Design Space Exploration for CNN Offloading to FPGAs at the Edge

Guilherme Korol*, Michael Guilherme Jordan*, Mateus Beck Rutzig†,
Jeronimo Castrillon‡§, Antonio Carlos Schneider Beck*
*Institute of Informatics, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
†Electronics and Computing Department, Universidade Federal de Santa Maria (UFSM), Santa Maria, Brazil
‡ Center for Advancing Electronics Dresden, TU Dresden, Dresden, Germany
§ Center for Scalable Data Analytics and Artificial Intelligence, Dresden, Germany
*{gskorol,mgjordan,caco}@inf.ufrgs.br,†mateus@inf.ufsm.br,‡jeronimo.castrillon@tu-dresden.de

***Abstract*—AI-based IoT applications relying on heavy-load deep learning algorithms like CNNs challenge IoT devices that are restricted in energy or processing capabilities. Edge computing offers an alternative by allowing the data to get offloaded to so-called edge servers with hardware more powerful than IoT devices and physically closer than the cloud. However, the increasing complexity of data and algorithms and diverse conditions make even powerful devices, such as those equipped with FPGAs, insufficient to cope with the current demands. In this case, optimizations in the algorithms, like pruning and early-exit, are mandatory to reduce the CNNs computational burden and speed up inference processing. With that in mind, we propose ExpOL, which combines the pruning and early-exit CNN optimizations in a system-level FPGA-based IoT-Edge design space exploration. Based on a user-defined multi-target optimization, ExpOL delivers designs tailored to specific application environments and user needs. When evaluated against state-of-the-art FPGA-based accelerators (either local or offloaded), designs produced by ExpOL are more power-efficient (by up to 2×) and process inferences at higher user quality of experience (by up to 12.5%).**

***Keywords*—Edge Computing, IoT, Offloading, CNN, FPGA.**

I. Introduction

AI-powered applications are one of the main drivers behind the Internet of Things (IoT). Deep Neural Networks, especially Convolutional Neural Networks (CNNs), have successfully delivered high-quality results for applications ranging from language to video processing. However, deploying such computationally expensive algorithms to IoT devices is a challenge. Usually constrained in power or processing capabilities, IoT devices are not guaranteed to scale up to the demand created by modern AI applications. Moreover, optimizing for metrics like power efficiency becomes a must to deal with modern CNNs, requiring over a billion multiply-accumulate (MAC) operations for a single input (e.g., 15.5G MACs in the VGG-16 model [1]).

In this context, edge computing is a system paradigm that proposes a multi-layered architecture, where the IoT devices can either process locally or offload their raw data over the network to more capable edge servers (sometimes called IoT gateways). These edge servers are placed physically close to the IoT devices, avoiding the long latency of the cloud and increasing security with less data exposure. To accelerate the CNN processing at reasonable power levels both IoT devices (when processing locally) and edge servers can make use of dedicated accelerators. In such scenario, FPGA platforms are one of the most popular alternatives, due to their performance, energy costs, and reconfigurability [2–4].

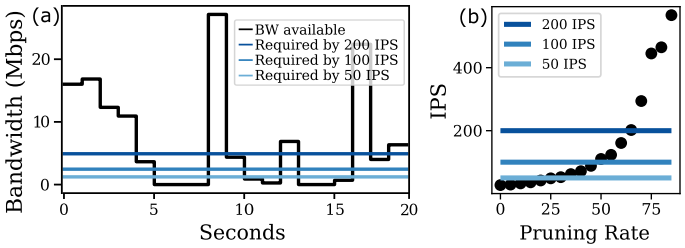

Figure 1. (a) Available 4G/LTE bandwidth (BW) and the BW required to offload 50, 100, and 200 IPS. (b) IPS for the CNVW2A2 CNN from 0 to 85% pruning running on the FINN [11] FPGA accelerator (CIFAR-10 dataset).

On top of that, IoT-Edge scenarios are highly heterogeneous, with multiple concurrent applications, variable workloads, and diverse environments. For instance, the user may experience a highly volatile bandwidth due to factors like signal strength fluctuation and changes in network load [5]. Figure 1(a) shows the available bandwidth (black curve) recorded on a 4G/LTE client [6] over 20 seconds (x-axis). The figure also shows three horizontal lines giving the minimum bandwidth required to offload 50, 100, or 200 inferences per second (IPS) to an edge server. From this example, it is clear that designing a system in charge of processing CNN inferences should take into account the expected application environment (e.g., bandwidth availability and workload). However, state-of-the-art has shown that relying on hardware improvements alone will not satisfy the efficiency levels demanded by modern CNNs in such environments, respecting the given power and energy constraints [7–10]. Therefore, the CNNs lying on top of the hardware layer must be optimized as well.

