INTEGRATION, the VLSI journal 107 (2026) 102625

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

An efficient open-source design and implementation framework for non-quantized CNNs on FPGAs

Angelos Athanasiadis [a,*], Nikolaos Tampouratzis [b], Ioannis Papaefstathiou [a]

[a] School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, 54124, Greece
[b] Department of Industrial Engineering and Management, International Hellenic University, Sindos, 57400, Greece

ARTICLE INFO

Keywords:
Convolutional Neural Networks
High-performance computing
Acceleration of matrix multiplication
AMD FPGAs
AMD Vitis

ABSTRACT

The growing demand for real-time processing in artificial intelligence applications, particularly those involving Convolutional Neural Networks (CNNs), has highlighted the need for efficient computational solutions. Conventional processors and graphical processing units (GPUs), very often, fall short in balancing performance, power consumption, and latency, especially in embedded systems and edge computing platforms. Field-Programmable Gate Arrays (FPGAs) offer a promising alternative, combining high performance with energy efficiency and reconfigurability. This paper presents a design and implementation framework for implementing CNNs seamlessly on FPGAs that maintains full precision in all neural network parameters thus addressing a niche, that of non-quantized NNs. The presented framework extends Darknet, which is very widely used for the design of CNNs, and allows the designer, by effectively using a Darknet NN description, to efficiently implement CNNs in a heterogeneous system comprising of CPUs and FPGAs. Our framework is evaluated on the implementation of a number of different CNNs and as part of a real world application utilizing UAVs; in all cases it outperforms the CPU and GPU systems in terms of performance and/or power consumption. When compared with the FPGA frameworks that support quantization, our solution offers similar performance and/or energy efficiency without any degradation on the NN accuracy.

1. Introduction

In recent years, Convolutional neural networks (CNNs) have been key components for many significant advancements in the field of artificial intelligence. They have proven to be highly effective in numerous fields like image, video, and natural language processing. In addition, they are effective in various tasks including image classification, object detection, and semantic segmentation.

The implementation of CNNs in real-world scenarios often encounters difficulties such as high processing power and high power consumption. The complicated structures of CNNs, which consist of several convolutional layers, fully connected layers, and a large number of parameters, require significant computational resources, which can be a substantial barrier, particularly for applications that demand real-time processing and need to be deployed on devices with limited resources, such as embedded systems and edge computing platforms.

On real-time and embedded AI applications, traditional processors (including CPUs and GPUs) are often insufficient in balancing the trade-offs between performance, power consumption, and latency. As a result, there has been growing interest in leveraging specialized hardware accelerators that can provide the necessary computational power while maintaining energy efficiency. In that respect, FPGAs have emerged as a viable solution to address these challenges. FPGAs offer a unique combination of high performance and reconfigurability which enables the implementation of custom arithmetic units, data paths and memory hierarchies that can optimize data flow and reduce bottlenecks. They also allow for the simultaneous execution of multiple operations, thus significantly speeding up the inference process. The ability to reconfigure the hardware logic allows for the implementation of highly optimized and parallelized versions of neural network algorithms, tailored specifically to the requirements of the application. Lastly, FPGAs can be configured to operate at lower power levels compared to traditional processors, making them ideal for power-constrained environments.

In recent years, various approaches have been proposed to implement CNNs focusing mainly in heavily quantized models. Non-quantized models maintain the full precision of the network parameters, ensuring high accuracy at the cost of higher resource utilization and power consumption. Quantized models, on the other hand, reduce the precision of the parameters, thereby lowering resource usage and power consumption but very often at the cost of reduced accuracy.

* Corresponding author.
E-mail addresses: angelathan@ece.auth.gr (A. Athanasiadis), ntampouratzis@ihu.gr (N. Tampouratzis), ygp@ece.auth.gr (I. Papaefstathiou).

https://doi.org/10.1016/j.vlsi.2025.102625
Received 20 June 2025; Received in revised form 24 November 2025; Accepted 1 December 2025
Available online 2 December 2025

Full precision in CNNs is a requirement for applications where even minimal accuracy loss can have significant consequences. Domains such as medical diagnostics [1] and optical computing [2] require highly reliable predictions to ensure safety, efficacy, and validity. Although quantization is beneficial for reducing computational complexity and resource requirements, it often introduces approximation errors that can compromise the integrity of results and as also demonstrated in [3], there are several cases that full arithmetic precision is required so as to ensure that critical tasks are performed with the highest possible accuracy. In addition, as highlighted in recent comprehensive surveys on CNNs for medical imaging and edge deep learning systems [4,5], achieving near-perfect arithmetic precision alongside optimized power consumption is essential for maintaining robust diagnostic performance, patient safety, and regulatory compliance. Such medical devices can be seamlessly implemented using our innovative flow, demonstrating the wide applicability of our approach.

