In depth in-network neural network performance analysis

Hesam Tajbakhsh
Dalhousie University
Halifax, Nova Scotia
hs762129@dal.ca

ABSTRACT

[add abstract]

In this project, we investigate running neural networks on *SmartNICs* without any accuracy loss. Previous works modified their neural networks to be compatible with their *SmartNICs* due to their device's limitation. Our goal is to figure out if we can execute an accurate model on a different *SmartNIC* to release some tasks from the host and gain the same or even better performance.

ACM Reference Format:
Hesam Tajbakhsh. 2021. In depth in-network neural network performance analysis. In *Proceedings of Final Report for the SDN course (SDN 2021).* ACM, Halifax, NS, Canada, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Reconfigurable network cards- or in other words *SmartNICs* , offering more resources in datacenters, are becoming increasingly popular in recent years. There are several kinds of *SmartNIC* , and in this report, first, we will briefly introduce two main architectures of them and their advantages. Afterward, we will review Artificial Neural Networks (ANN) to provide adequate background for our work. This work will focus on executing Machine Learning (ML) and Deep Learning (ML) applications on *SmartNICs* . Our primary reference [16] has examined ML/DL Neural Networks (NNs) on two types of *SmartNICs* , but they penalized some parameters (e.g., the accuracy of NNs); however, the throughput and latency are enhanced significantly.

By and large, each *SmartNIC* has its limitations and advantages. We aim to figure out whether we can execute these applications on a *SmartNIC* without sacrificing the accuracy. So, we will pick the *SmartNIC* which has not been investigated for these purposes and investigate if the *SmartNIC* helps us to achieve our goal or not. Our results show the selecting the suitable model for our *SmartNIC* 's type is crucial due to its limitation. We will evaluate two types of ANNs, and yield some information showing which one can be a better fit than the other.

With this in mind, the rest of the article is organized as the following. In section 2, *SmartNICs* are introduced. We will also talk about artificial neural networks in section 2. In 3 we talk about offloading ANN into *SmartNICs* and explain our goal. Our testbed details and results are studied in sections 4 and 5, respectively. Then, we will address a few related works in section 6. Eventually, in section 7, we will conclude our project based on the provided results.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SDN 2021, April 15, 2021, Halifax, Nova Scotion, Canada
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

We divide this section into two parts; introducing *SmartNICs* briefly and improving the overall performance by *SmartNICs* are discussed in the first part. Then, we talk about Artificial Neural Networks (ANN) in the second part.

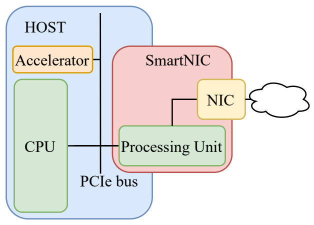
2.1 *SmartNICs* Overview

A glance at Fig. 1 reveals two architectures for *SmartNICs* . In the first architecture, *SmartNIC* 's processing is located on the path of networking, while in the latter one, Fig. 1b, the processing unit is located off the path, communicating with the NIC and the host through PCI Express slot (Peripheral Component Interconnect Express). There are several choices for the *SmartNIC* 's processing unit. On-the-shelve CPUs, FPGA, and ASIC are some instances that can be mounted on the *SmartNICs* . In the case of general-purpose CPUs, most of them are ARM-based or MIPS-based wimpy processors.

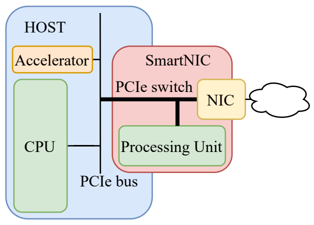
Owing to the high bandwidth demands of *SmartNICs* , they are connected to hosts over the PCI Express slot, as mentioned in the previous paragraph. However, PCI Express prompts for extra latency overhead in some cases. Keeping this in mind, either executing an application or offloading part of it into on-path *SmartNICs* can exclude PCI Express extra latency. Nevertheless, FPGAs and CPUs used in on-path

SmartNICs have limited resources and capabilities. For example, they usually do not support floating-point operations or have a limited memory footprint. It goes without saying that offloaded applications should be modified to be compatible with *SmartNICs* environments and may lose some traits. We will talk about different techniques of modification in the second half of this section, where we discuss ANNs.

Worth to mention, for each kind of *SmartNICs* there are a few brands available in the market; some of them are listed in Table 1.



