Journal of Signal Processing Systems (2026) 98:42
https://doi.org/10.1007/s11265-026-02012-w

RESEARCH

Joint Exploration of Neural Networks and Systolic Hardware for Improved AI Accelerator Performance

Annina Gutermann[1] · Alexey Serdyuk[1] · Foivos Paraskevas[1] · Hella Toto Kiesa[1] · Fabian Lesniak[1] · Jakob Schwarz[1] · Michael Hartmann[1] · Tanja Harbaum[1] · Juergen Becker[1]

Received: 20 March 2026 / Revised: 16 August 2026 / Accepted: 20 August 2026

Abstract

When executing common Neural Networks (NNs) on custom AI accelerators, the high performance suggested by advertised Giga or Tera Operations per Second (GOPS/TOPS) is typically not achieved, as low hardware utilization often leads to an effective performance in the single-digit percentage range of the theoretical peak. This discrepancy arises as NNs are typically designed without accounting for the target hardware, leading to inefficient mappings and software optimizations that fail to deliver the expected gains. Addressing this, we present a hardware-aware workflow that combines accurate latency modeling and design space exploration to optimize both neural network architectures and the underlying systolic-array-based accelerator. We develop and validate two high-precision latency models for two different Row-Stationary (RS) dataflows on our target accelerator. Using these models together with a structured search space generation, we generate Pareto-optimal search spaces in terms of achieved GOPS and latency for a given hardware target, and use these for a Bayesian Bayesian Hardware-Aware Neural Architecture Search. We further explore the accelerator design itself in a subsequent hardware DSE stage, varying the PE array dimensions and clock ratio to identify hardware configurations that maximize efficiency and minimize inference latency for each network and dataflow. We demonstrate our approach on ResNet-like networks. On the original hardware, the discovered ResNet-50-like architecture achieves an 85% relative increase in hardware utilization, reduces latency by 21% and parameter count by 18%, and maintains baseline ImageNet accuracy. On the optimized hardware, latency and area are further reduced while efficiency is increased by up to 94%, with up to 20% fewer PEs compared to the baseline configuration. For ResNet-34, similar trends are observed, with latency reductions exceeding 33% and efficiency gains up to 43%. To enable reproducibility, we open-source our complete workflow of latency models, search space generation, NAS and training pipeline.

Keywords Performance predictors · Neural architecture search · Hardware acceleration · Search Space Design

✉ Annina Gutermann
gutermann@kit.edu

Alexey Serdyuk
alexey.serdyuk@kit.edu

Foivos Paraskevas
foivos.paraskevas@kit.edu

Hella Toto Kiesa
hella.kiesa@kit.edu

Fabian Lesniak
lesniak@kit.edu

Tanja Harbaum
harbaum@kit.edu

Juergen Becker
becker@kit.edu

[1] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

1 Introduction

As Moore's Law and Dennard scaling continue to plateau, the performance improvements of General Purpose Processors (GPPs) are no longer sufficient to satisfy the requirements of modern data-intensive workloads, such as deep learning applications. Although Graphics Processing Unit have long been employed to advance AI performance, their high energy consumption makes them unsuitable for power-constrained scenarios, including embedded systems. Consequently, the focus is increasingly shifting toward heterogeneous systems that incorporate application-specific hardware accelerators, such as systolic array based architectures, which offer high throughput, energy efficiency, and

Published online: 03 September 2026

predictable data movement for dense matrix multiplications and convolutions [2].

However, the performance of systolic array accelerators is highly sensitive to the alignment between NN workload characteristics and hardware constraints. For example, operations optimized for GPUs, such as depthwise separable convolutions, often perform poorly on custom accelerators due to mismatches in dataflow strategies and microarchitectural details. Standard neural networks are typically designed without explicit consideration of the target hardware, therefore leading to inefficient mappings and software optimizations that fail to deliver the expected performance improvements. Recent studies increasingly indicate that no single NN architecture delivers optimal performance across all hardware platforms [3].

This motivates the joint optimization of neural network architectures and hardware characteristics in a resource-efficient manner. A widely adopted approach is Hardware-Aware Neural Architecture Search (HW-NAS) [3, 4], which typically employs performance predictors or proxy models to quickly estimate metrics such as accuracy and latency on a target platform. However, many existing predictors show estimation errors in the range of 20–30% [5], limiting reliable identification of optimal design points. Therefore, accurate and efficient latency estimation of convolutional operators for a given hardware target becomes a key prerequisite for effective HW-NAS. Beyond adapting the network architecture alone, FPGA-based accelerators provide additional flexibility by enabling adjustments to the hardware architecture itself. This reconfigurability, combined with HW-NAS, thus allows for an exploration of both the hardware configuration and the network structure for an optimal system performance.

In this work, we use FleXNNgine [6], an open-source FPGA-based systolic array accelerator that implements two variants of a RS dataflow [7] for convolution acceleration. We develop high-precision latency estimation models for both dataflows and propose a latency-centric methodology for generating a hardware-optimized search space for HW-NAS, as shown in Figure 1. To further refine the hardware configuration, we perform a Design Space Exploration (DSE) using a staged optimization flow, adjusting the Processing Element (PE)-array geometry and PE-array clock frequency to identify optimal configurations for the target networks and dataflows. To evaluate our approach, we perform a multi-objective Bayesian Neural Architecture Search (NAS) guided by a zero-cost accuracy proxy and the developed latency models. The results demonstrate that this methodology yields improved NNs with fewer parameters, reduced inference latency and reduced hardware cost, all while preserving similar accuracy.

2 Related Work

2.1 Latency Models for Hardware Architectures

Maximizing operations per latency necessitates a rigorous architecture exploration of the targeted hardware, which is typically completed through latency models. We categorize these into: empirical models requiring physical hardware, cycle-accurate models that are computationally intensive, and accelerated models derived from analytical methods or based on a given dataflow. A fundamental trade-off exists between simulation speed and precision: accelerated models often sacrifice accuracy for reduced execution time, whereas slower models provide higher fidelity.

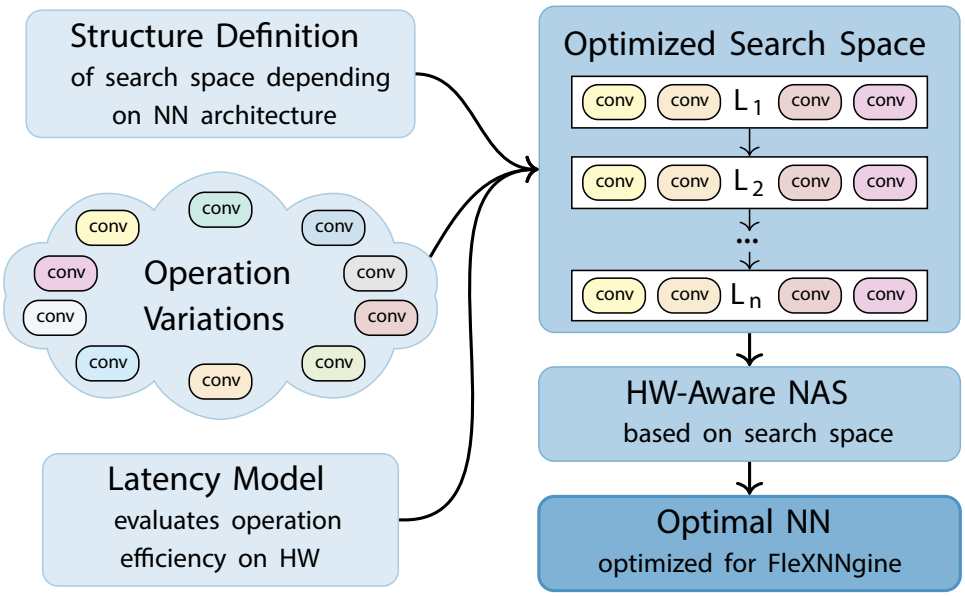


Fig. 1 Overview of our proposed workflow to generate an optimized NN for a given hardware platform. The search space depends on the target network structure, the operations supported by the hardware, and their respective latencies [1].

ZigZag [8] represents a latency-aware DSE framework for DNN accelerators, employing an intra-layer analytical model to estimate latency with 94.3% accuracy [9]. This framework enables rapid, simulation-free exploration of design variables such as loop tiling and memory hierarchy configurations. Timeloop [10] utilizes a template-driven approach to simulate spatial and temporal mappings across compute and memory structures, achieving a mean accuracy of 95%. Parallel tools include SCALE-Sim [11] for cycle-accurate performance estimation and MAESTRO [12], which focuses on modeling memory access patterns and their subsequent performance bottlenecks.

A white-box methodology, where internal hardware mechanics are explicitly defined, can surpass the fidelity of black-box or empirical strategies. In this work, we explicitly model the fixed hardware architecture and dataflow, identifying convolutional parameters (e.g., kernel size, stride, and channel dimensions) as the primary latency determinants. Our specialized latency model, validated against FPGA runtime measurements, achieves a near-perfect accuracy of 99.997% and 98.7% for the two RS dataflows, respectively, while maintaining high evaluation speed, facilitating both highly precise and fast performance forecasting.

2.2 Search Spaces in Neural Architecture Search

The definition of the search space is critical for the performance of the final model, as it delineates the set of attainable NN architectures. Early NAS methodologies focused exclusively on predictive accuracy, frequently yielding computationally intensive networks that were unsuitable for deployment in resource-constrained environments.

