Automated HW/SW Co-design for Edge AI: State, Challenges and Steps Ahead

Special Session Paper

Oliver Bringmann
oliver.bringman@uni-tuebingen.de
University of Tübingen
Tübingen, Germany

Wolfgang Ecker
Wolfgang.Ecker@infineon.com
Infineon Technologies AG
Neubiberg, Germany

Ingo Feldner
ingo.feldner@de.bosch.com
Bosch Corporate Research
Renningen, Germany

Adrian Frischknecht
adrian.frischknecht@uni-tuebingen.de
University of Tübingen
Tübingen, Germany

Christoph Gerum
christoph.gerum@uni-tuebingen.de
University of Tübingen
Tübingen, Germany

Timo Hämäläinen
timo.hamalainen@tuni.fi
Tampere University
Tampere, Finland

Muhammad Abdullah Hanif
muhammad.hanif@tuwien.ac.at
Technische Universität Wien, Austria
New York University Abu Dhabi, UAE

Michael J. Klaiber
michael.klaiber@de.bosch.com
Bosch Corporate Research
Renningen, Germany

Daniel Mueller-Gritschneder
daniel.mueller@tum.de
Technical University of Munich
Munich, Germany

Paul Palomero Bernardo
paul.palomero-bernardo@uni-tuebingen.de
University of Tübingen
Tübingen, Germany

Sebastian Prebeck
Sebastian.Prebeck@infineon.com
Infineon Technologies AG
Neubiberg, Germany

Muhammad Shafique
muhammad.shafique@nyu.edu
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates

ABSTRACT

Gigantic rates of data production in the era of Big Data, Internet of Thing (IoT), and Smart Cyber Physical Systems (CPS) pose incessantly escalating demands for massive data processing, storage, and transmission while continuously interacting with the physical world using edge sensors and actuators. For IoT systems, there is now a strong trend to move the intelligence from the cloud to the edge or the extreme edge (known as TinyML). Yet, this shift to edge AI systems requires to design powerful machine learning systems under very strict resource constraints. This poses a difficult design task that needs to take the complete system stack from machine learning algorithm, to model optimization and compression, to software implementation, to hardware platform and ML accelerator design into account. This paper discusses the open research challenges to achieve such a holistic Design Space Exploration for a HW/SW Co-design for Edge AI Systems and discusses the current state with three currently developed flows: one design flow for systems with tightly-coupled accelerator architectures based on RISC-V, one approach using loosely-coupled, application-specific accelerators as well as one framework that integrates software and hardware optimization techniques to built efficient Deep Neural Network (DNN) systems.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0131 and Business Finland Co-innovation project "SoC Hub".

CODES/ISSS '21, October 10–13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9076-7/21/10.
https://doi.org/10.1145/3478684.3479261

CCS CONCEPTS

• **Computing methodologies** → *Machine learning*; • **Hardware** → *Software tools for EDA*; • **Computer systems organization** → *Embedded systems.*

KEYWORDS

Edge Machine Learning, Edge Computing

ACM Reference Format:
Oliver Bringmann, Wolfgang Ecker, Ingo Feldner, Adrian Frischknecht, Christoph Gerum, Timo Hämäläinen, Muhammad Abdullah Hanif, Michael J. Klaiber, Daniel Mueller-Gritschneder, Paul Palomero Bernardo, Sebastian Prebeck, and Muhammad Shafique. 2021. Automated HW/SW Co-design for Edge AI: State, Challenges and Steps Ahead: Special Session Paper. In *2021 International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '21), October 10–13, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3478684.3479261

1 INTRODUCTION

As we are moving more and more into the era of Big Data, Internet of Thing (IoT), and Smart Cyber Physical Systems (CPS), rising rates of data production pose incessantly escalating demands for

massive data processing, storage, and transmission while stronger automation and autonomy requires systems to continuously interact with the physical world using edge sensors and actuators. The first generation of intelligent systems stream these large data sets into the cloud. For upcoming IoT systems, there is now a strong trend to move the intelligence from the cloud to the edge or the extreme edge, also referred to as TinyML. Such Edge AI systems can provide better privacy, higher autonomy, lower response latency, less cost, higher energy efficiency as well as lower bandwidth demand to the cloud.

Yet, the design of powerful edge AI systems is a challenging task as computing and memory resources are highly limited at the edge. Hence, designers need to optimize the complete system stack from machine learning algorithm, to model optimization and compression, memory planning, software implementation of kernels onto the hardware platforms with ML accelerators as well as the underlying hardware architecture design [14, 17, 21, 23, 48, 52]. We refer to this as HW/SW Co-design for Edge AI. The target of this paper is to discuss the current state, challenges in this field and propose the next steps to realize the next-generation Edge AI systems.

For that, the paper firstly provides an overview of related work in Section 2. Then, in Section 3 we discuss the current open research challenges in the field from the academic and industrial perspective. We outline the potential of current Open Source solutions for realizing a holistic HW/SW Co-design approach and outline the different steps required to achieve this goal. The paper then highlights the current state with three currently developed end-to-end flows:
In Section 4 we show that the selection of RISC-V processor extensions for tightly-coupled accelerator solution is a challenging process. The goal is to move the intersection between computational and communication bound as close as possible to the operational intensities for the concrete NN kernel. The presented, auto-generated tightly-coupled RISC-V solution enables the specific selection and combination of further configurable extensions for optimization purposes. The results of the early validation phase of this approach show a performance gain of $\sim 23x$.
In Section 5 we show how automatic generation of application-specific loosely-coupled AI accelerators can significantly increase design efficiency. To maximize performance, a joint co-optimization of both hardware and software is necessary. This can more than halve the power consumption at similar performance compared to hand-crafted approaches. An ongoing challenge for creating a true end-to-end design flow from AI use case to optimized accelerator is closing the gap between operator-level intermediate representations of the workload and a corresponding hardware implementation. The section lays out how this can be achieved using the machine learning compiler framework TVM.
Finally, in Section 6, we present a framework that systematically integrates software and hardware optimization techniques to build efficient Deep Neural Network (DNN) systems [16][30]. At the software level, hardware-aware optimization of DNNs is important, while at the hardware level, using optimized datapaths is the key to efficient DNN inference. Dataflow optimization is also essential to reduce the costly off-chip memory accesses. Each optimization technique has the potential to reduce the energy consumption of DNN execution on edge devices. However, systematic integration of these techniques uncovers the real benefits.