(a) On-path Architecture



(b) Off-path Architecture

Figure 1: Two architectures for *SmartNICs*

Table 1: A Few *SmartNICs* Available in the Market

	Available Brands in the Market
On-Path FPGA-based	NetFGPA-SUME [18]
On-Path CPU-based	Liquid IO [4]
Off-Path CPU-based	Stingray [3] and BlueField [11]

2.2 Artificial Neural Networks (ANNs)

[Let's focus our analyses on deep neural networks (DNNs) as these are currently more appealing. Give some background on them. What is a DNN? Where are they used (i.e., most common applications)? What about networking use cases? Give an overview about different architectures (i.e., show they are pretty varied - CNN, RNN, GNN, fully NN). Talk about different flavors (prunned, compressed, quantized)] Personally, I prefer ANN rather than DNN since some of our implementations are ANN, not necessarily DNN

In a nutshell, an artificial neural network- or ANN as we call in the rest of the report-is a network of neurons connecting with weighted vertices. An ANN can have one or more than one hidden layer. Also, the nodes/layers can organize different topologies. For instance, Fig. 2 depicts an ANN with two hidden layers, and each node is connected to all the following layer's nodes, knows as Fully Connected ANN. This architecture is use for Multi-layer Perceptron (MLP) classifiers.

Furthermore, an ANN will be called a Deep Neural Network (DNN) if it has many layers. In more detail, various

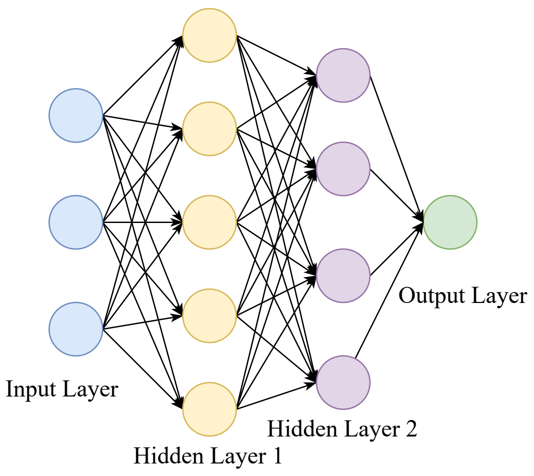


Figure 2: An Fully Connected ANN with 2 hidden layers

architectures are introduced for DNN solving different complex problems. Some notable examples are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The former one is vastly used for video analysis and image classification. Yet, Natural Language Processing is one of use cases for the latter one.

As has been mentioned before, *SmartNICs* have limited resources. On the other hand, ANNs are resource-hungry applications. Consequently, the original ANNs applications cannot execute on the *SmartNICs* without any modifications. Here, three solutions are listed for reducing the size of any ANNs, including fully connected NNs and DNNs.

- **Compressing**: In this method, the original model is converted to a compressed one via the framework's tools. The compressed one has 2x-4x less size without a prominent loss in the outputs (less than 0.5% in our results). An example of this method is TensorFlow lite [1].
- **Quantization**: In this method, more miniature data types are replaced with original data types. In other words, the quantized network has the same topology while the weights of vertices change. In the AI area, using 64bits or 32bits floating-point numbers is very common. Hence, replacing them with 8bit width data types drops the size significantly. Some studies go further and use only one bit for quantization, generating Binarized Neural Networks (BNNs). It goes without saying that the quantized model is not as accurate as of the original model.
- **Pruning**: The idea behind this method is to remove the vertices with small weights, or the vertices do not affect the output that much. Therefore, the topology becomes simpler here.

Up to here, we have provided adequate background regarding ANNs. The question coming to mind is **which ANN do we study in our project?** As we have mentioned, each ANN is suitable for a specific problem. The same in the Computer Networks area; based on the nature of the issue, the ANN's architecture would be chosen. For packet classification, fully connected ANNs provide acceptable accuracy. However, for analysis MPLS configuration, a deep NN provides a viable outcome [6]. Here, we will evaluate two kinds. The first one would be VGG16 [15], which is basically a deep NN for Image Classification. The second one would be a fully connected ANN for anomaly detection. We try to avoid quantizing or pruning the models, and utilizing the original model or the compressed one to keep the models accurate in our project.

