Hardware-aware Partitioning of Convolutional Neural Network Inference for Embedded AI Applications

Fabian Kreß, Julian Hoefer, Tim Hotfilter, Iris Walter, Vladimir Sidorenko, Tanja Harbaum, Jürgen Becker
Karlsruhe Institute of Technology
Karlsruhe, Germany
{fabian.kress, julian.hoefer, hotfilter, iris.walter, vladimir.sidorenko, harbaum, becker}@kit.edu

***Abstract*—Embedded image processing applications like multi-camera-based object detection or semantic segmentation are often based on Convolutional Neural Networks (CNNs) to provide precise and reliable results. The deployment of CNNs in embedded systems, however, imposes additional constraints such as latency restrictions and limited energy consumption in the sensor platform. These requirements have to be considered during hardware/software co-design of embedded Artifical Intelligence (AI) applications. In addition, the transmission of uncompressed image data from the sensor to a central edge node requires large bandwidth on the link, which must also be taken into account during the design phase.**

Therefore, we present a simulation toolchain for fast evaluation of hardware-aware CNN partitioning for embedded AI applications. This approach explores an efficient workload distribution between sensor nodes and a central edge node. Neither processing all layers close to the sensor nor transmitting all uncompressed raw data to the edge node is an optimal solution for each use case. Hence, our proposed simulation toolchain evaluates power and performance metrics for each reasonable partitioning point in a CNN. In contrast to the state of the art, our approach does not only consider the neural network architecture. In the evaluation, our simulation toolchain additionally takes into account hardware components such as special accelerators and memories that are implemented in the sensor node.

Exemplary, we show the simulation results for three commonly used CNNs in embedded systems. Thereby, we identify advantageous partitioning points regarding inference latency and energy consumption. With the support of the toolchain, we are able to identify three beneficial partitioning points for FCN ResNet-50 and two for GoogLeNet as well as for SqueezeNet V1.1.

***Index Terms*—convolutional neural networks, embedded systems, hardware accelerator, simulation toolchain, hardware/software co-design**

I. Introduction

In recent years, research on Deep Neural Networks (DNNs) has gained immense attention due to their superiority in image processing enabling applications and manifold use cases such as autonomous driving, intelligent prosthetics or assistive robotics [1]. However, the increasing complexity of neural networks and their intense workloads present unprecedented challenges to the embedded hardware.

These challenges can no longer be met by general-purpose CPUs or GPUs, but solely with dedicated hardware accelerators and methods of hardware/software co-design, allowing for efficient implementation of the DNN algorithms. The improvements in performance and energy efficiency can surpass 100x and 1000x, respectively [2]. However, the requirements of the use cases continue to increase, as applications like autonomous driving require multiple cameras with additional sensors instead of static images for accurate perception of the near environment and full surround view.

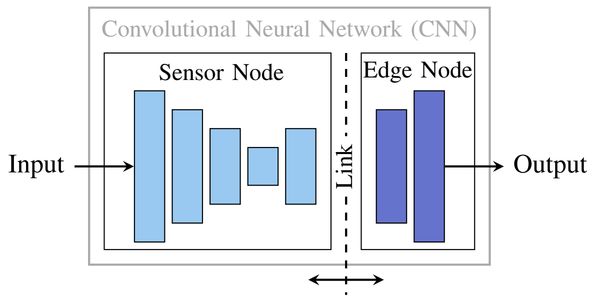

Fig. 1: The simulation toolchain evaluates different partitioning of a CNN model on different nodes. Both, sensor node and edge node are embedded platforms. The link layer transmits intermediate results from one node to the other and can be in between any network layer.

For example, in assistive robotics, personalization and providing an assistance in real-time is crucial for user acceptance. For personalization, the robot perceives its environment and recognizes people, which is often realized by observing the proximity with multiple distributed cameras and sensors. To realize a real-time person recognition while respecting privacy, local execution on dedicated accelerators [3] is preferred over remote computation.

Furthermore, Advanced Driver Assistance System (ADAS) systems is another emerging topic in this scope. Camera- and sensor-based systems with DNN-based object detection and semantic segmentation for evaluation are the foundation for lane assistants, face recognition [4] or fully autonomous vehicles. Cameras or sensors are connected over an on-board network to centralized Electronic Control Units (ECUs), which can be regarded as edge nodes. To reduce the amount of data that has to be transferred via the on-board network, the sensor node itself can process parts of the DNN. This not only helps reduce the amount of transferred data, but also increases energy efficiency and keeps communication latency low.