At the CNN algorithmic level, state-of-the-art optimizations aim at reducing the computational load in exchange for some controlled losses in accuracy. Early-exit [12] and pruning [13] are two popular techniques enabling the accuracy-resource trade-off. Early-exit exploits the fact that some inputs are easier to process than others (e.g., a cat in a clean background picture is easier to classify than one hiding in the bush). In an early-exit CNN, these easy inputs are output at earlier layers, producing faster inferences. Pruning, on the other hand, works by removing redundant parts of a CNN, from particular neurons to whole layers. By making the CNN smaller, pruning saves MAC operations and the storage required to run a CNN inference.

As an example, Figure 1(b) shows the throughput (in terms

of IPS, y-axis) for a CNN from 0 to 85% pruning (x-axis) running on an FPGA (early-exit was not depicted in the figure for the sake of simplicity). Figure 1(b) also shows the same three sample workloads of 50, 100, and 200 IPS as horizontal lines. Naturally, changing CNN optimization parameters like pruning to, for example, deliver 50, 100, or 200 IPS, will impact the quality of the delivered inferences (i.e., accuracy) as well as the power dissipation. Therefore, one can match the inference processing to the working environments of each application considering the full IoT-Edge spectrum (e.g., high workload tasks, requiring maximum performance, or scenarios constrained by energy or bandwidth).

To support the design of inference processing systems with such tight constraints and diverging goals, we propose **ExpOL**. ExpOL is a framework for Exploring the design space of Offloaded and Local FPGA-based inference processing for the IoT-Edge. Based on user-defined multi-target goals, ExpOL delivers FPGA designs at the *pareto front* for processing inferences on pruned early-exit CNNs running either locally, at the IoT device, or at the edge server.

Concretely, this work makes the following contributions:

- Presents a novel design space combining offloading and the CNN optimizations of pruning and early-exit to explore the accuracy-performance-power trade-off;
- Proposes ExpOL that leverages this design space to deploy optimized solutions to accelerate inference processing on FPGAs;
- Under an IoT-Edge application, ExpOL improves by up to 2× the power efficiency and 12.5% the user experience over locally or offloaded state-of-the-art FPGA-based inferences.

II. Background and Related Work

Two approaches are commonly used to handle the computational load of deep learning. One approach involves new system-level paradigms like the edge that moves data to be processed elsewhere and architectures that accelerate inference exploiting deep learning properties, such as their heavy dependence on matrix multiplications. The other approach is to optimize the algorithms by refining or creating techniques to trade off accuracy per computation. Next, we detail both approaches.

A. Offloading CNNs - Edge Computing

Edge computing allows offloading the inference (e.g., by sending images, audio samples, etc.) over the network from resource-limited devices to servers equipped with high-performance architectures. Examples of such systems include DjiNN [14] and Clipper [15]. DjiNN uses a multi-GPU system for scalability and low latency across multiple applications and DNN models. Clipper, on the other hand, employs a model selection mechanism to fuse the output of parallel CNNs based on the application feedback.

FPGA-based accelerators have been proposed as alternatives to energy-hungry GPU boards since they can increase power efficiency at small or no drops in accuracy [11, 16, 17]. Scylla [3] employs an FPGA for serving CNN inferences at the Edge. It exploits the reconfigurability capabilities of FPGAs for Quality of Experience (QoE) optimization. In [4], the authors propose a policy for allocating and scheduling multiple accelerators on a Xilinx ZCU104 board for particular workload levels, latency requirements, and available resources.

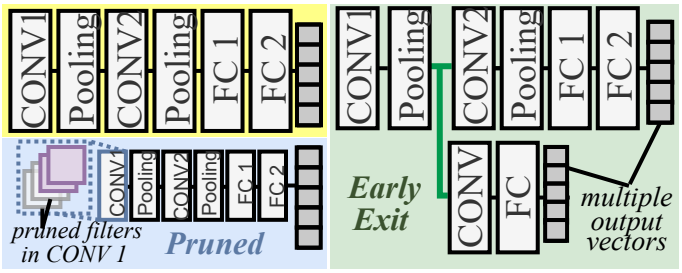

Figure 2. A sample CNN (yellow) with its pruned and early-exit versions.

