HW-FlowQ: A Multi-Abstraction Level HW-CNN Co-design Quantization Methodology

HW-FlowQ: A Multi-Abstraction Level HW-CNN Co-design Quantization Methodology

NAEL FASFOUS*, Technical University of Munich, Germany
MANOJ ROHIT VEMPARALA* and ALEXANDER FRICKENSTEIN*, BMW Autonomous Driving, Germany
EMANUELE VALPREDA*, Politecnico di Torino, Italy
DRITON SALIHU and NGUYEN ANH VU DOAN, Technical University of Munich, Germany
CHRISTIAN UNGER and NAVEEN SHANKAR NAGARAJA, BMW Autonomous Driving, Germany
MAURIZIO MARTINA, Politecnico di Torino, Italy
WALTER STECHELE, Technical University of Munich, Germany

Model compression through quantization is commonly applied to convolutional neural networks (CNNs) deployed on compute and memory-constrained embedded platforms. Different layers of the CNN can have varying degrees of numerical precision for both weights and activations, resulting in a large search space. Together with the hardware (HW) design space, the challenge of finding the globally optimum HW-CNN combination for a given application becomes daunting. To this end, we propose HW-FlowQ, a systematic approach that enables the co-design of the target hardware platform and the compressed CNN model through quantization. The search space is viewed at three levels of abstraction, allowing for an iterative approach for narrowing down the solution space before reaching a high-fidelity CNN hardware modeling tool, capable of capturing the effects of mixed-precision quantization strategies on different hardware architectures (processing unit counts, memory levels, cost models, dataflows) and two types of computation engines (bit-parallel vectorized, bit-serial). To combine both worlds, a multi-objective non-dominated sorting genetic algorithm (NSGA-II) is leveraged to establish a Pareto-optimal set of quantization strategies for the target HW-metrics at each abstraction level. HW-FlowQ detects optima in a discrete search space and maximizes the task-related accuracy of the underlying CNN while minimizing hardware-related costs. The Pareto-front approach keeps the design space open to a range of non-dominated solutions before refining the design to a more detailed level of abstraction. With equivalent prediction accuracy, we improve the energy and latency by 20% and 45% respectively for ResNet56 compared to existing mixed-precision search methods.

CCS Concepts: • **Computing methodologies** → **Modeling and simulation**; • **Computer systems organization** → *Architectures*; **Neural networks**.

*Contributed equally to this research.

Authors' addresses: Nael Fasfous, nael.fasfous@tum.de, Technical University of Munich, Germany; Manoj Rohit Vemparala, manoj-rohit.vemparala@bmw.de; Alexander Frickenstein, alexander.frickenstein@bmw.de, BMW Autonomous Driving, Germany; Emanuele Valpreda, emanuele.valpreda@polito.it, Politecnico di Torino, Italy; Driton Salihu, driton.salihu@tum.de; Nguyen Anh Vu Doan, anhvu.doan@tum.de, Technical University of Munich, Germany; Christian Unger, christian.unger@bmw.de; Naveen Shankar Nagaraja, naveenshankar.nagaraja@bmw.de, BMW Autonomous Driving, Germany; Maurizio Martina, maurizio.martina@polito.it, Politecnico di Torino, Italy; Walter Stechele, walter.stechele@tum.de, Technical University of Munich, Germany.

XXXX-XXXX/2021/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Additional Key Words and Phrases: Convolutional Neural Networks, Multi-Objective Optimization, Hardware Modeling, Genetic Algorithms, Quantization.

ACM Reference Format:
Nael Fasfous, Manoj Rohit Vemparala, Alexander Frickenstein, Emanuele Valpreda, Driton Salihu, Nguyen Anh Vu Doan, Christian Unger, Naveen Shankar Nagaraja, Maurizio Martina, and Walter Stechele. 2021. HW-FlowQ: A Multi-Abstraction Level HW-CNN Co-design Quantization Methodology. 1, 1 (October 2021), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Convolutional neural networks (CNNs) have become the state-of-the-art for many computer vision tasks, including image classification, object localization and semantic segmentation [4, 13, 29]. Along with their rise in popularity, their growth in computational complexity and memory demand has made the efficient deployment of CNNs on edge devices an increasingly challenging task. This led to a substantial effort in research to focus on the compression of CNNs.

A commonly used compression technique is quantization. The multitude of arithmetic operations typically required for modern CNNs can be executed on simplified hardware (HW) components, bringing benefits in terms of area, power and/or latency. Another advantage comes from lower bitwidth operands efficiently exploiting the data movement bandwidth available on the HW. However, the technique is not without its challenges. Low bitwidth representations have a lower information capacity, losing the fine details captured by gradient propagation at training time. Finding a suitable quantization scheme that achieves the target benefits without degrading task-related accuracy becomes more challenging at lower bitwidths. Moreover, not all layers require equal numerical precision [32], rendering the search space even larger.

Evaluating a quantization strategy based on metrics that are loosely correlated to HW benefits can lead to sub-optimal deployment setups. This has influenced recent works to optimize neural networks with hardware-in-the-loop (HIL) approaches [14, 32]. HIL-based optimization techniques necessitate the fully-functional and fabricated HW to be readily available when optimizing the CNN, leaving little to no room for adjusting the target execution platform. Look-up table (LUT) approaches have the same impediment, as filling up the table would require synthesized HW measurements [33].

HW-FlowQ bridges the gap between HW design and CNN quantization. In many real-time, safety-critical scenarios (e.g. robotics, autonomous driving), energy consumption and latency goals with respect to set resource constraints can only be met through hand-in-hand HW-CNN co-design. For such challenges, three large search spaces for the (1) HW-design, (2) the CNN structure and the (3) layer-wise quantization strategy need to be considered. Traversing all three in an unstructured manner would likely lead to sub-optimal solutions.

In a top-down approach, the tripartite design space is narrowed down iteratively with the help of an extensive HW-modeling tool and a genetic algorithm. HW-FlowQ provides the flexibility of studying the impact of mixed-precision quantization and HW-specific design parameters, without having to finalize the hardware platform in the early phases of development.

The contributions of this work can be summarized as follows:

- A HW-model-in-the-loop quantization methodology, allowing design space exploration of CNN quantization strategies and hardware platforms at different design phases.
- Exploring single and multi-objective genetic algorithms (SOGA, MOGA) for finding Pareto-optimal quantization strategies with respect to the underlying HW platform.
- Modeling vectorized and bit-serial accelerators, with varying resources and dataflows for mixed-precision quantization, enabling HW-design exploration *during* CNN optimization.

2 RELATED WORK

2.1 Quantization Methods

Many works have focused on improving the training scheme for quantized neural networks [2, 6, 17, 18, 36]. DoReFa-Net [36] adapts the straight-through estimator (STE) [2] and bounds the magnitude of latent weights and activation between $[0, 1]$. These latent datatypes are deterministically quantized. QNN [17] additionally quantizes the weights in the first and last layers of the CNN. The work in PACT [6] improves the training procedure of QNNs by learning the optimal clipping level for the activations of each layer at training time. The dynamic clipping function allows larger representational capability than DoReFa-Net, thereby increasing the prediction accuracy. The aforementioned works explore the effect of quantization on the accuracy of the model and the achieved compression rate. Claims on the degree of improvement in energy efficiency or latency are difficult to make, as these complex metrics highly depend on the underlying HW details (memory hierarchy, interconnect, technology, etc.).

2.2 Quantization & Search Schemes

Dong et al. [10] determine the bitwidths of activations and weights based on the Hessian spectrum obtained for individual layers. The quantization criterion is based on model size (`Params`), which loosely correlates to final energy or latency for an inference execution. Wu et al. [34] propose a framework that learns quantization levels for each layer during the training period. To encourage lower-precision weights and activations, a loss term is associated with the quantization objective, capturing the benefits in operation count (`OPs`) and model size (`Params`), but not necessarily the effectiveness on HW-level metrics such as energy and latency. The authors of HAQ [32] propose a reinforcement learning-based exploration scheme to determine HW-aware layer-wise quantization levels for weights and activations in a CNN model. The reward function is evaluated after executing the inference of the quantized CNN on an FPGA design. In APQ [33] a joint model architecture-pruning-quantization search is proposed. Pre-trained and pruned sub-networks are extracted from a once-for-all network. Then, mixed-precision quantization is applied and a predictor estimates the final accuracy. An energy/latency LUT is used to provide the HW feedback for a target accelerator during the search. Therefore, a set of pre-sampled data points is required from a readily available target hardware.

2.3 Hardware Modeling

Timeloop [27] is a HW-modeling tool that exploits CNN execution determinism to offer accurate estimates for a given hardware description. The tool provides the flexibility of changing the cost of hardware operations (e.g. read, write, multiply-accumulate, etc.) and the memory hierarchy, among other design parameters. Interstellar [35] proposes formal dataflow definitions. Different to Timeloop, Interstellar uses the Halide programming language to represent the memory hierarchy and data movement constraints. Tetris [11] and Tangram [12] make use of a dataflow scheduler for DNN workloads on spatial accelerators to test the potential of other manipulations possible for their memory hierarchies. The mentioned works have proven the effectiveness of HW-modeling of DNN workloads. Nevertheless, other aspects have not been explored as thoroughly, such as adding the effects of layer-wise mixed-precision quantization on the resulting dataflow or supporting mixed precision computation units. There is a need to extensively integrate HW-models with CNN optimization algorithms to aid the exploration of mixed-precision CNNs with respect to the HW-model under consideration, particularly when multiple HW-architectures are being considered as potential fabrication candidates.

2.4 Hardware-Software Co-Design

Jiang et al. [19] propose a HW-SW co-exploration framework, which includes the HW's performance in the reward function of an RL-agent and iteratively tunes both the CNN and HW architectures. Fine hardware-level details, such as scheduling schemes and quantized execution, are not explored, as the optimization loop targets optimally partitioning the CNN workload over a pool of FPGAs. The authors of [1] propose a HW-CNN co-design framework based on neural architecture search (NAS). Fine-level details such as dataflow mapping are not studied, nor the effect of layer-wise quantization on bit-serial or vectorized hardware. The authors of ALOHA [24] propose a multi-phase genetic algorithm-based framework for HW-aware CNN design, which takes into account the trade-off between model compression and adversarial robustness. Compression with mixed-precision or layer-wise quantization is not explored, nor the direct interactions between layer-wise quantization and different hardware designs and processing units. Although ALOHA considers the target hardware in early phases of the CNN design, the hardware itself is not actively being co-designed in the flow. FINN-R [3] is an accelerator generator for quantized CNN inference on Xilinx FPGAs. Two types of architectures are supported: a pipelined dataflow architecture with appropriately sized MAC units for each layer to exploit layer-wise quantization, and a single multi-layer array that is reused across all layers. Candidate architectures are evaluated using an analytical model for resource usage and throughput. The framework is HW-design focused, and does not explore the quantization and training search space of the CNN. NHAS [22] aims to find an optimal quantized CNN using an evolutionary algorithm. An efficient HW-architectural dimensioning for compute array and on-chip memory is searched to accelerate a given set of benchmark workloads. The HW-evaluation follows a LUT approach. Although NHAS promotes the idea of joint HW-CNN co-design, the approach can be enhanced by understanding the influence of each search decision against various aspects, such as the quantization method, mixed precision accelerator choice, and scheduling schemes, with the help of HW models during the optimization.