To address the requirements of practical deployability, contemporary research emphasizes the trade-off between accuracy and hardware efficiency, an approach called HW-NAS. Tan et al. [13] and Cai et al. [14] pioneered the integration of empirical hardware metrics, such as inference latency, directly into the optimization objective. Building upon these foundations, modern HW-NAS paradigms increasingly adopt cell-based NAS, which optimizes modular, reusable blocks that are subsequently stacked to construct the complete NN topology.

A significant portion of Convolutional Neural Network (CNN) optimization focuses on the convolution structure, which constitutes the primary computational bottleneck. While prior research has largely concentrated on square kernel dimensions or specific convolution types (e.g., standard vs. depthwise separable) [15], parameters such as input resolution and channel depth are often omitted to avoid prohibitive search space expansion. Conversely, Wang et al. [16] investigated asymmetric kernels (e.g., 1×3 or 1×5) within a multi-objective evolutionary search, demonstrating that such configurations can reduce computational overhead without compromising accuracy.

Zeng et al. [17] introduced an accelerator-aware NAS framework by synthesizing a unified search space from three manually designed configurations based on ResNet [18], MobileNet [19], and VGG [20]. By employing black-box profiling on the target accelerator to assess the accuracy-latency trade-off, the authors pruned the search space and utilized both reinforcement learning and differentiable NAS techniques to derive highly efficient models.

2.3 Search Methodology: Black-box Optimization with ZC-proxies

NAS search strategies are broadly categorized into black-box and one-shot methodologies [21, 22]. Black-box optimization treats the search space as a non-transparent objective function, utilizing surrogate models (e.g., Bayesian optimization) or heuristics (e.g., genetic algorithms, reinforcement learning) to guide architecture selection substituting gradient information. While versatile across diverse tasks and domains, these approaches incur significant computational overhead, as each candidate architecture necessitates independent training.

In contrast, one-shot methods involve training a single supernet with shared weights across all included candidate architectures, facilitating rapid evaluation through weight inheritance. This paradigm substantially reduces computational requirements compared to exhaustive individual training. Prominent state-of-the-art implementations include Differentiable Architecture Search (DARTS) [23] and Once-for-All (OFA) [24].

However, frameworks such as DARTS and OFA are incompatible with our defined search space, as their channel dimensions diverge from our analytically derived optimal values, which would necessitate the resource-intensive training of a new supernet. To circumvent this, we utilize Zero-Cost Proxies (ZC-proxies), which estimate performance metrics via surrogate indicators without requiring model training. Specifically, we employ proxies from AZ-NAS [25], which have demonstrated superior performance over state-of-the-art ZC-proxies on the ImageNet16-120 subset of the NAS-Bench-201 benchmark [26].

2.4 Neural Architecture and Accelerator Co-Search

The approaches discussed so far focus on accelerator DSE frameworks optimizing hardware for a given NN, whereas HW-NAS optimizes the NN for a given accelerator. Co-search removes this asymmetry by treating network and accelerator parameters as one design space. Jiang et al. [27] established this direction with a reinforcement-learning

co-exploration that couples architecture search to FPGA implementation feasibility. Subsequent work made the joint search tractable by making it differentiable: DNA [28] formulates a generic accelerator design space covering micro-architecture and mapping that is differentiable alongside an FBNet-style network space, and Auto-NBA [29] extends the joint space to include per-layer bitwidths, addressing the resulting drastic memory increase with a heterogeneous sampling scheme. NAAS [30] instead searches accelerator micro-architecture, mapping, and network in nested evolutionary loops, while CODEBench [31] couples a large CNN space with a simulator-based accelerator space. Common among these frameworks is that the accelerator is instantiated from a parameterized template and evaluated through an analytical or simulation-based cost model, so the searched hardware is a design point rather than an existing implementation.

The two most closely related studies to this work follow this template-based paradigm. NAF [32] and its journal extension [33] differentiably search network operations and block-level INT4/INT8 bitwidths jointly with heterogeneous multi-core FPGA accelerators and their mappings, over a layer-wise search space derived from FBNet [34] and an analytical model of their accelerator template. The searched pairs are implemented on a Xilinx ZCU102 and an Alveo U50. UniCoS [35] instead separates the problem into two stages: candidates from the ProxylessNAS and AutoFormer spaces are first ranked by a training-free proxy, and the layers of the selected network are then clustered into cores whose dataflows and mappings are searched. Hardware metrics are taken from the MAESTRO [12] cost model with Energy-Delay Product (EDP) as the objective, and the resulting accelerators are not implemented in RTL or on a device.

Our workflow differs from these frameworks in four aspects. First, both studies build their accelerators and search spaces around depthwise separable blocks. Such operators provide little reuse across channels and map poorly onto dense systolic arrays [36], and they are not part of the ResNet structure. Second, both derive an accelerator from a parameterized template that is defined together with the search, while we take the existing open-source RTL design [6] as fixed and vary only the parameters it exposes. The third, lies in the fidelity of the hardware feedback. Co-search frameworks guide the search with analytical models of their template, whose accuracy is either inherited from cost simulators such as MAESTRO, reported at 90–95 % [12], or not quantified, whereas our white-box model of a fixed data path is validated against on-board FPGA measurements to a Mean Absolute Percentage Error (MAPE) of $0.023\,\%$ for the Spatial Row-Stationary (SRS) and $1.3\,\%$ for the Temporal Row-Stationary (TRS) dataflow. Lastly, UniCoS optimizes EDP, NAF maximizes throughput under resource constraints, while we treat latency and PE-array efficiency as the objectives of the hardware DSE and combine latency with a training-free accuracy estimate in the subsequent NAS.

2.5 Summary and Contributions

Our work advances hardware-aware NAS by jointly optimizing both the NN architecture and the accelerator architecture in a staged co-design pipeline, whereas prior approaches treat one or the other as fixed (see Table 1).

While frameworks like Timeloop [10] and ZigZag [8] provide fine-grained latency models for fixed neural networks, our latency model simulates GOPS efficiency to map hardware and NN candidates under realistic constraints without requiring hardware deployment. Unlike HW-aware NAS approaches such as [37] and [38], which compute hardware metrics for fixed accelerators, our approach reconfigures PE array, dataflow and clock ratio alongside network parameters.

3 Hardware-aware CNN Design Space

3.1 Latency Prediction Model of Baseline Architecture

This work requires an existing accelerator architecture that efficiently computes convolutional layers and is available as open source to allow for detailed latency analysis. The systolic array-based accelerator *FleXNNgine* [6] meets these requirements and therefore serves as the baseline architecture in this work. Figure 2 provides an overview of the accelerator, which comprises a control unit, a scratchpad memory and a PE array. The number of PEs and their rectangular layout can be freely configured. The PE array is connected to the scratchpad via FIFOs, while the weight and input activation FIFOs are filled by a Round Robin (RR) arbitrator. The scratchpad interface supports a configurable word size to enable parallel byte loading. In this work, a scratchpad word size of 8 is used. The design employs multiple clock domains, with the scratchpad operating at a higher frequency than the PE array to mitigate stalling caused by memory bandwidth limitations. Clock domain crossings are indicated by grey dashed lines in Figure 2. The clock frequencies can be adjusted with a trade-off of higher possible Giga Operations per Second (GOPS) with increased PE array clock against an increased probability of stalling and an increased energy demand.

On the FleXNNgine, convolutions are calculated using the Row-Stationary (RS) dataflow [7], which is referred

Table 1 Comparison of related accelerator DSE, HW-NAS, and co-search frameworks with respect to optimization objective, search space, hardware modeling methodology and reported model accuracy. The last two columns indicate whether the NN and the accelerator are held fixed, searched, or reconfigured during optimization.

Framework	Model Acc.	Modelling Methodology	Objective	Search Space	DNN	HW
ZigZag [8, 9]	94.3%	Fine-grained analytical model	Energy, Performance, Area	MAC array, memory hierarchy	fixed	reconfigured
Timeloop [10]	95%	Highly fine-grained analytical model	Energy, Latency, Area	compute, memory, network	fixed	reconfigured
MAESTRO [12]	90-95%	Coarse-grained analytical model	Energy, Performance	PE count, L1/L2 buffer size, NoC bandwidth	fixed	reconfigured
FBNet [34]	unknown	Per-operator lookup table	Accuracy, Latency	Layer-wise, pre-defined layers/ blocks	NAS	fixed
Once-for-All [24]	unknown	Per-hardware latency lookup table	Accuracy, Latency	depth, width, kernel size, resolution	NAS	fixed
Kumar & Chandel [38]	89.9%	ML surrogate model	Accuracy, Area, Power, Latency	Compact HW-efficient building blocks (Conv2D, Dense, LSTM, Identity)	NAS	fixed
UniCoS [35]	inherited	MAESTRO analytical	Accuracy, EDP	ProxylessNAS, AutoFormer; layer clustering, loop order, tiling/parallelization	NAS	co-searched
Our Work [1]	99.997% (SRS) 98.7% (TRS)	Fine-grained analytical model	Accuracy, Latency, Area	ResNet blocks (channel values, kernel size), number of blocks, PE array geometry, clock ratio	NAS	reconfigured

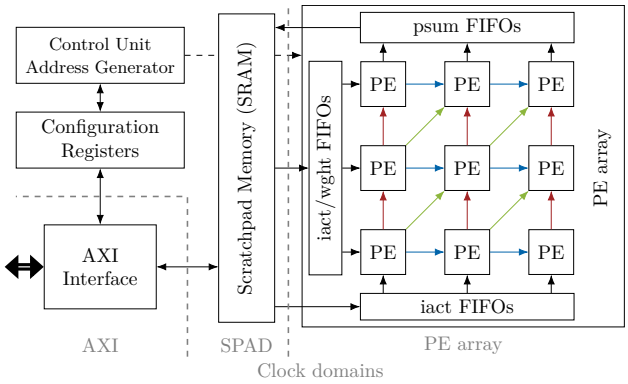


Fig. 2 Overview of the baseline accelerator architecture. In this example a 3×3 PE array is shown that is connected to a scratchpad memory via FIFOs. The SPAD domain operates in a higher clock frequency than the PE array domain. [6].

to as the Spatial Row-Stationary (SRS) in this work, or a custom modification thereof, the Temporal Row-Stationary (TRS) dataflow [6]. The difference of these is shown in Figure 3, that displays a convolution mapping to the accelerator for both dataflows. In the SRS, weights move left to right, input activations diagonally and partial sums upward through the array. In the TRS, weights and partial sums move identically, but the input activations move upwards. White squares represent idle PEs. In the example convolution, the SRS dataflow has 42% idle PEs, while the TRS dataflow has 36% idle PEs [6], showing that the dataflows can lead to a difference in utilization.