3 OFFLOADING NEURAL NETWORKS INTO *SMARTNICS*

[Talk about the necessities for accelerating NN/DNNs. Introduce the different opportunities/options of accelerators we have (GPU,TPU). Make the case for using SmartNICs as accelerators. Describe the efforts people already did in that sense (this related work, e.g., gianni's paper goes here. Other (less) related work goes into related work section. Mention we want to shed light into the capabilities of SmartNICs as NN accelerators. Give a very high overview of our evaluation and evaluation goals (say we tested different neural network types on different SmartNICs to understand their performance). Evaluation details (i.e., methodology) will be described in the next section).]

Emerging applications, e.g., Deep Learning based applications - those using DNN and video analytics and image classification, pose a challenge to cloud-based services. These applications demand high network bandwidth and low latency that may not be possible due to the long communication distance between mobile devices and cloud servers. One alternative is to deploy servers closer to mobile users, which is called edge computing. *SmartNICs* also can be used as edge servers providing fast services to the mentioned applications. In a typical fashion without any *SmartNIC* , the NIC, host's CPUs, and GPU/TPU (two standard accelerators for ANN) communicate through PCIe express slot. Once a packet reaches the NIC, it goes through PCIe to CPUs. Then it should be passed to GPU/TPU, again through PCIe. The response should traverse the reverse path, meaning PCIe is passed four times in this scenario.

What will happen if we execute the model on a *SmartNIC* and avoiding the PCIe slot's overhead. Siracusano *et al.* [16] conducted novel research focusing on running deep learning applications on *SmartNICs* . The offloading inference phase of three network-based machine learning applications to *SmartNICs* is studied in the mentioned work. They compared three implementations, on Netronome, on P4-NetFPGA, and HDL. In terms of IPC, CPUs of *SmartNICs* are more efficient than hosts'. Therefore, *SmartNICs* are viable candidates for processing those requests. Due to limited resources available in their hardware, they applied Binarized Neural Network to decrease the memory demand and resolving the lack of floating-point operations on *SmartNICs* . However, they lost some accuracy because of quantization. Though, they can offer much higher throughput and less latency by applying their approach.

In the following section, we evaluate different ANNs on a different *SmartNIC* having enough resources and supporting floating-point operations to understand whether we can have quicker latency without losing latency. Moreover, we alleviate the host's CPUs by executing models into the *SmartNIC* .

4 MEASUREMENT SETUP

[Describe the devices we are evaluating]

In this section, we addressed our testbed in detail. Firstly, we explain our hardware setup, devices' specs, and how they are connected. Knowing hardware testbed, we explain our software environments. Fig 3 shows an overview of both hardware and software of our experiments. Eventually, we speak about our performance metrics.

4.1 Hardware Testbed

As Fig. 3 exposes, on the left side, we use an off-path *SmartNIC* equipped with a Cortex-72 ARM CPU working at 800 MHz, with 16 cores and 16GB onboard DDR4 memory. It holds a dual-port SFP28 with 25Gbps for each one. It is mounted on the host, which owns an Intel(R) Xeon(R) Silver 4210R CPU working at 2.4GHz, with 10 cores and 32GB memory. Further details regarding cache are reported in Table 2. Either the host or the *SmartNIC* act as the server machine connecting to a client over a Tofino switch. The client is equipped with an Intel(R) Core(TM) i7-9700 CPU working at 3.0GHz, with 8 cores and 12GB memory.

Table 2: Size of Cache in the host and *SmartNIC*

Device	D-Cache	I-Cache	L2	L3
SmartNIC (ARM)	512KB	768KB	8MB	12MB
Host (x86)	320KB	320KB	10MB	14MB

4.2 Software Setup

On the server-side, Ubuntu 20.04.1 LTS with kernel 5.4.0-66-generic is installed on the host. The same OS runs on the client machine, as well. Nonetheless, the *SmartNIC* executes Ubuntu 20.0.1 with a modified kernel 5.4.44.

For implementing our ANNs, we used two frameworks;

- **TensorFlow** [2]: We implemented VGG16 [15] with TensorFlow. Our model recognizes type of pets based on the images. We adopted "Cats vs Dogs" dataset [12] to train and test our model. Moreover, we compressed the trained model with TensorFlow lite [12], and ran that version on our *SmartNIC* . Note to mention, both machines support floating-point operation, and we do not apply any quantization method.
- **scikit-learn** [14]: By this framework, we implemented the second used case in [16], with the same dataset, but without any quantization. The neural network is an MLP classifier for anomaly detection in the UNSW-NB15 dataset [13], having three layers with the size of (32, 16, 2).

