AdaFlow: A Framework for Adaptive Dataflow CNN Acceleration on FPGAs

Guilherme Korol*, Michael Guilherme Jordan*, Mateus Beck Rutzig†,
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
*{gskorol,mgjordan,caco}@inf.ufrgs.br,†mateus@inf.ufsm.br

***Abstract*—To meet latency and privacy requirements, resource-hungry deep learning applications have been migrating to the Edge, where IoT devices can offload the inference processing to local Edge servers. Since FPGAs have successfully accelerated an increasing number of deep learning applications (especially CNN-based ones), they emerge as an effective alternative for Edge platforms. However, Edge applications may present highly unpredictable workloads, requiring runtime adaptability in the inference processing. Although some works apply model switching on CPU and GPU platforms by exploiting different pruning rates at runtime, so the inference can adapt according to some quality-performance trade-off, FPGA-based accelerators refrain from this approach since they are synthesized to specific CNN models. In this context, this work enables model switching on FPGAs by adding to the well-known FINN accelerator an extra level of adaptability (i.e., flexibility) and support to the dynamic use of pruning via fast model switch on flexible accelerators, at the cost of some extra logic, or via FPGA reconfigurations of fixed accelerators. From that, we developed AdaFlow: a framework that automatically builds, at design time, a library from these new available versions (flexible and fixed, pruned or not) that will be used, at runtime, to dynamically select a given version according to a user-configurable accuracy threshold and current workload conditions. We have evaluated AdaFlow under a smart Edge surveillance application with two CNN models and two datasets, showing that AdaFlow processes, on average, 1.3× more inferences and increases, on average, 1.4× the power efficiency over state-of-the-art statically deployed dataflow accelerators.**

***Keywords*—Edge Computing, Adaptive Inference, CNN, FPGA.**

I. Introduction

Due to thermal and energy constraints, many IoT devices are restricted in their processing capabilities, requiring that computing-intensive tasks get processed elsewhere. In this scenario, numerous IoT devices send their raw data over the local network to Edge servers that are physically closer than the cloud, avoiding long latency and increasing security. Convolutional Neural Networks (CNNs) are a representative example of such heavy tasks and are used for many IoT applications, like smart video surveillance, intelligent manufacturing, smart cities, etc. FPGA platforms, due to their performance, energy costs, and reconfigurability, have successfully deployed these CNN models, emerging as scalable alternatives to be integrated into Edge servers [1–3]. However, technology scaling and hardware improvements alone cannot push forward the current levels of efficiency demanded by such applications [4]. Therefore, the CNN models that lie on top of the hardware layer must be optimized as well [5, 6].

Pruning [8] is one representative example of these optimizations. It consists in removing parts of a Deep Neural Network (DNN) to improve performance at the cost of accuracy. Figure 1(a) illustrates this effect in a CNN. It plots the accuracy and throughput (given in Frames per Second) of a CNN model over pruning rates varying from 0 to 85%. As we increase the pruned portion of the CNN, inference is processed faster as accuracy decreases. Some CPU and GPU-based Edge platforms

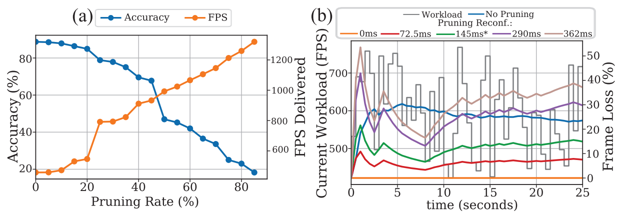


Figure 1. (a) Accuracy and FPS vs. Pruning Rate for CNVW2A2 on CIFAR10 over FINN [7]. (b) Edge server's workload and frame loss for not pruned and pruned CNVW2A2 models switched by FPGA reconfigurations of varied times. * Indicates the original CNVW2A2 FINN reconf. time on a ZCU104.

have been employing multiple CNN models with different pruning rates, so they can be dynamically switched according to the requirements at hand, which is called model switching [5, 6]. Therefore, pruning can be exploited to enable adaptation in highly unpredictable Edge environments with changing workload levels, available resources, or accuracy. However, state-of-the-art FPGA-based dataflow accelerators [7, 9], which are specifically synthesized for one CNN model, preclude fast model switching for FPGA-based modern Edge solutions: they require frequent use of time-consuming FPGA reconfigurations to switch from one accelerator to another. To better support this claim, let us take, as an example, an FPGA-based Edge server receiving frames from IoT devices to be processed (CNN inference). Due to factors like FPS fluctuation [10], network congestion [11], or variable number of connected nodes, the rate of incoming inference requests may change over time. Figure 1(b) shows this variance in the "workload" line, given as the incoming FPS (left y-axis) through time (x-axis). We simulated the following FPGA-based Edge servers (represented by different lines): "**No Pruning**" that uses the state-of-the-art FINN [7] dataflow CNN accelerator without pruning; and, "**Pruning Reconf.**" that can switch pruned models and their respective FINN accelerators by reconfiguring the FPGA with a specific reconfiguration time and frame loss rate (right y-axis).