3 HW-FLOWQ

HW-FlowQ is a HW-CNN co-design methodology which facilitates a top-down design approach, iteratively going through different levels of abstraction and performing some iterations of exploration, before fixing some parameters and refining the design to one abstraction level lower. Moving away from hardware-in-the-loop and LUT approaches in existing works, we propose *model-in-the-loop* design steps.

HW-FlowQ is based on an interaction between the individuals ρ of population $\mathcal{P}$, with the hardware model μ in the context of a genetic search algorithm $\mathcal{G}$ (Fig. 1). In detail, the genotype of an individual ρ encodes the layer-wise quantization levels for weights and activations (b_W, b_A) of the CNN. The individual's fitness $\mathcal{F}$ is measured through the HW-model estimations φ and CNN accuracy term ψ, computed w.r.t. the images and labels of a validation set. When $\mathcal{G}$ is a single-objective genetic algorithm (SOGA), the objectives φ and ψ are combined into a single cost function for fitness evaluation (Sec. 3.2). $\mathcal{G}$ can also take the form of a multi-objective optimization genetic algorithm (MOGA), such as the non-dominated sorting genetic algorithm (NSGA-II).

3.1 HW-Model Abstraction Levels

To enable the design steps of HW-SW co-design, different representations of the target HW platform need to be accessible based on the design phase. Starting from an abstracted, high-level representation makes it possible to coarsely search for HW-CNN combinations that may suit the application at hand. After some high-level parameters are fixed, a step of refinement can take place. This brings the exploration to a finer level of detail, but within the scope of the fixed parameters

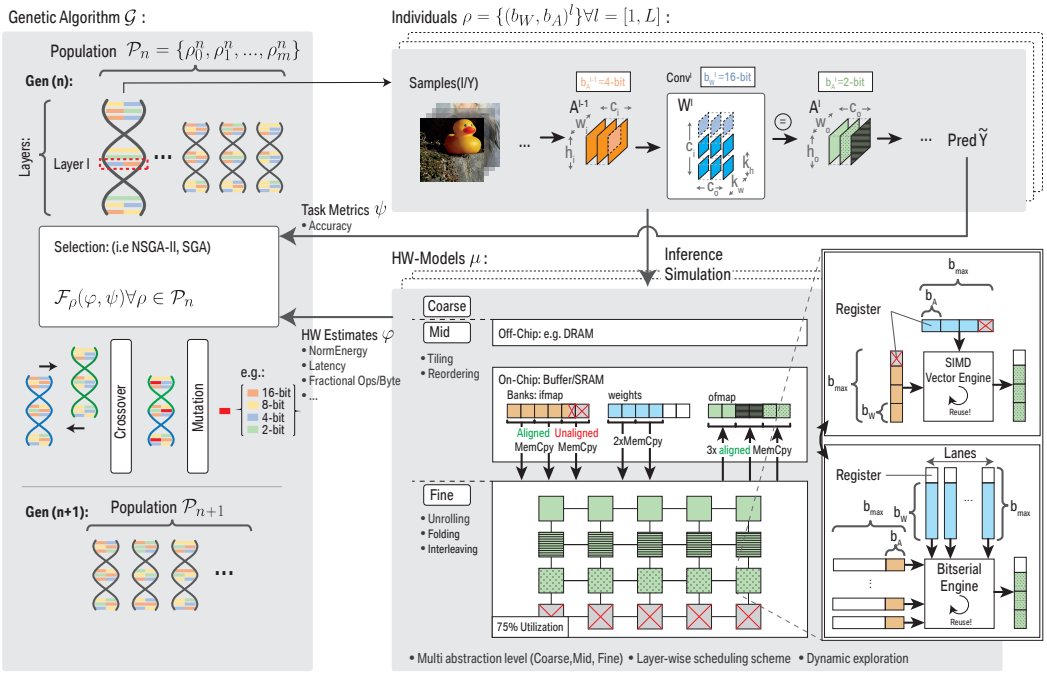


Fig. 1. Overview of the HW-FlowQ methodology. Population $\mathcal{P}$ is evaluated on task-related accuracy ψ and HW-metrics φ. The three proposed HW-modeling abstraction levels - *Coarse*, *Mid* and *Fine*, enable the genetic algorithm $\mathcal{G}$ to consider the HW-metrics of the CNN relative to the current design phase.

of the previous abstraction level (Fig. 2). More implementation-specific aspects can be considered after each refinement iteration. At any stage, if the exploration fails to find any suitable solutions, an abstraction step can take the design back to a coarser level and re-evaluate the higher-level parameters which were set. This type of design flow is commonly used in VLSI design, where complex, large design spaces must be explored at different levels of abstraction, from system-level down to transistor logic [25, 31].

Three levels of abstraction are offered in HW-FlowQ, namely *Coarse*, *Mid* and *Fine*. Starting with *Coarse*-level optimization, the framework can be used to test the effect of quantization on differently shaped/sized CNNs, given as an input. The total computations required and the task-related accuracy can be evaluated. The CNN parameters at this level heavily influence the start of the co-design

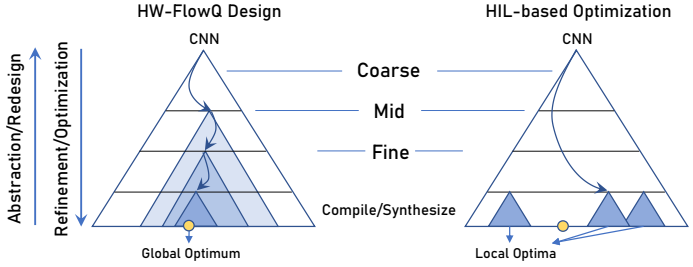


Fig. 2. Iterative refinement increases the likelihood of finding the global optimum. Flow inspired by [31].

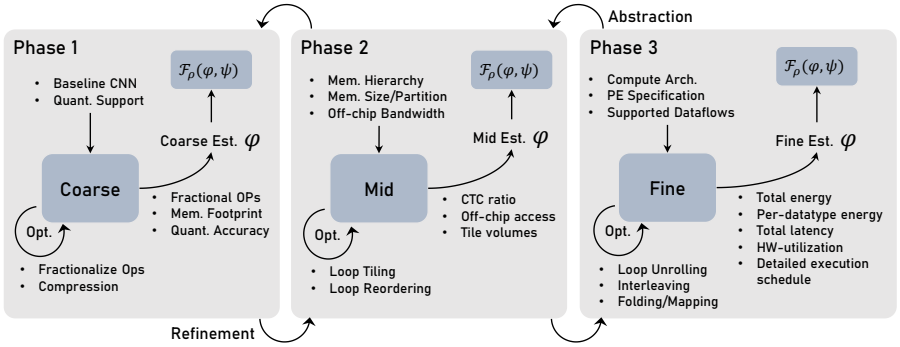


Fig. 3. Input, output and optimization details of the HW-model μ abstraction levels used at each phase. After refinement, the inputs of the preceding phase are inherited to the next.

process, as they set the upper-bound of task-related accuracies possible, as well as the range of fractional operations and on-chip memory the HW must accommodate. After quantization, if the target compression and/or task-related accuracy cannot be met, support for lower quantization levels needs to be considered and/or new CNN architectures need to be provided. The quantization training scheme can also be decided at this stage (e.g. DoReFa, PACT, QNN). It is important to note that HW-FlowQ does not constitute a neural architecture search (NAS) methodology, but is rather complementary to such techniques. As an example, a NAS framework can provide HW-FlowQ with high-accuracy CNN architectures as inputs at the *Coarse*-level. Then, HW-FlowQ can quantize them optimally for target HW-designs, as well as facilitate designing customized HW for the proposed CNNs. Once the CNN(s) meets the high-level requirements, the *Mid*-level evaluates the feasibility of different memory hierarchies to buffer and move data between different stages before reaching the on-chip computation units. Parameters such as data transfer volumes, computation-to-communication ratio (CTC) and off-chip memory accesses can be searched. This information can help in deciding which off-chip to on-chip communication infrastructure and bandwidth is suitable to meet the application constraints. The CNN can further be quantized with this HW-model-in-the-loop, in order to close the gap between the HW-constraints and computation/communication requirements, while maintaining the task-accuracy goals. Performing one more iteration of refinement takes the design to the *Fine*-level. At this stage, the HW computation architecture can be defined. Precise information on the supported quantization levels, number of computation units available, register file sizes, supported data movements, and more, can be provided. HW-FlowQ provides high-fidelity estimates of the benefits that can be achieved on the prospective HW-design, for a particular quantization strategy (Fig. 5). Details on the *Fine*-level modeling are provided in the next sections.

Considering all design parameters holistically would imply searching *all* possible quantization strategies for *all* candidate CNNs (*Coarse*), on *all* possible on-chip/off-chip communication and on-chip memory sizes (*Mid*) for *all* possible dataflows, compute array sizes, multiplier types and register dimensions (*Fine*). This would ultimately waste an immense amount of GPU hours, searching for solutions which could have been eliminated at the *Coarse*-level already. Additionally, with so many search parameters, the convergence of the search algorithm becomes more difficult to guarantee, potentially leading to sub-optimal results. To address this challenge, the step-wise optimization in HW-FlowQ's *Coarse*, *Mid* and *Fine* levels along with the Pareto-front-based quantization approach (NSGA-II) promotes a design-flow which leads to improved synergies in the final HW-CNN implementation and a more practical approach to searching the three large search spaces of (1) CNN structure, (2) quantization strategy, and (3) HW design.