2 RELATED WORK

There are multiple fundamentally different approaches for **hardware platforms** accelerating the inference of NNs. Some are based on systolic arrays [4] or enhance them by a reconfigurable architecture [57]. S2TA [29] is a method exploiting structured sparsity on systolic arrays. Wang et al. [50] showed an FPGA-based solution, similar to Sankaradas et al. [42] for CNNs. [50] made it applicable also to hybrid-NNs. Within the group of ASIC accelerators, it can be differentiated between processing-element based (PE-based), tightly-coupled and coprocessor-based. PE-based accelerators are designed for highly parallelized computation. Some were especially designed to calculate data in compressed format [8, 13, 36, 60] to gain efficient processing and power efficiency. Tightly-coupled ones are typically based on a light-weight CPU, preferred with an open-source ISA like RISC-V [2], which can be extended by additional instructions [37]. [11, 12] solve the bandwidth bottleneck of such systems by custom instructions loading data in incremental fashion while computing payload. Sub-byte custom instructions and tightly-coupled multicore clusters are further methods. Zhang et al.[61] presented a coprocessor extending the open-source Hummingbird E203 processor.

Several proposals exist for **design tools** of such hardware platforms. Timeloop [35] offers architecture exploration by automatically mapping the machine learning model to candidate HW architectures. It supports DNNs, fully connected layers and recurrent neural networks (RNNs). Another example is Gemmini, which is part of the chipyard [5] framework. It offers memory-mapped peripheral accelerators and Rocket Custom Co-processors (RoCC) based on RISC-V. Gemmini includes a matrix multiplication accelerator generator for systolic arrays. MAGNet [49] and AutoDNNchip [56] are examples for template-based generation of loosely-coupled ASIC accelerators. Given a DNN model and design budget they automatically co-optimize the operator mapping and underlying hardware architecture to create application-specific accelerator instances. Yu et al. [59] approaches the architectural design space exploration with a framework based on a hardware analytical model of individual DNN operations reducing the search to a multi-dimensional optimization problem. EasyQuant [54] is a post-training quantization (PTQ) technique utilizing scale optimization. Furthermore, Shomron et al [43] applies a sparsity-aware quantization (SPARQ) method.

In recent years, the **hardware-aware design of neural networks** has received increasing attention. In addition to techniques for model compression such as quantization-aware training (QAT) and pruning [14, 17, 21, 48, 52], hardware-aware neural architecture search (NAS), which also takes hardware characteristics like latency, power, or area into account, has become a central aspect in automating the process of designing new NN architectures for Edge AI. In [20] and [3] reinforcement learning based NAS is extended to include search for an accelerator configuration on FPGAs and optimize it for latency and area. [58] presents a similar approach for heterogeneous ASIC accelerators. Additionally, there exist academic approaches that address limited resources of edge platforms

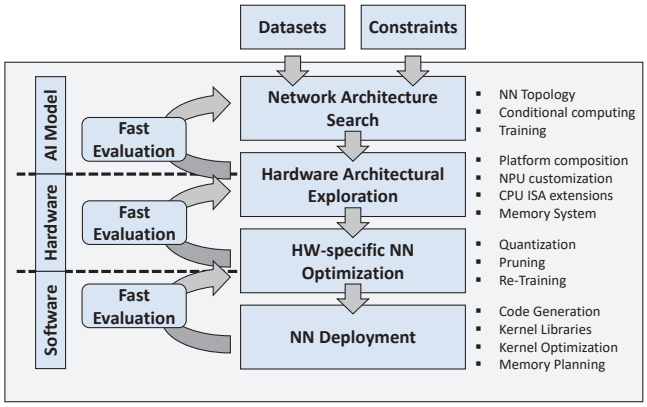

Figure 1: Generic Design Space Exploration Flow for HW/SW Codesign of Edge AI Systems

by pooling memory resources from several edge devices together and optimizing DNNs for distributed inference [41, 45].

3 RESEARCH AND INDUSTRIAL CHALLENGES IN EDGE AI

From an industrial perspective one of the main challenges to develop complex Edge AI systems is bridging the gaps between hardware (HW), software (SW) and AI engineers to define a common workflow for a joint Design Space Exploration (DSE). The diagram in Figure 1 shows such an envisioned DSE flow and the links between the three domains. It becomes apparently clear that the development of complex Edge AI systems requires the skills of software, hardware, and AI engineers. In most industrial projects, however, these skills are distributed among several people or even different teams.

Even-though effective HW/SW Co-design methodologies have been examined for decades, they are still not the norm in the development of large-scale (embedded) systems [27, 44]. As Edge AI systems are a relatively new device class, the same holds true for them. A major issue here is that the requirements between HW, SW, and AI domains cannot be easily exchanged. A result is that hardware is designed with assumptions about an algorithmic workload that might already be outdated and that the AI model is designed without an understanding of what operations provide good performance on the target hardware. This holds especially true in the field of embedded systems (which include Edge AI systems), as these systems often have limited computing and communication resources and, hence, use heterogeneous, application-specific platforms.

As of today, there is no established end-to-end development methodology for Edge AI systems [23]. The current fragmentation can lead to local optimization in the aforementioned domains, e.g. :

- optimization of AI model accuracy via Neural Architecture Search (NAS)[9],
- optimization of the hardware resources via Hardware Architecture Search (HAS) methods [19],
- optimization of the implementation of HW-specific AI kernel libraries (e.g. a Conv2D layer for a specific neural network accelerator) via auto-tuning methods[7]

A long-term challenge is to combine all of these optimization problems into a single holistic one, as envisioned by the flow in Figure 1. This may inherently change the task of HW/SW/AI designers. Their competence then needs to shift from implementing a single design point to defining search constraints for a global HW/SW/AI DSE flow.

Formulating and defining such constraints is the first step on the way to holistic optimization. As a means of realizing a flow as shown in Figure 1, compiler technology was identified to be the major asset to be able to achieve this goal [26]. All of the steps in Figure 1, from neural architecture search (NAS) to neural network deployment to hardware description can be seen as compiler passes. As of today, the quality and maturity of such compiler passes for industrial applications still varies. In neural network training TensorFlow as well as PyTorch are quite established, whereas passes for neural architecture search (e.g. DARTS [28]), tools for neural network deployment (e.g. ApacheTVM [7]) and tools for hardware generation (e.g. CIRCT [1, 62]) are in early industrialization phases or in the research phase. However, these approaches already show that the developed compiler technology is going to provide an end-to-end Edge AI flow soon and possibly even in the open-source domain.