From Figure 1(b), we point out that switching CNN models is mandatory since unforeseeable changes in workload require adapting the inference processing. However, model switching has to be fast enough so it does not undermine the overall performance. For example, even though servers with 290ms (purple curve) and 362ms (brown curve) switch models to increase their throughput, their reconfiguration times cause frame losses that are higher than using no model switching (i.e., use the original, not pruned FINN). As the reconfiguration time decreases, the model switching gains start to appear, and, in an ideal scenario with no reconfiguration overhead (0ms - orange curve), all frames would get processed, achieving zero frame loss. While a zero-cost reconfiguration time is unrealistic, this experiment shows that fast model switching (i.e., faster than traditional FPGA reconfiguration times) is mandatory for highly unpredictable workload scenarios.

Nevertheless, enabling such extra adaptability to allow for shorter switching times requires extra logic and power, creating a trade-off between flexibility and efficiency for FPGA-based dataflow accelerators. Given that, our contribution is twofold: 1) we implemented an extra level of adaptability to FINN on top of its framework, so pruning and fast model switching are possible. With that, flexible-pruning accelerators (i.e., allow for fast switching) and accelerators fixed to particular pruned models (fixed-pruning) become available; 2) These new versions enabled the development of AdaFlow, which is a hybrid, two-step approach: *at design time*, AdaFlow automatically generates a library composed of the now available pruned CNN models with different resource and accuracy profiles, and dataflow accelerators that can either support fast model switching or model switching via FPGA reconfiguration; *at runtime*, it offers a management technique that automatically chooses the best CNN model and accelerator type from the library to dynamically adapt the inference serving, aiming to increase the number of processed inferences with less energy or higher throughput, according to a configurable accuracy threshold and current environmental workload demands.

Therefore, this work makes the following contributions:

- A dataflow-aware *pruning method* for generating multiple CNN models and a novel dataflow accelerator implemented on top of FINN that enables *fast model switching*;
- AdaFlow, a framework for adaptive inference on dataflow accelerators, which comprises two steps: the static generation of a library of pruned CNN models with flexible and fixed accelerators; and the dynamic exploitation of this library by switching, at runtime, between CNN models and accelerators to approach the best balance of accuracy, performance, and power given a set of requirements;
- We show that AdaFlow processes, on average, 1.3× more inferences at minor accuracy penalties, resulting in Quality of Experience improvements of 12%, while improving power efficiency by 1.27×, on average, when compared to FINN accelerators deploying CNVW2A2 and CNVW1A2 CNN models on the CIFAR10 and GTSRB datasets.

II. Background

CNN Optimizations. With millions of weights and multiply-accumulate (MAC) operations, modern CNNs require extreme amounts of computation and memory transfers. To enable more efficient CNNs, works have proposed several optimizations to reduce CNN memory and computation requirements. Compression methods like non-structured, structured pruning, and quantization are leading examples of such optimizations. Pruning is especially recommended when there is a need to reduce not only the model's memory footprint (e.g., by quantizing weights), but also computation (i.e., number of MACs) at inference. In particular, filter pruning is the structured pruning technique used in this work. It removes entire filters from the CNN weight matrices, creating no sparsity (keeping memory access regular), which facilitates the use of the existing hardware infrastructure. Removing filters from a convolutional (CONV) layer also reduces the number of channels of the output feature map. This correlation between pruned filters and channels grant filter pruning a roughly quadratic effect on reducing the CNN's footprint and computations.

FPGA-Based CNN Acceleration with FINN. FPGA-based accelerators can be divided into two groups: *single-engine* (where a single convolutional engine performs all layers, one at a time), and *dataflow accelerators* that rely on a pipeline-like, feed-forward, architecture mapping each CNN layer to a dedicated module [7, 9]. We adopt dataflow accelerators since they have shown the highest performance levels compared to the single-engine ones due to their specialization and greater ability to explore parallelism [12]. Tools like fpgaConvNet [9] and FINN [7] are frameworks for mapping DNNs to dataflow accelerators on FPGA. These frameworks take a graph-based description of a CNN to map it to a sequence of hardware modules connected in a *dataflow* fashion. FINN, which is used in this work, employs hardware modules implemented as a set of High-Level Synthesis (HLS) template classes configured with each layer's parameters. Therefore, each generated dataflow accelerator is "hard-wired" to its CNN model.

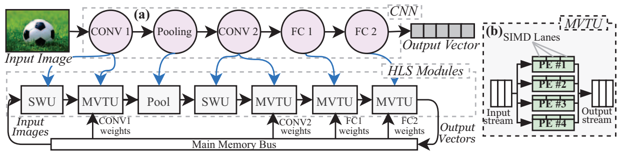


Figure 2. A sample CNN-dataflow mapping.