To support the implementation of DNN-based applications into embedded and distributed systems, we present a simulation toolchain for hardware-aware Convolutional Neural Network (CNN) partitioning. In our tool flow, the previously described use cases are to be approximated with the help of a simplified model, shown in Fig. 1. A first partition of DNN layers is calculated on a sensor node, while the second partition is computed on an edge node. The output feature map of the last CNN layer processed in the sensor node is transferred over a wired link, such as Ethernet. Our contributions in this paper are as follows:

- We present our simulation toolchain for evaluating energy and latency of CNN partitioning in embedded systems
- We show, how we apply the toolchain to obtain latency and energy consumption of applications running on distributed embedded systems
- We evaluate and identify beneficial partitioning points of multiple, commonly used CNNs, namely FCN-ResNet50, GoogLeNet and SqueezeNet

II. Related Work

In many vision-based applications, DNN inference on sensor devices is not a feasible solution due to the limited compute performance and tough constraints regarding power consumption [5]. Consequently, computational intensive operations such as inference of Convolution (CONV) layers are offloaded to more powerful platforms. However, several accelerators for sensor platforms have already been proposed in recent years to enable inference of small CNNs on sensor devices. As an example, the analog accelerator RedEye is directly connected to a pixel array of a camera [6]. It consists of modules for processing CONV, max pooling and quantization operations of which each can be used multiple times for single inference. Furthermore, the authors evaluated partitioning GoogLeNet based on their accelerator design, showing an improved latency and slightly reduced overall power consumption when paired with a Jetson TK1 GPU.

Besides of designing specialized accelerators for embedded platforms to meet latency and energy constraints, little research has been carried out considering DNN inference partitioning over multiple nodes.

An edge-host partitioning approach has been presented by Ko et al. [7]. They designed and synthesized an inference engine containing 144 16-bit MAC units, an on-chip buffer and a JPEG encoder and decoder which allows storing weights in compressed format. They were able to prove that partitioning of DNNs can be beneficial regarding throughput and energy-efficiency. However, they do not provide a hardware-aware design space exploration for DNN partitioning.

Hu et al. presented a partitioning algorithm for CNNs optimizing throughput and accuracy of a pipelined inference [8]. Their approach includes an autoencoder-based compression scheme to improve the inter-device communication performance. Similar to CNN partitioning presented in this paper, the first part of this DNN, i.e. the encoder, is executed by the transmitting node while the decoder section is processed in the receiver. However, the proposed methodology neither considers power consumption nor the actual hardware implementation used for inference in the sensor node. Such an approach would be beneficial in identifying an optimal partitioning of the CNN for a specific hardware platform.

The open-source framework DeepThings, proposed by Zhao et al. [9], is targeting dynamic balancing of workloads in edge clusters, achieving reduced local memory usage and significant inference performance improvement. Under consideration of the platform constraints and the CNN parameters, this approach divides the CONV layers into independent tasks that can be distributed over the edge nodes available in the network. Nevertheless, DeepThings only considers optimizations of scheduling tasks over multiple edge platforms. This methodology neglects the actual hardware implementation, such as specialized hardware accelerators. Therefore, this approach only allows for latency and throughput improvement based on dynamic workload balancing, but does not enable evaluation of overall energy efficiency across sensor and edge nodes.

Yang et al. proposed a co-exploration methodology of hardware design space and neural architecture for realtime Artificial Intelligence (AI) applications [10]. Thereby, the execution of DNNs is partitioned on multiple Field-Programmable Gate Arrays (FPGAs) based on a network-on-chip infrastructure. Applying a feedback loop for exploration, their approach can lead to improved accuracy of the DNN and increased hardware efficiency. However, this methodology again disregards energy consumption of the hardware and, therefore, is not applicable for optimizing energy efficiency in embedded AI.

III. Simulation Toolchain

Recent embedded systems are composed of distributed compute platforms, which enable data preprocessing locally at the sensor to reduce the amount of data to be transmitted to centralized control units. However, in many multi-camera based embedded applications, e.g. automotive or robotics, preprocessing data in each node is not always sufficient to maintain the overall bandwidth limitation of the on-board network. Such systems are therefore constrained by a maximum number of possible sensors in the entire system.

Though, a rising number of sensors deployed in embedded systems can be observed in recent years. By processing parts of the workload of the central edge platform already in the sensor node the demand for additional sensors in e.g. assistive robots or autonomous driving cars can be fulfilled. Nevertheless, embedded systems are constrained in energy consumption and latency in order to operate on limited energy budget and real-time requirements. In this paper, we address this major challenge by integrating highly specialized hardware accelerators into the sensor node and by investigating optimal workload distribution in a distributed system.

Partially offloading the CNN workload to different platforms while respecting the constraints, though, increases the complexity of the hardware/software co-design. Therefore, we pro-