Fig. 3 summarizes the inputs required at each level, the optimization that the HW-Model μ can perform internally at each phase and the output estimates which can be used to evaluate the HW-related fitness $\mathcal{F}_\rho$ of different individuals in population $\mathcal{P}$. Traversal between the levels is indicated by refinement and abstraction arrows. The decision on whether the search takes a step of refinement can be inferred from a list of application constraints. For example, a maximum number of fractional operations needs to be met, before a transition between *Coarse* to *Mid* can take place. Similarly, a desired off-chip communication constraint can be set, before the design transitions between *Mid* to *Fine*. If a certain constraint cannot be met, the framework must reconsider the inputs of the current level (e.g. at *Coarse* reconsider baseline CNN architecture, at *Mid* reconsider memory hierarchy, etc.). If changing the inputs of the level does not meet the targets, the inputs of the level above are reconsidered (abstraction). Through this progressive filtering of design decisions at each level, the output of the overall framework meets the desired application targets at the end of the flow.

3.2 Genetic Algorithm

Finding the correct layer-wise quantization strategy for both weights and activations w.r.t. a target HW-model is a complex problem which would benefit from gradient-free optimization due to the discrete nature of the solution space. The search space for the quantization strategy alone consists of $|q|^{2L}$ solutions, where q is the set of possible quantization levels and L is the number of layers. Quantizing some layers leads to larger drops in accuracy than others, and different accuracy drops can take place at different quantization levels for the same layer. Moreover, quantization strategies change the mapping and scheduling space of the accelerator. For example, a quantization strategy might make new schedules possible, which lead to sudden drops in latency and energy, as soon as a particular computation tile fits the on-chip memory after quantization. Therefore, we leverage genetic algorithms (GAs) to tackle our quantization strategy search problem, as they are known to be resilient to noisy search spaces, quick to prototype, and do not need smooth, continuous search spaces to perform well.

Explicit, bijective encoding is used to create the genomes of potential solutions as shown in Fig. 4. A single genome represents a potential CNN quantization strategy and has as many genetic loci as there are layers in the CNN. Each genetic locus encapsulates a tuple of integer bitwidth values for weights and activations (b_W , b_A) at the corresponding layer. The set of possible alleles at each genetic locus is defined by the bitwidths supported by the HW-model. Bitwidth-to-layer encoding can be captured intuitively in sequential genomes, which leads to a sensible use of GA operators, such as single-point crossover (example in Fig. 4). Neighbouring CNN layers have higher feature correlation than distant layers. Therefore, quantized layer relationships encoded in neighboring genetic loci can survive in a population and be reused through single-point crossover to create more efficient offspring. The more fit the parents become throughout the generations, the better genetic localities they will have to create better individuals. Mutation further allows offspring to escape local minima of their parents.

Referring back to Fig. 1, on the top-left an initial population $\mathcal{P}_0$ is randomly generated at the start of the genetic algorithm $\mathcal{G}$, with each individual encoding the quantization levels of each layer of the CNN in its genes. The individuals of $\mathcal{P}_0$ are briefly fine-tuned and evaluated based on their task accuracy ψ on a validation set (Fig. 1 top-right), as well as HW-estimates φ of the HW-model through inference simulation (Fig. 1 bottom-right). Based on the GA configuration, ψ and φ define the fitness of each individual $\rho^0 \in \mathcal{P}_0$. As depicted in Fig. 1, ψ and φ are fed back to a *selection* phase in $\mathcal{G}$, to constrain the cardinality of the population to $|\mathcal{P}| = m$. Individuals survive this phase based on their fitness. Survivors are allowed to *mate* and produce offspring in $\mathcal{P}_1$, which inherit alleles from two survivor parents through *crossover*. A round of *mutation* takes place, altering alleles of

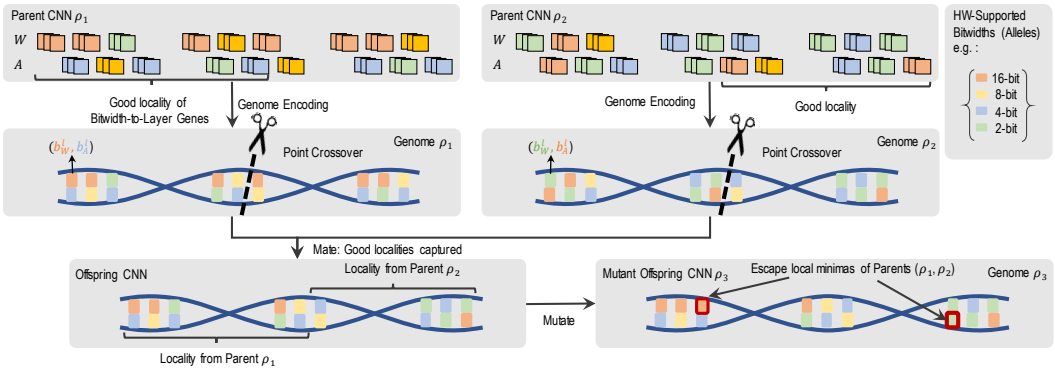


Fig. 4. Layer-wise genome encoding allows for intuitive use of genetic operators (crossover, mutation) to capture and maintain good localities of bitwidth-to-layer encodings from two fit parents into their offspring.

the offspring in $\mathcal{P}_1$. The population goes through the same phases of fitness evaluation, selection and crossover for n subsequent generations.

3.2.1 Fitness Evaluation. We explore both a SOGA and MOGA approach in the form of the non-dominated sorting genetic algorithm (NSGA-II). Both GAs share the same evolutionary flow described earlier, but are different in their observation of fitness. By definition, SOGA maximizes a *single* reward. Since our problem inherently involves multiple objectives (ψ and φ), a balanced reward function must be defined to combine them into a single fitness value $\mathcal{F}$ to apply SOGA.

$$\mathcal{F}_\rho = \begin{cases} (1 - \frac{\psi^* - \psi}{t}) \cdot \log(\frac{\varphi^*}{\varphi}), & \text{if SOGA} \\ \psi, \varphi & \text{otherwise NSGA-II} \end{cases} \tag{1}$$

The fitness definition of SOGA in Eq. 1 is inspired by the cost function proposed in [16]. ψ^* and φ^* are the task-related accuracy and the HW-estimates of the uncompressed CNN respectively. The function balances the improvements in HW efficiency $\log(\varphi^*/\varphi)$ while trying to maintain task-related accuracy through the term $(1 - (\psi^* - \psi)/t)$. t sets a threshold on accuracy degradation, where a difference between ψ^* and ψ equal to or greater than t turns the accuracy term negative and renders the fitness $\mathcal{F}_\rho$ of individual ρ unacceptable.

In the case of NSGA-II optimization, the algorithm evaluates the *Pareto optimality* of each individual w.r.t. the population $\mathcal{P}$. This relieves the burden of crafting a single fitness function, which may not always guarantee a fair balance between multiple objectives. Additionally, having an array of potential solutions in a Pareto-front is a better approach for design space exploration, compared to having a single solution suggested by the search algorithm. Design space exploration is a fundamental part of HW-SW co-design making NSGA-II an attractive alternative to SOGA.

Considering the accuracy-related fitness term ψ, the quantization strategies of a population $\mathcal{P}$ need to be evaluated in a reasonable amount of time, to avoid a bottleneck in the search process. To tackle this challenge, we circumvent the need to fully train each quantized network ρ in $\mathcal{P}$. Instead, we instantiate the CNN model, load it with pre-trained floating-point weights, then quantize it according to the genome of ρ and briefly fine-tune it to recover from the accuracy loss introduced by the quantization. This process can also be parallelized, as the individuals within a population can be fine-tuned at the same time, on a single or multiple GPUs. We study the learning behavior of 2-bit, 4-bit and 6-bit networks, to see how early their training curves can be differentiated. This gives us an indication of how well the accuracy will be at the end of a full-training cycle. The accuracy fitness evaluation epochs in Sec. 6 were chosen accordingly. The GA essentially evaluates

the learning capacity of the individual, not its final fully-trained accuracy. At the end of the search, when a solution is chosen, we train it from scratch, without loading any pre-trained weights. It is worth mentioning that fast accuracy predictors, such as the ones proposed in [8, 33], could also be used for the purpose of fast accuracy-related fitness evaluation in HW-FlowQ's GA.

3.2.2 Genetic Operators. The mutation, crossover and selection operations are pivotal to the GA's efficacy. We apply single-point crossover, which intuitively has a high probability of capturing attractive bitwidth-to-layer encodings of two fit individuals and maintains inter-layer dependencies across segments of the CNN, as shown in Fig. 4. With mutation probability p_{mut} a single allele at a randomly selected genetic locus is replaced by another from the set of possible alleles. All individuals conform with the CNN and the quantization levels supported on the HW. Tournament selection is used for SOGA, where m tournaments take place to decide all the survivors. On the other hand, NSGA-II selection is based on the crowded-comparison-operator [9].

3.3 Modeling Mixed Precision Inference

In this work, we focus on modeling spatial architectures similar to [5, 23, 27, 35], with an on-chip buffer and a compute core with an array of PEs, as depicted in Fig. 1. The energy cost of data accesses depends on the technology and the size of the memory. HW-FlowQ supports independent read-write costs for off-chip communication, memory blocks, as well as the register files (RFs) of the PEs on the compute blocks. For our cost models, we adhere to the approach proposed in [5, 15, 27, 35]. A normalized energy cost is set for each operation that can take place on the architecture. The HW-model attempts to map the computations of a particular CNN workload efficiently onto the HW-model. For each schedule, the HW-model is able to extract the number of *actions* (reads, writes, MACs) required at each level of the accelerator. The number of actions is multiplied by the cost of each action on each type of memory/compute unit. The exact normalized energy costs we choose in this work align with the Timeloop [27] framework and the Eyeriss model in [5], as shown in Tab. 1. HW-FlowQ also supports manually setting costs for each action, depending on different fabrication technologies.