In detail, the design constraints for such a holistic optimization are discussed in [47] for accuracy, throughput, latency, energy efficiency, power consumption, flexibility, and scaleability in seven design steps. In Edge AI chips, the most expensive resources are usually on-chip memory and high-speed IO pins. Therefore, the maximum achievable parallelism must first be studied at all levels between the application data and the computing micro-architecture, while exploring the alternatives. The challenge is the very large space when all the factors are optimized at the same time. Additional challenges arise from the heterogeneity of the platforms. Complex Neural Networks may be deployed partly on specialized accelerators, partly on the embedded processors, possibly with AI-specific ISA extensions. Such mixed-deployment scenarios require research into compiler passes that can explore different mappings as well as consider communication overheads. Additionally, memory planning methods are required that consider the memory resources of all available compute units.

Further major burden is the AI-compile flow for the EDGE AI accelerator. It begins with training, since EDGE devices capture data with non-ideal sensors, each with a slightly different behavior. Ideally, training data is collected in parallel with different sensor variants. To further improve, a data engineering/processing is established to equalize the data variation. Next, network definition and network optimization (as weight scaling) should consider the execution time on the AI accelerator and the precision of the AI accelerator in a neat fashion, respectively. Ultimately, tensor level optimization and adoption of embedded code generation should be adopted to the EDGE AI accelerator in an clean way.

Finally, in some applications, it is not only required to run inference on the edge device but also training [10], which may require additional training accelerators. Here, especially federated learning is in the focus [55], in which training data from several edge devices can be aggregated into a global training process while being able to maintain the privacy of the training data, as only model updates and not the data itself are sent from the edge to the cloud.

4 AUTOMATIC BUILD OF A RISC-V SYSTEM WITH TIGHTLY-COUPLED ACCELERATOR

AI applications require the processing of neural network (NN) inferences on proper platforms. Optimizing throughput, hardware utilization as well as memory requirements and power consumption are key to an efficient and fast setup. Processing is generally bound to system's capabilities in both communication and computation. Thus, a methodology for adjusting those design aspects for a certain application is required in order to reach high utilization. Since the kernel properties have a strong influence on the balance of the two major throughput bounds, it also plays a major role in designing an optimized system.

Williams et al. [53] showed along their roofline model, that the overall processor performance for a certain kernel can be estimated according to (1), using the minimum of both computational and communicational bound.

$$\begin{aligned} Attainable\ GFlops/sec = Min(&Peak\ Floating\ Point\ Perf,\\ &Peak\ Memory\ Bandwidth\ *\ Operational\ Intensity) \end{aligned} \tag{1}$$

The roofline model hereby focuses on floating point operations at computation side and off-chip memory accesses from communication perspective, where operational intensity describes the kernel specific Flops/byte. The benefits of the model are in its visualization of throughput limitations over various kernels and unambiguous bottleneck analysis. Since the roofline model doesn't consider memory footprint, area, latency and power, those parameters have to be kept in mind additionally, while taking optimization decisions.

4.1 RISC-V tightly-coupled approach

We focus on a tightly-coupled accelerator solution, back-boned by RISC-V. On one hand with its base instruction set the RISC-V serves as a baseline in order to compare bandwidth and computation limitations against extended/accelerated solutions, on the other hand it represents a fallback solution for untreated corner cases. Due to RISC-V's open instruction set architecture the integration of additional custom extensions is simple and goes hand in hand with high configurability. The accelerator is targeting low power, low cost applications, which either are too heavy to fulfill the latency requirements on a micro controller or are not energy efficient enough, especially when it comes to advanced data formats.

Sze et al. [46] rates a DRAM read to be more than 100x more expensive than on SRAM. For the targeted low power accelerator, this infers strict avoidance of off-chip memory accesses by reducing model size, instead enabling the storage of model data on-chip. Thus, the discussed extensions are developed at the background of supporting reduced model footprints. In addition, reducing the model size to an extent of manageable loss in precision leads to several other desirable effects. A smaller memory footprint reduces the static memory power consumption and chip area. Since the CPU core and its extensions are supposed to be small in terms of area, memory is the major driver for chip area. In addition, a reduced model footprint leads to a decreased number of required accesses causing a lower dynamic memory power consumption.

Since different kernels constitute various operational intensities, providing optimized hardware support for a certain application is challenging. We address this issue by offering a highly extendible and configurable system, giving the freedom to optimize the SoC for a concrete kernel by interchanging extension modules and thus moving computational and communicational bounds closer to the theoretical maximum.

4.2 Modified roofline model

Figure 2 shows the modified roofline model for the tightly-coupled RISC-V accelerator. Instead of FLOPS the axis refer to MACS. Efficient low-weight NN inference models use quantized fixed point integer instead of floating point [25], since they allow smaller model footprints at acceptable accuracy loss. Furthermore, the examined system doesn't provide a floating point extension, for area and power saving reasons. Therefore, we refer to MACS as a more feasible metric.

In extent to Williams et al. [53] roofline model we introduce the terminology of a virtual bandwidth. Data of optimized models are typically not stored in a ready-to-process fashion in memory and alignment restrictions are softened in favor of a minimized memory footprint. Thus, a data preprocessing step preceding the actual payload computation is mandatory. This preprocessing step has major influence on the system throughput by reducing the communicational bound. While considering the physical bandwidth bound being the absolute maximum a memory/cache can deliver, the virtual bound is defined by the amount of data delivered ready-to-process to the computation unit. The virtual bound will always be below or equal to the physical bound.

4.3 Discussion of the extended model

In the following we discuss the optimization for two different kernels along with the modified roofline model in Figure 2. The first makes use of a compression technique, namely sparsity. In this technique dropping of zero weights is considered, but the results can be extrapolated for other compression algorithms like Huffman coding. The second one focuses on a kernel using packed data. This feature describes the common storage of multiple datasets within the same memory location. One can imagine a packing of 4/8/16/32 data sets per word with respectively 8/4/2/1 bits in width.

Equation 2 gives the formula for the achievable memory footprint for different compression ratios and packing formats. r_p and r_s define the packing ratio and sparsity ratio respectively, whereas c_{se} considers the cost introduce by sparsity encoding.

$$Footprint = \#Datasets * (1/r_p * (1 - r_s) + c_{es}) \tag{2}$$

$$r_p = \#Packed\ Datasets/Byte$$
$$r_s = \#Zero\ Datasets/\#Datasets$$
$$c_{se} = Sparsity\ Encoding\ Cost/Dataset$$

4.3.1 Raising bounds. For both computational as well as communicational bounds, we elaborated on three levels of implementation. The baseline is embodied by a plain RISC-V 32bit base instruction set CPU, in Figure 2 referred to as *rv32i* together with its processor dedicated load store unit. The load store unit is the limiting factor at the communication side, whereas the instruction set defines the capabilities of the compute units, thus bounds computation.