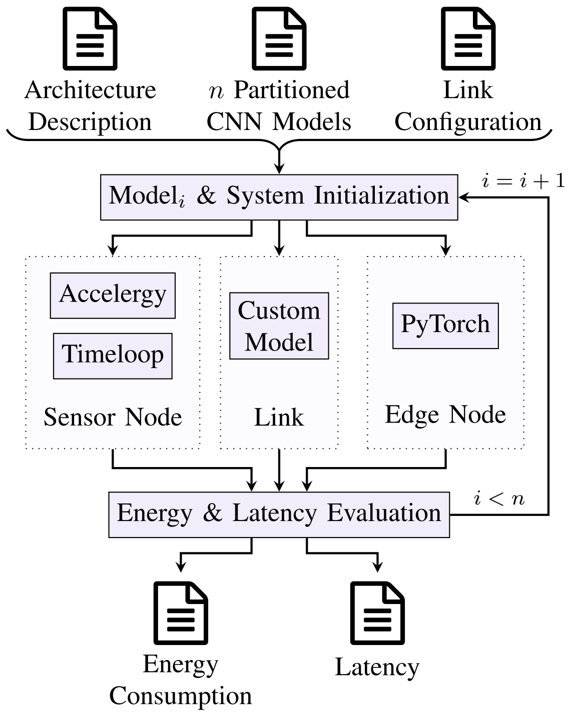

Fig. 2: Overview of our simulation toolchain estimating energy consumption and latency for each system module independently. It expects the architecture description of the sensor node, n partitioned CNN models and the link configuration as input, loops through the models and outputs the estimations in spreadsheet format.

pose a simulation toolchain, which allows fast exploration of the design space for distributed compute platforms consisting of sensors and central edge node. An overview of the toolflow is provided in Fig. 2.

With this simulation toolchain we aim at optimizing the hardware architecture of the sensor node as well as the CNN partitioning efficiency. For this reason, consideration of a single sensor node connected to the central edge node via a link is sufficient. However, dedicated constraints, such as the maximum available bandwidth, can be used to account for the impact of additional sensor nodes in the system.

Our toolchain is based on established tools and custom models to guarantee for compatibility with widely used open-source deep learning libraries, such as PyTorch, and designed to fit a wide range of use cases. As input, the toolchain takes:

- an architecture description of the hardware deployed in the sensor node including memories and accelerators.
- a set of CNN models for exploration that have to be partitioned in advance, as demonstrated in section IV.
- a link configuration containing important characteristics for estimating energy consumption and latency.

The actual simulation is divided into three independent segments that run in parallel to significantly reduce the overall simulation time. In this process, each system module, i.e., sensor node, link, and edge node, simulates or measures energy consumption and latency for the given CNN model independently. The evaluation module collects the results of the preceding simulations and calculates overall energy consumption and latency for the partitioned CNN model. Afterwards, the remaining models are evaluated analogously.

Finally, the simulation toolchain provides energy consumption and latency of all given partitioned CNN models in spreadsheet format to the system designer.

A. Sensor Node

Hardware-aware evaluation of CNNs is particularly important in the scope of embedded systems, which are constrained by both energy consumption and latency. Hence, these nodes have to offer specialized hardware accelerators for inference. There are many open-source hardware accelerators available [11]–[14], each having its own advantages and drawbacks in terms of power, performance and area consumption. Thus, determining an optimal design for a given application is not trivial. Enabling rapid simulation of low power and low latency Application-Specific Integrated Circuit (ASIC) designs, consequently, requires a fast and flexible framework that allows evaluation of various CNN hardware accelerators. Frameworks such as FLECSim [15] provide cycle-accurate simulation and evaluation of a broad range of accelerators. However, analyzing each cycle leads to enormous simulation time and is therefore not applicable for our proposed simulation toolchain. In contrast, higher-level simulation frameworks for evaluating CNN hardware accelerators [16]–[18] offer significantly reduced simulation time but less accuracy of evaluation metrics. Moreover, most of these tools are tied to just one hardware accelerator architecture, e.g. systolic arrays, and only allow to adjust few architectural parameters neglecting optimization of processing elements or the memory hierarchy.

To overcome the aforementioned drawbacks, we use Timeloop [19] as tool to evaluate most important design metrics of the sensor node. It takes an analytical model of the hardware accelerator and searches for an optimal inference mapping of a single DNN layer. Thereby, Timeloop offers different search algorithms and optimizations metrics, e.g. energy and delay, as well as tuneable exploration termination conditions. Consequently, this approach enables fair evaluation of various designs and applications in a reasonable time span.

Usually, CONV layers contribute by far the majority of the computational operations and weights to most CNNs [20]. Therefore, the evaluation of the embedded platform in the sensor node focuses on CONV layers to estimate energy consumption and latency as well as the required bandwidth on the link between sensor and edge node. This methodology hence offers a reduced simulation time, which is very important in the context of hardware/software co-design. The architecture description also features hardware constraints, the configuration of Timeloop's internal mapper and the layer input shape. Latter may be different for each CONV layer of the CNN. Therefore, the simulation toolchain has to extract every CONV layer and the corresponding input shape first, before launching Timeloop. Based on this information, Timeloop evaluates the amount of cycles required by the hardware accelerator for single CONV layer inference. Even though, it outputs detailed statistics about each individual module instantiated in the accelerator, such as memories, buffers and processing elements, additional tools need to be included to account for their energy consumption.

Therefore, Timeloop inherently offers an interface to Accelergy [21], an open-source tool for analyzing energy consumption based on a hardware accelerator description and action counts of each module. Accelergy uses primitive component libraries provided by estimation plug-ins to derive the overall energy consumption. In the proposed toolchain, we use CACTI [22] for modelling memory components and Aladdin [23] to evaluate energy consumption of accelerator structures. As a result, Accelergy provides reliable estimations regarding the energy consumption per layer.