Figure 2(a) shows a mapping from a sample CNN to a simplified FINN dataflow. FINN maps CONV layers to the Sliding Window Unit (SWU) and Matrix-Vector-Threshold Unit (MVTU) modules. The SWU module prepares the input feature map to be multiplied with the weight matrix in the MVTU, performing the actual convolution operation. Its output corresponds to that layer's output feature map and is fed to the next module in the dataflow. Figure 2(b) details the FINN's MVTU module. Each MVTU can be configured with the number of processing elements (PEs) and SIMD lanes, which directly affects MVTU's performance and resource usage. The number of PEs and SIMD in each layer's MVTU can be defined by the user (through a configuration file) but are also constrained by parameters of the CNN model, like the layer's number of channels and kernel size. These restrictions will guide the pruning process presented in Subsection IV-A1.

III. Related Work

To cope with the resource-hungry processing, CNN inferences can be offloaded from resource-limited devices to servers equipped with high-performance architectures, the so-called inference serving systems. At the Edge, FPGA-based accelerators have been largely used to accelerate inferences [1–3], enabling HLS optimizations [13], consuming less energy than GPU boards with equivalent accuracy [14], or even superior performance with small accuracy drops [15]. Scylla [1] employs an FPGA for serving CNN inferences at the Edge, exploiting the reconfigurability capabilities of FPGAs for Quality of Experience (QoE) optimization. In [2], the authors exploit the regularity of inference requests for allocating and scheduling multiple accelerators over a Xilinx ZCU104.

To further increase the performance/efficiency of inference processing, some works have approached optimizations on the FPGA-based accelerators. Seyoum et al. make use of dynamic partial reconfiguration to break a FINN dataflow into chunks and schedule them to execution on low-cost FPGAs [16]. In [17], a toolflow for statically customizing CNNs to the underlying dataflow is proposed. On the other hand, GPU-based Edge platforms have employed optimizations like CNN pruning dynamically to adapt the inference on the accuracy-resource trade-off. ReForm [6] provides resource-aware inference mechanisms by reconfiguring a CNN model according to the mobile device's computation resource. Dynamic-OFA [5] dynamically prunes a CNN to adapt the inference in embedded GPUs.

Wrap-up and Our Contributions. CPU and GPU-based inference systems have shown that adapting the inference processing (e.g., by switching pruned CNN models) is crucial at the Edge [5, 6]. Even though works have already shown that pruning can be used at the design stage of FPGA-based dataflow accelerators [17, 18], these systems have not been able to leverage such optimizations at runtime since they are synthesized to particular CNN models [7, 9]. In this context, AdaFlow enhances the state-of-the-art of FPGA dataflow accelerators with runtime flexibility, enabling fast model switching on dataflow accelerators and enabling the advantages of adaptive pruning at runtime.

IV. AdaFlow

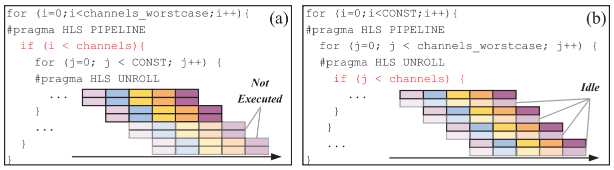
A. AdaFlow's CNN and Dataflow Optimizations

This section details our pruning technique and our modifications to FINN. Both contributions enable the generation of multiple pruned CNN models and the novel flexible dataflow accelerator used in the AdaFlow framework.

1) Dataflow-Aware Pruning: To provide multiple design points on the accuracy-resource trade-off for a single dataflow accelerator, we propose a filter pruning mechanism that, besides the CNN model, takes into account properties of the dataflow accelerator. In FINN, there are two main constraints refraining a dataflow accelerator from loading a CNN model that had its CONV layers *freely* pruned: for each MVTU performing a CNN layer, we have that the number of PEs has to be divisible by the number of CONV filters (or neurons, in the case of a fully-connected layer); and, the number of SIMD lanes has to be divisible by the number of input channels. Such constraints guarantee the correctly feeding of all PEs and SIMD lanes, ensuring full parallelism (i.e., no idle PEs or SIMD lanes). To build models that respect such constraints, we implemented the Dataflow-Aware Pruning mechanism, which, starting from an initial CNN model, prune filters to generate a pruned model version with particular accuracy and resource profile.

For each pruned model, our Dataflow-Aware Pruning takes an initial CNN model, a FINN configuration file (containing the dataflow parameters), and a pruning rate (percentage specifying how many filters to prune). Then, for every CONV layer, the procedure attempts to prune a certain amount of filters r_i in such a way that it respects the $({ch_{out}}^i - r_i)\, mod\, (PE_i) = 0$ and $({ch_{out}}^i - r_i)\, mod\, (SIMD_{i+1}) = 0$ constraints, where PE_i and $SIMD_{i+1}$ give the MVTU's number of PEs and SIMD lanes (see Figure 2(b)) of current i and next layer $i+1$, respectively. ${ch_{out}}^i$ gives the not-pruned number of channels for that layer (from the initial CNN). If the constraints are not met, the procedure iteratively decreases r_i until they are met. Dataflow-Aware Pruning leverages the filter selection proposed in [8] that measures the relative importance of a filter in each channel by calculating, from the floating-point representation, the sum of its absolute weight values (ℓ_1-norm). After all CONV layers have been pruned, the model can be retrained and exported as an ONNX file. We call the dataflow accelerators synthesized from pruned CNN models **Fixed-Pruning** since they can only execute that particular model. AdaFlow's pruning is implemented on top of Brevitas [19], which is a PyTorch-based tool for quantization-aware training from Xilinx and is part of the FINN infrastructure.