B. Optimizing CNNs - Pruning and Early-Exit

A CNN is a Deep Neural Network (DNN) like the one pictured over the yellow background in Figure 2. The goal of a CNN is to read an input and predict from within a finite set of problem-defined classes which one correctly describes that image. To do so, the first step is the definition of the CNN topology (i.e., the CNN model), which specifies the number and type of layers and their parameters. As in the example from Fig. 2, the Convolutional (CONV), Fully Connected (FC), and pooling are the three main CNN building blocks. CONV and FC layers process inputs with their weights and biases, producing multi-dimensional matrices called feature maps. During training, these weights are updated through iterations until a certain quality threshold is reached. Once training is complete, the model can predict new inputs (that were not seen during training). This phase is called inference and is the focus of our work.

At inference, the feature map received by a CONV layer is convolved with weights (organized as 3D filters). These filters act as "feature extractors" as they slide over the input feature map. The features or patterns will support the inference in following CONV and FC layers. FC layers can be viewed as matrix-vector multiplications between their input and weights. They are usually used for flattening the output of CONV layers and calculate the probabilities of each class (output vector).

CONV and FC layers require extreme amounts of computation and memory transfers. Compression methods, like pruning, have proven to be effective in reducing such requirements while only incurring small accuracy costs. Pruning reduces the memory and computation required by a CNN. This work focuses on filter pruning [13] (see a pruned CNN in Figure 2 with removed filters). The percentage of CONV filters to be removed on each layer is defined as the *Pruning Rate*. Also note that removing filters from a CONV layer also reduces its number of output feature map channels (each filter generates one output channel), granting a roughly quadratic effect on reducing the CNN memory footprint and its respective computations.

Either on GPUs [18, 19] or on FPGAs [20, 21], pruning has been used to improve inference processing. ReForm [18] provides a resource-aware mechanism reconfiguring a CNN according to the device's resources. DMS [19], on the other hand, prunes CNN filters for Quality-of-Service (QoS) optimization. Targeting FPGAs, authors in [21] propose a framework that adapts the inference processing by switching the pruned model

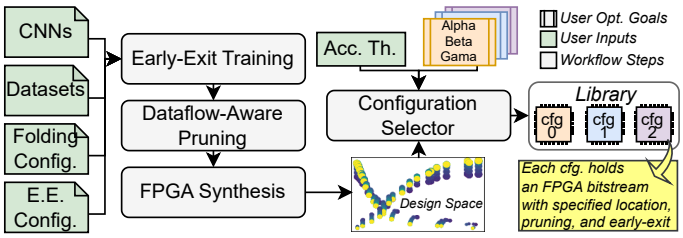

Figure 3. ExpOL's workflow.

at runtime. In [20], a toolflow that statically customizes the CNN pruning to the underlying FPGA accelerator is proposed.

While pruning is typically applied statically, early-exit [12] is a dynamic optimization that exploits the ease of processing certain inputs. Early-exit allows the CNN to finish processing earlier (i.e., in a layer before the last) on exits connected to the original CNN layers (called backbone layers) - see an early-exit CNN with two exits in Figure 2 (with the branch from backbone to early exit highlighted). During inference, the early-exit CNN must decide whether to take early exits or not based on the confidence that the input has been correctly classified with the layers processed so far. The *Confidence Threshold* is used to make this decision, and when an exit outputs a confidence above this threshold, the inference is completed. Lowering the confidence threshold allows more inputs to be classified earlier, relaxing the expected confidence. The softmax function of the exit output vector is a popular method for measuring confidence. It can be calculated from the exits output as $\sigma(y)_i = e^{y_i} / \sum_{j=1}^{K} e^{y_j}$ for the output vector y of K classes.

Early exits have been used by many works, such as FlexDNN [7] and Hapi [8], that design and deploy early-exit models on embedded GPUs focusing mainly on performance improvements. SPINN [22] proposed using early-exit to help offload inferences from devices with limited processing capabilities. In SPINN, a not-taken early exit can be followed by the offloading of that feature map that would otherwise continue onto the backbone layers. To coordinate exits and offloading, SPINN uses a scheduler that tunes early-exit and offloading policies. For FPGAs, [9] reconfigures the FPGA at each not-taken exit to load the next set of layers. In [10], early exits are placed in ResNets targeting the accuracy-computational cost trade-off.

Wrap-up and Our Contributions. Some works optimize the CNNs with pruning only [18–21], early-exit only [7–10], focus on the system-level offloading only [3, 4, 14, 15], or even leverage the early exits as feedback to tune the offloading [22]. In this work, however, we propose ExpOL to combine all of those approaches into a single design space. And, in contrast to the state-of-the-art, ExpOL enables an automatic multi-target search so that it can adapt the inference processing to the highly diverse environments and applications of the IoT-Edge.