Our latency analysis of convolutions on the accelerator is divided into four phases: *Initialization, Calculation, Output* and *Stalling* that are explained individually in the following for both the SRS and the TRS dataflow. If equations are specific to one of the dataflows, this is denoted by *trs* or *srs* in the variable name. To refer to both dataflows simultaneously, the abbreviation *xrs* is used. The parameters that are used in the following to describe convolutions are listed in Table 2.

3.1.1 Initialization

During the initialization phase, data is loaded into the array. The scratchpad supplies data to the FIFOs, which forward the data to the PE array until the local line buffers of the PEs on the left and bottom edges of the array are filled. Once data begins to accumulate in the FIFOs instead of being directly consumed by the array, the preloading phase is considered complete and the calculation phase begins.

The duration for FIFO preloading is equal for both dataflows and is estimated by:

$$L_{init,xrs} = \left\lceil \frac{clk_{sp}}{clk_{pe}} \cdot \left\lfloor \frac{lb_{size}}{P_{sp}} + 1 \right\rfloor \cdot F_{num} \right\rceil \cdot clk_{pe} \quad (1)$$

If the array is small and operates on a low clock frequency, data may accumulate in the FIFOs because it cannot be processed fast enough, rather than because the array is full. Our model ignores this case, assuming it can be avoided through proper design.

3.1.2 Calculation

The computation time is determined by the loop structure of the RS dataflows and their mapping onto the PE array. For most convolutional layers, tiling across spatial dimensions

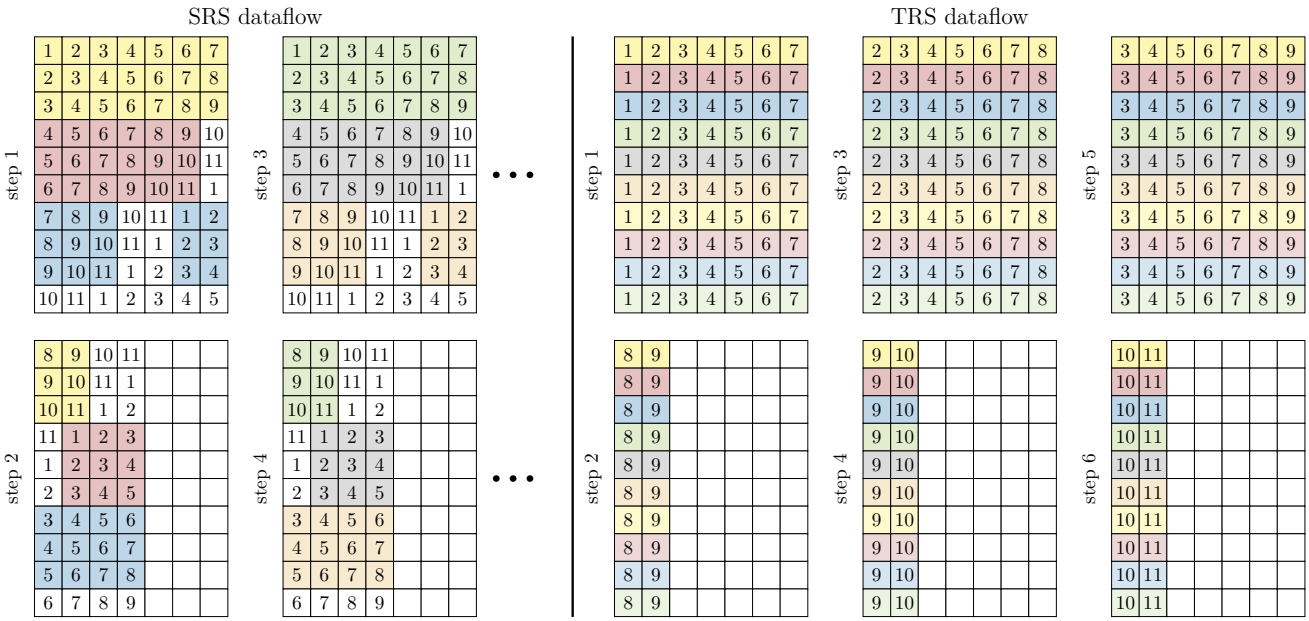


Fig. 3 Mapping of a convolution ($H = 11$, $K = 3$ and $C_{out} = 10$) on a 10×7 PE array according to the SRS and the TRS dataflows. Squares represent single PEs, numbers the input image rows and colors the different kernels [6].

Table 2 Parameters describing convolutional operations on the FleXNNgine. Images and kernels are assumed to be square.

Variables	Description
$H \times H, W \times W$	Input/Output image size
$K \times K$	Kernel size
C_{in}, C_{out}	Input/Output channel count
S	Stride
H_{pe}	Input rows that are output per iteration
H_i	Iterations to process all image rows
$C_{in,lb}$	Input channels that fit into linebuffers
$C_{out,pe}$	Output channels that fit onto PE array
$C_{in,i}, C_{out,i}$	Iterations to process all C_{in}/C_{out}
P_{sp}	Parallelism of scratchpad (word size)
F_{num}, F_{size}	Number/Size of FIFOs
lb_{size}	Linebuffer Size
$size_y \times size_x$	PE array size

and channels is required, as typical problem sizes exceed the array dimensions.

The outermost loop iterates over the output channels. As illustrated in Figure 3, up to $C_{out,pe,srs} = 3$ output channels can be processed in parallel for the SRS dataflow and up to $C_{out,pe,trs} = 10$ for the TRS dataflow. Therefore, $C_{out,i,srs} = 4$ ($C_{out,i,trs} = 1$) iteration(s) are required to compute all $C_{out} = 10$ output channels.

The next loop level corresponds to spatial tiling of the input feature map. In Figure 3, $H = 11$ input rows are processed, while only $H_{pe} = 7$ can be output simultaneously in both dataflows. Therefore, $H_i = 2$ tiling iterations are necessary. Although the second iteration processes fewer rows, it requires the same number of cycles as a full iteration. In the TRS dataflow, kernel rows are mapped onto a single PE-array row only, resulting in additional cycles to process all kernel rows.

The innermost loop performs tiling across input channels. The total number of input channels C_{in} is divided into $C_{in,i}$ groups of $C_{in,lb}$ channels that fit into the linebuffers within the PEs. The last iteration $C_{in,i,last}$ may be shorter if fewer channels remain to be processed.

Based on these considerations, the computation time can be expressed by the following equations:

$$L_{calc,H_i,srs} = ((C_{in,i} - 1) \cdot C_{in,lb} + C_{in,lb,last}) \cdot K \cdot clk_{pe} \cdot W \tag{2}$$

$$L_{calc,H_i,trs} = ((C_{in,i} - 1) \cdot C_{in,lb} + C_{in,lb,last}) \cdot K^2 \cdot clk_{pe} \cdot W \tag{3}$$

$$L_{calc,xrs} = L_{calc,H_i,xrs} \cdot H_i \cdot C_{out,i,xrs} \tag{4}$$

In case of padded images, the latency impact depends on the specific padding strategy. In general, padding increases the spatial dimensions of the input feature map by $(K - 1)$ per dimension, which leads to higher latency. However, for zero padding, multiplications involving padded values can often be skipped in hardware, reducing the effective computation time compared to a naive implementation. This optimization is only applicable under symmetric quantization, since otherwise the zero-point is non-zero and the corresponding multiplications cannot be skipped.

In the FleXNNgine, an optimization for zero padding is implemented for the horizontal padding in the SRS dataflow. Zero-padded rows at the top and bottom are treated as regular data, and in the TRS dataflow, all padded pixels are processed as regular data.

As the padding width depends on the kernel size, both the number of affected edge pixels and the number of zero multiplications per pixel vary with the kernel size K. To account for this, inner and padded pixels are modeled separately: central pixels are computed using Eqs. 2 and 4 without increasing the original image width, whereas the latency contribution of padded pixels is calculated separately as:

$$L_{calc,pad,H_i,srs} = 2 \cdot \left(\sum_{i=1}^{\frac{K-1}{2}} C_{in,lb} \cdot K - C_{in,lb} \cdot i \right) \cdot C_{in,i} \cdot clk_{pe} \tag{5}$$

per H_i iteration which is then added to L_{calc,H_i}.

The stride S is applied after the convolution by discarding every S-th value, making computation latency independent of the stride.