Recent studies demonstrate that post-training quantization can significantly compromise inference accuracy. Specifically, authors in [6] show that reducing three "tiny" CNNs from FP32 to 8-bit precision decreases CIFAR-10 accuracy by roughly 19%, while an additional downgrade to 4-bit yields a drastic 67% loss and comparable double-digit deficits on other benchmarks. Similarly, authors in [7] report that integer quantization diminishes MNIST performance from 96.79% to 94.43% and JAFFE emotion-recognition accuracy from 94.44% to 83.33% in otherwise identical network topologies. Although quantization offers model-size and resource savings, it often incurs a non-negligible accuracy penalty. Consequently, our work retains full-precision parameters and attains efficiency through FPGA-centric architectural optimization.

Moreover, larger and/or more complex convolutional architectures also experience substantial accuracy drops, when quantization is applied. For instance, in [8], SA-BNN attains only 68.7% when quantization is utilized, compared with 74.7% for the full-precision version. Similarly, in the domain of fast convolution algorithms, [9] demonstrates that applying quantization in large-tile Winograd convolution introduces pronounced accuracy degradation relative to standard convolutional schemes; the quantized model reaches up to 69.7% accuracy while the full precision configurations achieve 73.3%.

Even advanced quantization-aware and mixed-precision methods still introduce measurable accuracy losses: for instance, Mishchenko et al. [10] report 40% AUC degradation for a 4-bit QAT keyword-spotting model, Xu et al. [11] show 4-bit Q-DETR suffers a 4.5 point Average Precision (AP) drop (with mixed-precision variants still 3 AP below full precision), and Cai et al. [12] demonstrate that state-of-the-art mixed-precision quantization yields 1–3.7% top-1 loss on ResNet-50 and MobileNetV2 and up to 5%–7% on ShuffleNet.

Beyond accuracy considerations, several recent works show that full-precision CNNs are frequently deployed in energy-constrained embedded systems where power consumption is the primary limiting factor. For example, Xu et al. [3] report that the FPGA achieves approximately 3× higher power efficiency compared to the NVIDIA GTX 1080 Ti under the same full-precision workload, while Qi et al. [13] demonstrate that FPGA accelerators achieve a significantly better performance–power ratio than GPUs (e.g., 3.27GOPS/W on FPGA vs. 0.978GOPS/W on GPU), highlighting that FPGAs become preferable to GPUs when total energy budget is the dominant constraint.

In this paper we present a novel design and implementation framework that allows for the seamless FPGA implementations of non-quantized CNNs with high performance, energy efficiency and accuracy. By maintaining full precision in the neural network parameters, the proposed framework retains the accuracy of the original CNN models while leveraging the parallel processing capabilities of FPGAs.

The main benefits of our approach are:

- An **open-source library**[1] designed to accelerate CNN algorithms. This library provides very high configurability and flexibility in order to take full advantage of the resources of modern AMD FPGAs.
- **High Design Productivity, Flexibility and Adaptability**: The presented efficient design flow is based on the widely used DarkNet NN design framework, it uses purely C/C++ and targets the whole range of FPGAs from the smallest to the largest ones. Since it allows for seamless implementation of virtually any CNN architecture, the proposed framework is fully versatile and adaptable to different application requirements.
- **High Performance**: Real-time applications require rapid processing of data to meet stringent latency requirements. The proposed framework can fully exploit the full parallelism of any FPGA to accelerate the inference process of CNNs, ensuring timely and efficient processing.
- **Energy Efficiency**: In many deployment scenarios, particularly in embedded systems and edge devices, power consumption is a critical constraint. The proposed framework optimizes the power efficiency of ML inference on FPGAs, making it suitable for power-sensitive applications, while retaining full precision, which is crucial for applications where precision is paramount.