III. ExpOL

ExpOL is a fully automatic multi-target optimization tool for deploying FPGA-based CNN accelerators in IoT-Edge systems. ExpOL works before deployment to output a library of tuned design *configurations* generated based on a set of user goals. An ExpOL configuration specifies the location of deployment (IoT device or edge) with the corresponding FPGA bitstream and CNN model (with defined pruning and early-exit parameters). This section will take our use case to help with the explanation of ExpOL. The use case consists of an IoT device with a small footprint FPGA board, the PYNQZ1, while the edge server holds a more powerful board, the ZCU104, that can hold wider accelerators and larger CNN models. The offloading is considered over a wireless communication channel.

A. Workflow

Figure 3 presents the workflow. It starts by receiving the user inputs: original CNN models, training datasets, folding and early-exit (E.E.) configuration files, an accuracy threshold, and the user optimization goals. Optimization goals are specified as a tuple of α, β, and γ weights for combining the designs accuracy, performance, and power ("User Opt. Goals" in Figure 3). The inputs are, first, sent to the "Early-Exit Training" before they get pruned in the "Dataflow-Aware Pruning" step, creating multiple versions of the original CNNs. Those versions are then input to the "FPGA Synthesis" step to build the design space. Finally, the "Configuration Selection" searches for the best-scoring configurations according to the user optimization goals. Below, we detail these steps.

1) Early-Exit Training: The first step is the addition of early-exits to the user-supplied CNNs (defined in pytorch/Brevitas [23]). To that end, configuration files (*E.E. Config.* in Figure 3) set in ExpOL where and how the exits should be added. Exits can be any combination of convolutional, pooling, and fully-connected layers. These layers are appended as modules to the original CNN. After all exits have been added, the early-exit CNN can be trained. ExpOL follows the training proposed in [12] to train CNNs with multiple exits. In [12], all exits are trained simultaneously by combining their loss in a Joint Loss Function: $J_{loss} = \sum_{n=1}^{N} w_n L(\hat{y}_{exit_n}, y, \theta)$, where N is the number of exits, w_n the exit's weight, and L the traditional loss function accepting the exit's softmax $\hat{y}_{exit_n}$, ground-truth y, and the weights θ. Early-Exit steps are carried out as python scripts.

In our use case, we take the CNV, a VGG-like CNN, quantized and tuned for FPGA execution [11]. This model has six convolutional layers followed by three fully-connected ones. For the CNV, we have set two early-exit configurations: one for the IoT device that adds a single early exit after the second convolutional layer. The exit consists of another convolutional layer with the same parameterization (stride, kernel size, etc.) of the previous layer, a max-pool layer with a kernel size of $k = \lfloor \frac{DIM}{2} \rfloor$, where DIM is the dimension of the feature map, and two fully-connected layers with the same parameterization of the original fully-connected layers in the original model. For the edge server with a larger FPGA, we set two exits to be added: after the second and fourth convolutional layers. Both exits follow the same configuration described above.

2) Dataflow-Aware Pruning: Once the early-exit CNNs have been trained, they can be pruned. ExpOL does that by ranging the pruning rate at fixed intervals. At each pruning rate, ExpOL creates a CNN with a different accuracy-resource trade-off. To prune the CNNs that will later run on the FPGA, the "Dataflow-Aware Pruning" is used [21]. It consists of a filter pruning technique that, besides the CNN, considers the properties of the FPGA accelerator. In general lines, this pruning method

guarantees that the final number of filters (and, consequently, of convolutional channels) is compatible with the parallelization set on the accelerator. After pruning, the CNNs are retrained and can be exported as Open Neural Network Exchange (ONNX) files suitable for FPGA synthesis. Pruning is also performed as python scripts.

3) FPGA Synthesis: To synthesize these CNNs into FPGA accelerators, ExpOL uses the FINN framework from AMD/Xilinx [11]. FINN accelerators are called dataflow or streaming accelerators since they rely on a pipelined architecture. FINN compiles and synthesizes CNNs to hardware modules implemented as a set of High-Level Synthesis (HLS) parameterizable template classes. On top of FINN, ExpOL adds the ability to compile CNN models with early exits. When synthesizing an early-exit model, branches are added at the specified exit locations so the data (i.e., feature maps) can be split into two, feeding both backbone and early exit.

In FINN, the user can tune the parallelism of every HLS module through a JSON folding configuration file (*Folding Config.* in Figure 3). By increasing the parallelism in the folding configuration, it is possible to increase the accelerator throughput without changing the CNN models, trading FPGA resources (and power) per performance. In our case study, we have input two folding configurations: one to synthesize accelerators for the IoT FPGA and another file to set the parallelism on the larger, edge server, FPGA. The folding used for the IoT FPGA is $4\times$ narrower than the one for the edge server (given the lower parallelism configured across the HLS layers). As will be shown in Section V, this lower parallelization already requires almost all slices of the FPGA on the IoT Device. During synthesis, reports on power dissipation (from Vivado synthesis) and performance (from RTL simulations) are produced and stored to support the configuration search.