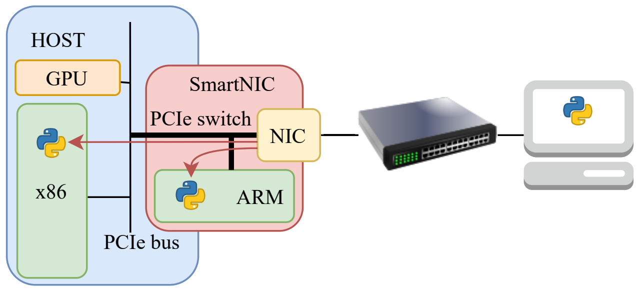


Figure 3: The testbed

4.3 Workload

[Describe how we generate the workload in our experiments]

For evaluating the system, we have developed a socket-based client-server application in Python3. The client program, running on the right-hand machine of Fig. 3, data is read from the dataset; Then, a request is transmitted to the server program, running either on the host or on the *SmartNIC* on the left side Fig. 3, over a TCP connection. The server code runs the model and waits for requests. Once it receives any, it captures and decodes the data, performs the corresponding operations, and finally returns the response to the client. Besides, we can break down the overall latency by adding timestamps in our code. Our implemented codes are released in our Git repository [17]

4.4 Measured resources

[Describe our metrics and how we measure them in each device]

Once a neural network is implemented, the first point coming to mind is how accurate it is. With this in mind, in our experiments, we divide our datasets into two parts, training data (80% of whole data) and test data (the rest 20% of data). The trained models are evaluated by unseen test data at the end of the training, and the accuracy is measured.

The following parameter reserving some words here is the latency. As we addressed in previous lines, we implemented our code in Python3. We can measure the overall latency in our client program. Once the code sends a request, it also starts the timer. Whenever it captures the response, stops the timer, and calculates the overall latency. With the same timestamping approach, we can break down the latency in the server code to extract communicating time and executing time, as a further illustration.

To clarify the results, we report roofline results. Roofline, proposed in [10], is a benchmark estimating the performance in different levels of caches, memory, and CPU. It reports the maximum bandwidth of caches and memory in terms of GB/s, the maximum computation power of the CPU in terms of GFLPs/sec.

5 RESULTS

In this section, we report the results gained in the given testbed. First, we lecture about VGG16's results. Then, we report roofline results to clarify the results we provided. At the end of this section, we express the MLP classifier's results.

5.1 VGG16's Results

Our evaluation after training depicts the original model and the lite version have 90.6% and 90.1% accuracy, respectively. Besides, Table 3 shows the latencies for VGG16 applications. First, we measured the overall latencies for both the *SmartNIC* and the host. The host is 5x faster to calculate the response with the VGG16 model due to results compared to the *SmartNIC* . Considering this, we break down the overall latency to find out which time slot needs more time. As we can understand from the fourth column of Table 3, performing the model operations takes 5x times inside *SmartNIC* 's CPU. For better understanding, we perform Roofline benchmark to figure out if the results make sense. The results are reported in Table 4. The results in the given table show that the host is 8x-10x quicker than the *SmartNIC* . Since VGG16 is a very deep neural network demanding many complex operations, and the *SmartNIC* has limited resources, the overall latency and computation time for the *SmartNIC* are reasonable.

5.2 MLP Classifier's Results

In these experiments, we load exactly the same model either on the *SmartNIC* or on the host for anomaly detection. Consequently, in both cases, the accuracy is the same, nearby 96.03%. Since the load for this experiment is not much, we run the server code on one core. In addition to that, we increased the number of connections on the client-side. The measured results are expressed in Table 5 illustrating how long it takes

Table 3: VGG16's Results

Device	Framework	Total Latency	Computation Time	Loading the Image	Communication Time
SmartNIC (ARM)	TensorFlow lite	493.58 ms	470.99 ms	16.88 ms	5.71 ms
Host (x86)	TensorFlow	107.05 ms	97.98 ms	5.21 ms	3.86 ms