B. Link

Since the partition interconnection link is meant to transport information that is critical to the modelled CNN, a separate link model is provided by the toolchain to estimate its impact on overall network performance. Currently, a copper-based Ethernet link model has been implemented with two evaluation parameters: transmission latency and interface power consumption. Nevertheless, more link models can be added easily, e.g. wireless Ethernet, Bluetooth, etc.

Since our simulation toolchain targets embedded platforms, the link model assumes a common embedded scheme with a microprocessor system directly connected to the Physical layer (PHY) device. With this connection scheme, overall power consumption of the interface is predominantly defined by energy demands of the PHY device. For this reason, emphasis has been given to modelling the properties of these particular devices.

The Ethernet model is based on the BASE100-T, BASE1000-T and 10GBASE-T. In its baseline "legacy" configuration, power consumption is evaluated based on typical values for PHY devices [24].

Aside from power consumption, transmission latency T_{link} of the link is evaluated. It primarily depends on data size N_{data} and line rate B and is described by equation (1) [25].

$$T_{link} = \frac{N_{hdr} + N_{stf} + max\{N_{data}, N_{min}\}}{B} + T_{wire} \qquad (1)$$

$N_{hdr} = 20$, $N_{stf} = 18$ and $N_{min} = 46$ are sizes of the header, stuffing section and minimum payload size as per Ethernet standard. Wire propagation delay T_{wire}, on the other hand, must be computed with a particular use case in mind. Considering signal propagation delay in a copper twisted pair to be $\approx 6\ ns/m$, the cable delay is described as $T_{wire} = l_{line} \cdot 6 \cdot 10^{-9}\ s$, where l_{line} is the length of the cable in meters.

C. Edge Node

In contrast to the sensor node and link, the performance of the edge node does not need to be simulated with a high-level tool. Instead, Commercial-Off-the-Shelf (COTS) compute platforms for real-time evaluation can be used, providing realistic data for exploring CNN partitioning. For evaluation of the CNN latency, our toolchain uses the PyTorch Benchmark module, which allows to run multiple threads while still ensuring reliable performance metrics by adding a warm-up run and CUDA synchronization. To achieve a reliable inference latency estimation, our toolchain determines the median value of 1000 measurements performed on the GPU. Since Python is available on many platforms, this approach is applicable for a broad range of embedded systems. Depending on the workload, designers can incorporate either high-performance embedded computing platforms such as the NVIDIA Jetson TX-2 or embedded CPU platforms, e.g. Raspberry Pi.

Apart from latency, power consumption of the edge node must also be taken into account in the evaluation. Since COTS platforms usually implement general-purpose components, however, measuring the power consumption of CNN inference would only provide rough estimates. Consequently, the overhead introduced by the operating system and peripherals has to be considered for evaluation as well. Since this depends on the application, we assume a constant power consumption in our evaluation.

IV. Evaluation

In the following, we show the results of CNN partitioning for an exemplary workload that will be specified in the subsequent subsection. The system used for executing the sensor node and link evaluation is running CentOS on an AMD EPYC 7702P using Python, PyTorch, Timeloop and Accelergy including its plug-ins, i.e. CACTI and Aladdin. Timeloop is configured to run eight threads in parallel to find a delay- and energy-optimal mapping of each layer of the CNN for the accelerator architecture. Thereby, it uses the linear-pruned search algorithm, a more efficient version of linear search due to its ability to prune irrelevant permutations of unit-factors. The search is terminated by the victory condition which we set to 100 valid but suboptimal mappings. This configuration provides a good trade-off between quality of the results and the simulation time.

The computationally powerful edge node is represented by an NVIDIA Jetson TX-2 platform. For machine learning tasks, it is equipped with a GPU that offers 256 Pascal cores, running at 1.3 GHz. In combination with an ARM-based Quadcore CPU, the Jetson hardware has a high performance to power ratio, as it runs at 15 W in the MAX-P configuration for maximum performance. From a software perspective, the node is operating on unmodified 64-bit Jetson Linux. The measurement of the partitioned neural network workloads is performed using PyTorch accelerated by CUDA libraries and the aforementioned PyTorch Benchmark module, taking the median latency out of 1000 consecutive runs.

A. Workloads

As exemplary workloads, we select three network topologies: Fully Convolutional Network (FCN) ResNet-50 realizing