4) Configuration Selector: The Configuration Selector performs the last step in the framework (Figure 3). It will exhaustively navigate the design space looking for the best configuration. The search is based on a profit equation configured by the user with three weights for combining accuracy (α), throughput (β), and power (γ) as

$$\begin{aligned} \text{profit}_i = \alpha \cdot \text{config}^i_{\text{accuracy}} + \\ \beta \cdot \text{config}^i_{\text{throughput}} + \gamma \cdot (1 - \text{config}^i_{\text{power}}) \end{aligned} \quad (1)$$

where $\text{config}^i_{\text{throughput}}$, $\text{config}^i_{\text{power}}$, and $\text{config}^i_{\text{accuracy}}$ are the i-th configuration's throughput, power dissipation, and accuracy normalized w.r.t all configurations in the design space (min-max normalization).

The search starts by filtering out all design points with accuracy below the minimum specified by the user (*Acc. Th.* in Figure 3). After that, ExpOL evaluates the profit equation on all design points left. These design points are then sorted, so the Configuration Selector outputs the configuration of the highest profit. In case the user inputs more than one tuple, a library of optimal design points is generated by outputting one configuration for each tuple. These configurations represent the *pareto front* for the specified optimization goal and are ExpOL main product. It is the user responsibility to make the best use of the ExpOL library. For example, ExpOL can enable different "operating modes". Assuming a battery-powered IoT device, the ExpOL library can be used to switch to a low-power configuration (e.g., one generated for a tuple with a high γ value) when the battery level approaches a critical value.

Table I
OPTIMIZATIONS TUPLES AND THEIR GENERATED CONFIGURATIONS.

Opt. Tuple	Tuple Description	Name	ExpOL Configurations		
{0.5,0.5,0.0}	Acc. and Throughput	Cfg. 0	Edge	20% P.R.	60% C.T.
{0.0,1.0,0.0}	Throughput	Cfg. 1	Edge	20% P.R.	65% C.T.
{0.5,0.0,0.5}	Acc. and Power	Cfg. 2	IoT Device	15% PR.	50% C.T.
{0.0,0.0,1.0}	Power	Cfg. 3	IoT Device	20% P.R.	30% C.T.
{0.3,0.3,0.3}	All important	Cfg. 4	IoT Device	20% P.R.	45% C.T.

IV. METHODOLOGY

Accelerators used across our experiments were synthesized within the FINN design flow [11] with Vivado targeting an XCZU7EV FPGA for the edge server and an XC7Z020 for the IoT device, both at 100MHz. We used Xilinx Vivado for resource usage and power extraction and Verilator RTL simulations for performance. We adopted the CNV CNN from FINN with 2-bit quantization (CNVW2A2) on the CIFAR-10 dataset (3x32x32 images). ExpOL generates 18 models for the early-exit CNN with pruning rates from 0% (not-pruned) to 85% (5% steps). Each model generates a specific FINN accelerator. Each pruned CNN's confidence threshold vary from 0 to 100% at 5% steps. Accuracy results are reported on Brevitas TOP-1 test accuracy. The early-exit training procedure follows [12], weighting the first exit at 1.0 and the remaining at 0.3. Pruned early-exit CNNs are retrained for 40 epochs [13], with standard data augmentation and a learning rate of 0.001 with decay of 0.1. Training was performed on Intel Xeon E5-2640 with NVIDIA Tesla K20m GPU.

We base our evaluation on an IoT smart video application that can request inferences to an edge server or process it locally. Evaluations are 15 seconds long. The IoT device produces 30, 60, and 90 Inferences per Second (IPS) during the first 5, 10, and 15 seconds, respectively. The 4G/LTE Bandwidth Logs dataset [6] is used to model the communication channel. It provides the measured quality of 4G/LTE connections recorded along different routes in a city while downloading a large file over HTTP. Two traces were chosen for evaluation, representing two scenarios: a *stable* one recorded on a walking person and a *unstable* one recorded on a moving tram. For the evaluation, we have two baselines: **Original-IoT Device**, the original CNV running locally on the IoT FPGA; and, **Original-Edge** when all inferences are offloaded to the edge FPGA running the original CNV. The user can set any combination of search parameters in ExpOL tuples. However, here we illustrate its capability with five combinations presented in Table I (with descriptions and their generated configurations). The accuracy threshold to search configurations is set to 10%.