Table 4: Roofline's Results

Device	CPU Bound	Max L1 Speed	Max L2 Speed	Max L3 Speed	Max DRAM Speed
SmartNIC (Arm), One Core	1.1 GFLOP/s	14.5 GB/s	4.7 GB/s	3.2 GB/s	2.4 GB/s
SmartNIC (ARM), 16 Cores	17.6 GFLOP/s	231 GB/s	19.9 GB/s	14.1 GB/s	10.6 GB/s
Host (x86), One Core	9.1 GFLOP/s	101.0 GB/s	73.2 GB/s	36.3 GB/s	19.7 GB/s
Host (x86), 16 Cores	91 GFLOP/s	1009.4 GB/s	49.9 GB/s	35.2 GB/s	26.8 GB/s

for the server to respond. Excluding eight connections, both the *SmartNIC* and the host have more or less the same latencies. Averages are less for the *SmartNIC* than the host, while the standard deviation is higher than the host. Once we set the number of connections to eight, the *SmartNIC* works fine, with reasonable changes in the results. However, the host's CPU utilization reaches 100%; consequently, the connections are refused, and the numbers would not be acceptable anymore.

Further, the communication times demonstrate that the *SmartNIC* would be a good fit for the applications demanding high communication, such as I/O intensive loads. In other words, the *SmartNIC* has faster access to the PCI Express slot than the host; keeping in mind that several devices can be mounted on the host through the slot, and x86 should handle the challenge as well.

Evaluating these applications is beyond our project. Eventually, owing to these results, we can understand, MLP classifiers are a good fit for executing on the *SmartNIC* ; one wimpy core of the *SmartNIC* can have the same performance compared to one powerful core of the host.

6 RELATED WORK

Offloading other applications to SmartNICs.

Many researchers studied different aspects of *SmartNICs* , and how *SmartNICs* can be used wisely to provide better performance even with comparison hosts with powerful CPUs. Long story short, we picked a few of them here.

Ming *et al.* in [8] extended three well-known distributed applications to use the proposed framework's APIs (called iPipe) and benefit *SmartNICs* interests, such as saving up the Host's CPU and more agile latency. The applications explored are a replicated key-value store, a distributed transaction system, and a real-time analytics engine. Furthermore, two on-path and two off-path *SmartNICs* using embedded CPU have been investigated. Characterizing the performance of each *SmartNICs* , they acquainted iPipe framework for distributed applications. The framework providing actor-based programming consists of three major components, an actor scheduler, a distributed memory object abstraction for actor migration, and, eventually, a security isolation mechanism. Carried out experiments manifest that on-path *SmartNICs* show better performance than off-path ones.

Daniel *et al.* [5] conferred a solution to offload networking in a virtualized environment into *SmartNICs* and accepted the FPGA-based one as the best decision for both speed and programmability. Single Root I/O Virtualization is introduced in a nutshell, allowing VMs to have direct access to *SmartNICs* . Moreover, the NIC is responsible for all network management. Their given method provided the quickest latency for a public cloud at the time the article was published.

E3 paper [9] employed Cavium LiquidIO *SmartNICs* , which is equipped with MIPS architecture CPUs, to offload a few microservice-based applications. The goal, as same as other works, is to reduce the latency and power consumption. In a nutshell, the tool, by knowing the system's topology and eliminating extra overhead, distributes the load among the resources in heterogeneous systems with various architectures, x86, and ARM. The idea is to run microservices on *SmartNICs* as long as enough traffic is passed to the hosts.

Many IoT wireless devices have ultra-limited resources; nevertheless, many deep learning applications hoping to be run on the edge need huge resources; take wireless cameras as an example that captures photos to be processed. Moreover, the network's bandwidth may fluctuate for the devices. CLIO paper [7] introduces a prototype for those kinds of tools. Previous papers had tried to either reduce the raw data or compress the models; the accuracy dropped, consequently. Considering a deep learning model, CLIO splits the model, and a part of the model with a few layers is offloaded into devices. Further, different slices, with different sizes, from the last layer have been created. The prototype decides to transmit the best choice based on the available bandwidth.

Table 5: Average Latencies for the MLP classifier to Detect the Anomaly.

Average (Standard Deviation) of the Latancies in ms				
Device:	*SmartNIC* (ARM)		Host (x86)	
The Number of Connections	Overall Latency	Communication Time	Overall Latency	Communication Time
1	3.89 (2.99)	1.14 (0.18)	4.01 (1.01)	1.67 (0.19)
2	4.00 (7.98)	0.62 (0.42)	4.02 (1.46)	1.69 (0.9)
4	4.2 (19.41)	0.2 (0.12)	4.6 (3.13)	1.10 (1.49)
8	12.56 (31.04)	0.47 (4.44)	N/A	N/A

Hence the bandwidth is used efficiently; simultaneously, reasonable accuracy is achieved. Keep in mind that the outputs of middle layers have much less data compared to raw data.