In order to increase the computational bound, two further features namely *rv32im* comprising an integer multiplier and *rv32im + simd mac* are shown in Figure 2. Since for typical NN inference

kernels multiplication and addition are heavily utilized, we decided to evaluate the explicit multiplication support and further make use of a custom extension supporting single instruction multiple data (SIMD) format in a dedicated integer vector multiply-accumulate unit. Those two features define the medium and maximum bound in computational perspective.

With respect to the communication bound two further extensions are added. The un/repacked extension supports the system by preparing data for kernels making use of the packed data feature. It especially offloads the address computation, data fetch and repacking for alignment reasons. The sparsity extension skips activations with a corresponding zero weight from computation or inserts alignment zero weights skipped in memory according to a sparsity vector. The kernel features influence the computation intensity, whereas the mentioned extensions optimize the bounds, according to the concrete kernel properties. Optimization goal is to move the intersection between computational and virtual bandwidth bound close to the kernel computation intensity. In the desired operation point utilization of both aspects is high and throughput as well as latency is optimized.

4.3.2 Compression feature. Figure 2 shows decreasing computational intensity for the sparse kernel compared to the original kernel. The reason is the additional storage of the sparsity information in a dedicated sparsity vector in memory. Additionally, zero weights are dropped out of the memory. This overall heavily decreases the memory footprint. Nevertheless, the computational intensity does not get affected by that. The amount of MACs reduces proportionally to the sparsity ratio, due to zero skipping in multiplications. Without further analysis, a positive influence on memory size and latency can be stated.

Since a sparse kernel requires additional preprocessing steps (blue/white dots), the virtual bandwidth bound decreases in comparison to the non-sparse case (green squares), since less ready-to-process data gets delivered to the computation unit in the same duration. On the other hand, a dedicated extension (red/white dots), even increases the virtual bound, since it prepares data more efficiently than CPU does.

Given these bounds, we discuss the consequences for a sparse kernel in comparison to its original. We find the original kernel to be computationally limited by *rv32i*. Extending to *rv32im* leads to minor throughput gain, since the bandwidth bound is close. Moving to the sparse kernel the throughput reduces, limited by virtual bandwidth due to more expensive data-preparation at first. Nevertheless, adding the sparsity extension lifts the bound to computation. By enabling the *rv32im* extension the compute and bandwidth throughput evolves close to equality, thus the concrete kernel becomes an almost ideal one. It is important to mention, for different sparsity ratios the computational intensities deviate. Thus, this trade-off analysis may result in different conclusions.

4.3.3 Packing feature. In contrast to sparsity, Figure 2 shows an increased computational intensity of the packed kernel compared to the original one. The exact deviation depends on the concrete packing format. Without further discussion, it can be stated that packing reduces kernels memory footprint. By packing, the number of datasets stays untouched, while multiple are stored into one common memory location.

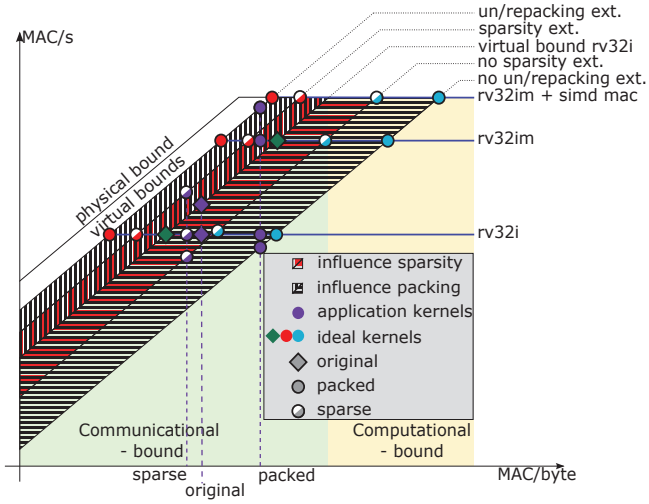

Figure 2: Extended roofline model

Using packing doesn't necessarily lift the bandwidth bounds. Theoretically, the memory can deliver more datasets, but the unpacking or repacking process is complex, especially with odd but efficient packing formats. Thus the virtual bandwidth bound even drops (blue dots), since data preprocessing becomes the bottleneck. Instead, a dedicated extension taking over un/repacking tasks resolves the issue and drives the virtual bandwidth limitations closer to the physical bound (red dots). Due to packing, payload fetching inevitably leads to accesses of unutilized data as side effect, especially when combined with sparsity.

Especially when it comes to SIMD processing units, the packing extension plays a crucial role, since it allows data to be prepared in a proper ready-to-process SIMD format in a direct fashion. Similar to sparse kernel support, the packing extension helps to optimize the throughput bounds for a concrete kernel, by simultaneously reducing latency, memory footprint and system power consumption.

4.4 Evaluation

We did an early validation of the proposed work-in-progress flow for an audio localization application based on a sparse packed kernel with 70% weight sparsity and a packing format of four 8-bit datasets per addressable memory location. Both sparsity as well as packing extension were enabled. The computation performance level was set to *rv32im + simd mac* with support of four parallel MAC operations matching the packing format.

The packing reduced the model footprint by ~75% in comparison to the original 32bit data format, whereas weight sparsity led to a weight footprint reduction of further ~60%, In contrast the *rv32im* support added ~30% area overhead on top of the baseline and *rv32im + simd mac* further ~20%. The extension for sparsity and packing support increased the area by another ~70%. This setup achieved a latency reduction of ~23*x* in comparison to a non-optimized *rv32im*.

5 AUTOMATIC GENERATION OF APPLICATION-SPECIFIC AI ACCELERATORS FOR EDGE DEVICES

Deep neural network accelerators help bring intelligent data processing to edge devices. To meet the strict performance and power requirements, many accelerators use hand-crafted architectures tailored to specific use cases. Designing and extending such architectures is time-consuming and requires strong domain knowledge. Generator-based approaches that automate the development of loosely-coupled AI accelerators can help increase design efficiency. To fully leverage their potential, end-to-end solutions from hardware-aware DNN training and deployment down to the actual hardware synthesis are needed.

We present HANNAH (Hardware Accelerator and Neural Network seArcH) to automatically co-optimize neural network architectures and a corresponding neural network accelerator (Section 5.1) and outline our current efforts of integrating TVM into the existing deployment and hardware generation flow (Section 5.2). The potential of joint co-optimization of hardware and software is highlighted on an audio use case (Section 5.3).