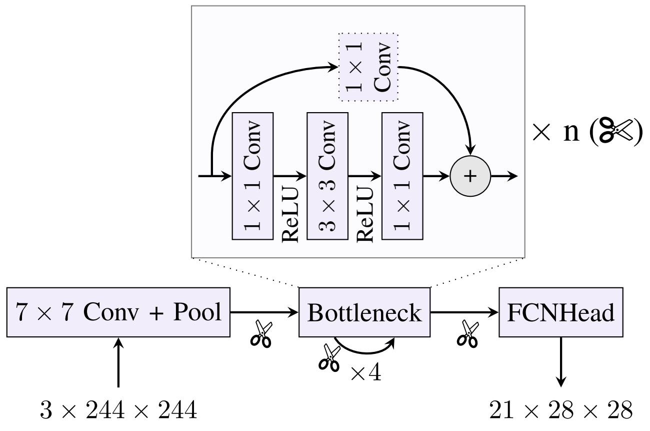

Fig. 3: Exemplary overview of the FCN ResNet structure with the partitioning points highlighted by scissors for distribution of the network among different platforms

semantic segmentation in e.g. autonomous driving, as well as GoogLeNet [26] and SqueezeNet V1.1 [27] both performing image classification tasks. Partitioning of neural networks is not advisable for each individual layer or layer type, respectively. State-of-the-art networks often use *skip connections* to prevent vanishing gradients during training. Skip connections are usually grouped into building blocks. Partitioning the network within these building blocks would introduce a larger amount of data to transfer and a greater memory footprint. For this reason, we choose to introduce partitioning points at which a separation of the neural network is favorable. These are usually placed in between the aforementioned building blocks and in layers without skip connections. In addition, we added an *Identity* layer in the beginning of each of our selected topologies to evaluate an entire execution on the remote node as well. For a deeper understanding of the partitioning points, Fig. 3 shows a practical example of an FCN ResNet-50 in more detail, with small scissor icons highlighting the favorable partitioning points.

ResNets are available in different configurations and thus varying complexity levels [28]. For FCN ResNet, usually ResNet-50 and ResNet-101 configurations are used, reaching 91.4% or 91.7% pixel accuracy, respectively. In the topology level, the network consists of a configuration independent head and tail, as well as four *Bottleneck* blocks. Each *Bottleneck* block has n residual blocks (upper part of Fig. 3), which can be partitioned individually but not in itself. The first residual block has a downsampling feature, which is realized by an additional 1×1 convolution in the branch. The amount of residual blocks in each *Bottleneck* is determined by the configuration. ResNet-50, for instance, has 3, 4, 6 and 4 residual blocks in each *Bottleneck*. The network head consists of a large convolution and max pool to reduce the dimensions, as well as a batch normalization layer. Whereas the tail has two convolutions with batch normalization to build the segmented output. Our network is trained for 21 segmentation classes. Head and tail can be partitioned for each individual layer, resulting in 25 partitioning points for FCN ResNet-50.

TABLE I: Median simulation time to evaluate system modules for different CNN architectures out of 50 runs. The edge node evaluation includes 1000 measurements for each element in the given set of partitioned CNN models. Simulation time of the link is very short and is therefore neglected.

Architecture	**Partitioning points**	**Sensor Node** (sim. time)	**Edge Node** (sim. time)
FCN ResNet50	25	610 s	3320 s
GoogLeNet	19	410 s	395 s
SqueezeNet V1.1	17	167 s	54 s

Partitioning points for GoogLeNet and SqueezeNet V1.1 are defined in a similar fashion. GoogLeNet consists of multiple *Inception* modules that offer different paths between the layers, with different convolution filter sizes. This approach thereby allows for minimal weight sparsity. In addition to the *Inception* modules, the networks consist of simple convolutions, pooling and dense layers. Inserting a partitioning point after each of these modules, sums up to 19 partitioning points in GoogleNet.

SqueezeNet's topology aims for a very low amount of parameters to reduce the memory footprint. Similar to *Inception* modules of GoogLeNet, it therefore incorporates *Fire* modules which perform an efficient feature extraction with concurrent 1x1 and 3x3 convolutions. SqueezeNet also stacks multiple *Fire* modules with pooling operations upon each other, with a dense layer for classification at the end. In total, SqueezeNet V1.1 provides 17 partitioning points.

B. Simulation Time

In general, the main purpose of simulation is accelerating the design process by providing good estimates in terms of latency and energy for hardware/software co-design. Therefore, in the case of a large design space, fast simulation enables broad exploration to determine an optimal system architecture. Our simulation toolchain only adds a marginal simulation time overhead to the integrated tools for CNN analysis and partitioning between sensor and edge node.

In contrast, the estimates for inference of CNNs on a non-parameterizable off-the-shelf accelerator, such as the NVIDIA Jetson TX-2 GPU platform, can be directly generated and need no modeling by Timeloop and Accelergy. However, a certain reliability of the estimations has to be ensured. We therefore ran each simulation 50 times in total and determined the median value for the corresponding metric. In order to allow fair comparison between different architectures, the toolchain configuration is the same for all applications, including the accelerator analyzed using Timeloop. Thereby, our sensor node is based on a Simba-like accelerator architecture [13].

Since the evaluation of the link is based on a simple mathematical model, the computation takes less than a millisecond and can therefore be neglected. The results are shown in Table I. FCN ResNet50 takes the longest time for evaluating the set of partitioning points (3320 s), which is obvious due to being the largest network examined in the scope of our work. In contrast to GoogLeNet and SqueezeNet V1.1, Timeloop finds an optimal mapping for each CONV layer significantly faster than the Jetson TX-2 can perform 1000 CNN inferences for each partitioned model in total.