3.1.3 Output

After each H_i iteration, partial sums are written back to the scratchpad. The duration for this phase can be obtained directly from the source code and is equivalent for both dataflows:

$$L_{out,xrs} = (W \cdot (K + C_{out,pe,xrs} + 1) + 4) \cdot H_i \cdot C_{out,i,xrs} \cdot clk_{pe} \tag{6}$$

However, a special case occurs when the output rate, meaning the rate at which the PE array writes data to the FIFOs, exceeds the store rate at which the scratchpad reads the data from the FIFOs. Since no backpressure mechanism is implemented, this situation must be handled separately. In this case, the output phase has to be slowed down by introducing a throttle factor. The factor is calculated as follows:

$$r_{throttle} = \frac{P_{sp} \cdot \frac{clk_{pe}}{clk_{sp}}}{size_x \cdot (1 - \frac{F_{size} \cdot size_x \cdot P_{sp}}{size_x \cdot C_{out,pe} \cdot W})} \tag{7}$$

If the factor lies between 0 and 1, output throttling is required. Otherwise, no throttling is necessary as either the store rate exceeds the output rate or the output data fits entirely into the FIFOs. The adjusted output duration is then given by:

$$L_{out,trs} = L_{out,trs} \cdot \frac{1}{0.9 \cdot r_{throttle}} \tag{8}$$

3.1.4 Stalling

Stalling occurs when the FIFOs supplying data to the PE array become empty. To analyze this condition, we first estimate the FIFO occupancy at the point when new data is reloaded into the array, which happens after a $C_{in,i}$ calculation is completed. The occupancy is determined by the ratio between the duration of a $C_{in,i}$ iteration and the scratchpad FIFO refill rate, while ensuring that the value does not exceed the FIFO capacity:

$$F_{occ,xrs} = \min \left(\left\lfloor \frac{(C_{in,lb} \cdot K + 2) \cdot clk_{pe}}{F_{num} \cdot \frac{clk_{sp}}{P_{sp}}} \right\rfloor, F_{size} \right). \tag{9}$$

Stalling can occur in both dataflows only if the required reload data $C_{in,lb}$ exceeds the available FIFO occupancy F_{occ}. If this condition is met, the stall duration cannot be determined exactly because it depends on the current position of the scratchpad arbitrator and on which FIFOs must be refilled. For the SRS dataflow, the required scratchpad wait cycles to refill all empty FIFOs are approximated as $\frac{F_{num}+size_y}{2}$ whereas for TRS they are approximated as $2 \cdot (F_{num} + size_y)$. This results in the following stall duration:

$$L_{stall,srs} = \frac{F_{num} + size_y}{2} \cdot \left\lceil \frac{\max(C_{in,lb} - F_{occ}, 0)}{P_{sp}} \right\rceil \cdot clk_{sp} \tag{10}$$

$$L_{stall,trs} = 2 \cdot (F_{num} + size_y) \cdot K \cdot \left\lceil \frac{\max(C_{in,lb} - F_{occ}, 0)}{P_{sp}} \right\rceil \cdot clk_{sp} \tag{11}$$

The total latency estimate is given by the sum of the four phases: $L_{total} = L_{init} + L_{calc} + L_{out} + L_{stall}$.

Some minor effects, such as single idle cycles caused by state machine overhead in the hardware, are omitted from the formulas for clarity but are included in the final model.

3.2 Search Space Generation

Our latency model enables the evaluation of a large number of convolution and hardware-mapping combinations without requiring deployment on the target platform. Our objective is to construct a NN composed of layers that maximize efficiency, which we define as the ratio between achieved Giga Operations per Second (GOPS) and the theoretical peak performance, GOPS_{max}. However, this efficiency measure is relative to the theoretical peak performance and does not directly account for the absolute execution time or the number of Giga Operations (GOPs) of the layer. To avoid constructing networks that achieve high efficiency at the cost of excessive latency and GOPs, we additionally consider latency as an optimization objective.

Simply selecting Pareto-optimal convolutional layers is insufficient, as not every combination results in a valid or well-performing network architecture. We therefore constrain the search space to a Resnet-like architecture, which introduces structural constraints summarized in Figure 4. The figure illustrates so-called blocks, each representing one layer of the network. We consider two cases: ResNet-34 and ResNet-50. The main difference betweeen these is that the latter employs bottleneck blocks, which are often regarded as computationally cheap due to their use of 1×1 convolutions. This optimization, however, is largely motivated by GPU architectures and does not necessarily translate well to specialized accelerators, where limited data reuse can reduce the efficiency of 1×1 convolutions.

Each ResNet architecture comprises two block types: one that preserves spatial resolution and channel dimensions (S1), and one that reduces spatial resolution while increasing the number of channels (S2). The arrows on the right side of each block respresent the skip connections, where the block input is added directly to its output. These connections facilitate information flow across layers and improve training stability in deep networks [18]. For the addition to be valid, input and output tensor shapes must match. This condition is inherently satisfied in S1 blocks, whereas S2 blocks require an additional convolution in the skip path to align the tensor dimensions.

S1 blocks are inherently more efficient than S2 blocks, as a stride of $S = 2$ limits convolution efficiency to 50%. Moreover, S1 blocks occur more frequently, since they can be repeated multiple times. In the original ResNet architectures, S1 blocks typically appear in sequences of two to five consecutive repetitions. Consequently, we prioritize the efficiency of S1 blocks, with particular emphasis on the convolutions highlighted in green in Figure 4. The highlighted convolution within the S2 block of ResNet-34 is identical to the convolutions in the S1 block of the subsequent stage and is therefore implicitly covered by the S1-based prioritization. The block search is executed individually for each permutation of dataflows, hardware configurations and network architectures, as all of these influence the optimal NN structure.

3.2.1 ResNet-34 block definition

In the ResNet-34 architecture, S1 blocks consist of two convolutions with identical dimensions. For a given input resolution H, multiple S1 block candidates can therefore be constructed by varying the kernel size K and channel count, with $C_{in} = C_{out}$. Figure 5 illustrates the latency and efficiency of S1 block candidates with $H = 7$ for different kernel sizes and channel counts, evaluated on the baseline architecture described in Section 4. The original ResNet-34 configuration with $H = 7, K = 3, C_{in} = C_{out} = 512$ is indicated by the dashed red lines. The results show that kernel size and channel count both affect latency and efficiency. Increasing the channel count generally improves efficiency slightly, but the resulting latency increase is considerably larger.

This observation motivates the use of an asymmetric search range for the channel count favoring smaller channel counts than in the baseline architecture. To remain close to the original ResNet-34 configuration, channel values are selected from the interval $I = [(1 - r_L) \cdot C_{R34} ... r_U \cdot C_{R34}]$ where C_{R34} denotes the original ResNet-34 channel values, and r_L and r_U define the relative lower and upper bounds of the interval, respectively. All channel configurations that increase efficiency while reducing latency compared to the baseline are included in the search space. In addition,

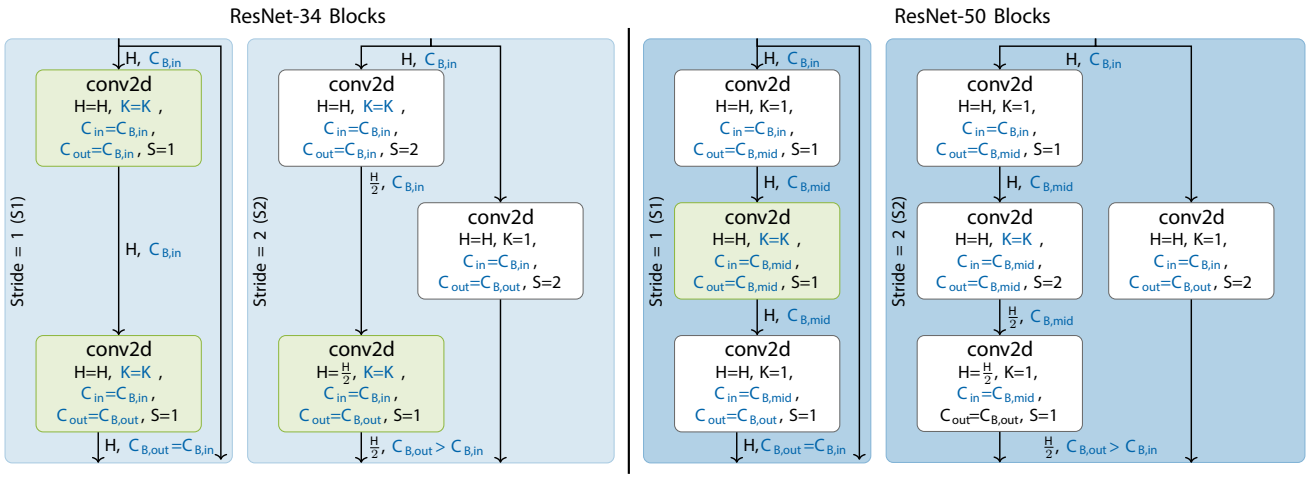


Fig. 4 Abstract block description for ResNet-like networks. ResNet-34 employs two convolutions per block when stride = 1 and three when stride = 2, while ResNet-50 adds an additional convolution per block. Convolutions highlighted in green represent the most efficient operations. Parameters marked in blue indicate deviations from the original ResNet architectures within our search space [1].

highly efficient configurations are retained even if they do not yield latency improvements, to preserve diversity within the search space.

The channel count of each S1 block determines the C_{out} of the preceding S2 block and the C_{in} of the subsequent S2 block. Thus, once the S1 blocks are defined, the convolutional structure of the network is fully specified.

3.2.2 ResNet-50 Block Definition

In contrast to ResNet-34, S1 blocks in ResNet-50 are heterogeneous: only the central convolution operates efficiently, whereas the surrounding 1×1 are less favorable on the target architecture. Since the middle convolution of an S1 block is equivalent to the convolutions in ResNet-34 S1 blocks with the same input resolution, the optimized channel values identified for ResNet-34 can be reused for $C_{B,mid}$. However, in ResNet-50 these values are independent of the layer interfaces $C_{B,in}$ and $C_{B,out}$ introducing an additional degree of freedom. We therefore evaluate S1 blocks with varying $C_{B,in} = C_{B,out}$ within an interval I centered around the original ResNet-50 interface dimensions. The interface configuration is then fixed to the setting that provides the most favorable trade-off between latency and efficiency.