5.1 HANNAH Framework

The HANNAH design flow is shown in Figure 3. Neural networks are instantiated and trained in the training component employing quantization aware training (QAT) and dataset augmentation. Trained neural networks are then handed over to the deployment component for target code generation. Here, the neural network is quantized to a low word width representation, the neural network operations are scheduled, and on-device memory is allocated for the neural network. Along with the target network architecture, a specialized hardware accelerator for the neural network processing is instantiated from a configurable Verilog template. Hardware dependent performance metrics like power consumption and chip area are then either generated by running the neural network on a gate-level simulation or estimated using an analytical performance model. The evolution-based search strategy implemented in the HANNAH optimizer is then used to incrementally search the neural network and hardware accelerator co-design space.

5.1.1 HANNAH - Train. A hardware-aware neural network design and training process is key for an efficient DNN execution on edge devices. HANNAH addresses this by providing a constrainable neural network design space and a QAT flow.

Neural networks within the design space are build from configurable building blocks. These blocks contain common DNN operations (e.g., convolution, activation, pooling) and topology information that defines the order and structure of the operations. By constraining the operands and topology of each block, it is possible to ensure that neural networks of a particular archetype are generated that are supported by the target hardware.

Deploying the neural networks on efficient integer hardware involves quantization of the floating-point data used during training. To account for the quantization effects, quantization-aware training is used to simulate a symmetric fixed-point quantization of weights, biases, and features during the forward pass of the training.

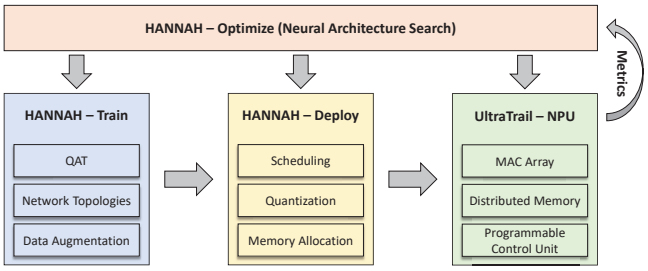

Figure 3: Overview of the HANNAH framework.

Finally, datasets and data augmentation such as the addition of noise is also handled by the training component.

5.1.2 HANNAH - Deploy. To execute a trained neural network, a target-specific mapping onto the hardware is necessary. This includes quantization, operator scheduling, memory allocation and layout, as well as the generation of corresponding configuration files. HANNAH supports two separate deployment options. The machine learning compiler framework TVM and a custom deployment backend optimized for the neural processing unit (NPU) described in the following section. For the hardware accelerator search the use of the custom backend is mandatory. In addition to the target-specific mapping, the custom backend also handles the parameterization and validation of the NPU architecture. Given the network model, quantization intent, and desired number of functional units it derives fitting memory macros and scales port widths and the control unit appropriately. If desired, a simulation, synthesis, and power estimation are run automatically. The simulation results are compared with the reference output of the training component to validate the functional correctness of the NPU.

5.1.3 UltraTrail - NPU. The used NPU architecture is based on UltraTrail [6], a configurable accelerator designed for the TC-ResNet. It consists of an array of multiply-and-accumulate (MAC) units for convolutional and fully-connected layers, an output processing unit for post-processing operations such as activation functions or pooling, a distributed memory structure, and a programmable control unit. The baseline architecture has been heavily parameterized (e.g., arbitrary word width, MAC array dimension, memory sizes) which opens up a large NPU design space.

To avoid time-consuming simulation and synthesis of the hardware when exploring this design space, analytical models are provided. This includes a cycle-accurate performance model, as well as models for power and area.

5.1.4 HANNAH - Optimize. The search for a target neural network and accelerator architecture configuration is implemented as an evolution-based multi-objective optimization. The joint search space combines the neural network and NPU design space. It can be further restricted by reducing the number of certain design choices. During the iterative search process the error rate of the trained neural network and estimated power, latency, and area of the hardware models are used as target metrics for the optimization function.

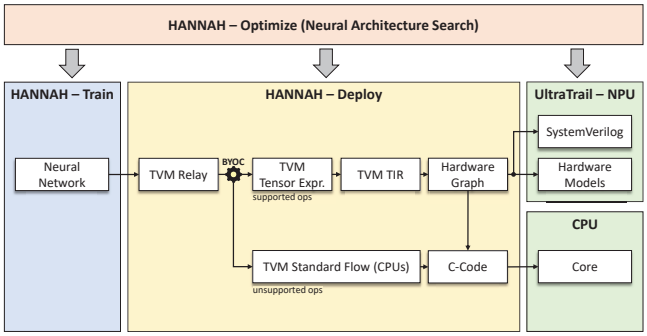

Figure 4: TVM-based hardware generation flow.

5.2 TVM-based Hardware Generation

Although highly parameterizable, the core architecture of the NPU and in particular the control and data flow is based on a hand-crafted design process. While this is sufficient for single application-specific scenarios, extending the architecture is a time consuming procedure that requires expert knowledge which makes a fast adoption to different application areas difficult. The same applies to the hardware models and custom deployment backend, due to their direct dependence on the underlying hardware. With TVM already integrated into HANNAH, the obvious step would be to streamline the deployment flow by moving the NPU-specific deployment to TVM. However, without a mechanism to automate the generation of a specific NPU architecture from the operator-level intermediate representation (IR) of a neural network used in TVM, manual design of custom backends and accelerators would still be required.

To close the gap between the operator-level IR and a corresponding hardware implementation we propose a combined TVM-based hardware generation flow. This includes (1) an IR hardware synthesis, (2) abstract hardware modeling, and (3) automatic deployment. Combining these three aspects comes with various advantages. First, changes to the framework such as updates or extensions are represented in all components, which significantly reduces the implementation overhead. Second, adding new features is a one-time effort. Once implemented within the generator framework, design changes like the dataflow or memory management are handled on an algorithmic level within TVM and must no longer be manually crafted for each accelerator instance. Finally, it enables a true end-to-end compilation flow from neural network down to the actual hardware. This allows rapid joint prototyping of both hardware and software, which in turn can be used for automatic design space exploration in HANNAH. The greater flexibility on the hardware target also opens up the possibility to extend the search from application-specific to domain-specific accelerators with an optimized support for a set of application areas.

Another advantage that comes from using TVM is the rich and growing feature set available in the framework. For example, support for mixed-deployment allows operations not supported by the NPU to be added to the NAS.

Figure 4 shows an abstract view on the TVM-based hardware generation flow. A trained neural network gets passed from the training to the deployment stage, where it is transformed to RelayIR. Using the Bring Your Own Codegen (BYOC) framework, the graph is partitioned into NPU supported and unsupported operations. For the supported operations, custom tensor expressions are defined. Scheduling and memory layout transformations are performed in this stage. The lowered TensorIR (TIR) is analyzed and from it a hardware graph, i.e., an abstract hardware representation, is build. The hardware graph forms the basis for deployment, hardware modeling, and hardware synthesis. Deployment is realized through C-Code emission using BYOC. Hardware synthesis uses a combination of parameterizing generalized hardware templates and actual RTL-Code generation. Finally, TVM is used to execute the program on the target system.