3.3.1 Scheduler and Mapper: Modern compute architectures allocate a considerable amount of their power budget for memory accesses and data movement [15]. Moreover, redundant data movement can have a significant impact on latency. This has made efficient scheduling of CNNs on spatial hardware an active field of research [5, 11, 12, 23, 27, 35]. Eq. 2 represents the operation required for computing a single output pixel of the output activation tensor A^l from the two input tensors A^{l-1} and W^l for input activations and weights respectively. The bias addition and batch dimension are not shown for simplicity. As depicted in Fig. 1, C_i and C_o represent the input and output channel dimensions, H_i and W_i the spatial height and width of the input feature maps, H_o and W_o equivalently for output feature maps, and K_h and K_w are the height and width of the 2-D kernels. Lastly, s represents the stride of the kernel over the input feature map for the convolution operation. The equation can also be interpreted as a nested loop, with the tensor and summation indices and limits representing the for-loops' iterators and loop bounds. Consequently, similar to other for-loop algorithms, CNNs inherently present many data reuse opportunities.

$$A^l[c_o][w_o][h_o] = \sum_{c_i}^{C_i} \sum_{k_h}^{K_h} \sum_{k_w}^{K_w} A^{l-1}[c_i][w_o \cdot s + k_w][h_o \cdot s + k_h] \cdot W^l[c_o][c_i][k_w][k_h] \tag{2}$$

Three main techniques are commonly used to optimize a nested loop's execution on hardware, namely *loop tiling, reordering and unrolling*. HW-FlowQ's scheduler and mapper components handle loop optimization techniques largely in a similar manner to the popular frameworks Interstellar [35]

and Timeloop [27]. For brevity, we will only discuss the added considerations to capture the effect of mixed-precision quantization, and refer the interested reader to [35] and [27] for further reading on CNN inference modeling on spatial architectures.

Quantization shrinks the bitwidth of datatypes allowing larger computation tiles to fit in a given lower-level memory. This increases the number of possible loop tiling and reordering schedules. Loop optimization through *unrolling* is handled by HW-FlowQ's mapper component and is dependent on both the dataflow supported by the accelerator and the mixed-precision computation technique. It is important to note that when unrolling fractionalized (quantized) computations on a vectorized or multi-lane bit-serial PE-array, a single PE may handle more spatially distributed computations, as long as its register files fit the operands/partial sums needed/generated by said computations. This can be exploited by HW-FlowQ's mapper to find more efficient schedules which fit on a smaller physical computation array and require less PE-to-PE data movement.

Depending on the defined HW-model, the scalar or single-instruction multiple-data (SIMD) vector-engines in the PEs can be word-aligned, making some quantization degrees less attractive than others. An example of sub-optimal SIMD-register usage is marked with a red-cross in Fig. 1 (middle-right). HW-FlowQ can also model bit-serial compute units, similar to [30], in which case, a relative improvement for any quantization level for weights and/or activations can be achieved on the compute block. The word alignment on the compute block can be set differently to that of the outer memory blocks and the off-chip memory interconnect.

In the convolution operation, partial sums (`psum`) can grow after each accumulation to a maximum of $2b + C_i$, where b is the bitwidth of the operands. The HW-model considers instances of the largest possible `psum`, according to the maximum bitwidth b_{max} supported by the accelerator. The increase in vector throughput due to quantization of W^l and A^{l-1} is constrained to the maximum amount of `psum` RF memory available on the PE. After complete accumulation, a speed-up can be achieved in writing back the output pixels at the quantization level of the input of the following layer b_A^l of A^l.

3.3.2 Vectorized and Bit-Serial Computation: To estimate the benefits in the computation of low-bitwidth and mixed-precision CNNs, vectorized and bit-serial compute units are modeled. The choice of the computation unit has a direct influence on the schedule, as it affects how many computation cycles are required for a particular operation and how many unique computations can be assigned to the same HW at different bitwidths.

For vectorized accelerators, we model an aligned SIMD-MAC unit which has a maximum bitwidth of b_{max} for both weights and activations. A speed-up through data-level parallelism at the PE-level can happen at V_{speedup} integer steps as shown in Eq. 3.

$$V_{\text{speedup}} = \left\lfloor \frac{b_{max}}{\texttt{max}(b_W, b_A)} \right\rfloor \tag{3}$$

V_{speedup} is the vectorization degree aligned with the wider operand between b_W and b_A. This not only allows for more parallel computations in the same cycle, but also reduces the memory access cost at the register file level, which would now access V_{speedup} data that fit into the SIMD-register with bitwidth b_{max} in a single read operation. The limitation of vectorized computation units is that they can only perform complete operations, and therefore cannot always fully exploit any arbitrary bitwidth. For example, a 16-bit vector unit can perform 2 complete 8-bit computations, however, if the operands were 6-bits each, a non-integer speed-up of ~ 2.67 would not be possible. Another limitation is that variable bitwidths of $b_W \neq b_A$ cannot be exploited for higher parallelism even if both are aligned, due to the $\texttt{max}(b_A, b_W)$ term in Eq. 3. The wider of the two operands dictates the number of parallel computations which fit in the vector engine.

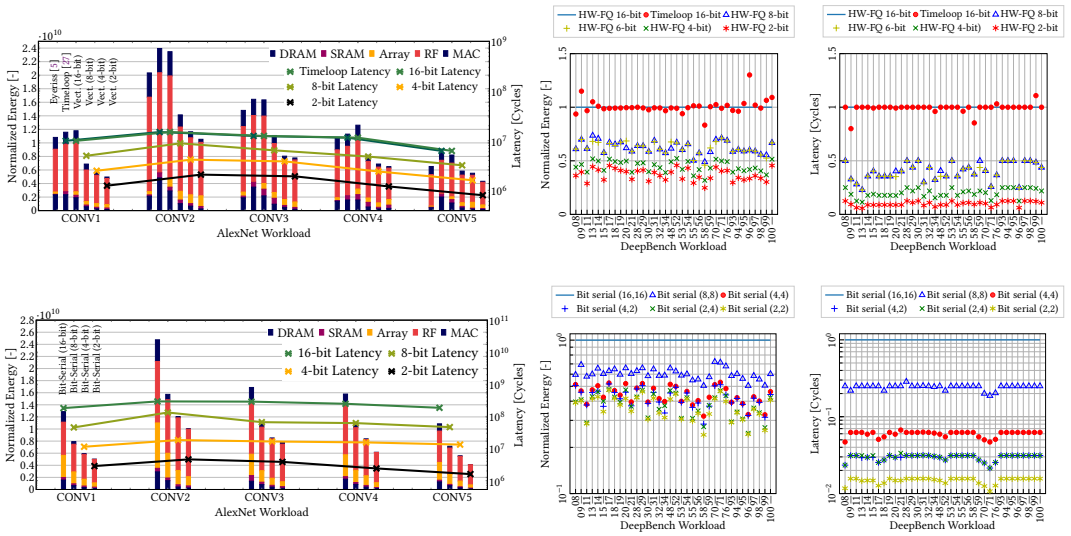


Fig. 5. 16-bit AlexNet validation of HW-FlowQ with Eyeriss [5] and Timeloop [27] (top-left), as well as 8, 4 and 2-bit vectorized execution. Validation with Timeloop on DeepBench workloads [26] (top-right). Bit-serial execution of AlexNet (bottom-left) and DeepBench (bottom-right).

Bit-serial computation units can fully exploit any level of quantization for both operands. Their performance is enhanced w.r.t. a b_{max} computation according to

$$BS_{\text{speedup}} = \frac{b_{max}^2}{b_W \times b_A}. \tag{4}$$

It is important to note that BS_{speedup} cannot be directly compared to V_{speedup} due to the inherent differences between how both architectures perform a single b_{max} computation. Bit-serial units require $b_W \times b_A$ cycles to complete a single computation, whereas bit-parallel (irrespective of the vectorization degree) produce a complete result(s) in every computation cycle. To compensate for this, most bit-serial accelerators employ more computation *lanes* to make up for their slower method of computation, and try to match the *throughput* of bit-parallel architectures, while benefiting from the flexibility of supporting and getting a speedup at any reduced bitwidth for either operand b_W or b_A. We model each PE to have a fixed number of independent lanes to perform parallel computations. The bits trickle in from the register files over $b_W \times b_A$ cycles.

4 MIXED-PRECISION MODEL VALIDATION

To validate HW-FlowQ's *Fine*-level modeling and mapping components, we compare its estimates to the Eyeriss architecture [5] for AlexNet [21] inference. We would like to stress that AlexNet is not used for quantization experiments (Sec. 6), as it is severely outdated. We only use AlexNet's diverse CNN layer shapes (large/small kernels, strides, group convolutions) to prove the fidelity/precision of our hardware-modeling framework on varying computation workloads. Similarly, Eyeriss provides sufficiently complex on-chip data movement, which is also challenging to model (e.g. vertical, horizontal, and diagonal data movement on spatial array). This diversity in workloads and data movement helps us validate our HW-modeling framework. We further validate our model with Timeloop [27] across workloads from the DeepBench benchmark suite [26], on Eyeriss-256-DB (configuration in Tab. 1).

Fig. 5 (top-left) shows the breakdown of normalized energy contributions at each memory level for the AlexNet convolutional layers. Our 16-bit HW-model demonstrates high fidelity and accuracy w.r.t. [5] and [27]. We additionally show the effects of quantization on latency and energy, for vectorized and bit-serial HW-models. Extending the comparison with Timeloop, the DeepBench workloads demonstrate the consistency between the modeling techniques (Fig. 5 (top-right)). Deviating points occur when both frameworks resolve to different mapping solutions (*i.e.* not related to the correctness of the model, but rather the schedule search). For both bit-serial and vectorized accelerators, a non-trivial variation can be observed for the energy benefits of different quantization schemes. Quantization allows for larger computation tiles to fit on the on-chip memory, making more loop-tilings legal for a given workload. This changes the schedule search space and introduces new solutions for the model's mapper to consider. A slightly more direct relationship can be observed for latency, particularly for bit-serial accelerators, which have a high parallelism potential (due to computation lanes) that is not easily saturated. It is important to note that the y-axis follows a logarithmic scale for bit-serial latency, due to the exponential speed-up gains (Eq. 4). [27] does not support bit-serial PEs.

5 CROSS-ABSTRACTION LEVEL INTERACTIONS

HW-FlowQ groups hardware-related design parameters into abstraction levels to facilitate the interpretation of HW-CNN interactions (recall Fig. 3). The neighbouring abstraction levels must be cohesive to create a sensible flow between them, which can guide the designer and the genetic algorithm towards a more efficient co-design strategy. For example, CNN workloads (*Coarse*) directly affect the on-chip memory (*Mid*). However, the effect of a CNN workload (*Coarse*) on the dataflow (*Fine*) is hard to understand without knowing the on-chip/off-chip interconnect (*Mid*) or the on-chip memory size (*Mid*) in between. The criteria are separated to divide the complexity of CNN structure search (*Coarse*), interconnect/memory hierarchy search (*Mid*) and compute architecture search (*Fine*).