3.3 Design Space Exploration of the Accelerator Hardware

The underlying accelerator hardware has a strong impact on overall AI system performance. Moreover, no single hardware architecture can efficiently execute all types of AI models. Therefore, we perform a DSE of key hardware parameters to identify systolic array configurations that are better suited for ResNet-34 and ResNet-50 using Bayesian Multi-Objective Optimization (MOO). In this exploration, both $size_x$ and $size_y$ of the PE array, as well as the ratio between the scratchpad clock clk_{sp} and the PE array clock clk_{pe} are varied. clk_{sp} is fixed at its maximum achievable frequency, while clk_{pe} is varied. The exploration is conducted for both the original ResNet-34 and ResNet-50 architectures and for both dataflow variants, SRS and TRS.

To avoid configurations with excessive hardware resource usage, the total number of PE is used as a hardware resource proxy and is constrained to be less than or equal to that of the baseline design. The objective is to identify hardware configurations that maximize computational efficiency while minimizing inference latency.

3.4 Neural Architecture Search

To perform NAS over the constructed search space, we employ Bayesian MOO guided by the latency model and the fitness error function introduced in [39]. As illustrated in Fig. 6, the fitness function measures the deviation of a candidate architecture from reference architectures that are known to achieve high accuracy on the same dataset, considering both the number of residual blocks and the channel sizes within each block and expressed as a simple linear function:

$$w_j = 48.(j+1), \tag{12}$$

where j is the block number and w_j is the per-block channel width. The error function is formulated as follows:

$$e_{fit} = \sum_{j=1}^{N} |w'_j - w_j|, \tag{13}$$

where N is the number of layers of the candidate network configuration, w'_j is the channel width of j-th layer of the candidate configuration.

To maintain feasibility and guide the search toward high-quality networks, additional soft constraints are applied to penalize architectures with excessive parameter counts, high latency, or a low total number of residual blocks, as these factors are strongly correlated with ResNet accuracy. Following the MOO process, feasible architectures are

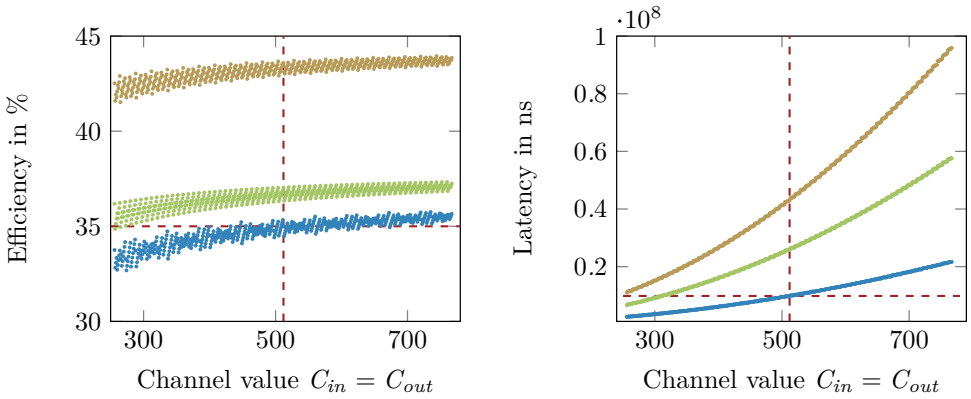


Fig. 5 Efficiency and latency values for S1 blocks with $H = 7$ and varying channel and kernel values. The colors represent kernel sizes: $K = 3$ is marked in blue, $K = 5$ in green and $K = 7$ in brown. The red dashed lines mark baseline values using $C_{in} = C_{out} = 512$ and $K = 3$.

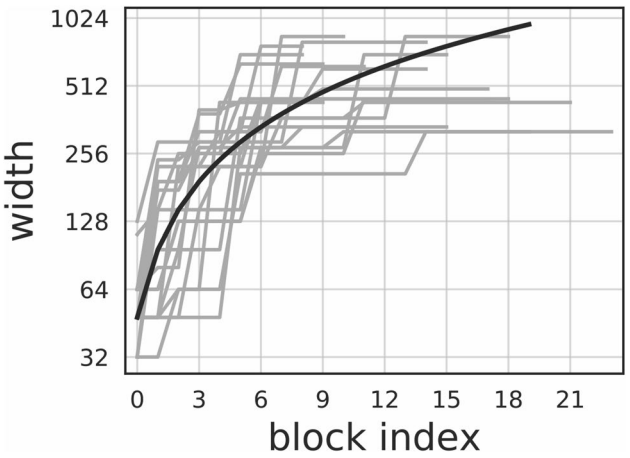


Fig. 6 Fitness function derivation as the linear fit over block widths of best 20 models from $AnyNetX_E$ search space [39]. Width axis is logarithmic.

sampled from the resulting Pareto front and subsequently trained on the ImageNet1K dataset [40].

4 Implementation

All evaluations are performed using revision 6 of the FleXNNgine accelerator available on GitHub [41]. Both the latency model and the search space generation framework are implemented in Python 3.10.18 and executed on Rocky Linux 9.6. The latency models were validated using an FPGA implementation of the accelerator with a 10×7 PE array on a Zynq UltraScale+ XCZU7EV device.

To efficiently support a wide variety of convolution configurations, the search space generation assumes a hardware layout that has been identified as well suited for general convolution workloads [36]. Specifically, a 21×20 PE array operating at $clk_{pe} = 160$ MHz and $clk_{sp} = 400$ MHz is used. This setup enables efficient mapping of all kernel sizes and most image resolutions while remaining suitable for FPGA-based prototyping without significant PE array stalling. Under this configuration, the accelerator reaches a theoretical peak performance of 134.4 GOPS_{max}. Resource utilization was evaluated on a VCU128 evaluation kit with the XCVU37P FPGA. The channel search ranges around the original network configurations are defined by $r_{L,R34} = 0.2$, $r_{L,R50} = 0.15$ and $r_{U,R34} = r_{U,R50} = 0.02$.

The DSE of accelerator configurations is performed using Optuna with a Treestructured Parzen Estimator (TPE) sampler. The configuration space includes variations in both dimensions of the systolic array within the range [7, 64], as well as the PE array clock clk_{pe} within [50, 200] in steps of 10 MHz. This results in a total of 48,735 possible configurations. For the MOO, we consider efficiency, as defined in Section 3.2, and latency as the primary objectives. Configurations exceeding the PE count of the baseline design are penalized to discourage excessive hardware resource usage. For each combination of model and dataflow, 5000 configurations are sampled and evaluated.

We implement a multi-fidelity NAS pipeline using Optuna with a TPE. An initial warm-up phase of 15 Monte Carlo trials is performed to gather statistics for the TPE model. The search aims to identify more efficient variants of ResNet-34 and ResNet-50 for corresponding hardware variants. To guide the optimization, soft constraints are applied that penalize architectures exceeding 27 M parameters or exhibiting higher latency than the original models when executed on the FleXNNgine accelerator. Additionally, we penalize models that lead to worse latency and efficiency than the baseline architectures in order to guide the search towards improved configurations. The MOO is conducted over 2000 trials, after which several architectures are selected from the resulting Pareto front.

Model training follows the schedule and data augmentation strategy described in [42]. Networks are trained for 50 epochs using Stochastic Gradient Descent (SGD) with a momentum of 0.875. The learning rate follows a cosine decay schedule with five warm-up epochs and an initial rate of 0.768, adjusted for training on four GPUs. A weight decay of $3.0517578125 \cdot 10^{-5}$ is applied to all non-normalization layers. Training is performed using Torch Lightning on nodes equipped with four NVIDIA A100 GPUs. The best architecture identified by the search is subsequently trained on the full dataset, requiring 140 GPU-hours. To verify the correctness of our pipeline implementation, we also train a baseline ResNet-50 model. In addition, ResNet-34 is trained, as its accuracy under this training pipeline is not reported.

5 Experimental Results

5.1 Latency Model

To evaluate the runtime of the latency model, we tested all convolution configurations within the following parameter ranges: $H \in \{112, 56, 28, 14, 7\}$, $K \in \{1, 3, 5, 7\}$, $C_{in} \in [1, 1024]$, $C_{out} \in [1, 2048]$, $S \in \{1, 2\}$,both with and without padding. In total, this resulted in 167,526,480 evaluations.The full sweep was completed in 808.12s, corresponding to an average evaluation time of 4.82µs per convolution.

The accuracy of the latency model was assessed by comparing its predictions with measured runtimes on the FPGA. Although some variability is introduced by

non-deterministic effects such as clock domain crossings and differing starting positions of the scratchpad arbitrator, the model closely matches the observed hardware latency, as illustrated in Figure 7. Across all experiments, the SRS model achieves a mean absolute error of 47.12ns and a root mean squared error of 65.08ns. This corresponds to a mean absolute percentage error of approximately 0.023%. The TRS model shows a mean absolute error of 28.23µs and a root mean squared error of 50.0µs, corresponding to a mean absolute percentage error of 1.3%.

5.2 Discovered Hardware Architectures

For each combination of dataflow and baseline NN, we performed a DSE and obtained Pareto fronts of optimized configurations that meet the maximum array size constraint of 420 PEs. Figure 8 shows an exemplary Pareto front for the ResNet-50 model computed with the SRS dataflow, illustrating the trade-off between latency and efficiency, with the number of PEs indicated by the point color. Configurations were selected by prioritizing high efficiency while favoring low latency and a reduced PE count. In the example shown in Figure 8, the highest-efficiency configuration of the pareto front is therefore selected with 384 PEs, an efficiency of 13.99% and a latency of 543.4ms.