Table 1: Comparison of Results on Keyword Spotting Task

	UltraTrail [6]	HANNAH HA	HANNAH LP
Area	0.20 mm^2 [a]	0.13 mm^2 [b]	0.09 mm^2 [b]
Word Width (W)	6	6	6
Word Width (F)	8	6	6
MAC Array Size	8×8	8×8	6×6
Accuracy	93.09 %	94.33 %	93.37 %
Power	8.2 µW	6.38 µW	3.79 µW

[a] Area for layout/chip [b] Cell area for synthesis

5.3 Case Study

We evaluate the impact of a joint hardware/software co-optimization using the HANNAH flow of Section 5.1 on the task of real-time keyword spotting. We use the Google Speech Commands Dataset V2 (GSDC) [51] with 10 keyword classes and 1 class each for "silence" and "unknown". The real-time constraint is set to 10 inferences per second. Soft constraints for area, power, and accuracy are set to 150 000 μm^2, 5 µW, and 93 %, respectively. Candidates that violated them are penalized by being ranked worse than those that meet the soft constraints. The search space contains a wide variety of neural network architectures, different quantization word widths for weights (W) and features (F), and different NPU MAC-array sizes. The search ran with a search budget of 2000 individual architectures and used a population size of 100.

The NPUs are implemented using the 22FDX technology by Globalfoundries with low-leakage standard cells and SRAMs by Invecas. Synthesis and power estimation was done with Cadence Genus 20.10 and Synopsys PrimeTime Q-2019.12, respectively. All accelerator instances are set to operate at 250 kHz and are evaluated at a 25 °C TT corner with 0.8 V supply voltage and no body bias voltage. The NPU uses clock-gating and low-power modes of the SRAM during idle times.

Table 1 shows a comparison of the hand-crafted baseline architecture UltraTrail and the two Pareto points with highest accuracy (HA) and lowest power (LP) selected from the NAS results. Both co-optimized instances of neural network and NPU reach better results for area, power, and accuracy compared to the baseline architecture. HA achieves an accuracy increase of 1.24 p.p. while reducing the total power by 22.2 %. Using a reduced number of processing elements, LP reaches a 53.8 % reduction in power with a slight 0.28 p.p. increase in accuracy.

6 ENERGY-EFFICIENT EDGE AI: FROM ALGORITHMS TO HARDWARE ARCHITECTURES

Modern systems perform most of the data processing close to the sensors at the edge due to privacy and latency concerns. However, edge devices are highly resource-constrained and cannot run complex DNNs, specifically for long durations. Designing a highly efficient yet effective DNN system is challenging and requires systematic integration of hardware and software optimization techniques. We present a cross-layer framework for generating optimized DNNs and optimizing the deployment of the DNN onto a given hardware platform [16][30] (Section 6.1). After presenting the framework, we will highlight the effectiveness of different parts of the framework with the help of practical case studies (Section 6.2).

6.1 Cross-Layer Optimization Framework

Our cross-layer EdgeAI design and optimization framework is illustrated in Figure 5. It considers user requirements such as accuracy and performance along with hardware constraints to explore the architecture space and to generate an appropriate / Pareto-optimal DNN model(s). The DNN is then further optimized by exploiting pruning and quantization techniques. The optimized model is then processed for error-resilience analysis that guides the hardware approximation phase to achieve further energy efficiency. Finally, an effective dataflow scheme and layer partitioning and scheduling are devised / employed to optimize the off-chip memory accesses. The details of different steps of our framework are explained in the following subsections.

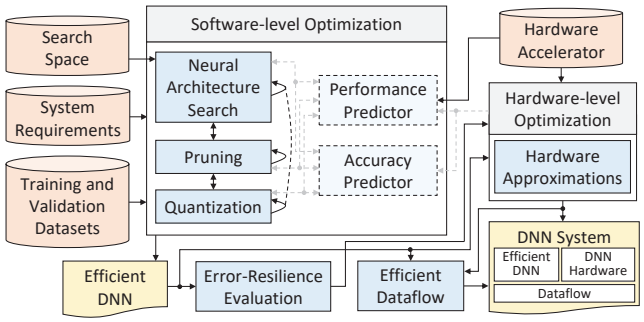

Figure 5: Overview of our cross-layer EdgeAI framework.

6.1.1 Software-level optimization. Hardware-aware neural architecture design is the key to efficient data processing at the edge. Therefore, based on the user requirements and hardware constraints, a restricted search space is first defined. Then using a multi-objective optimization algorithm that jointly considers validation accuracy and performance metrics, the overall design space is explored to generate a suitable DNN. For fast exploration, accuracy and performance estimation models are constructed and used to characterize candidate models that guide the optimization algorithm. Our results in [31] and [38] show that different Pareto-optimal models can be generated using such a methodology based on user-defined constraints/requirements.

The generated DNN is then passed for pruning and quantization to further improve the efficiency of DNN execution on edge devices. Network pruning is usually performed iteratively, where, in each iteration, a small percentage of network parameters are removed and the network is retrained for a limited number of epochs to regain the lost accuracy. The pruning policy is selected based on the target hardware and the user constraints to remove the correct set of parameters that offer maximum efficiency gain without affecting the accuracy. After pruning, an effective quantization policy is selected to enable low-precision computations and further reduction in the network size. In [38], we show that model compression through pruning and quantization enables a further reduction in storage requirement by up to 53× for an accuracy loss of less than 0.2%. Note that pruning and quantization can be closely integrated with the NAS algorithm to significantly reduce the design cost.

6.1.2 Hardware-level optimization. Specialized hardware accelerators are typically used for efficient DNN execution on edge devices. These accelerators can be further optimized by exploiting the error-resilience characteristics of DNNs. Towards this, different types of approximations can be employed at the hardware level, e.g., memory approximation by reducing the supply voltage below the standard specification [24][22] or functional approximation of the computational units [34]. These approximations are usually effective for less complex applications or require extensive retraining of DNNs. Therefore, we focus on data-driven approximations such as [18] or approximations with additional error compensation [15][33] to avoid any accuracy loss.