In Fig. 6, we investigate the effect of changing the numerical precision of ResNet-18 for ImageNet on (1) off-chip (DRAM) accesses and (2) computational throughput. These metrics can be evaluated at all three abstraction levels, which makes them useful in highlighting the *flow* between the levels. At the *Coarse*-level, the DRAM accesses are estimated as the CNN's total necessary reads and writes for all datatypes (inputs, weights and outputs). Since the *Coarse*-level abstraction is agnostic to

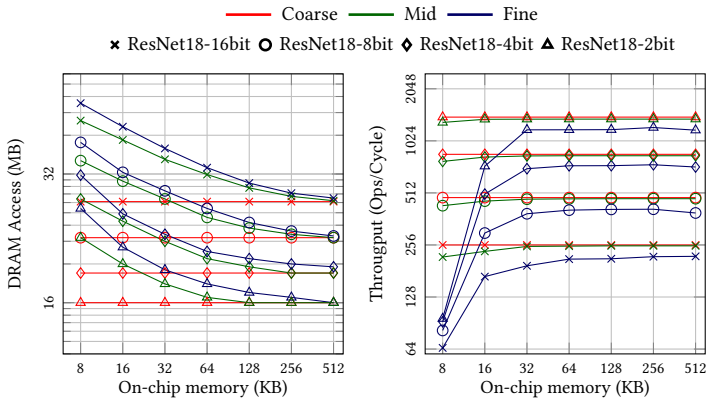


Fig. 6. Analysis of DRAM access and throughput on on-chip buffer size at different levels of hardware abstraction and quantization.

the on-chip memory details, we see that all its corresponding red curves are constant. However, among the *Coarse*-level curves, the difference in read/write volumes at each quantization level (left plot), as well as the speed-up possible through vectorization (right plot) is still captured.

Moving on to the *Mid*-level, the model can capture more details of the HW. In this case, the limitations of an under-dimensioned on-chip memory or an insufficient off-chip to on-chip communication bandwidth can be detected. The *Mid*-level estimates are sensitive to on-chip memory and communication, but semi-agnostic to the computation architecture. For this example, we set the bandwidth of off-chip to on-chip communication to 8 bytes/cycle. On the DRAM accesses plot, the green lines approach the red (ideal) curves, as the on-chip memory grows. More importantly, at lower numerical precisions, we notice the *Mid*-level estimates meeting the corresponding *Coarse*-level estimates at smaller on-chip memory sizes. Moreover, the *Mid*-level's limited information on the computation architecture can still be used to detect bottlenecks in communication and/or on-chip memory size. The *Mid*-level abstracts the details of the computation architecture through the assumption that all processing elements are fully utilized and can always perform computations, if sufficient data is available. In the throughput plot, we observe communication bottlenecks for small on-chip memory sizes, which are not able to provide the ideal computation architecture with enough data to fully utilize it. These communication bottlenecks are more evident for CNN models consisting of multiple fully-connected layers (AlexNet and VGG-16). This behaviour does not change with numerical precision, since smaller bitwidths also increase the ideal computation throughput of vectorized processing elements (*i.e. Coarse*-level estimates get higher).

At the *Fine* abstraction level, the model considers the CNN, the memory hierarchy, and the computation architecture details (register files, dataflow, mapping, etc.). For this example, we used the validated Eyeriss-256-DB from Tab. 1, with varying SRAM sizes. The *Fine*-level blue curves approach the *Mid* and *Coarse* curves at a slower rate, as the on-chip memory increases. This is due to the other limitations of the computation architecture, which the *Fine*-level takes into account (e.g. sub-optimal unrolling, limited register file sizes, etc.). The *Fine*-level provides much more information (as shown in Fig. 2), but for the purpose of highlighting the cross-abstraction level interactions, these are not covered in this section.

The different bitwidth ResNet-18s have different *Coarse* lines (red), which limit the theoretical optimum DRAM accesses and throughput. The *Mid* and *Fine* lines (green and blue), which capture more HW details, never surpass their respective red lines which are defined at the *Coarse* stage. The search at the *Coarse* level provides these theoretical optimal performance levels for a range of mixed-precision CNN quantization strategies, while subsequent levels try to reach that optimum, by parameterizing the HW. For example, if our target was to achieve the theoretical best performance w.r.t. 4-bit *Coarse*, we could either over-dimension our hardware (blue-diamond line at 512KB of on-chip memory) or quantize down to 2-bit and dimension the on-chip memory to 32KB (*Mid* and *Fine* triangle lines of 2-bit touch/surpass the 4-bit *Coarse* line). Both options allow us to reach the theoretical optimum set by *Coarse* for 4-bit, but each option would have a different effect on accuracy, where over-dimensioned HW would achieve higher accuracy due to larger bitwidths, while 2-bit would have lower accuracy but a smaller on-chip memory design.

From Fig. 6, we can clearly see a multi-abstraction *flow* which can help the designer eliminate HW and CNN candidates at earlier stages of the co-design, without having to spend costly GPU hours on training or synthesis and HIL-based testing.

6 EXPERIMENTS

HW-FlowQ is evaluated based on CIFAR-10 [20] and ImageNet [29] datasets for the classification task and Cityscapes [7] for the semantic segmentation task. The 50K train images of CIFAR-10 are used for training and accuracy fitness ψ evaluation, while the 10K test images are used for

final accuracy evaluation at the end of the search. The images have a resolution of 32×32 pixels. ImageNet consists of $\sim$ 1.28M train and 50K validation images with a resolution of 256×256 pixels. The Cityscapes dataset consists of 2975 training images and 500 test images. The images of size 2048×1024 show German street scenes along with their pixel-level semantic labels of 19 classes.

In Sec. 6.1 and Sec. 6.3, we aim to highlight the flexibility of our HW-modeling tool and search approach. To isolate and identify the effects of changing the HW-model on the resultant quantization strategy, we fixed all other variables of the experiment, including the CNN workload (ResNet20). In Sec. 6.2, we focus on understanding the hyper-parameters of NSGA-II and its convergence. Here, we complicate the task by enlarging the quantization search space, and employing a deeper 56-layer CNN. In Sec. 6.4, we apply HW-FlowQ to a different task domain, namely semantic segmentation. We use the DeepLabv3 [4] model to study the effects of layer-wise quantization on the encoder, bottleneck layers (incl. atrous spatial pyramid pooling (ASPP) block), and the decoder layers of the segmentation network. Finally, in Sec. 6.5, we compare our work with state-of-the-art methods of uniform and variable quantization, further testing HW-FlowQ on wide and high resolution CNNs (ResNet18 for ImageNet). If not otherwise mentioned, all hyper-parameters specifying the task-related training were adopted from the CNN's base model and its corresponding quantization method. The first and last layers are kept at 16-bits, following the heuristic of other quantization works [6, 28, 36]. The HW metrics are generated based on the HW configurations described in Tab. 1. The vectorized Spatial-256 HW-model with row-stationary dataflow is used in Sec. 6.5 with additional support for 1-bit (XNOR-Net). As a *Coarse*-level metric, we use fractional operations (Frac. OPs) as measure of CNN computation compression, with respect to the hardware it is executed on. For example, we compute the Frac. OPs of a vectorized accelerator as the layer-wise sum of operations over speed-up due to the respective layer's quantization:

$$Frac.OPs = \sum_{l=0}^{L} \frac{OPs_l}{V^l_{speedup}}. \tag{5}$$

Table 1. Hardware configurations and normalized access energy costs used for experiments and validation.

Specs / HW-Model	PE Array	DRAM Cost	SRAM Size	SRAM Cost	Array Cost	Registers Size (filter, ifmap, psums)	Registers Cost
Spatial-168*	12×14	200	128KB	6	2	224, 12, 16 Words	1
Spatial-256*	16×16	200	256KB	13.84	2	224, 12, 16 Words	1
Spatial-1024*	32×32	200	3072KB	155.35	2	224, 12, 16 Words	1
Eyeriss-1024	32×32	200	3072KB	155.35	2	224, 37, 16 Words	1
Eyeriss-256 - DB	16×16	200	128KB	7.41	0	192, 12, 16 Words	0.99

*: Same dimensioning for bit-serial (BS) and other dataflows (RS, OS, WS)

For experiments on CIFAR-10, we set the population size $|\mathcal{P}|$ to 25 and 50 for exploration and SoTA comparison experiments respectively. The number of generations n is fixed to 50 for all CIFAR-10 experiments. Probabilities for mutation and crossover are set to 0.4 and 1.0 respectively. For ImageNet experiments, $|\mathcal{P}|$ and n are scaled down to 10, while Cityscapes experiments have $|\mathcal{P}| = 25$ and $n = 10$. The CNNs trained on CIFAR-10 are fine-tuned for 2 epochs and their accuracy fitness is evaluated on 10K random samples from the train-set during the search. For ImageNet, we fine-tune for 0.4 epochs before evaluating on the valid-set. For Cityscapes, 10 epochs are necessary to evaluate the candidate population. As explained in Sec. 3.2, the accuracy fitness (ψ) is the GA's measure of the learning capacity of an individual. To avoid artificially biasing the search algorithm towards individuals that perform well on the test set, we keep the accuracy fitness restricted to train or validation set. This way, the framework does not indirectly "see" the test set during the

search. After the search concludes, we fully train the chosen individual from scratch and report its test set accuracy as "Accuracy Top-1" in the result tables.

6.1 A HW-CNN Co-design Example

This experiment serves as a simple example of how the design levels can be used to iteratively narrow down the range of solutions that could suit a potential application. Additionally, we compare SOGA and MOGA (NSGA-II) variants of the search algorithm presented in Sec. 3.2. In a real use case, a set of different CNNs can be considered at the start of the exploration, proposed by a NAS algorithm for example. For simplicity, we start with ResNet20 as our baseline, with task-accuracy of 92.47% on the CIFAR-10 dataset. At the *Coarse*-level, we start the compression with Frac. OPs being our target optimization criterion. Considering a vectorized accelerator, we allow the GA to maintain individuals that have bitwidths which we plan on supporting in our target HW-accelerator. In this example, we restrict $b_A = b_W \in \{16, 8, 6, 4, 2\}$. In Tab. 2, the potential of operation fractionalization w.r.t. the bitwidth restrictions given is around 75% at a task-related accuracy of around 89-90%. Assuming a higher accuracy/fractionalization or lower CNN memory footprint was necessary, we can consider relaxing the condition $b_A = b_W$, allowing the GA to find solutions with varying b_A and b_W values. We can also consider supporting more quantization levels and expand the range $\{16, 8, 6, 4, 2\}$ with intermediate quantization levels and/or binary OPs. This would result in a more fine-grained search that results in new solutions that achieve our desired task-accuracy and CNN memory footprint, at the cost of potentially more complex HW (supporting more b_W and b_A options for example). It is important to note that NSGA-II offers a *range* of Pareto-optimal Frac. OPs vs. accuracy solutions (*Accuracy Pareto-leader*: Top-1 accuracy of 90.70% at 63% φ_{OPs}; *Compression Pareto-leader*: Top-1 accuracy of 89.34% at 77.09% φ_{OPs}).