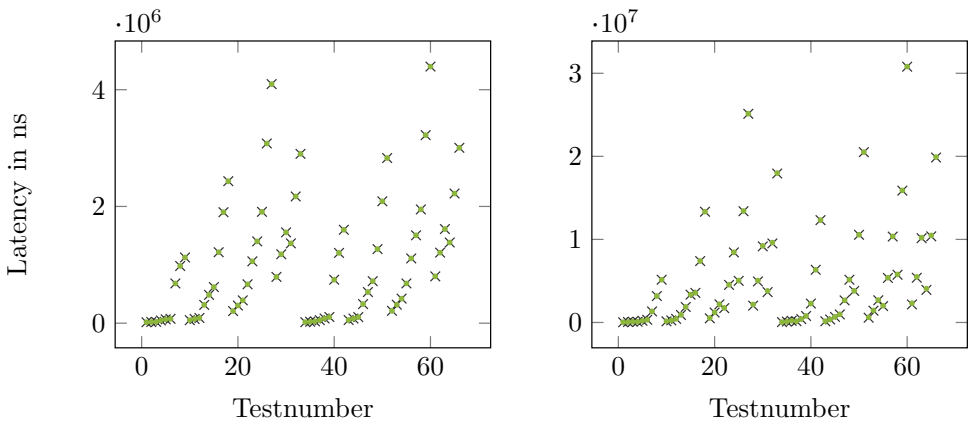
The selected configurations are summarized in Table 3 and are used for further evaluation. For nearly all combinations of dataflow and baseline model, the selected architectures achieve lower latency and higher efficiency than the baseline while requiring fewer PEs, with the largest gains observed in PE reduction. An exception is the ResNet-34 model with the SRS dataflow, for which the baseline hardware configuration remains optimal in terms of latency within the explored design space. In this case, a configuration with latency close to the baseline and a substantially reduced PE count is selected to compensate for the slight latency degradation in the subsequent NAS stage.

The number of PEs was used as a proxy for hardware area during the DSE. To assess the validity of this proxy, the corresponding FPGA resource utilization was evaluated in Table 4. LUT and FF utilization scale approximately with the number of PE; for example, reducing the PE count by 20% for SRS ResNet-34 reduces LUTs by approximately 19.2% and FFs by 18.3%. DSP utilization remains below the baseline, while URAM utilization remains constant across configurations. These results support the use of PE count as a hardware-area proxy for the explored configurations.

5.3 Search Space

The final search space for ResNet-50, executed with the SRS dataflow on the baseline hardware, is summarized in Table 5. Layers 1 and 2 are omitted, as they do not conform to the multi-convolution block structure: Layer 1 consists of a single convolution with $H \times W = 224^2,\ K = 7,\ C_{in} = 3$ and $C_{out} = 63$ and Layer 2 is a max-pooling layer. Layer 3 has an input size of $H \times W = 56^2$ and an initial input channel count of $C_{B,in} = 63$, which is increased to $C_{B,in} = 224$ by the first S1 block. This is the only instance of an S1 block where input and output channels differ. Channel notations $C_{B,in}$ and $C_{B,mid}$ correspond to the definitions in Figure 4. For S1 blocks, $C_{B,out}$ is equal to $C_{B,in}$ and for S2 blocks, $C_{B,out}$ is equal to $C_{B,in}$ of the following layer. Convolutions that improve both latency and efficiency compared to the baseline are observed only with kernel size $K = 3$. However, the highest efficiency is often achieved with $K = 7$. Therefore, select convolutions with $K = 7$ that provide high efficiency were included in the search space, with the corresponding $C_{B,mid}$ values marked with an asterisk.

Search spaces for the TRS dataflow differ slightly from the SRS search space, but are still similar. For the discovered hardware architectures, the search spaces align more closely with the original ResNet values, reflecting the fact that the hardware was optimized specifically for these networks.



(a) SRS dataflow (b) TRS dataflow

Fig. 7 Comparison of our latency model (black crosses) with measured FPGA execution times (green dots) across 66 tests with varying image size, kernel size, channel count, and padding.

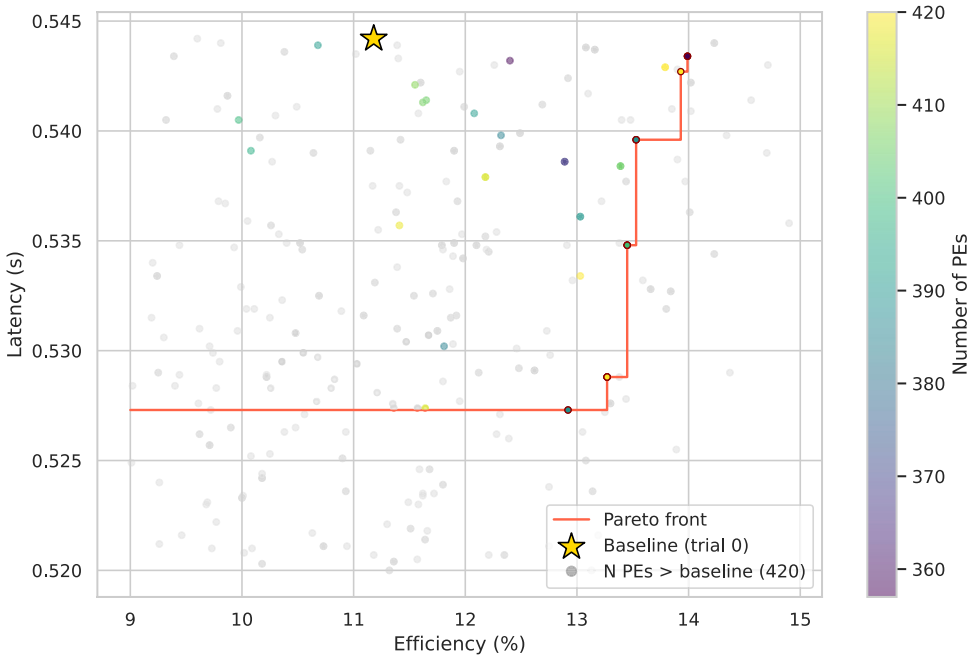


Fig. 8 DSE of different hardware configurations for ResNet-50 with the SRS dataflow. Grey points denote configurations outside the defined constraints, with their intensity indicating the number of PEs (darker points correspond to fewer PEs). The red line marks the Pareto front.

Table 3 Results of the accelerator hardware DSE compared to the baseline configuration with following parameters: 21×20 PE array (420 PE in total), $clk_{pe} = 160$ MHz and $clk_{sp} = 400$ MHz. Corresponding baseline values for efficiency and latency are provided in parenthesis.

	ResNet-34		ResNet-50	
	SRS	TRS	SRS	TRS
$size_x$	16	14	16	14
$size_y$	21	26	24	26
clk_{pe} (MHz)	170	150	140	100
GOPS_{max}	114.2	109.2	107.5	72.8
Efficiency (%)	**46.27** (40.53)	**48.20** (32.86)	**13.99** (11.18)	**7.02** (3.59)
Latency (ms)	138.6 (134.5)	**152** (165.6)	**543.4** (544.2)	**1599.3** (1696.2)

Table 4 Resource utilization comparison of the baseline hardware with 420 PEs and the selected hardware configurations on an XCVU37P board.

	Baseline	ResNet-34		ResNet-50	
		SRS	TRS	SRS	TRS
PEs	420	336	364	384	364
LUTs	388.923	314.382	338.113	355.465	338.113
FFs	212.707	173.785	186.098	195.278	186.098
URAM	16	16	16	16	16
DSPs	299	236	250	232	250

5.4 Discovered Neural Network Architectures

Table 6 summarizes the performance of the baseline and discovered architectures for both dataflows on the initial hardware setup from [36] and on the individually optimized hardware configurations as defined in Table 3. All networks are trained using the pipeline described in [42]. The accuracy results from three independent training runs are reported separately in Table 7 as mean and standard deviation. The accuracy achieved by ResNet-50 closely matches the reported results for this training pipeline, confirming the validity of our implementation.

On the initial hardware configuration, the discovered architectures generally reduce latency and parameter count while maintaining or improving efficiency. For ResNet-34 with the SRS dataflow, the discovered architecture reduces latency by 32.8% and parameter count by 42.8%, while maintaining a similar efficiency of approximately 40%. The TRS variant achieves a 15.6% latency reduction and a 6.3% reduction in parameters, while slightly increasing efficiency from 32.86% to 33.37%. The largest performance improvement is observed for ResNet-50 with SRS, where the discovered architecture reduces latency by 21.3% and parameters by 17.6%, while increasing efficiency by 84.5%

Table 5 Definition of the search space for a ResNet-50-like network tailored to the hardware configuration from [36] executed with the SRS dataflow. Input channels $C_{B,in}$ are defined per image size $H \times W$ and match the output channels $C_{B,out}$ of the preceding layer. $C_{B,mid}$ and S refer to middle channels and stride, respectively (see Figure 4). Kernel sizes of middle convolutions default to 3, except where indicated by an asterisk, which corresponds to a kernel size of 7.