V. RESULTS

In this section, we start by analyzing the design space created with optimized inference processing at the local or the offloaded FPGA. We then present the configurations generated by ExpOL according to the optimization goals in Table I. Lastly, we compare ExpOL to the two baselines under the aforementioned application scenarios.

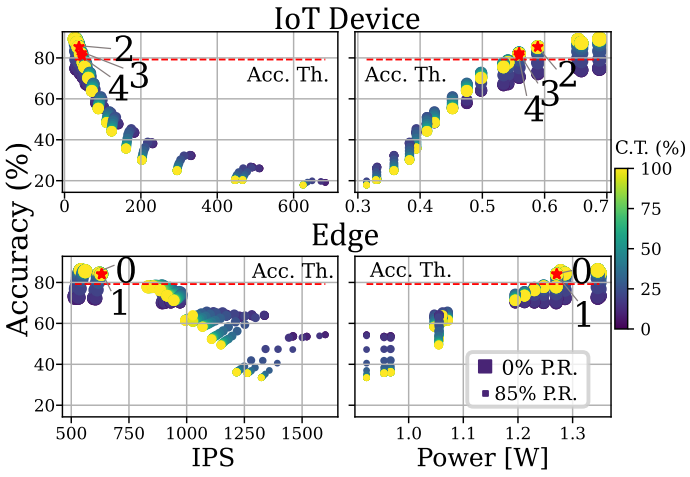

Figure 4. Inferences per Second or IPS (left plots) and Power (right plots) for the IoT device (upper plots) and edge (bottom plots) on the CIFAR-10 dataset with CNVW2A2 model (note the different x-axis ranges).

A. Design Space

Figure 4 presents the accuracy *versus* Inferences per Second (IPS) and *versus* Power for the FPGAs at the IoT device (a PYNQZ1 board, upper plots) and at the edge server (a ZCU104 board, bottom plots). The size and color of the design points represent their pruning rate and confidence threshold, respectively. We start by highlighting performance and efficiency differences between the two platforms. The larger edge server FGPA delivers the highest throughput levels, achieving more than 1500 IPS (at the highest pruning rate and the lowest confidence threshold). The IoT device, in turn, achieves a maximum of 683 IPS. However, the lower throughput delivered by the smaller IoT FPGA comes with a significantly lower power dissipation, and a higher power efficiency for most design points.

Table II shows the resource usage for the baselines and ExpOL generated configurations. For instance, when comparing two configurations with an equal pruning rate of 20% (configurations 0, running at the edge and 3 in the IoT), we see that the larger parallelization at the edge consumes almost 50% more LUTs and 55% more FFs than the same pruning rate on the IoT. Due to small FPGAs at IoT devices, accelerators quickly exhaust their resources. For example, the largest accelerator in the design space (not presented in Table II) occupies 96.07% of the available BRAMs and over 97% of the slices in the IoT FPGA. Such a design leaves no room for more parallelism. Also noticeable is the overhead of the early-exit layers in terms of resource usage. If we compare the Original-Edge to ExpOL configuration 0, we see an increase of LUT (34.86%) and FF (38.78%) utilization due to early-exits - despite configuration 0 being synthesized for a 20% pruned model. In summary, ExpOL creates a design space that ranges from fast and power-consuming to slower but more efficient inference processing. In this space, ExpOL can generate configurations covering a wide range of optimization targets.

B. ExpOL configurations

Figure 4 also shows the five configurations generated by ExpOL according to the optimization tuples from Table I. The horizontal line (*Acc. Th.*) gives the 10% accuracy threshold used throughout our evaluation. By setting the weights of the optimization tuple, the user can trade-off accuracy for performance and power. For instance, tuples that consider performance (e.g., tuples 0 and 1) get allocated at the edge since its larger FPGA can deliver higher throughput levels. In contrast, for tuples targeting power only (tuple 3) or a compromise between power and accuracy (tuple 4), ExpOL deploys solutions that processes inferences locally. From Figure 4, we note that the user could increase gains by allowing a lower accuracy threshold, as a lower line would enable more design points to be selected. In summary, by tuning the optimization parameters in ExpOL, the user can tune the design according to any goal, allowing it to match the inference processing characteristics to the application demands.

Table II
FPGA RESOURCE USAGE OF SELECTED CONFIGURATIONS AND BASELINES.