Table 2. ResNet20 for CIFAR-10 quantized at different abstraction levels of the Spatial-256 HW with SOGA and NSGA-II.

Configuration (< φ >;< **level** >;< $\mathcal{G}$ >)	**Accuracy** Top-1 [%]	**Accuracy** ψ Fitness [%]	**HW-**φ Fitness [%]*	**N. Energy** [$\times 10^7$]	**Latency** [$\times 10^3$cyc.]
Baseline (16 bit)	92.47	-	-	32.84	191
Frac. OPs; Coarse; SOGA	89.28	88.44	79.64	-	-
Frac. OPs; Coarse; NSGA-II	**90.09**	**92.80**	73.09	-	-
DRAM acc.; Mid; SOGA	89.18	91.93	67.79	-	-
DRAM acc.; Mid; NSGA-II	**90.00**	**95.33**	65.56	-	-
N. Energy; Fine; SOGA	89.45	91.18	51.05	16.07	52
N. Energy; Fine; NSGA-II	**90.09**	**94.75**	48.12	17.04	61
Latency; Fine; SOGA	88.44	86.21	78.71	14.94	41
Latency; Fine; NSGA-II	**89.99**	**94.78**	68.77	17.05	59

*: Measured as (1-(Compressed Metric/Baseline Metric))*100

Once satisfied with the CNN's compression potential, the search can be refined to take the off-chip to on-chip memory movement into consideration. Here we can estimate how many processing passes (rounds of communication between off-chip to on-chip) would be necessary to complete all the computations of a CNN. The layer tiling and loop ordering can be searched for different quantization strategies. For this example, we check the *Mid*-level estimates based on DRAM accesses when the on-chip buffer is dimensioned to 256KB. The CNN's DRAM accesses can be reduced by around 65% w.r.t. the 16-bit baseline CNN, while maintaining the same accuracy that was targeted at the *Coarse*-level. Based on the bandwidth of the off-chip to on-chip communication infrastructure considered, this can confirm that our dimensioning of the on-chip buffer is in a good range to achieve a significant reduction of DRAM accesses, without having to *over-quantize* our CNN and

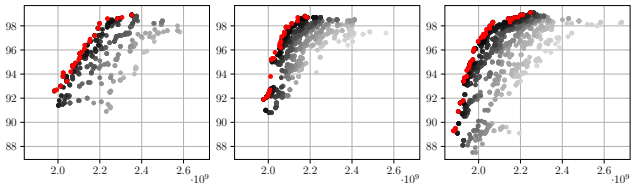
lose the task-related accuracy goal. Finally, the *Fine*-level estimates give us a better understanding of how our CNN can be scheduled on a particular HW-Model. For this example, we proceed with the Spatial-256 configuration presented in Tab. 1, with row-stationary dataflow. Normalized energy can be reduced by around 50%, while maintaining the target Top-1 accuracy from the higher abstraction levels. When considering latency, we observe the drawback of the SOGA approach, not being able to decently balance accuracy and the HW-reward. Although the emerging solution maximizes the latency reward significantly (78.71%), it leads to a considerable accuracy degradation (88.44%). The reward function (Eq. 1) was designed to balance both accuracy and HW-rewards, however, due to the high potential of improving latency through quantization, we find that the SOGA algorithm was willing to sacrifice the train reward ψ (down to 86.21%) to get a much larger overall reward through latency φ_L. This highlights the weakness of handcrafting reward functions to achieve multi-criteria optimization. On the other hand, NSGA-II offers a range of solutions, from which a well-balanced solution is shown in Tab. 2, reducing latency by (68.77%) and maintaining the task-related accuracy since the *Coarse*-level. The Pareto-optimal solutions for latency vs. accuracy range from Top-1: 90.63% at 60.00% φ_L to Top-1: 89.37% at 72.74% φ_L. The set of all Pareto-front solutions is not shown in the table for brevity.

6.2 Multi-Objective Genetic Optimization using NSGA-II

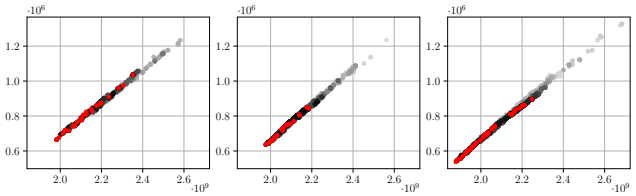
To relieve the burden of designing a fair cost function, facilitate design space exploration, and maintain a diverse set of Pareto-optimal solutions, we leverage NSGA-II as the search technique for the next experiments. We also exploit the multi-objective capabilities to simultaneously construct a Pareto-front which optimizes for task-related accuracy, energy, and latency, concurrently in a single search experiment. To collect more information on the hyper-parameters of the GA, the characteristics of the search space, and the relationship between quantization and the optimization targets, we present the following experiments on the Spatial-256 HW-model with row-stationary dataflow and the deeper ResNet-56 CNN, trained on the CIFAR-10 dataset. As mentioned in Sec. 3, the quantization search space has a size of $|q|^{2L}$, where q is the set of possible quantization levels and L the number of layers. The larger search space of ResNet-56 (5^{54} solutions for $b_A = b_W \in \{16, 8, 6, 4, 2\}$) allows us to verify the scalability of the GA search approach.

Fig. 7 shows 2-D projections of the 3-D Pareto-fronts produced by three experiments, each with a different population size $|\mathcal{P}|$ and/or number of generations n. An increase in generation count (left vs. middle) allows the Pareto-front to take a more convex form. On the other hand, increasing the population size $|\mathcal{P}|$ results in an extended Pareto-front, finely covering a wider surface and a larger hypervolume, however with similar form as the middle configuration. The solutions which are most attractive, are those which offer a trade-off among the optimization criteria. For $(|\mathcal{P}|, n) = (25, 50)$ and $(50, 50)$, we notice that the points which contribute the most to the total Pareto-front hypervolume (lie at the apex of the convex Pareto-front) are comparable in HW-metrics and accuracy fitness. The $(|\mathcal{P}|, n) = (25, 25)$ configuration has solutions of equivalent accuracy fitness, however, their hardware metrics are worse. With these insights, we fix $n = 50$ for all CIFAR-10 experiments to get better convergence. $|\mathcal{P}|$ is set to 25 for exploration experiments and 50 for comparison with state-of-the-art experiments.

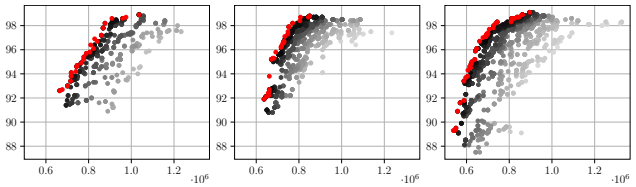
The hypervolume occupied by the 3-D frontier can be measured at each generation to derive the search convergence. As a decision-making technique, we extrapolate a reference point from the polar solutions (worst in each dimension) of the final Pareto-front, and find the farthest solution from it in the frontier, based on Euclidean distance. We refer to this solution as the hypervolume-leader (HV-leader), which offers a balanced trade-off among the Pareto-points of the frontier.



(a) Accuracy Fitness ψ (%) (y-axis), Normalized Energy φ_E (unitless) (x-axis)



(b) Latency φ_L (Cycles) (y-axis), Normalized Energy φ_E (unitless) (x-axis)



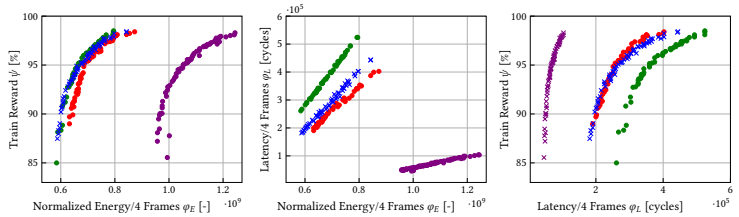
(c) Accuracy Fitness ψ (%) (y-axis), Latency φ_L (Cycles) (x-axis)

Fig. 7. 2-D projections of three 3-D Pareto-fronts for ResNet56 quantization: left to right ($|\mathcal{P}|$, n) = (25, 25), (25, 50), (50, 50). Grey to black shades represent Pareto-fronts of older to newer generations, red points belong to the final Pareto-front.

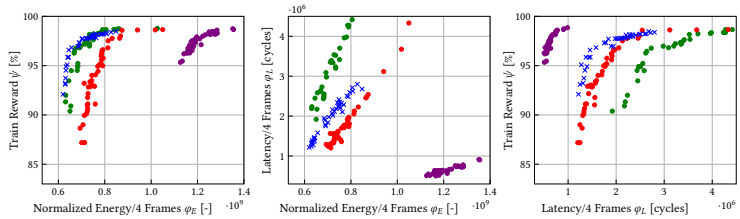
6.3 HW Modeling and Exploration

6.3.1 Quantization on Different HW Dimensions. In this experiment, three candidate HW-accelerators are modeled to observe the effect of HW-dimensioning (computing units, buffer sizes and memory access costs) on quantizing ResNet20 for CIFAR-10. Fig. 8(a) shows three 2-D projections of four 3-D Pareto-fronts optimizing task-related train reward ψ, normalized energy φ_E and processing cycle latency φ_L. φ_E and φ_L are measured for processing 4 frames, to compare with a batch-processing HW-model.

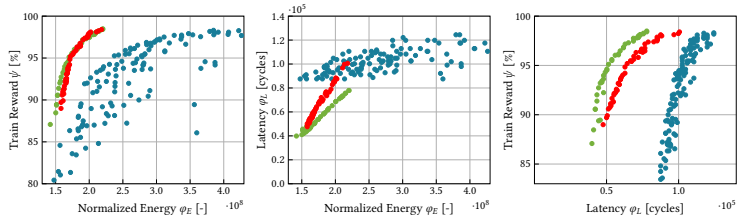
The three differently dimensioned spatial compute arrays (details in Tab. 1) show similar characteristics in the shape and form of their Pareto-fronts. A slight difference can be observed for Spatial-1024, where its Pareto-front has a narrower latency range w.r.t. different quantization strategies. This indicates that the loop unrolling capacity is already exploited at a high degree due to the large PE-array, and cannot be improved much further through quantization. This hints to Spatial-1024 being slightly over-dimensioned for the task. On the other hand, Spatial-168 and 256 HW-models show a wider range of solutions for φ_E and φ_L, leaving more room for exploiting quantization to meet a given constraint for the CNN under consideration (ResNet20). In all three plots, a gap can be noticed between the 1024 model and the others, indicating a good potential for testing a model dimensioned in between (e.g. 512 PE array).