$H \times W$	$C_{B,in}$	$C_{B,mid}$	
		$S = 1$	$S = 2$
56^2	$\{63, 224\}$	$\{63\}$	$\{104^*, 112, 126\}$
28^2	$\{440\}$	$\{104^*, 112, 126\}$	$\{207^*, 224, 238\}$
14^2	$\{880\}$	$\{207^*, 224, 238\}$	$\{414^*, 432, 448, 511\}$
7^2	$\{1740\}$	$\{414^*, 432, 448, 511\}$	-

Table 6 Evaluation of the discovered architectures on baseline and individually optimized hardware configurations. The analysis uses the latency model and considers only convolutional layers. Bold values indicate improvements over the corresponding baseline.

		ResNet-34				ResNet-50			
		SRS		TRS		SRS		TRS	
		orig.	disc.	orig.	disc.	orig.	disc.	orig.	disc.
Baseline HW	Parameters (M)	21.80	**12.48**	21.80	**20.43**	25.56	**21.06**	25.56	**22.32**
	GOPs	7.327	**4.886**	7.327	**6.072**	8.174	11.91	8.174	**6.385**
	Latency (ms)	134.5	**90.36**	165.9	**140**	544.2	**428.5**	1696.2	**1288.0**
	GOPS	54.48	54.08	44.16	**44.84**	15.02	**27.79**	4.82	**4.96**
	GOPS_{max}	134.4	134.4	134.4	134.4	134.4	134.4	134.4	134.4
	Efficiency (%)	40.53	40.23	32.86	**33.37**	11.18	**20.67**	3.59	**3.69**
	Number of PEs	420	420	420	420	420	420	420	420
Optimized HW	Parameters (M)	21.80	**13.03**	21.80	**17.93**	25.56	**20.33**	25.56	**20.70**
	GOPs	7.327	**4.445**	7.327	**5.687**	8.174	**6.942**	8.174	**6.022**
	Latency (ms)	138.6	**89.00**	152.0	**110.1**	543.4	**425.0**	1599.3	**1186.7**
	GOPS	52.86	49.94	48.20	**51.65**	15.04	**16.33**	5.11	5.07
	GOPS_{max}	114.2	114.2	109.2	109.2	107.5	107.5	72.8	72.8
	Efficiency (%)	46.27	43.72	44.14	**47.30**	13.99	**15.19**	7.02	6.97
	Number of PEs	336	336	364	364	384	384	364	364

Table 7 Top-1 accuracy of the original and discovered networks. Results are measured on the full ImageNet dataset over three independent training runs with different seeds (42, 222, 333) and are expressed as mean ± standard deviation.

Network	NAS Target		Accuracy (%)
	Dataflow	Hardware	
ResNet-34	*original ResNet-34*		72.04 ± 0.07
	SRS	baseline	71.01 ± 0.22
	TRS	baseline	72.03 ± 0.24
	SRS	optimized	70.49 ± 0.21
	TRS	optimized	71.59 ± 0.16
ResNet-50	*original ResNet-50*		75.37 ± 0.21
	SRS	baseline	75.34 ± 0.13
	TRS	baseline	74.87 ± 0.40
	SRS	optimized	74.93 ± 0.17
	TRS	optimized	74.89 ± 0.11

from 11.18% to 20.67%. For TRS, latency and parameter count are reduced by 24.1% and 12.7%, respectively, with a smaller efficiency improvement from 3.59% to 3.69%. Since the baseline hardware uses the maximum array size of 420 PE, these improvements are achieved while simultaneously reducing hardware resources.

The accuracy results in Table 7 show that the performance improvements generally come with limited changes in classification accuracy, although the impact varies across architectures and dataflows. For ResNet-50, all discovered networks remain within 0.5 percentage points of the baseline accuracy. The three independent training runs in Table 7 exhibit standard deviations between 0.11 and 0.40 percentage points, indicating consistent results across seeds. The largest accuracy reduction occurs for ResNet-34 with SRS, where accuracy decreases from $72.04 \pm 0.07\%$ to

$71.01 \pm 0.22\%$ on the initial hardware. In contrast, the TRS variant retains essentially the baseline accuracy, changing from $72.04 \pm 0.07\%$ to $72.03 \pm 0.24\%$. Thus, the observed accuracy changes are dependent on the configuration rather than a consistent consequence of the architecture search.

The effect of the hardware DSE can be separated from that of NAS by comparing the original networks on the initial and the optimized hardware. Hardware optimization reduces the PE count by 20% for ResNet-34 with SRS, 13.3% for ResNet-34 with TRS, 8.6% for ResNet-50 with SRS, and 13.3% for ResNet-50 with TRS. These reductions are accompanied by changes in both latency and efficiency. For example, the optimized hardware for ResNet-50 with SRS maintains almost the same latency as the baseline hardware while increasing efficiency from 11.18% to 13.99%, whereas the TRS configuration increases efficiency from 3.59% to 7.02% while reducing latency by 5.7%. The hardware optimization therefore enables substantial PE reduction while maintaining or improving both latency and efficiency.

The effect of NAS can in turn be observed by comparing the original and discovered networks on the same hardware configuration. The resulting gains depend strongly on the hardware target. For example, on the initial hardware, NAS increases the efficiency of ResNet-50 with SRS from 11.18% to 20.67%, while reducing latency by 21.3%. When NAS targets the individually optimized hardware, however, the corresponding efficiency increase is much smaller, from 13.99% to 15.19%, while latency is reduced by 21.8%. For ResNet-50 with TRS, the discovered network even slightly reduces efficiency on the optimized hardware, from 7.02% to 6.97%, despite reducing latency by 25.8%. This demonstrates that the benefits of NAS and hardware optimization are not fully additive: the hardware configuration optimized for the baseline network does not necessarily provide the same relative benefit for the subsequently discovered network.

Despite this interaction, the combined optimization reduces latency and hardware requirements for all evaluated configurations. Compared with the original networks on the initial hardware, the discovered networks on their respective optimized hardware reduce latency by 21.9% to 33.8% across the four dataflow and network combinations, while reducing the PE count by 8.6% to 20%. Efficiency increases by 7.9% and 43.9% for the ResNet-34 SRS and TRS configurations, respectively, and by 35.9% and 94.2% for the corresponding ResNet-50 configurations. The results therefore demonstrate that the sequential workflow can simultaneously reduce latency and hardware requirements while improving efficiency for the evaluated configurations. At the same time, the differing gains across hardware targets highlight the interaction between network and hardware optimization and show that their benefits cannot be assumed to be additive.

6 Conclusion and Outlook

In this paper, we introduced a hardware-aware workflow for jointly considering neural network and accelerator characteristics during system design. The workflow combines latency modeling, hardware-constrained search space generation, multi-objective Bayesian NAS, and hardware DSE. For the FleXNNgine accelerator, the resulting architectures achieve substantial reductions in latency and hardware requirements while maintaining comparable accuracy across the evaluated ResNet NNs.

The presented workflow is implemented as a staged optimization process: accelerator configurations are first optimized for the baseline networks, followed by a NAS targeting the selected hardware configurations. The results demonstrate that this sequential approach improves latency, efficiency, and hardware cost, while also showing that the benefits of the two stages are not fully additive. A unified optimization of network and hardware configurations is therefore a promising direction for future work.

While the evaluation is limited to ResNet networks on the FleXNNgine accelerator, the underlying methodology is applicable to other systolic-array accelerators. The latency model is specific to FleXNNgine and its dataflows and would need to be adapted and validated for other architectures. The use of PE count as a hardware-cost proxy can also be applied to other architectures when its relationship with FPGA resource utilization is established. These principles provide a basis for constructing hardware-aware search spaces for other accelerator designs, with the specific models and search spaces adapted to the target architecture and workload.

Further improvements are limited by operations that map inefficiently onto systolic arrays but remain prevalent in modern neural networks. For instance, as reported in [36], EfficientNet [43] and MobileNetV2 [19] achieve only 2.83% and 3.14% hardware utilization, respectively, on FleXNNgine with the baseline configuration due to their reliance on depthwise separable convolutions. Future work will address these limitations by designing networks that avoid inefficient operations and by adapting accelerator hardware to handle these specific operations more effectively.

Author Contributions J.S. provided the initial latency model, which was largely refactored by A.G. A.G. also defined the search space for generating the neural networks and wrote a large part of the manuscript, including most figures and tables. A.S. conducted the hardware design space exploration, NAS, and neural network training, and wrote

the corresponding sections of the manuscript. H.T.K. contributed to neural network training. F.P. and H.T.K. performed the state-of-the-art research and wrote the corresponding part of the manuscript. F.L. provided guidance on implementation details of the baseline architecture and designed Figures 2 and 3. M.H. developed the multi-objective optimization code for TPE Bayesian optimization with soft constraints. T.H. and J.B. provided supervision and general guidance and contributed to the general manuscript. All authors reviewed, proofread, and approved the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability All data and code supporting the findings of this study are publicly available. The full workflow, including latency models, search space generation, NAS, and training pipelines, is open-sourced at https://github.com/itiv-kit/zc-nas-for-flexnngine. Neural networks were trained and evaluated on the publicly available ImageNet dataset. Researchers can reproduce all experiments using the resources provided.

Declarations

Conflicts of interest The authors declare no competing interests.

Competing interests The authors declare no competing interests.