7 CONCLUSION

In this project, we first talked about *SmartNICs* and a few architectures for them; then, we discussed each architecture has its advantages. However, all of them release resources from the host. So, data center administrators take this advantage to move applications into *SmartNIC* , freeing more resources on the servers for tenants.

Later, we spoke about the importance of neural networks and their popularity in different applications. Focusing on *SmartNIC* , we provided a project explaining some case studies requiring fast latency and lower bandwidth. In the project, the researchers adopted on-path *SmartNICs* having very limited resources. So, they quantized their neural networks to reduce the model size. However, those *SmartNICs* had not supported floating-point operations. Their limitations lead them to use binarized neural networks, losing accuracy and being compatible with the devices.

In this project, we chose an off-path *SmartNIC* which has more resources and supports floating-point operations. Therefore we do not lose accuracy more than 0.05%. Our results show that ARM-based *SmartNICs* still have limited resources and computation power than the hosts, and they are not a suitable environment for executing complex neural networks such as VGG16. Running VGG16 on our *SmartNIC* takes approximately 5x more time than the host. Conversely, ARM-based *SmartNICs* would be competitive resources for executing simpler neural networks such as MLP classifiers due to their quicker communication time. We were able to offload a workload running on one core of the host to one core of the *SmartNIC* , without any change in the outputs, and still have more space in the *SmartNIC* .

REFERENCES

[1] 2021 (accessed April 15, 2021). *TensorFlow Lite | ML for Mobile and Edge Devices.* https://www.tensorflow.org/lite.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16).* 265–283.

[3] Broadcom. 2021 (accessed April 15, 2021). *Stingray PS225.* https://docs.broadcom.com/doc/PS225-PB.

[4] Cavium. 2021 (accessed April 15, 2021). *LiquidIO-II 10/25G Smart NIC Family.* https://www.marvell.com/content/dam/marvell/en/public-collateral/ethernet-adaptersandcontrollers/marvell-ethernet-liquidio-ii-cn23xx-10g-25g-product-brief-2017.pdf.

[5] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18).* USENIX Association, Renton, WA, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[6] Fabien Geyer and Stefan Schmid. 2019. DeepMPLS: fast analysis of MPLS configurations using deep learning. In *2019 IFIP Networking Conference (IFIP Networking).* IEEE, 1–9. https://doi.org/10.23919/IFIPNetworking46909.2019.8999396

[7] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. 2020. CLIO: Enabling Automatic Compilation of Deep Learning Pipelines across IoT and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20).* Association for Computing Machinery, New York, NY, USA, Article 58, 12 pages. https://doi.org/10.1145/3372224.3419215

[8] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using IPipe *(SIGCOMM '19).* Association for Computing Machinery, New York, NY, USA, 318–333. https://doi.org/10.1145/3341302.3342079

[9] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19).* USENIX Association, Renton, WA, 363–378. https://www.usenix.org/conference/atc19/presentation/liu-ming

[10] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. 2015. Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond (Eds.). Springer International Publishing, Cham, 129–148.

[11] Mellanox. 2021 (accessed April 15, 2021). *BlueField SmartNIC for Ethernet High Performance Ethernet Network Adapter Cards*. https://www.mellanox.com/sites/default/files/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf.

[12] Microsoft. [n. d.]. Cats and Dogs Dataset. https://www.kaggle.com/karakaggle/kaggle-cat-vs-dog-dataset. ([n. d.]). Accessed: April 15, 2021.

[13] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). https://doi.org/10.1109/MilCIS.2015.7348942

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[15] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[16] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Hamed Haddadi, Gianni Antichi, and Roberto Bifulco. 2020. Running Neural Networks on the NIC. *arXiv preprint arXiv:2009.02353* (2020).

[17] Hesam Tajbakhsh. [n. d.]. SDN2021-InNetworkNN. ([n. d.]). https://github.com/hesam4g/SDN2021-InNetworkNN.git

[18] Xilinx. 2021 (accessed April 15, 2021). *NetFPGA-SUME Virtex-7 FPGA Development Board*. https://www.xilinx.com/products/boards-and-kits/1-6ogkf5.html.