(a) HW-Dimensioning and ψ, φ_L, φ_E potential: Spatial 168 Spatial 256 Spatial 1024 Spatial 256-Batch4



(b) Mixed-precision arch.: BitSerial-168 BitSerial-256 BitSerial-1024 BitSerial-256-Batch4



(c) Dataflow exploration: Output Stationary Row Stationary Weight Stationary

Fig. 8. 2-D projections of 3-D Pareto-fronts of 3 exploration experiments on ResNet20 for CIFAR-10 for HW dimensioning, bit-serial processing and dataflow variants.

Increasing the batch size to 4 on the Spatial-256 configuration shows an improvement in terms of energy, bringing the Spatial-256 configuration closer to the energy efficiency of Spatial-168, while maintaining the latency of the single-batch Spatial-256 configuration. This can be seen in Fig 8(a).

In Tab. 3, we show the results of the baseline 16-bit ResNet20 executing on the 3 hardware configurations and a larger batch-size. Maintaining our desired accuracy threshold of 90%, we choose solutions from the Pareto-front of each hardware configuration. We can see that with this accuracy requirement, the Spatial-1024 configuration can achieve a very low latency with respect to other configurations, without over-quantizing the CNN. This comes at the cost of more energy required by the execution, due to larger memories and more expensive access costs (see Tab. 1). We notice that for the small Spatial-168 hardware configuration, the latency is the highest, as it needs to maintain an accuracy of 90% (i.e. cannot over-quantize), while processing on fewer PEs. Nevertheless, the smaller design reduces the energy requirements of the execution. Finally, taking a look at Spatial-256 with batch-size 4, we notice an improvement in latency and energy, due to better reuse of the weights w.r.t. the batched inputs, bringing the energy of the execution close to the small Spatial-168 accelerator.

Table 3. Quantization of ResNet20 for CIFAR-10 on different HW-Dimensions.

Configuration (< **choice** >;< μ >;< **batch** >)	**Accuracy** Top-1 [%]	**Accuracy** ψ Fitness [%]	**N. Energy** φ_E / 4 Frames [$\times 10^7$]	**Latency** φ_L / 4 Frames [$\times 10^3$cyc.]
Baseline (16-bit); Spatial-168; 1	92.47	-	130.28	1128
Baseline (16-bit); Spatial-256; 1	92.47	-	131.36	764
Baseline (16-bit); Spatial-1024; 1	92.47	-	190.40	204
Baseline (16-bit); Spatial-256; 4	92.47	-	119.94	764
Pareto-Choice; Spatial-168; 1	90.31	94.22	64.80	343
Pareto-Choice; Spatial-256; 1	90.26	96.31	72.68	282
Pareto-Choice; Spatial-1024; 1	90.25	95.08	105.41	68
Pareto-Choice; Spatial-256; 4	90.03	95.19	66.09	259

Table 4. Quantization of ResNet20 for CIFAR-10 on Bit-Serial Accelerators.

Configuration (< **choice** >;< μ >;< **batch** >)	**Accuracy** Top-1[%]	**Accuracy** ψ Fitness [%]	**N. Energy** φ_E / 4 Frames [$\times 10^7$]	**Latency** φ_L / 4 Frames [$\times 10^3$cyc.]
Baseline (16-bit); BS-168; 1	92.47	-	141.07	20365
Baseline (16-bit); BS-256; 1	92.47	-	147.65	12296
Baseline (16-bit); BS-1024; 1	92.47	-	208.26	3326
Baseline (16-bit); BS-256; 4	92.47	-	137.57	12296
Pareto-Choice; BS-168; 1	90.17	94.90	68.36	2468
Pareto-Choice; BS-256; 1	90.37	92.91	75.58	1406
Pareto-Choice; BS-1024; 1	90.19	96.95	116.63	563
Pareto-Choice; BS-256; 4	90.33	92.10	62.01	1216

6.3.2 Quantization on Bit-Serial Mixed-Precision Accelerators. So far, vectorized accelerators have been considered, which support $b_A = b_W \in \{16, 8, 6, 4, 2\}$. In this experiment, we change the underlying computation unit to observe the benefits that can be achieved for a bit-serial accelerator which supports any $b_A, b_W \leq 16$ for any layer. All PEs have 16 computation lanes to allow for higher throughput and partially compensate for the slower, serialized operations. The results of this experiment are shown in Fig. 8(b).

An interesting difference can be observed when changing the dimensioning of the accelerator. A larger bit-serial accelerator produces an even more compact Pareto-front, due to its ability to maximize loop unrolling over the large PE-array, extended further with the computation lanes. For the energy/latency graph (middle) more solutions appear for a particular energy and/or latency, breaking the almost linear relationship between optimal energy and latency mapping observed for vectorized accelerators (Fig. 8(a)). This can be attributed to both the change in compute architecture and the variations possible for both b_A and b_W. In Tab. 4, we observe similar latency and energy trends for bit-serial computation, as with vectorized computation for batch-size 1. A Pareto-optimal solution is chosen for each hardware and trained to achieve an accuracy above 90%. The smallest BS-168 is the slowest, yet the most energy efficient, while BS-1024 significantly reduces the latency at the cost of more energy for data movement. The improved effect of batch processing is more prominent for bit-serial accelerators. The 256-PE bit-serial accelerator with batch processing offers a significant improvement in terms of energy, bringing the 256-PE configuration to better energy efficiency than the smaller 168-PE counterpart executing batch-size 1 inputs. Additionally, latency also gets a decent improvement of 13.5%. This shows the advantages of relaxing the $b_A = b_W$ constraint on the HW and the GA search.

To further analyze this aspect, we study the layer-wise quantization strategy chosen by the GA for batch sizes 1 and 4 on bit-serial accelerators. Layers with large activation volumes can have lower bitwidth activations (low b_A), while the weights can be kept at a slightly higher bitwidth (higher b_W). The opposite can be done for layers with large filter volumes. This extends the improvements to be gained on mixed-precision accelerators and larger batch sizes (*i.e.* larger activation volumes). In Fig. 9, the quantization strategy chosen for batch size of 4 reflects the GA's attempt to compress

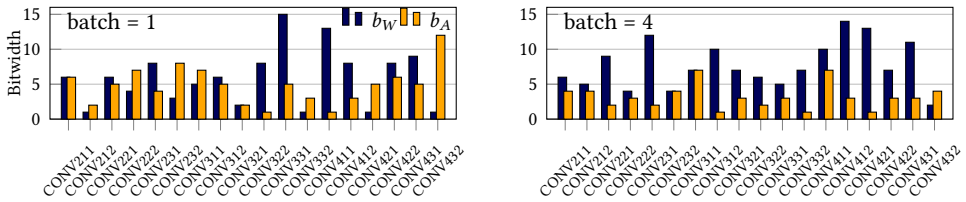


Fig. 9. Layer-wise bitwidth strategy for BS-256 hardware. Batch size 1 (left) and 4 (right). NSGA-II compensates for larger activations (batch=4) by lowering b_A and maintains accuracy by increasing b_W, when compared to batch=1 inference.

the large activations more aggressively than for batch size of 1 (particularly for the first half of the CNN), while increasing the weight bitwidths b_W to maintain accuracy. The resulting quantized CNNs of both batch 1 and batch 4 have an equivalent accuracy (~90.3%), but with a noticeable improvement in HW-metrics for batch size of 4, due to the GA taking the capabilities of the HW into account.

Table 5. Quantization of ResNet20 for CIFAR-10 on Different Dataflows.

Configuration (< `choice` >;< μ >;< `batch` >)	**Accuracy** Top-1[%]	**Accuracy** ψ Fitness [%]	**N. Energy** φ_E [$\times 10^7$]	**Latency** φ_L [$\times 10^3$cyc.]
Baseline (16-bit); Fine; OS-256	92.47	-	46.33	159
Baseline (16-bit); Fine; WS-256	92.47	-	69.40	166
Baseline (16-bit); Fine; RS-256	92.47	-	32.84	191
HV-Leader; Fine; OS-256	89.99	93.98	16.36	48
HV-Leader; Fine; WS-256	89.11	90.87	20.54	97
HV-Leader; Fine; RS-256	89.65	95.19	17.20	64

6.3.3 Quantization on Different Dataflows. To demonstrate the effect of quantization on dataflows, a weight stationary (WS) dataflow and an output stationary (OS) dataflow are presented. WS unrolls computations in dimensions C_o and C_i over the processing element array, while OS unrolls H_o and W_o, and replicates the unrolling over C_o. Both WS and OS support channel interleaving in order to maximize their register utilization, similar to the row stationary (RS) dataflow.

The baselines in Tab. 5 show RS is the most energy-efficient, while OS offers the best latency. WS is placed in the middle in terms of latency but has worse energy efficiency when compared to the other considered dataflows. The Pareto-fronts of quantization strategies in Fig. 8(c) demonstrate the effect of dataflows on three accelerators, which are otherwise identical in dimensioning. WS proves to be highly sensitive to quantization, having many unique non-dominated combinations of φ_E, φ_L and ψ. Generally, WS is the least efficient in terms of latency and energy, for a particular train accuracy ψ. OS dataflow enjoys its lead in latency, due to a higher potential of unrolling as a result of quantization over vectorized PEs (each vectorized PE acts as V_{speedup} virtual PEs). Consequently, its energy rivals that of RS. The higher parallelism degree on a single SIMD-vector engine reduces the total cost of MAC operations. Furthermore, since the loop unrolling is taking place across the array as well as within the vectorized PEs, fewer PE-to-PE hops are required to achieve the unrolling of the mapper, resulting in less array data movement energy.