The development of a power-efficient, non-quantized FPGA-enabled CNN framework represents a significant advancement in the field of deep learning acceleration. Our comprehensive design and implementation flow which is built upon the DarkNet framework allows rapid integration and deployment, offering a user-friendly environment where researchers and engineers, with very limited HW Design experience, can effortlessly design and implement CNNs tailored to their specific needs with 0% accuracy loss. In other words our design ensures a high level of abstraction and flexibility, enabling users to efficiently prototype their CNN architectures, optimize and exploit FPGA boards without specific HW knowledge. By addressing the challenges of performance, power efficiency, and accuracy, the proposed framework has the potential to enhance the deployment of CNNs in real-time and embedded applications, paving the way for more efficient and effective AI solutions.

The rest of this paper is structured as follows: Section 2 presents a summary of related research in the field of FPGA-based and GPU-based CNN design frameworks as well as in dense matrix multiplication systems. Section 3 covers the background needed for CNNs. Section 4 outlines the design flow and the implemented architecture of our framework, emphasizing on the design optimizations of the convolutional layer of the CNNs and on the matrix multiplication algorithm involved in CNNs. Section 5 offers comprehensive performance evaluation that demonstrates the advantages of our solution. Lastly, Section 6 concludes our work and suggests potential areas for future research.

2. Literature review

2.1. Convolutional Neural Networks

2.1.1. Quantized and mixed-precision accelerators

In [14] a uniform architecture that accelerates both 2D and 3D CNNs by mapping convolutions to matrix multiplications, is presented. It addresses the high computational demands of 3D CNNs and proposes strategies to optimize on-chip memory usage and computational efficiency. The architecture demonstrates high throughput and energy efficiency on FPGA platforms, emphasizing the potential of non-quantized CNN acceleration.

Authors in [15] focus on mixed-precision FPGA-based CNN accelerators providing valuable insights into the design space of the required

[1] https://github.com/angelosathanasiadis/Open-source-non-quantized-CNNs-framework-on-FPGAs

processing elements (PEs) and dataflow optimizations. They highlight the complexity of balancing precision and energy efficiency without compromising performance, which is relevant for non-quantized CNN implementations aiming for high accuracy and throughput.

Authors in [16] delve into various optimization strategies for implementing CNNs on FPGAs, focusing on balancing computational efficiency and memory usage. They discuss the challenges of non-quantized CNNs, such as the high demand for computational resources and the need for efficient data reuse and memory access patterns. The paper provides a comprehensive overview of techniques to enhance the performance of FPGA-based CNN accelerators.

Another significant contribution to this domain is the study in [17] which delves into various optimization techniques that circumvent the need for quantization while preserving the precision of the CNN models. The research highlights the challenges of balancing computational efficiency with accuracy, emphasizing the importance of maintaining high precision in scenarios where model accuracy cannot be compromised.

Recent advancements in quantized CNNs on FPGAs have focused on optimizing both power consumption and computational efficiency while maintaining model accuracy. Authors in [18] introduce FPQNet, a fully pipelined and quantized CNN implementation optimized for ultra-low latency and high throughput on FPGAs. Their approach leverages channel parallelization and layer pipelining, achieving significant performance improvements for video processing applications. Similarly, [19] proposes a logarithmic quantization method for CNNs, which reduces hardware complexity and power consumption, demonstrating the effectiveness of their approach on FPGA-based accelerators. In addition, [20] further advanced the field with MBFQuant, a mixed-precision quantization technique tailored for mobile CNN applications, optimizing hardware utilization without compromising accuracy. Finally, [21] presents an FPGA implementation based on fixed-point quantization approaches, including SqueezeJet-3, which utilizes 8-bit dynamic fixed-point arithmetic for CNN acceleration on edge devices.

2.1.2. Full-precision (non-quantized) accelerators

Recent advancements in implementing CNNs on FPGAs have highlighted the critical balance between model precision on one hand and performance and power optimizations on the other. One notable study is [3] which underscores the trade-offs associated with maintaining full precision in CNNs as opposed to adopting quantization techniques. The authors demonstrate that while quantized models can significantly reduce power consumption and enhance processing speed, they often suffer from a noticeable loss in accuracy. Full precision models, although more resource-intensive, retain the original accuracy of the network, which is paramount in precision-sensitive applications.

[22] offers a comprehensive exploration of strategies for implementing CNNs on FPGAs. The paper provides a comparative analysis of full-precision networks against their quantized counterparts, revealing that although quantization offers substantial improvements in resource efficiency and processing speed, it typically results in a significant drop in model accuracy. This study advocates for a meticulous evaluation of the precision requirements in applications, stressing that the accuracy of full-precision networks makes them indispensable in contexts where precision is crucial.