2) Enabling Runtime Flexibility on FINN: In our work, besides the accelerators generated from pruned CNN models that require FPGA reconfiguration whenever a model switch is needed, we enable fast switching of those models through a novel dataflow accelerator, the **Flexible-Pruning**, so FPGA reconfigurations are not needed. For that, we modified FINN's HLS classes with runtime-controllable parameters. Since the pruning technique used by AdaFlow affects only the number of CONV filters and its respective number of output channels, the number of channels is the only parameter that needs to be configured at runtime in the dataflow accelerator. In that sense, Flexible HLS templates differ from regular FINN HLS templates only in the loops in which bounds are affected by the number of CONV channels. Then, at runtime, it can be configured to process a smaller number of channels. Therefore, flexible accelerators are synthesized to the worst case in terms of model size, given by the initial, not-pruned, CNN model. Below, we focus on the difference between Flexible and FINN HLS templates. The modifications made to FINN's HLS classes can be divided into two cases: when the runtime-controllable parameter (i.e., number of CONV channels) affects HLS *unroll* directive and when it affects the *pipeline* directive.



(a)
```
for (i=0;i<channels_worstcase;i++){
#pragma HLS PIPELINE
  if (i < channels){
    for (j=0; j < CONST; j++) {
    #pragma HLS UNROLL
      ...
    }
    ...
  }
}
```

(b)
```
for (i=0;i<CONST;i++){
#pragma HLS PIPELINE
  for (j=0; j < channels_worstcase; j++) {
  #pragma HLS UNROLL
    if (j < channels) {
      ...
    }
  }
  ...
}
```

Figure 3. Loops with variable control (red shaded) in Flexible HLS templates with `channels` half the `channels_worstcase` in both examples.

We bring two representative examples for presenting each case: the MVTU module, responsible for executing all convolutional and fully-connected layers, and the MaxPool module. MVTU's unroll is independent of the runtime-controllable parameter (MVTU unroll is given by the number of PEs and SIMD lanes - see Figure 2). On the other hand, MaxPool unrolling depends on the number of channels. Figure 3 presents these two examples, where `channels_worstcase` gives the full number of input or output channels (given by a not pruned model) and `channels` is the runtime-controllable parameter that gives the current number of channels (particular to the currently loaded CNN model version, variable between models). Red shaded `if` statements give the runtime-controllable behavior to both modules. In Figure 3(a) we have a simplified version of the MVTU, which is unrolled on a fixed parameter `CONST` (i.e., PE/SIMD values). In this case, the runtime-controllable parameter only affects the pipeline feeding, causing fewer pipeline iterations and a shorter execution time. On the other hand, for the MaxPool module that is unrolled on the runtime-controllable parameter, the loop has to be synthesized to the worst case, and, when `channels` < `channels_worstcase`, some of the units performing the unrolled operation will not be fed, as depicted in Figure 3(b).

Information on the number of channels of every model's layer is attached to the model description when AdaFlow prunes a CNN model. Then, during inference, at runtime, the number of channels can be passed to every flexible module in the dataflow (those have an extra 16-bit interface port to set the runtime-controllable, `channels`, parameter). However, the extra circuitry enabling flexible execution creates a small overhead in both resource usage (making the accelerator larger) and performance (increased latency). This trade-off between accelerators' flexibility and efficiency is exploited by AdaFlow to process inferences at the highest levels of adaptability.

We added the set of new flexible HLS template classes to the original FINN framework. Also, FINN's design steps [7] were modified to integrate the new runtime-controllable functionalities and ensure synthesis and correct processing of the Flexible-Pruning accelerators.

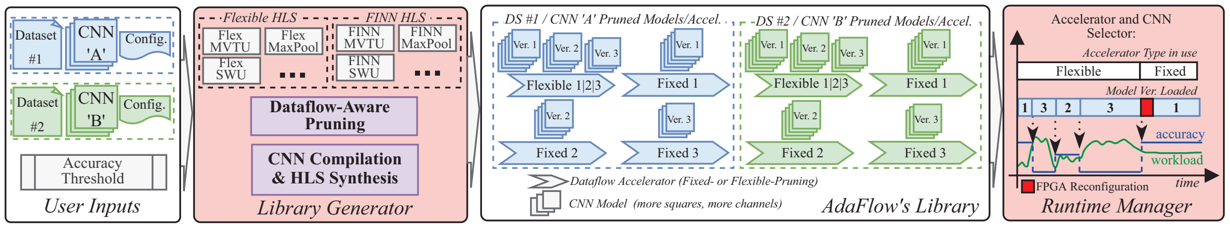


Figure 4. AdaFlow's Workflow.