C. Experimental Results

For our evaluation, we assume a common embedded system which can be found in automotive or robotics. It consists of a centralized off-the-shelf GPU deployed in a high power domain and specialized sensor node accelerators that are integrated in a low power domain. Both platforms are connected via a five-meter Gigabit Ethernet cable. The low power domain for the sensor nodes leads to a simpler and far less error-prone on-board network. This leaves us with several possible optimization goals. Since we assume constant power consumption in the edge node, we only have to consider the sensor node and link for evaluating energy consumption. Due to the size of the CNNs, the accelerator deployed on the sensor node to evaluate FCN ResNet-50 and GoogLeNet is a Simba-like architecture [13] clocked at 500 MHz. In contrast, SqueezeNet can be inferred on a smaller, less energy consuming accelerator. Consequently, we choose an Eyeriss-like architecture [11] clocked at 200 MHz to obtain the corresponding evaluation metrics that can be found below.

1) FCN ResNet-50: The estimated energy of the sensor node and the communication link is shown in Fig. 4a. As can be seen, if a single CONV layer is followed by a Batch Normalization (*BatchNorm2d*) and *ReLU* layer, e.g. in case of *Conv2d:2-1*, partitioning between these layers makes no difference for the data amount to be transferred over the link.

When choosing an early partitioning point, the energy consumed in the sensor node is very low, and due to the large output feature maps, the communication link dominates the energy demand. In the following layers, the output feature maps are getting smaller, leading to fewer data on the communication link, with local minimums at partitioning point *Maxpool2d: 2-4* and partitioning point *Bottleneck: 3-7*. These mark our first points of interest. In case communication bottlenecks on the link are identified and energy demand on the sensor node has to be kept low, we favor one of these solutions.

Our estimated latency in Fig. 4b shows that only minor differences can be noticed between execution of the CNN mainly on the specialized sensor node accelerator compared to execution mainly on the off-the-shelf GPU. Therefore, the design decision has to be made with respect to energy consumption, e.g., regarding the maximum amount of energy that should be consumed on the sensor node.

Executing the majority of the DNN on the sensor node contradicts the optimization goal of low energy consumption on the sensor node. Obviously, the maximum of energy is consumed, when the backbone network is entirely executed on the sensor node. Therefore, we define partitioning point *Bottleneck: 3-13* as our third point of interest, due to the steep rise of sensor energy demand when we select later partitioning

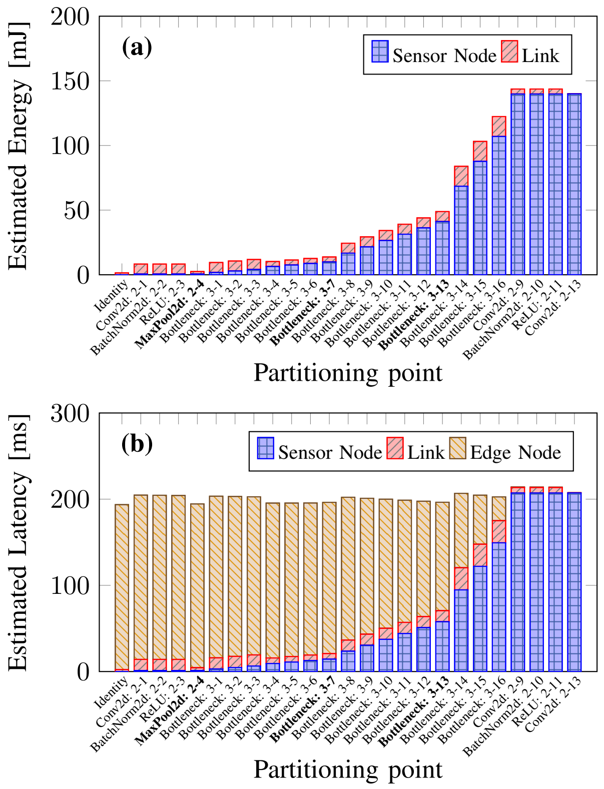

Fig. 4: FCN ResNet-50 evaluation results of each reasonable partitioning using a Simba-like architecture clocked at 500 MHz in the sensor node. Since we assume static power consumption in edge node, energy estimation is only shown for sensor node and link. The partitioning points *MaxPool2d: 2-4* and *Bottleneck: 3-7* are favorable in terms of energy consumption in the sensor node. *Bottleneck: 3-13* marks a point of interest in terms of optimal latency balancing.

points. This point fulfills a good trade-off between latency and energy consumption on the sensor node and should be chosen when a well-balanced system is preferred.

2) GoogLeNet: In contrast to the FCN ResNet-50, our evaluation of GoogLeNet, depicted in Fig. 5, clearly shows two anomalies which highlight the relevance of our proposed simulation toolchain. When partitioning at *BasicConv2d: 1-1* or *BasicConv2d: 1-4*, the necessary amount of data to be transferred occupies the link significantly, which leads to peaks in both energy and latency. As the output feature maps of the layers get smaller, the energy consumption of the link reduces, which makes partitioning point *Inception: 1-9* and the succeeding layers interesting regarding our optimization goal. Additionally, *Inception: 1-6* provides a good trade-off for both small link and sensor node energy consumption, marking another point of interest.

