurgeon* displays linear behavior, as it utilizes 100% of the dataset during the training phase.

Next, as illustrated in Fig. 5, we compare the execution time



(a) Latency per DNN model (b) Energy per DNN model (c) Latency per Edge Device (d) Energy per Edge Device

Fig. 5: Execution Latency and Energy Consumption Prediction Accuracy for alternative DNN workloads and Edge nodes.



(a) Latency Objective (b) Energy Objective

Fig. 6: DNN percentage offloading over heterogeneous Edge nodes for latency and energy optimization objectives.

accuracy and energy prediction accuracy of `RoaD-RuNNer` (Fig. 5a, 5b) and *Neurosurgeon* per deep learning model and Edge node (Fig. 5c, 5d), respectively. Our framework achieves 68.01% less execution time prediction error and 63.8% less energy prediction error per DNN on average, while we achieve up to 69.6% less execution time prediction error and up to 34.9% less energy prediction error per edge device. In contrast to our prediction mechanism, *Neurosurgeon* is based on linear and logarithmic regression, making it impossible to provide high accuracy (or low error) and capture non-linear behaviors in the system, thus leading to high RMSE.

DNN Offload Analysis: Further discussion can be conducted on the decision making of `RoaD-RuNNer`, in order to achieve latency and energy optimization objectives. Thus, in Fig. 6 the layer offloading percentage in terms of the number of layers offloaded for each model and edge node is depicted, in order to achieve each optimization objective, respectively. First, as depicted in Fig. 6a, we see that the more powerful devices (AGX, NX) offload less computation for remote execution, compared to less powerful devices (Nano, TX1). More specifically, for AGX and NX the 58.7% of the target DNN is offloaded on average, opposed to Nano and TX1, where the 87.1% is offloaded. Similar observations are extracted for the energy optimization objective, as shown in Fig. 6b. Furthermore, having execution latency as the optimization objective, the 74.45% of layers is offloaded on average, while for the energy optimization objective the corresponding percentage rises to 90.1%. This is due to the fact that the network imposes high latency overhead, thus making the data and layer transmission prohibitive in order to meet latency optimization objectives.

V. Conclusion

This work presents `RoaD-RuNNer`, a novel resource management framework for DNN partitioning and offloading over heterogeneous CPU/GPU edge computing systems. Our framework strongly leverages collaborative filtering techniques to estimate performance and energy requirements of individual DNN layers over heterogeneous devices. By aggregating this information, it specifies a set of Pareto optimal DNN partitioning schemes that trade-off between performance and energy consumption. Our approach outperforms existing state-of-the-art approaches by achieving 9.58× speedup on average and up to 88.73% less energy consumption and high prediction accuracy by limiting the prediction error down to 3.19% and 0.18% for latency and energy, respectively.

References

[1] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[2] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2018.

[3] Y. Liang, D. O'Keeffe, and N. Sastry, "Paige: Towards a hybrid-edge design for privacy-preserving intelligent personal assistants," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 55–60, 2020.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[6] 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.

[7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE communications surveys & tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[8] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[9] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[10] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 401–411, 2018.

[11] 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," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[12] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–37, 2020.

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

[14] L. Zhou, H. Wen, R. Teodorescu, and D. H. Du, "Distributing deep neural networks with containerized partitions at the edge," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[15] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.

[16] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 77–88, 2013.

[17] E.-I. Christoforidis, S. Xydis, and D. Soudris, "Cf-tune: Collaborative filtering auto-tuning for energy efficient many-core processors," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 25–28, 2017.

[18] "Edge computing: gaining the digital edge." https://atos.net/en/solutions/edge-computing-infrastructure. Accessed: 10-09-2022.

[19] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[20] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, 2019.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.