B. AdaFlow Workflow

AdaFlow's goal is to provide an efficient and adaptive solution for CNN inference processing. Figure 4 shows AdaFlow's two-step workflow from the generation of its Library up to the execution of inferences. AdaFlow starts by receiving the following user inputs: initial CNN models, training dataset, and FINN configuration files as well as an accuracy threshold. Then, the *Library Generator* creates a library containing multiple pruned CNN models and dataflow accelerators. In this step, user datasets and initial CNN models are used to generate multiple pruned CNN models and their respective dataflow accelerators (**Fixed**- and **Flexible**-Pruning). AdaFlow's Library supports the second step, the *Runtime Manager*, that performs runtime adaptation of the inference processing based on the user's accuracy threshold and workload conditions. Next, we detail the AdaFlow's Library Generator and Runtime Manager.

1) Library Generator: In this step, the Library Generator reads the user's inputs to feed the Dataflow-Aware Pruning (presented in Subsection IV-A1). Then, by ranging the pruning rate at fixed steps for each dataset/initial CNN model, it gathers multiple pruned versions of each initial CNN model. These CNN model versions offer multiple design points on the accuracy-resource trade-off (the larger and less pruned a CNN is, the higher is its accuracy). After all pruned CNN models are generated, the Library Generator passes them to the "CNN Compilation & HLS Synthesis" to synthesize their respective accelerators: Fixed-Pruning (one for each pruned CNN model, using fixed modules - FINN HLS in Figure 4) and Flexible-Pruning (one for all pruned models of the same initial CNN model, using the new Flexible HLS classes). With CNN models and accelerators generated, a library in the form of a table containing a list of pruned CNN models (rows) with their accuracy (extracted after pruning) as well as the throughput values (extracted during synthesis) is created.

2) Runtime Manager: The Runtime Manager is the software module in charge of selecting and loading the pruned CNN models and accelerator type (Fixed or Flexible). The choice for which CNN model to load (i.e., which pruned version) is tightly coupled to the application at hand. Generally, the conflicting targets of accuracy and performance are desired for inference - in other words, it should process the most inferences at the highest accuracy. To that end, the Runtime Manager dynamically selects CNN models taking their accuracy and throughput into account. The search on the Library takes as input the user's accuracy threshold (see user's input in Figure 4) and information the server's current workload (i.e., incoming FPS). Then, it selects, among the pruned models with accuracy above the threshold, the model that delivers the highest throughput (i.e., FPS). The only exception happens when the desired FPS (currently incoming) can be matched by more than one model. In that case, the model with the best accuracy will be selected. The Runtime Manager will act every time there is a change in either accuracy threshold (set by the user) or incoming FPS (that can be flagged by performance monitors added to the software in charge of the incoming inferences). For selecting the accelerator type, the Runtime Manager uses a rule-based criteria: Fixed-Pruning accelerators are only selected when models need to be switched at intervals greater than a predefined value, which can be fine-tuned depending on the application and FPGA at hand. For example, if the user sets this criteria to 1 second and FPGA reconfiguration is 100ms long, the Runtime Manager will only select Fixed-Pruning if, at least, 1 second has passed since the last model switch. Otherwise, Flexible-Pruning accelerators are used. It is important to note that fine-tuning this criteria is possible for other applications.

As an illustrative example, Fig. 4's Runtime Manager displays workload and accuracy curves for a sample application. First, let us see how the Runtime Manager uses the trade-off created by pruned CNN model versions (ver. 1 to 3 in Fig. 4). We can see that the Runtime Manager responds to workload increases by switching to *faster* models (at the cost of lower accuracy). For instance, the model first loaded (version 1) is switched by model version 3 as soon as the workload rises. As version 3 was pruned more aggressively, it has fewer channels (fewer squares in Fig. 4), resulting in a higher throughput that can better accommodate the current workload level. For unpredictable and fast changes in workload, CNN models can be switched fast, requiring no FPGA reconfiguration, thanks to the Flexible accelerator in use. On the other hand, when the application shifts to a more stable phase, the Runtime Manager can load a Fixed-Pruning dataflow since the reconfiguration overhead will not harm the overall performance (no subsequent model switches are expected). Fixed-Pruning accelerators consume less power as they do not have any of the logic due to the runtime-controllable parameters.

V. Methodology

Setup and Tools. Accelerators used across our experiments were synthesized within the Xilinx's FINN design flow [7] with Vivado targeting a Xilinx Zynq Ultrascale+ MPSoC ZCU104 board (XCZU7EV) at 100MHz. We used Xilinx Vivado for resource usage and power extraction and Verilator RTL simulations for performance (FINN infrastructure).