6.4 Mixed-Precision Quantization for Semantic Segmentation

The semantic segmentation task is critical to applications in robotics and autonomous driving. High-quality segmentation can be more computationally complex by several orders of magnitude, when compared to classification tasks (see Tab. 6). This is related to both, the typically larger input

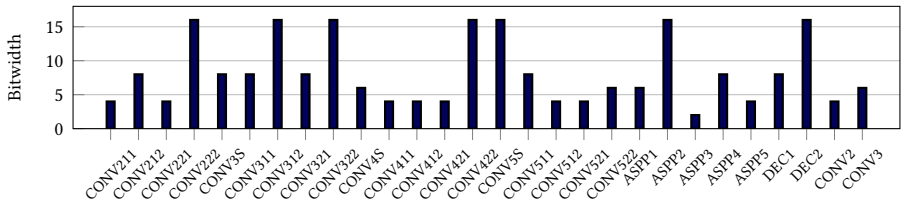


Fig. 10. Layer-wise bitwidths (b_W=b_A) of a DeepLabv3 Pareto-choice strategy with 67.3% mIoU on Cityscapes. Short and parallel layers have b_A equal to their respective bottom layer.

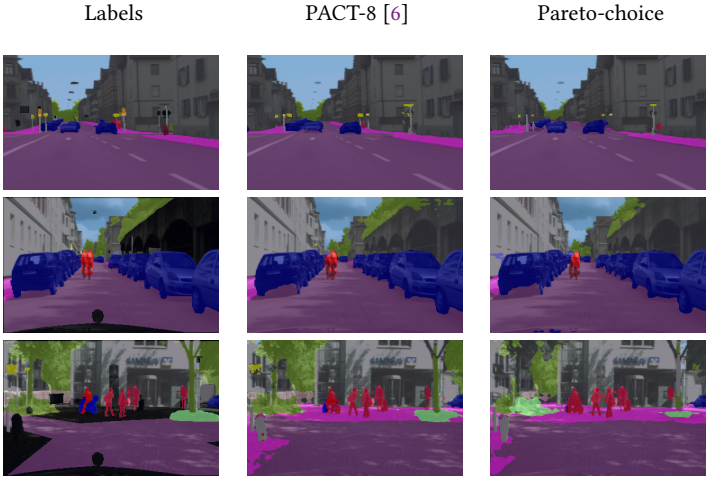


Fig. 11. Qualitative results of DeepLabv3 quantization on Cityscapes scenarios. Black regions have no ground-truth labels. Pareto-choice has 21.6% compression compared to uniform 8-bit PACT.

image resolution and the additional layers needed for semantic segmentation (bottleneck, ASPP block and decoder layers).

For the DeepLabv3 network executing on Eyeriss-1024 (details in Tab. 1), HW-FlowQ must adapt to the task's training challenges, particularly on low-bitwidth (≤4-bit) configurations for PACT quantization, which often lead to exploding gradients. Despite this difficultly of PACT, HW-FlowQ produced the Pareto-choice candidate shown in Fig. 10, which achieved 67.30% mean intersection over union (mIoU) with a 21.6% reduction in fractional operations over uniform 8-bit PACT quantization of DeepLabv3 (shown in Tab. 6). In Fig. 11, we show qualitative results in semantic predictions between uniform 8-bit PACT and our HW-FlowQ Pareto-choice, for three example scenes in the Cityscapes dataset. These results show an impressive potential of mixed-precision low-bitwidth quantization on complex semantic segmentation tasks, potentially with more advanced quantization techniques under HW-FlowQ in future work.

6.5 Comparison with State-of-the-Art

In this section, we compare HW-FlowQ against state-of-the-art quantization approaches on shallow, deep, and wide CNNs for classification and semantic segmentation tasks. DoReFa-Net [36] and PACT [6] indicate uniform quantization with the respective method. We found that PACT propagates the gradients better during training for deeper and wider networks; therefore, we compared against it for ResNet-56, ResNet-18 and DeepLabv3. We reimplemented the work in HAQ [32] and adapted the reward in Eq. 1 to the reinforcement learning (RL) agent. The RL-agent was integrated with our HW-model μ and thoroughly tested with different agent hyper-parameters to achieve the

convergence behavior depicted in [32]. We found that the agent was achieving a better result when optimizing for latency, as such, we quote the respective results. Finally, XNOR-Net is presented as a binary neural network variant to compare with a highly HW-efficient implementation.

For ResNet-20, our HV-leader provides energy and latency reductions of 48% and 69% while maintaining a Top-1 accuracy of 90.15%. In contrast, HAQ achieved a reduction of 42% and 57% for energy and latency. For ResNet-56, our HV-leader achieved equivalent energy and latency to PACT

Table 6. Comparison of HW-FlowQ with state-of-the-art quantization methods on Eyeriss-256 Vectorized.

Model/ Dataset	Method	Accuracy Top-1 [%]	F.Ops [$\times 10^6$]	N. Energy [$\times 10^7$]	Latency [$\times 10^3$cycles]
ResNet20 CIFAR-10	Baseline (16-bit)	92.47	41	33	191
	DoReFa-Net (4-bit) [36]	89.75	10	16	51
	DoReFa-Net (2-bit) [36]	87.16	5	14	43
	XNOR-Net (1-bit) [28]	83.98	3	15	17
	HAQ [32]	89.75	17	19	83
	HV-Leader [Ours]	**90.15**	**12**	**17**	**60**
ResNet56 CIFAR-10	Baseline (16-bit)	93.89	125	101	588
	PACT (4-bit) [6]	90.96	32	48	155
	PACT (2-bit) [6]	90.43	16	42	80
	XNOR-Net (1-bit) [28]	85.61	8	47	48
	HAQ [32]	92.07	56	61	279
	HV-Leader [Ours]	**92.00**	**30**	**49**	**154**
ResNet18 ImageNet	Baseline (16-bit)	69.01	1814	1489	9498
	PACT (4-bit) [6]	66.59	542	822	3070
	PACT (2-bit) [6]	63.59	330	703	1897
	XNOR-Net (1-bit) [28]	52.51	224	676	1230
	HV-Leader [Ours]	**67.02**	551	821	3191
DeepLabv3 CityScapes	Baseline (16-bit)*	69.68	147367	140057	273588
	PACT (8-bit) [6]*	69.95	76028	92035	134395
	XNOR-Net (1-bit) [28]*	58.51	13606	39148	71559
	Pareto-Choice [Ours]*	67.30	59616	87475	121422

*: Executed on Eyeriss-1024

(4-bit), while maintaining better accuracy. Our HV-Leader achieves an improvement of 20% and 45% over HAQ for energy and latency, at an equivalent Top-1 accuracy of 92%. The improvements over HAQ go beyond these results. The reinforcement learning-based approach in HAQ has the same limitations as SOGA, namely the aggregation of multiple criteria into one cost function. Our MOGA approach through NSGA-II inherently supports multi-criteria optimization. The designer does not need to handcraft a reward function which fairly captures all the optimization targets in one reward value. For the ImageNet experiment, a HV-leader with an accuracy of 67.02% and HW-estimates comparable to PACT (4-bit) was achieved in only $n = 10$ and $|P| = 10$.

7 CONCLUSION

HW-FlowQ optimizes CNNs by finding quantization strategies based on high-fidelity HW-model-in-the-loop setups. Abstraction levels and design phases inspired by VLSI design flows help in systematically narrowing down hyper-parameters for both the CNN and HW-design, exposing HW-CNN co-design synergies. Exploring vectorized and bit-serial compute engines, we exploit the performance trade-offs for different mixed precision workloads. We demonstrate the effectiveness of NSGA-II, which offers a Pareto-optimal set of quantization strategies for different HW-models during the optimization process. As future work, the proposed framework and the HW-model can be reused to investigate other compression techniques, such as pruning. The genomes can be reformulated to assign a sparsity rate for each layer, which could further extend the compression

capabilities of the co-design framework. Compared to existing mixed-precision methods, our GA-based quantization improves the latency of ResNet20 and ResNet56 by 49% and 45%, respectively, with equivalent prediction accuracy.

REFERENCES

[1] Mohamed S. Abdelfattah, undefinedukasz 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 *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 192, 6 pages.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *ArXiv* abs/1308.3432 (2013).

[3] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.* 11, 3, Article 16 (Dec. 2018), 23 pages. https://doi.org/10.1145/3242897

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking Atrous Convolution for Semantic Image Segmentation. arXiv:1706.05587 [cs.CV]

[5] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[6] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *ArXiv* abs/1805.06085 (2018).

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha. 2019. ChamNet: Towards Efficient Network Design Through Platform-Aware Model Adaptation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11390–11399.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[10] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.

[11] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Xi'an, China). ACM, New York, NY, USA, 751–764. https://doi.org/10.1145/3037697.3037702

[12] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 807–820. https://doi.org/10.1145/3297858.3304014

[13] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[14] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *The European Conference on Computer Vision (ECCV)*.

[15] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 10–14.

[16] Qiangui Huang, Shaohua Kevin Zhou, Suya You, and Ulrich Neumann. 2018. Learning to Prune Filters in Convolutional Neural Networks. *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018), 709–718.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research (JMLR)* 18, 1 (Jan. 2017), 6869–6898.

[18] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[19] 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. https://doi.org/10.1109/TCAD.2020.2986127

[20] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. University of Toronto.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[22] Yujun Lin, Driss Hafdi, Kuan Wang, Zhijian Liu, and Song Han. 2019. Neural-Hardware Architecture Search. In *NeurIPS Workshop*.

[23] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo. 2018. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 7 (2018), 1354–1367.

[24] P. Meloni, D. Loi, G. Deriu, A. D. Pimentel, D. Sapra, B. Moser, N. Shepeleva, F. Conti, L. Benini, O. Ripolles, D. Solans, M. Pintor, B. Biggio, T. Stefanov, S. Minakova, N. Fragoulis, I. Theodorakopoulos, M. Masin, and F. Palumbo. 2018. ALOHA: An Architectural-Aware Framework for Deep Learning at the Edge. In *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications* (Turin, Italy) *(INTESA '18)*. Association for Computing Machinery, New York, NY, USA, 19–26. https://doi.org/10.1145/3285017.3285019

[25] G. De Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti. 1987. *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*. Martinus Nijhoff Publishers.

[26] S. Narang. 2016. DeepBench - Baidu Research. https://github.com/baidu-research/DeepBench.

[27] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *The European Conference on Computer Vision (ECCV)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 525–542.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *ACM International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[30] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. 2018. Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 20, 6 pages. https://doi.org/10.1145/3195970.3196072

[31] F. Vahid and T.D. Givargis. 2001. *Embedded System Design: A Unified Hardware/Software Introduction*. Wiley.

[32] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[33] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[34] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search. *CoRR* abs/1812.00090 (2018). arXiv:1812.00090 http://arxiv.org/abs/1812.00090

[35] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 369–383. https://doi.org/10.1145/3373376.3378514

[36] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. arXiv:1606.06160 [cs.NE]