Configuration	FPGA	LUTs	FFs	BRAMs
Cfg. 0	Edge (XCZU7EV)	42898 (18.62%)	54167 (11.75%)	111 (35.58%)
Cfg. 1	Edge (XCZU7EV)	42898 (18.62%)	54167 (11.75%)	111 (35.58%)
Cfg. 2	Device (XC7Z020)	30135 (56.64%)	35990 (33.83%)	98 (70.36%)
Cfg. 3	Device (XC7Z020)	29106 (54.71%)	34842 (32.75%)	93 (66.79%)
Cfg. 4	Device (XC7Z020)	29106 (54.71%)	34842 (32.75%)	93 (66.79%)
Original-IoT Device	Device (XC7Z020)	26051 (48.97%)	30730 (28.88%)	140 (83.93%)
Original-Edge	Edge (XCZU7EV)	31807 (13.81%)	39029 (08.47%)	102 (32.69%)

C. Evaluation

We now evaluate the configurations generated by ExpOL under the two IoT-Edge application scenarios from Section IV. First, the *stable* scenario where the available bandwidth (49.2 Mbps on average) has fewer variations and stays at all times above the minimum required to offload the inferences (49.1 Mbps at the highest load). The configurations generated by ExpOL are compared to the baselines in Figure 5 on the Power Efficiency (inferences per Watt) and the total of Processed Inferences. Figure 5 also presents results on Quality of Experience[1] (QoE, right y-axis). Each configuration (ExpOL and baselines) in Figure 5 represents one whole execution under the same scenario. Power Efficiency and Processed Inferences (bars) are normalized w.r.t Original-IoT Device baseline.

In this scenario, we notice a significant performance difference between the edge (Original-Edge baseline and ExpOL configurations 0 and 1) and IoT device (Original-IoT Device and configurations 2-4). Since the network can accommodate all frames offloaded from the IoT device, it does not harm the final throughput and all inferences can be fed to the edge. Also, because of such high bandwidth and the accelerators deployed by the edge configurations having enough throughput, there is no difference in the number of processed inferences between Original-Edge baseline and ExpOL configurations 0 and 1 in Figure 5. The high performance also led to high QoE levels with a slight advantage of Original-Edge over configurations 0 and 1. This is due to the original CNN running on the baseline having accuracy higher than the ones deployed by ExpOL configurations 0 (4.64% accuracy loss) and 1 (4.7% loss). We note that in case of heavier workloads (e.g., from multiple IoT clients) we would see the Original-Edge performance (and QoE) decrease. For such heavier workloads, the throughput can only be improved with the ExpOL optimized models.

[1]Defined as the product of accuracy by the percentage of processed inferences, it measures the user experience demanding fast inferences at high quality.

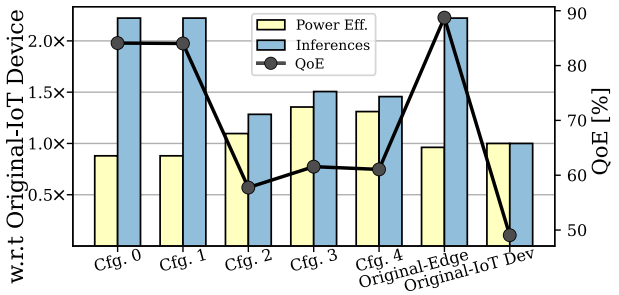

Figure 5. Power Efficiency and Total of Processed inferences w.r.t Original-IoT Device (bars, left y-axis) and QoE (right y-axis) under stable network scenario. Higher is better.

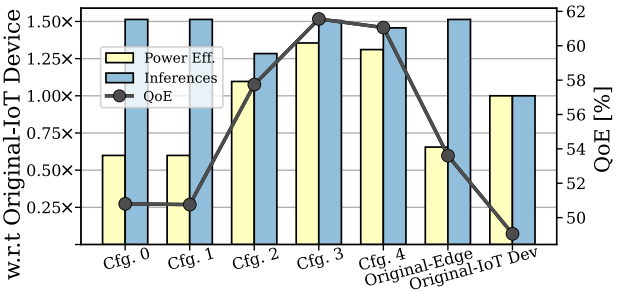

Figure 6. Power Efficiency and Total of Processed inferences w.r.t Original-IoT Device (bars, left y-axis) and QoE (right y-axis) under unstable network scenario. Higher is better.

Regarding efficiency in Figure 5, the power-oriented configuration, ExpOL configuration 3, is 1.4× and 1.35× more power-efficient than Original-Edge and Original-IoT Device baselines, respectively. However, the smaller IoT accelerators cannot process the full workload. The Original-IoT Device baseline showed an inference loss of 55% throughout the evaluation. While ExpOL configurations 2, 3, and 4 improved it to 42.2%, 32.2%, and 34.4% of inference losses, respectively. Considering those system configurations running inferences locally, ExpOL overcomes the Original-IoT Device baseline thanks to the pruning and early-exit optimizations. Such examples show that optimizations at the CNN level can alleviate the performance losses for power-oriented goals. As we are going to see next, local and optimized configurations will also be helpful when the network shows a not-so-stable behavior.