We adopted two CNN models supported by our accelerator of choice for evaluation: CNVW2A2 and CNVW1A2. Models were adapted to the CIFAR-10 and the German Traffic Sign Recognition Benchmark (GTSRB) datasets. All images consider CIFAR-10's image resolution (3x32x32). Accuracy results are reported on Brevitas TOP-1 test accuracy. Each pruned CNN model is retrained for 40 epochs after pruning [8], with standard data augmentation (padding, crop, and flipping) and learning rate of 0.001 with decay of 0.1. Retraining was performed on Intel Xeon E5-2640 with NVIDIA Tesla K20m GPU. AdaFlow generates 18 models for each initial CNN with pruning rates from 0% (not-pruned) to 85% (5% steps). Each model generates a *Fixed-Pruning* dataflow accelerator. Four *Flexible-Pruning* accelerators were synthesized, one for each dataset/CNN.

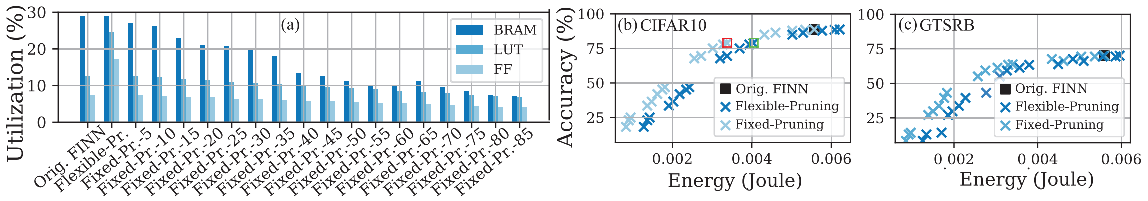


Figure 5. (a) FPGA Resource for FINN, Flexible and Fixed accelerators. Accuracy vs. Energy for CNV2W2A on CIFAR-10 (b) and GTSRB (c).

Evaluation Scenario. Our case study is based on typical smart video surveillance systems that have to deal with numerous cameras (IoT devices) sending inference requests (frames) to a local Edge server. Therefore, we have set 20 IoT devices to produce inference requests at the real-time rate of 30 FPS. Evaluations are 25 seconds long. Due to factors like FPS fluctuation [10], network congestion [11], or variable number of connected nodes, the rate of incoming inference requests (workload's incoming FPS) in an inference server changes over time [20]. Based on such environments [10, 11, 20], we evaluate AdaFlow under two scenarios: **Scenario 1** that represents a more stable Edge environment with 30% random workload deviation every 5 seconds; and **Scenario 2** that represents a more unpredictable environment with 70% workload deviation every 500ms. Experiments are executed 100 times, and average values are reported. We have set the maximum accuracy loss (AdaFlow's accuracy threshold) to 10%. The Runtime Manager selects accelerator type given a predefined criteria (see Subsection IV-B2). Based on our experiments, we set this value to 10× the reconfiguration time. We note that both parameters are user-defined and can be tuned to specific applications and user goals. In the next section, we evaluate AdaFlow over the Original FINN (baseline) on performance, power efficiency, and Quality of Experience, here defined as the product of accuracy by the percentage of processed frames. With QoE, we assess the user experience that targets low frame loss levels and high accuracy levels.

VI. Results

A. AdaFlow's Design Space

Before we evaluate AdaFlow under the Edge application, this section shows the design space enabled by AdaFlow's Library. Figure 5(a) shows the FPGA resource usage (y-axis) for the original FINN and AdaFlow's Flexible and Fixed-Pruning accelerators for CNVW2A2 CNN on the CIFAR-10 dataset. Figures 5(b) and (c) plot the energy per inference (x-axes) vs. accuracy (y-axes) for CNVW2A2 on CIFAR-10 and GTSRB datasets. CNVW1A2 follows the same behavior. As it can be noticed in Figure 5(a), the Flexible-Pruning accelerator presents the highest resource usage compared to FINN and its Fixed-Pruning counterparts as it requires extra logic to implement the runtime-controllable behavior. Still, as the space taken by feature maps and weights only decreases with pruned models when compared to FINN, Flexible-Pruning shows no increase in BRAM usage (which is often the limiting factor for FPGA-based CNN accelerators - i.e., the resource with the highest usage, see Figure 5(a)). Flexible-Pruning increases in 1.92× the number of used LUTs compared to the original FINN. Fixed-Pruning, on the other hand, presents reductions in LUT usage ranging from 1.5% (at 5% pruning rate) to 46.2% (at 85% pruning rate) compared to the original FINN.

Although requiring more resources, AdaFlow's Flexible-Pruning allows switching to pruned models that will deliver higher performance and lower energy consumption at runtime. Overall, as the pruning rate increases, the energy consumption decreases at the cost of accuracy w.r.t the FINN baseline. For example, it would be possible to switch to a 25% pruned model on Flexible-Pruning (green square in Fig. 5(b)), reducing the energy per inference by 1.38× with accuracy loss of only 9.9% when compared to FINN. The same pruning rate with a Fixed-Pruning accelerator (red square in Fig. 5(b)) reduces by 1.64× the energy consumption w.r.t FINN. When comparing AdaFlow's accelerators, the Fixed-Pruning ones present slightly better inference latency (up to 3.7% difference, 0.67% average), meaning that Fixed-Prune's lower energy consumption is mainly due to having no runtime-controllable logic.