6.1.3 Dataflow optimization. The energy consumption of off-chip memory accesses is orders of magnitude higher than other operations involved in DNN execution. Dataflow and layer partitioning and scheduling define the number of off-chip memory accesses required per inference. Therefore, it is important to optimize them to suppress the dominating factor in the system's energy consumption. To achieve this, we study the reuse factor of all the datatypes (i.e., weights, activations, and partial sums) of each layer of the given DNN, and based on the DNN accelerator design, choose a suitable adaptive layer partitioning and scheduling scheme to minimize the memory accesses. Our results in [40] show that, with the selection of an appropriate layer partitioning and scheduling scheme, it is possible to reduce the number of off-chip memory accesses for the VGG-16 by 36% and for the MobileNet by 45% over conventional schemes. Apart form layer partitioning and scheduling, application-specific memory designs can also be developed by exploiting application characteristics and techniques like power-gating to further boost the efficiency gains [32]. Moreover, selection of an appropriate data mapping policy that orderly prioritizes to maximize the row buffer hits, bank- and subarray-level parallelism have also shown to be highly effective in reducing the energy consumption associated with DRAM accesses [39].

6.2 Case Studies

6.2.1 Designing efficient DNNs for wearable healthcare devices. Based on the methodology presented in Section 6.1, we developed BioNetExplorer framework [38] to systematically generate

and optimize DNN architectures for bio-signal processing in wearable edge devices. The overview of the framework is shown in Figure 6. It deploys genetic algorithms to search for embedded DNN architectures with low hardware overhead to ensure their deployment in wearables devices to extract health information like the occurrence of arrhythmia or seizures. The framework enables hardware-aware DNN architecture search by considering hardware constraints (i.e., storage and FLOPs) during the exploration stage. The genetic algorithm-based search reduces the exploration time by 9x (on average) compared to exhaustive search and identifies Pareto-optimal designs that can reduce the storage requirement by 30MB for an accuracy loss of less than 0.5%. The model compression stage enables a further reduction in storage requirement by up to 53× for an accuracy loss of less than 0.2%.

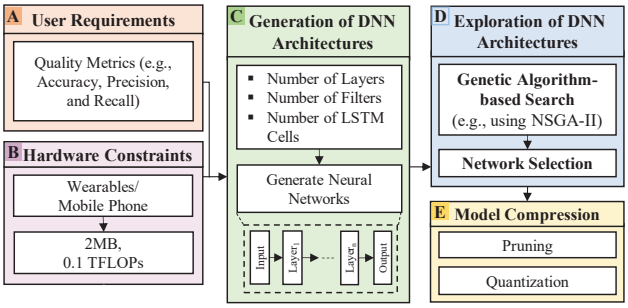

Figure 6: Overview of the BioNetExplorer framework [38].

6.2.2 Optimizing off-chip memory accesses for image classification application. To evaluate the impact of selecting appropriate dataflow and layer partitioning and scheduling on the energy efficiency and the number of off-chip memory accesses of a DNN system, we employed our ROMANET methodology [40] to identify the dataflow and computation schedule for image classification using the MobileNet and the VGG-16 networks. The evaluation shows that selection of effective dataflow and layer partitioning and scheduling leads to 36% reduction in the number of off-chip memory accesses for the VGG-16 and 45% for the MobileNet over conventional schemes, which translates to around the same amount of access energy savings.

7 CONCLUSIONS

This paper outlined the state, challenges and step aheads, which needs to be addressed within a holistic DSE process to enable an automated HW/SW Co-design for Edge AI systems. Three currently developed flows were presented in an exemplary fashion to illustrate the current state, but many more such flows or individual tools are under development by various groups world-wide.

REFERENCES

[1] 2021. CIRCT: Circuit IR Compilers and Tools. https://github.com/llvm/circt

[2] 2021. RISC-V. https://riscv.org/

[3] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. 2020. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[4] Trio Adiono, Grasia Meliolla, Erwin Setiawan, and Suksmandhira Harimurti. 2018. Design of Neural Network Architecture using Systolic Array Implemented in Verilog Code. In *2018 International Symposium on Electronics and Smart Devices (ISESD)*. 1–4. https://doi.org/10.1109/ISESD.2018.8605478

[5] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Jonathan Bachrach, Sophia Shao, Borivoje Nikolić, and Krste Asanović. 2020. Invited: Chipyard - An Integrated SoC Research and Implementation Environment. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218756

[6] Paul Palomero Bernardo, Christoph Gerum, Adrian Frischknecht, Konstantin Lübeck, and Oliver Bringmann. 2020. Ultratrail: A configurable ultralow-power tc-resnet ai accelerator for efficient keyword spotting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4240–4251.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 578–594.

[8] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[10] Frederik Funk, Thorsten Bucksch, and Daniel Mueller-Gritschneder. 2020. ML Training on a Tiny Microcontroller for a Self-adaptive Neural Network-Based DC Motor Speed Controller. In *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*.

[11] Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. 2021. XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V based IoT End Nodes. *IEEE Transactions on Emerging Topics in Computing* (2021), 1–1. https://doi.org/10.1109/TETC.2021.3072337

[12] Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gürkaynak, and Luca Benini. 2017. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2700–2713. https://doi.org/10.1109/TVLSI.2017.2654506

[13] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 243–254. https://doi.org/10.1109/ISCA.2016.30

[14] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[15] Muhammad Abdullah Hanif, Faiq Khalid, and Muhammad Shafique. 2019. CANN: Curable approximations for high-performance deep neural network accelerators. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[16] Muhammad Abdullah Hanif and Muhammad Shafique. 2021. A cross-layer approach towards developing efficient embedded deep learning systems. *Microprocessors and Microsystems* (2021), 103609.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[18] Shubham Jain, Swagath Venkataramani, Vijayalakshmi Srinivasan, Jungwook Choi, Kailash Gopalakrishnan, and Leland Chang. 2019. BiScaled-DNN: Quantizing long-tailed datastructures with two scale factors for deep neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[19] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4154–4165.

[20] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4805–4815.

[21] Qing Jin, Linjie Yang, and Zhenyu Liao. 2019. Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207* (2019).

[22] Sung Kim, Patrick Howe, Thierry Moreau, Armin Alaghi, Luis Ceze, and Visvesh Sathe. 2018. MATIC: Learning around errors for efficient low-voltage neural network accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[23] Michael J. Klaiber, Axel J. Acosta, Ingo Feldner, and Falk Rehm. 2021. Enabling Cross-Domain Communication: How to Bridge the Gap between AI and HW Engineers. *CoRR* abs/2104.03780 (2021). arXiv:2104.03780 https://arxiv.org/abs/2104.03780

[24] Skanda Koppula, Lois Orosa, A Giray Yağlıkçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. 2019. EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 166–181.

[25] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv* abs/1806.08342 (2018).

[26] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2020. MLIR: A compiler infrastructure for the end of Moore's law. *arXiv preprint arXiv:2002.11054* (2020).