Our estimated latency in Fig. 5b clearly shows that executing the CNN entirely on the specialized sensor node accelerator is preferred over execution on the off-the-shelf GPU. A global minimum can be observed for the partitioning point

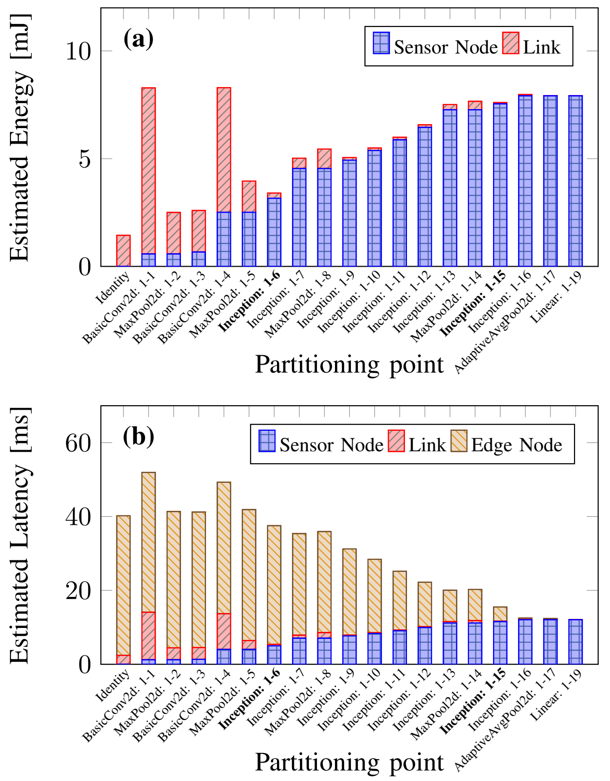

Fig. 5: GoogLeNet evaluation results of each reasonable partitioning using a Simba-like architecture clocked at 500 MHz in the sensor node. Large output feature maps of *BasicConv2d: 1-1* and *BasicConv2d: 1-4* lead to peaks in both link energy consumption and latency. If minimal latency is required, the network has to be executed entirely on sensor node. Additionally, partitioning point *Inception: 1-6* provides a good trade-off in terms of energy consumption, latency and link usage.

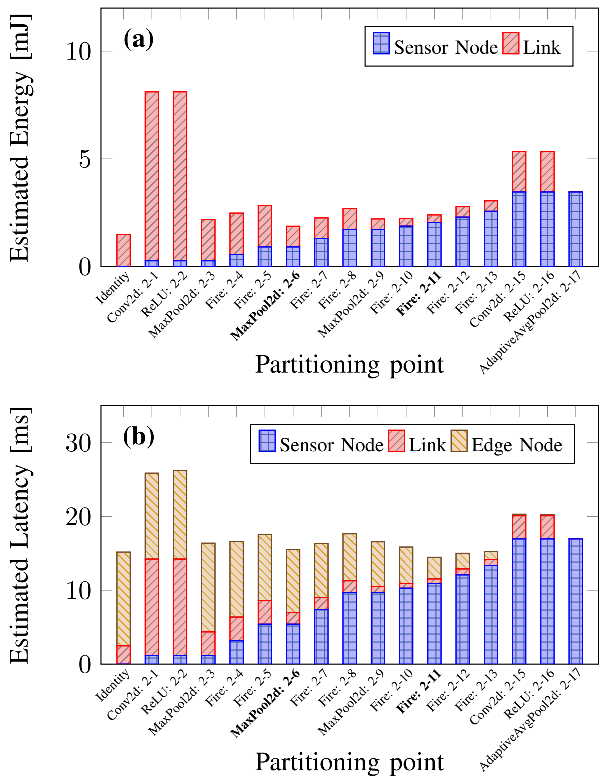

Fig. 6: SqueezeNet V1.1 evaluation results of each reasonable partitioning using an Eyeriss-like architecture clocked at 200 MHz in the sensor node. Large output feature maps of *Conv2d: 1-2* and *ReLU: 2-2* lead to peaks in both link energy consumption and latency. Partitioning point *Fire: 2-11* offers minimal latency by also providing very low energy consumption. A good trade-off between latency and energy consumption can be achieved by considering partitioning point *MaxPool2D: 2-6*.

Inception: 1-15 marking our second point of interest. When a minimum latency is preferred, this point is advantageous. Since the results here move in opposite linear directions, most of the partitioning points are Pareto-like. Therefore, it must be decided whether lower latency or lower energy consumption is more important for the specific use case.