Even though Flexible accelerators present energy consumption higher than their Fixed counterparts, they require no FPGA reconfigurations to switch models at runtime. Consequently, in scenarios where the application requirements change rapidly, Flexible-Pruning is the only alternative for runtime adaptation, as will be presented next.

B. AdaFlow at the Edge

In this section, we evaluate AdaFlow under the two Edge scenarios presented in Section V. Table I reports frame loss, QoE, and power for AdaFlow and Original FINN averaged over the full 25 seconds run. Table I's right-most column gives AdaFlow's power efficiency (number of processed inferences per Watt) w.r.t Original FINN. First, we see that AdaFlow achieves greater performance (i.e., lower frame loss) and power efficiency than the original FINN for all evaluated datasets and CNN models. The higher performance levels (frame loss up to 27.22% lower than Original FINN - GTSRB/CNVW1A2) and efficiency (up to 1.40× more efficient - GTSRB/CNVW2A2) are enabled by AdaFlow's dynamic adaptation. By switching to models of higher throughput whenever needed, AdaFlow can adapt the inference processing to eventual workload increases with minimal accuracy loss. This minimal accuracy loss and improved performance also cause AdaFlow to deliver QoE higher than FINN. As it is not always necessary to keep accuracy close to the threshold (e.g., for a currently low workload level), AdaFlow's Runtime Manager (Subsection IV-B2) grants a 7.07% (CIFAR-10/CNVW2A2) maximum accuracy drop over all evaluations (4.6% on average). We also would like to note that for applications that tolerate accuracy thresholds larger than the one in use (10%), larger performance and efficiency gains are expected since, in that case, CNN models of more aggressive pruning would be allowed.

Now, let us consider a representative evaluation to detail the AdaFlow behavior. In Figure 6(a), we show FINN and AdaFlow frame losses (y-axis) over the 25 seconds run (x-axis) for CIFAR-10/CNVW2A2 under Scenarios 1 and 2 as well as Scenario 1+2 that provides an additional evaluation. To show how AdaFlow adapts to a change of environment, Scenario 1+2 starts with a stable condition up to 15 seconds (same setting of Scenario 1) and then shifts to a more unpredictable phase (same setting of Scenario 2) that lasts until the end of the evaluation. In Figure 6(a), we also show the pruned models used by AdaFlow for Scenario 1+2 and the moment it changes the accelerator type. Pruned models switched during

Table I
FRAME LOSS, QOE, POWER, AND POWER EFFICIENCY FOR ALL DATASETS AND CNN MODELS ON THE FULL 25 SECONDS RUN.

		Frame Loss (%)		QoE (%)		Power (W)		Power Eff.
Dataset / Model	Scenario	AdaFlow	Orig. FINN	AdaFlow	Orig. FINN	AdaFlow	Orig. FINN	w.r.t FINN
CIFAR-10 / CNVW2A2	1	0	23	81.74	68.32	1.01	1.07	1.39x
	2	5.11	30.99	78.54	61.23	1.2	1.07	1.25x
GTSRB / CNVW2A2	1	0	23.53	65.12	53.55	1.01	1.07	1.4x
	2	3.64	29.91	63,21	49.08	1.14	1.07	1.3x
CIFAR-10 / CNVW1A2	1	12.27	23.68	73.58	66.63	0.98	1	1.17x
	2	21.89	31.73	66.12	60.47	1.125	1	1.01x
GTSRB / CNVW1A2	1	0	22.57	65.85	69.86	0.94	0.96	1.35x
	2	4.14	31.36	62.88	47.95	1.11	0.97	1.23x

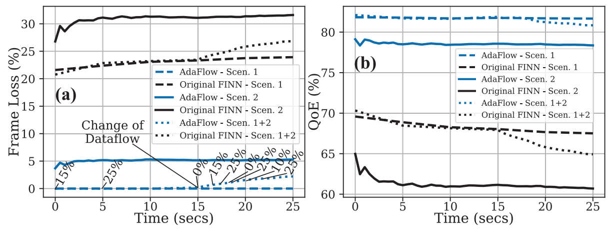


Figure 6. Frame Loss (a) and QoE (b) for CNVW2A2 on the CIFAR10 dataset. AdaFlow's model switches are indicated in (a).

the execution of other scenarios are not displayed for the sake of clarity. Figure 6(b) follows the same idea of (a) for showing Orignal FINN and Adaflow QoE curves.