References

1. Gutermann, A., Serdyuk, A., Paraskevas, F., Kiesa, H. T., Lesniak, F., Schwarz, J., Hartmann, M., Harbaum, T., & Becker, J. (2025). Improving ai accelerator performance through co-designing neural networks and systolic hw. In: *2025 IEEE Nordic Circuits and Systems Conference (NorCAS)* (pp. 1–7). https://doi.org/10.1109/NorCAS66540.2025.11231285
2. Chen, Y., Xie, Y., Song, L., Chen, F., & Tang, T. (2020). A survey of accelerator architectures for deep neural networks. *Engineering,6*(3), 264–274. https://doi.org/10.1016/j.eng.2020.01.007
3. Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., & Lin, Y. (2021). HW-NAS-Bench:Hardware-Aware Neural Architecture Search Benchmark. arXiv. arXiv:2103.10584 [cs] version: 1. https://doi.org/10.48550/arXiv.2103.10584. Accessed 10 Jul 2025
4. Chitty-Venkata, K. T., & Somani, A. K. *Neural Architecture Search Survey: A Hardware Perspective*, *55*(4), 1–36. https://doi.org/10.1145/3524500. Accessed 05 May 2025
5. Osterwind, A., Droste-Rehling, J., Vemparala, M.-R., & Helms, D. (2023). Hardware execution time prediction for neural network layers. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (pp. 582–593). Cham: Springer.
6. Lesniak, F., Gutermann, A., Harbaum, T., & Becker, J. (2024). Enhanced accelerator design for efficient cnn processing with improved row-stationary dataflow. In *Proceedings of the Great Lakes Symposium on VLSI 2024. GLSVLSI '24*. New York, NY, USA: Association for Computing Machinery.
7. Chen, Y.-H., Emer, J., & Sze, V. (2016). Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. *SIGARCH Computer Architecture News,44*(3), 367–379. https://doi.org/10.1145/3007787.3001177
8. Mei, L., Houshmand, P., Jain, V., Giraldo, S., & Verhelst, M. (2021). ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators. *IEEE Transactions on Computers,70*(8), 1160–1174. https://doi.org/10.1109/TC.2021.3059962. Accessed 2025-05-07
9. Mei, L., Liu, H., Wu, T., Sumbul, H. E., Verhelst, M., & Beigne, E. (2022) A uniform latency model for dnn accelerators with diverse architectures and dataflows. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 220–225). https://doi.org/10.23919/DATE54114.2022.9774728
10. Parashar, A., Raina, P., Shao, Y. S., Chen, Y.-H., Ying, V. A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S. W., & Emer, J. (2019). Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 304–315). IEEE, Madison, WI, USA. https://doi.org/10.1109/ISPASS.2019.00042 . https://ieeexplore.ieee.org/document/8695666/ Accessed 2025-05-07
11. Raj, R., Banerjee, S., Chandra, N., Wan, Z., Tong, J., Samajdar, A., & Krishna, T. (2025). SCALE-Sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis. arxiv:2504.15377
12. Kwon, H., Chatarasi, P., Pellauer, M., Parashar, A., Sarkar, V., & Krishna, T. (2019). Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-52* (pp. 754–768). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3352460.3358252
13. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). MnasNet: Platform-Aware Neural Architecture Search for Mobile. arXiv. https://doi.org/10.48550/arXiv.1807.11626
14. Cai, H., Zhu, L., & Han, S. (2019). ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. arXiv. arXiv:1812.00332 [cs]. https://doi.org/10.48550/arXiv.1812.00332. Accessed 24 Jul 2025
15. Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M., & Wang, N. (2021) A comprehensive survey on hardware-aware neural architecture search. arXiv. arXiv:2101.09336 [cs]. https://doi.org/10.48550/arXiv.2101.09336. Accessed 04 June 2025
16. Wang, Z., Trefzer, M. A., Bale, S. J., & Tyrrell, A. M. (2022). A Multi-objective Evolutionary Approach for Efficient Kernel Size and Shape for CNN. In *2022 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). https://doi.org/10.1109/IJCNN55064.2022.9892201. ISSN: 2161-4407. https://ieeexplore.ieee.org/document/9892201/. Accessed 24 Jul 2025
17. Zeng, S., Sun, H., Xing, Y., Ning, X., Shan, Y., Chen, X., Wang, Y., & Yang, H. (2020). Black Box Search Space Profiling for Accelerator-Aware Neural Architecture Search. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 518–523). IEEE, Beijing, China. https://doi.org/10.1109/A

SP-DAC47756.2020.9045179. https://ieeexplore.ieee.org/document/9045179/ Accessed 2025-06-23
18. He, K., Zhang, X., Ren, S., & Sun, J. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/arXiv.1512.03385 . arxiv:1512.03385. Accessed 10 Jul 2025
19. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. https://doi.org/10.48550/arXiv.1801.04381 . arxiv:1801.04381. Accessed 10 Jul 2025
20. Simonyan, K., & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://doi.org/10.48550/arXiv.1409.1556 . arxiv:1409.1556. Accessed 29 Aug 2025
21. Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research,20*(55), 1–21.
22. White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., & Hutter, F. (2023). Neural Architecture Search: Insights from 1000 Papers. arXiv. https://doi.org/10.48550/arXiv.2301.08727
23. Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable Architecture Search. arXiv. https://doi.org/10.48550/arXiv.1806.09055
24. Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*. https://arxiv.org/pdf/1908.09791.pdf
25. Lee, J., & Ham, B. (2024). AZ-NAS: Assembling Zero-Cost Proxies for Network Architecture Search. arxiv:2403.19232
26. Dong, X., & Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=HJxyZkBKDr
27. Jiang, W., Yang, L., Sha, E.H.-M., Zhuge, Q., Gu, S., Dasgupta, S., Shi, Y., & Hu, J. (2020). Hardware/Software Co-Exploration of Neural Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,39*(12), 4805–4815. https://doi.org/10.1109/TCAD.2020.2986127
28. Zhang, Y., Fu, Y., Jiang, W., Li, C., You, H., Li, M., Chandra, V., & Lin, Y. (2020). DNA: Differentiable Network-Accelerator Co-Search. arXiv. arXiv:2010.14778 [cs.LG]. https://doi.org/10.48550/arXiv.2010.14778
29. Fu, Y., Zhang, Y., Zhang, Y., Cox, D., & Lin, Y. (2021). Auto-NBA: Efficient and Effective Search Over the Joint Space of Networks, Bitwidths, and Accelerators. In *Proceedings of the 38th International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research* (vol. 139, pp. 3505–3517). https://proceedings.mlr.press/v139/fu21d.html
30. Lin, Y., Yang, M., & Han, S. (2021). NAAS: Neural Accelerator Architecture Search. *2021 58th ACM/IEEE Design Automation Conference (DAC)* (pp. 1051–1056). https://doi.org/10.1109/DAC18074.2021.9586250
31. Tuli, S., Li, C.-H., Sharma, R., & Jha, N. K. (2023). CODEBench: A Neural Architecture and Hardware Accelerator Co-Design Framework. *ACM Transactions on Embedded Computing Systems,22*(3), 1–30. https://doi.org/10.1145/3575798
32. Lou, W., Qian, J., Gong, L., Wang, X., Wang, C., & Zhou, X.: NAF: Deeper Network/Accelerator Co-Exploration for Customizing CNNs on FPGA. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1–6). https://doi.org/10.23919/DATE56975.2023.10137094
33. Lou, W., Gong, L., Wang, C., Qian, J., Wang, X., Li, C., & Zhou, X. (2024). Unleashing Network/Accelerator Co-Exploration Potential on FPGAs: A Deeper Joint Search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,43*(10), 3041–3054. https://doi.org/10.1109/TCAD.2024.3391688
34. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., & Keutzer, K. (2019). FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. arXiv. arXiv:1812.03443 [cs.CV]. https://doi.org/10.48550/arXiv.1812.03443. Accessed 23 Jul 2026
35. Fu, W., Lou, W., Tang, C., Wen, H., Qin, Y., Gong, L., Wang, C., & Zhou, X. (2025). UniCoS: A Unified Neural and Accelerator Co-Search Framework for CNNs and ViTs. *2025 62nd ACM/IEEE Design Automation Conference (DAC)* (pp. 1–6). https://doi.org/10.1109/DAC63849.2025.11133418
36. Serdyuk, A., Gutermann, A., Lesniak, F., Oberhoff, F., Dreher, M., Majer, M., Pschorr, A., Fürst-Walter, I., Harbaum, T., & Becker, J. (2025). *On Benchmarking Systolic-Array-based Accelerators for Automotive Applications*. Springer: Accepted for publication at Driving the Future Symposium.
37. Hendrickx, L., Symons, A., Van Ranst, W., Verhelst, M., & Goedemé, T. (2023). Hardware-aware NAS by Genetic Optimisation with a Design Space Exploration Simulator. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 2275–2283). https://doi.org/10.1109/CVPRW59228.2023.00222 . ISSN: 2160-7516. https://ieeexplore.ieee.org/document/10208795/. Accessed 22 Jul 2026
38. Kumar, A., & Chandel, R. (2026). Hardware-Aware Neural Architecture Search (NAS) with Area, Power, and Latency Constraints for ASIC Implementation. In *2026 9th International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 95–102). https://doi.org/10.1109/ICOEI68323.2026.11542183 . https://ieeexplore.ieee.org/document/11542183/
39. Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., & Dollar, P. Designing Network Design Spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 10425–10433). IEEE. https://doi.org/10.1109/CVPR42600.2020.01044 . https://ieeexplore.ieee.org/document/9156494/ Accessed 2025-08-17
40. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). https://doi.org/10.1109/CVPR.2009.5206848
41. Institute of Information Processing Technology (ITIV). (2025). FleXNNgine. https://github.com/itiv-kit/flexnngine. Accessed 08 Aug 2025
42. Nvidia. (2025). ResNet-50 v1.5 for MXNet. https://github.com/NVIDIA/DeepLearningExamples/tree/master/MxNet/Classification/RN50v1.5. Accessed 28 Aug 2025
43. Tan, M., & Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://doi.org/10.48550/arXiv.1905.11946. arxiv:1905.11946. Accessed 15 Jul 2025