Similar to the previous plot, Figure 6 shows the results under the *unstable* scenario. The lower average bandwidth of this scenario (6.93 Mbps) represents a more challenging environment for edge offloading. Over the 15 seconds of evaluation, 287 frames were lost, representing 31.89% inference loss for edge-based configurations (Original-Edge baseline and ExpOL configurations 0 and 1). For those configurations, the inference loss is also perceived as a significant drop in the delivered Quality of Experience (QoE). In this scenario, the ExpOL configurations running inferences locally achieve QoE *and* power efficiency levels higher than *both* baselines. It becomes clear that tuning the design decisions of the inference processing is crucial for IoT-edge applications and the pruning and early-exit optimizations help navigate the design space when offloading inferences is not beneficial.

VI. Conclusions

We showed that combining the CNN optimizations to system-level design decisions can deliver efficient designs and improve the users' quality of experience (QoE). From a multi-target search on the design space, ExpOL delivers designs at the pareto front with power efficiency up to 2× and QoE up to 12.5% better than SoTA solutions either local or at edge servers.

Acknowledgments

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, São Paulo Research Foundation (FAPESP) grant #2021/06825-8, FAPERGS, CNPq, by the AI competence center ScaDS.AI Dresden/Leipzig (01IS18026A-D), and by the BMBF programme "Souverän. Digital. Vernetzt.", joint project 6G-life (16KISK001K).

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[2] E. Nurvitadhi, G. Venkatesh, J. Sim *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *FPGA*. ACM, 2017, pp. 5–14.
[3] S. Jiang *et al.*, "SCYLLA: QoE-aware Continuous Mobile Vision with FPGA-based Dynamic Deep Neural Network Reconfiguration," in *INFOCOM*. IEEE, 2020.
[4] H. Ting *et al.*, "Dynamic sharing in multi-accelerators of neural networks on an FPGA edge device," in *ASAP*. IEEE, 2020.
[5] A. Bokani *et al.*, "Comprehensive mobile bandwidth traces from vehicular networks," in *MMSys*. ACM, 2016, pp. 1–6.
[6] J. van der Hooft *et al.*, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.
[7] B. Fang *et al.*, "FlexDNN: Input-Adaptive On-Device Deep Learning for Efficient Mobile Vision," in *SEC*. IEEE, 2020, pp. 84–95.
[8] S. Laskaridis, S. I. Venieris *et al.*, "HAPI: hardware-aware progressive inference," in *ICCAD*. IEEE, 2020, pp. 91:1–91:9.
[9] M. Farhadi, M. Ghasemi, and Y. Yang, "A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on FPGA," in *HPEC*. IEEE, 2019, pp. 1–7.
[10] M. Wang *et al.*, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *SiPS*. IEEE, 2019, pp. 178–183.
[11] M. Blott *et al.*, "Finn-*R*: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM TRETS*, vol. 11, no. 3, pp. 16:1–16:23, 2018.
[12] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *ICPR*. IEEE, 2016, pp. 2464–2469.
[13] H. Li *et al.*, "Pruning filters for efficient convnets," in *ICLR*. OpenReview.net, 2017.
[14] J. Hauswald *et al.*, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *ISCA*. ACM, 2015, pp. 27–40.
[15] D. Crankshaw, *et al.*, "Clipper: A low-latency online prediction serving system," in *USENIX-NSDI*, 2017, pp. 613–627.
[16] C. Baskin *et al.*, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," in *IPDPS*, 2018.
[17] G. Korol *et al.*, "Synergistically exploiting cnn pruning and hls versioning for adaptive inference on multi-fpgas at the edge," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, 2021.
[18] Z. Xu *et al.*, "Reform: Static and dynamic resource-aware DNN reconfiguration framework for mobile device," in *DAC*. ACM, 2019.
[19] W. Kang, D. Kim, and J. Park, "DMS: dynamic model scaling for quality-aware deep learning inference in mobile and embedded devices," *IEEE Access*, vol. 7, pp. 168 048–168 059, 2019.
[20] J. Faraone *et al.*, "Customizing low-precision deep neural networks for fpgas," in *FPL*, 2018.
[21] G. Korol *et al.*, "Adaflow: A framework for adaptive dataflow CNN acceleration on fpgas," in *DATE*. IEEE, 2022, pp. 244–249.
[22] S. Laskaridis *et al.*, "SPINN: synergistic progressive inference of neural networks over device and cloud," in *MobiCom*. ACM, 2020, pp. 1–15.
[23] A. Pappalardo, "Xilinx/brevitas," https://doi.org/10.5281/zenodo.3333552.