[27] Glaydson Luiz Bertoze Lima, Guilherme Augusto Lopes Ferreira, Osamu Saotome, Adilson Marques Da Cunha, and Luiz Alberto Vieira Dias. 2015. Hardware development: Agile and co-design. In *2015 12th International Conference on Information Technology-New Generations*. IEEE, 784–787.

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[29] Zhi-Gang Liu, Paul N. Whatmough, Yuhao Zhu, and Matthew Mattina. 2021. S2TA: Exploiting Structured Sparsity for Energy-Efficient Mobile CNN Acceleration. arXiv:2107.07983 [cs.AR]

[30] Alberto Marchisio, Muhammad Abdullah Hanif, Faiq Khalid, George Plastiras, Christos Kyrkou, Theocharis Theocharides, and Muhammad Shafique. 2019. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 553–559.

[31] Alberto Marchisio, Andrea Massa, Vojtech Mrazek, Beatrice Bussolino, Maurizio Martina, and Muhammad Shafique. 2020. NASCaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[32] Alberto Marchisio, Vojtech Mrazek, Muhammad Abdullah Hanif, and Muhammad Shafique. 2020. DESCNet: Developing efficient scratchpad memories for capsule network hardware. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[33] Sana Mazahir, Osman Hasan, and Muhammad Shafique. 2017. Adaptive approximate computing in arithmetic datapaths. *IEEE Design & Test* 35, 4 (2017), 65–74.

[34] Vojtech Mrazek, Zdenek Vasícek, Lukás Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. 2019. ALWANN: automatic layer-wise approximation of deep neural network accelerators without retraining. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[35] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[36] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 27–40. https://doi.org/10.1145/3140659.3080254

[37] Saman Payvar, Mir Khan, Rafael Stahl, Daniel Mueller-Gritschneder, and Jani Boutellier. 2019. Neural Network-based Vehicle Image Classification for IoT Devices. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. 148–153. https://doi.org/10.1109/SiPS47522.2019.9020464

[38] Bharath Srinivas Prabakaran, Asima Akhtar, Semeen Rehman, Osman Hasan, and Muhammad Shafique. 2021. BioNetExplorer: Architecture-Space Exploration of Bio-Signal Processing Deep Neural Networks for Wearables. *IEEE Internet of Things Journal* (2021).

[39] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. 2020. DRMap: A generic DRAM data mapping policy for energy-efficient processing of convolutional neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[40] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. 2021. ROMANet: Fine-Grained Reuse-Driven Off-Chip Memory Access Management and Data Organization for Deep Neural Network Accelerators. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 4 (2021), 702–715.

[41] Robert M Radway, Andrew Bartolo, Paul C Jolly, and et. al. 2021. Illusion of large on-chip memory by networked computing chips for neural network inference. In *Nature Electronics*, Vol. 4. 71–80.

[42] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. 2009. A Massively Parallel Coprocessor for Convolutional Neural Networks. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. 53–60. https://doi.org/10.1109/ASAP.2009.25

[43] Gil Shomron, Freddy Gabbay, Samer Kurzum, and Uri Weiser. 2021. Post-Training Sparsity-Aware Quantization. *arXiv preprint arXiv:2105.11010* (2021).

[44] Frank Slomka, Matthias Dorfel, Ralf Munzenberger, and Richard Hofmann. 2000. Hardware/software codesign and rapid prototyping of embedded systems. *IEEE Design & Test of Computers* 17, 2 (2000), 28–38.

[45] Rafael Stahl, Alex Hoffman, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. 2021. DeeperThings: Fully Distributed CNN Inference on Resource-Constrained Edge Devices. *International Journal of Parallel Programming* 49, 4 (2021).

[46] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2020. *Efficient Processing of Deep Neural Networks*. Morgan & Claypool Publishers. https://doi.org/10.2200/S01004ED1V01Y202004CAC050

[47] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Magazine* 12, 3 (2020), 28–41. https://doi.org/10.1109/MSSC.2020.3002140

[48] Frederick Tung and Greg Mori. 2018. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7873–7882.

[49] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Magnet: A modular accelerator generator for neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[50] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. 2018. Design Flow of Accelerating Hybrid Extremely Low Bit-Width Neural Network in Embedded FPGA. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 163–1636. https://doi.org/10.1109/FPL.2018.00035

[51] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. arXiv:1804.03209 [cs.CL]

[52] Simon Wiedemann, Suhas Shivapakash, Daniel Becking, Pablo Wiedemann, Wojciech Samek, Friedel Gerfers, and Thomas Wiegand. 2021. FantastIC4: A Hardware-Software Co-Design Approach for Efficiently Running 4Bit-Compact Multilayer Perceptrons. *IEEE Open Journal of Circuits and Systems* 2 (2021), 407–419. https://doi.org/10.1109/OJCAS.2021.3083332

[53] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785

[54] Di Wu, Qingming Tang, Yong le Zhao, Ming Zhang, Yingnan Fu, and Debing Zhang. 2020. EasyQuant: Post-training Quantization via Scale Optimization. *ArXiv* abs/2006.16669 (2020).

[55] Qi Xia, Winson Ye, Zeyi Tao, Jindi Wu, and Qun Li. 2021. A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Computing* 1, 1 (2021).

[56] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. 2020. AutoDNNchip: An automated dnn chip predictor and builder for both FPGAs and ASICs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 40–50.

[57] Rui Xu, Sheng Ma, Yaohua Wang, and Yang Guo. 2020. CMSA: Configurable Multi-directional Systolic Array for Convolutional Neural Networks. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 494–497. https://doi.org/10.1109/ICCD50377.2020.00089

[58] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[59] Ye Yu, Yingmin Li, Shuai Che, Niraj K. Jha, and Weifeng Zhang. 2021. Software-Defined Design Space Exploration for an Efficient DNN Accelerator Architecture. *IEEE Trans. Comput.* 70, 1 (2021), 45–56. https://doi.org/10.1109/TC.2020.2983694

[60] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-x: An Accelerator for Sparse Neural Networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 20, 12 pages.

[61] Shuhua Zhang, Jie Tong, Jun Zhang, Yuqing Lei, Minghao Zhang, Dang Li, and Lanruo Wang. 2020. A RISC-V Based Coprocessor Accelerator Technology Research for Convolution Neural Networks. *Journal of Physics: Conference Series* 1631 (sep 2020), 012002. https://doi.org/10.1088/1742-6596/1631/1/012002

[62] Ruizhe Zhao and Jianyi Cheng. 2021. Phism: Polyhedral High-Level Synthesis in MLIR. *arXiv preprint arXiv:2103.15103* (2021).