As explained in Subsection IV-B2, the Runtime Manager selects Fixed-Pruning accelerators for stable environments (i.e., the ones that cause relatively few model switches). Scenario 1 represents such a condition, allowing that AdaFlow reconfigures the FPGA with Fixed-Pruning accelerators. For example, considering the first run for CIFAR-10/CNVW2A2 out of the 100 runs averaged ("AdaFlow - Scen. 1" in Figure 6), AdaFlow switched models five times, resulting in a total od five FPGA reconfigurations (around 725ms total). Even though they need FPGA reconfigurations, Fixed-Pruning accelerators offer higher power efficiency: see AdaFlow's lower frame loss and lower power dissipation in Scenario 1 (when using Fixed-Pruning) in contrast to AdaFlow in Scenario 2 in Figure 6(a). In Scenario 2, when changes in workload happen at a higher rate, FPGA reconfigurations are not allowed. Thus, AdaFlow employs the Flexible-Pruning accelerator. A total of 31 model switches were performed over Scenario 2's 25 seconds run. For those model switches, however, no FPGA reconfiguration is required thanks to the Flexible-Pruning accelerator. In summary, enabling fast model switches on the FPGA-based server made it possible to achieve power efficiency 1.25× greater and frame loss 25% lower than the Original FINN in Scenario 2.

Unlike the first two scenarios, Scenario 1+2 presents a shift in the workload condition at runtime (as explained earlier). This change of workload condition causes AdaFlow to change from Fixed- to Flexible-Pruning accelerator. During the more stable phase of Scenario 1+2 (from 0 to 15secs), AdaFlow switched models twice (15 and 20% pruning rates, indicated in Figure 6(a)) with Fixed-Pruning accelerators. After that, the Runtime Manager flagged a change in workload at a shorter time interval and reconfigured the FPGA with the Flexible-Pruning accelerator ("Change of Dataflow" in Figure 6(a)). Once the Flexible-Pruning was loaded, six more model switches were performed with no need for FPGA reconfigurations. Overall, in Scenario 1+2, AdaFlow achieved a frame loss 24% lower, and increased QoE and power efficiency by 15% and 1.21×, respectively, over FINN. Therefore, in contrast to inference on traditional dataflow accelerators, AdaFlow adapts the inference processing by changing the pruning rate at runtime. Moreover, as shown in Scenario 1+2, AdaFlow can exploit power-efficient accelerators (e.g., from 0 to 15secs) or fast model switching (e.g., from 15secs onwards) dynamically, depending on the workload conditions.

VII. CONCLUSIONS

We proposed the AdaFlow framework that adapts at runtime the inference processing with transparently generated pruned CNN models and new accelerator designs that enable fast model switching and power-efficient execution. Overall, AdaFlow improves, on average, power efficiency in 1.27×, performance in 21%, and QoE in 12% over the state-of-the-art FINN.

ACKNOWLEDGMENTS

This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Brazil - Finance Code 001, FAPERGS and CNPq.

REFERENCES

[1] S. Jiang *et al.*, "SCYLLA: QoE-aware Continuous Mobile Vision with FPGA-based Dynamic Deep Neural Network Reconfiguration," in *INFOCOM*. IEEE, 2020.
[2] H. Ting *et al.*, "Dynamic sharing in multi-accelerators of neural networks on an FPGA edge device," in *ASAP*. IEEE, 2020.
[3] 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.
[4] A. C. S. Beck, C. A. L. Lisbôa, and L. Carro, *Adaptable embedded systems*. Springer, 2012.
[5] W. Lou *et al.*, "Dynamic-ofa: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms," in *CVPR Workshops*, 2021.
[6] Z. Xu *et al.*, "Reform: Static and dynamic resource-aware DNN reconfiguration framework for mobile device," in *DAC*. ACM, 2019.
[7] 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.
[8] H. Li *et al.*, "Pruning filters for efficient convnets," in *ICLR*. OpenReview.net, 2017.
[9] S. I. Venieris and C. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *FCCM*, 2016, pp. 40–47.
[10] Y. Shuai *et al.*, "Memtv: a research on multi-level edge computing model for traffic video processing," in *CAC*, 2020.
[11] B. Fu *et al.*, "A survey of cross-layer designs in wireless networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 110–126, 2014.
[12] M. Blott *et al.*, "Evaluation of optimized cnns on heterogeneous accelerators using a novel benchmarking approach," *IEEE Trans. on Comp.*, vol. 70, no. 10, pp. 1654–1669, 2021.
[13] B. N. Lignati *et al.*, "Exploiting hls-generated multi-version kernels to improve CPU-FPGA cloud systems," in *ASP-DAC*, 2021.
[14] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for iot intelligence on the edge," in *DAC*, 2019.
[15] C. Baskin *et al.*, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," in *IPDPS*, 2018.
[16] B. Seyoum *et al.*, "Spatio-temporal optimization of deep neural networks for reconfigurable fpga socs," *IEEE Trans. on Comp.*, pp. 1–1, 2020.
[17] J. Faraone *et al.*, "Customizing low-precision deep neural networks for fpgas," in *FPL*, 2018.
[18] T. Peres, A. Gonçalves, and M. P. Véstias, "Faster convolutional neural networks in low density fpgas using block pruning," in *ARC*, 2019.
[19] A. Pappalardo, "Xilinx/brevitas," https://doi.org/10.5281/zenodo.3333552.
[20] V. J. Reddi, C. Cheng, D. Kanter *et al.*, "Mlperf inference benchmark," in *ISCA*. IEEE, 2020, pp. 446–459.