3) SqueezeNet V1.1: Similar to GoogLeNet, the generated results of the SqueezeNet, depicted in Fig. 6, reveal highly disadvantageous partitioning points. Partitioning after the first or last CONV layer (or *ReLU* layer), the large amount of output feature data to be transferred over the link leads to peaks in both energy consumption and latency. When the energy consumed by the link is to be minimized, the partitioning point *Fire: 2-11* marks a point of interest, leaving out sending only CNN results after the last layer. Looking at the latency plot, this point also marks the global minimum, providing a very good trade-off. Another partitioning point worth mentioning is *MaxPool2d: 2-6* having the lowest energy consumption for sensor node and link combined, leaving out the first layer (*Identity*). Since CNN inference solely on the edge node is not favorable in multi-sensor systems due to high uncertainties introduced regarding latency, this partitioning point should be considered offering a good trade-off between latency and energy consumption.

V. Conclusion

The deployment of AI in embedded systems faces constraints such as latency restrictions and limited energy consumption which have to be considered during hardware/software co-design of neural networks. In this paper, we presented a simulation toolchain for fast evaluation of hardware-aware CNN partitioning for embedded AI applications. Since it is based on open-source tools, it offers straightforward integration into existing toolflows. Our approach allows exploring offloading CNN inference while explicitly taking the hardware integrated in the sensor node into account. As a result, our proposed simulation toolchain can be used to determine an optimal partitioning point with respect to the application and its requirements.

The evaluation results presented thereby proved the effectiveness of the simulation toolchain to find multiple points of interest for three, commonly used CNNs. Based on the output metrics provided, we were able to identify several optimal partitioning points for FCN ResNet-50, GoogLeNet, and SqueezeNet V1.1.

In particular, the evaluation results showed interesting insights into CNN partitioning for distributed systems. These findings could be beneficially taken into account during Neural Architecture Search (NAS) for embedded AI applications. Hence, future work will include developing and evaluating an automated design space exploration framework for hardware-aware CNN partitioning to support NAS and integration of other DNN models enabling the simulation toolchain to evaluate a broader range of applications.

Acknowledgment

SPONSORED BY THE
Federal Ministry of Education and Research

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0096 (ZuSE-KI-mobil). The responsibility for the content of this publication lies with the authors.

References

[1] N. Fasfous, M.-R. Vemparala, A. Frickenstein, M. Badawy, F. Hundhausen, J. Höfer, N.-S. Nagaraja, C. Unger, H.-J. Vögel, J. Becker, T. Asfour, and W. Stechele, "Binary-lorax: Low-latency runtime adaptable xnor classifier for semi-autonomous grasping with prosthetic hands," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 13 430–13 437.

[2] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," *CoRR*, vol. abs/1602.01528, 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[3] I. Walter, J. Ney, T. Hotfilter, V. Rybalkin, J. Hoefer, N. Wehn, and J. Becker, "Embedded face recognition for personalized services in the assistive robotics," in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Cham: Springer International Publishing, 2021, pp. 339–350.

[4] T. Hotfilter, F. Kempf, J. Becker, D. Reinhardt, and I. Baili, "Embedded image processing the european way: A new platform for the future automotive market," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 2020, pp. 1–6.

[5] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809919306356

[6] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 255–266, jun 2016. [Online]. Available: https://doi.org/10.1145/3007787.3001164

[7] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018, pp. 1–6.

[8] D. Hu and B. Krishnamachari, "Fast and accurate streaming cnn inference via communication compression on the edge," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020, pp. 157–163.

[9] 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.

[10] L. Yang, W. Jiang, W. Liu, E. H. M. Sha, Y. Shi, and J. Hu, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 85–90.

[11] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[12] NVIDIA Corporation. (2018, Nov.) The nvidia deep learning accelerator. [Online]. Available: http://nvdla.org/

[13] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302

[14] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," *arXiv preprint arXiv:1911.09925*, vol. 3, 2019.

[15] T. Hotfilter, J. Hoefer, F. Kreß, F. Kempf, and J. Becker, "Flecsimsoc: A flexible end-to-end co-design simulation framework for system on chips," in *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, 2021, pp. 83–88.

[16] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: Automl codesign of a cnn and its hardware accelerator," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[17] F. Muñoz-Martínez, J. L. Abellán, M. Acacio, and T. Krishna, "Stonne: A detailed architectural simulator for flexible neural network accelerators," *ArXiv*, vol. abs/2006.07137, 2020.

[18] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[19] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[20] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network accelerator," 2018.

[21] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[22] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 694–701.

[23] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 97–108.

[24] P. Reviriego, J. Maestro, J. Hernández, and D. Larrabeiti, "Study of the potential energy savings in ethernet by combining energy efficient ethernet and adaptive link rate," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 3, pp. 227–233, 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.1526

[25] J. Gámiz-Caro and A. Grau, "Message delay in distributed control systems through ethernet," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 36–41, 2005, 16th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016371737

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv:1409.4842 [cs]*, Sep 2014, arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842

[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *arXiv:1602.07360 [cs]*, Nov 2016, arXiv: 1602.07360. [Online]. Available: http://arxiv.org/abs/1602.07360

[28] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *arXiv:1605.06211 [cs]*, May 2016, arXiv: 1605.06211. [Online]. Available: http://arxiv.org/abs/1605.06211