Finally, these studies collectively illuminate the importance of precision in FPGA-based CNN implementations and the potential drawbacks of quantization. They highlight the necessity of optimizing FPGA resources while ensuring that the inherent accuracy of the CNN models is preserved, thereby offering valuable insights for the development of high-performance, precision-critical FPGA-based neural network accelerators. In summary, to the best of the authors' knowledge, no design framework for FPGA accelerators exists that can efficiently implement non-quantized CNNs on virtually any FPGA (of one of the two major vendors) while also emphasizing on power efficiency.

2.1.3. High-level-to-RTL translation tools

High-level CNN-to-FPGA compilers have emerged to streamline accelerator development. IBM's AccDNN [23] (DNNBuilder) is a seminal example, converting Caffe CNN models directly into optimized RTL with no manual coding. It instantiates pre-optimized layer IPs and pipelines them for maximum throughput, even performing design-space exploration to tune parallelism under resource constraints. This yields very high performance and strong energy efficiency in published designs. However, AccDNN's flexibility is limited by supporting only standard layer types and Caffe-trained networks (up to 15 layers), often requiring model quantization for feasible FPGA deployment. In contrast, Rivera et al. [24] developed an automatic CNN-to-RTL framework with a push-button workflow. Their CINVESTAV platform provides a GUI for model configuration and handles everything from training to bitstream generation using hand-written Verilog templates for each layer. This greatly reduces the effort to integrate new CNN models – the user simply defines the architecture and lets the tool produce a custom accelerator "quickly and easily". The generated designs leverage parameterized parallelism but are restricted to the supported layer set and primarily fixed-point precision, thus the designed hardware modules does not support full floating point precision. A more recent tool is AutoVCoder [25], which explores large-language-models to generate HDL code automatically. AutoVCoder is an open-source framework that uses domain-specific training and retrieval-augmented generation to improve Verilog code correctness from LLMs. In principle, it can reduce development time – the tool itself writes the CNN accelerator RTL given a high-level description – and offers unprecedented flexibility to target novel architectures. However, this approach remains nascent, as it lacks built-in power or resource optimizations and the quality of results depends on AI training, often requiring verification and potentially manual tuning.

Our Darknet-based framework focuses mainly on developer productivity and architectural flexibility. Specifically, it supports full-precision CNN models and open-source deployment on diverse Xilinx devices (from the edge-class Kria KR260 to the large Alveo U55C), enabling low-effort mapping of new networks without quantization. Unlike prior tools, which often trade off between precision and performance, our approach prioritizes ease of use and model fidelity – new CNNs can be integrated by simply using their Darknet definitions, while adjustable design parameters allow scaling to available FPGA resources. This flexibility comes at a cost in resource usage compared to heavily quantized or manually optimized solutions, but as proved, it delivers practical throughput with minimal hand engineering.

2.2. Matrix multiplication algorithm

Since the most processing demanding tasks in CNNs are the required Matrix multiplications, several researchers have focused on proposing optimization methods and approaches that take full advantage of the underlying characteristics of modern FPGAs. Since we have also tackled this task in order to implement our efficient design framework in this subsection we list the most relevant papers in this field.

Ahmad and Pasha [26] investigated several optimization techniques for hardware-accelerated general matrix multiplication on FPGAs, with a specific focus on Convolutional Neural Networks (CNNs). In contrast to [26], our approach provides a comprehensive library that empowers users to customize FPGA-based matrix multiplication to their unique computational requirements, irrelevant of the application domain. Although Haghi et al. [27] present a reconfigurable FPGA assistant for in-network computations, with an accompanying case study on distributed matrix multiplication, their study focuses on reconfigurable compute-in-the-network FPGA assistance for collective support. Our work is orthogonal to this since it mainly aims at taking full advantage of the hardware resources of a single FPGA while providing adaptability for (i) implementation in different FPGA devices and (ii) easy integration with design flows which focus on distributed FPGAs.

The study presented in [28] focuses on high-level synthesis (HLS) techniques to optimize blocked floating-point matrix multiplication, a fundamental operation in many scientific computing applications. It explores strategies for improving the efficiency of matrix multiplication through HLS, specifically targeting architectures where floating-point operations are crucial. The work demonstrates how optimized synthesis can lead to significant performance improvements in hardware implementations, highlighting the benefits of HLS for computationally intensive tasks.

De Fine Licht et al. [29,30] presented their research results on transformations of HLS code for HPC and flexible communication respectively. However, their approach has certain limitations: (i) they are dependent on the HW implementation tool (e.g. they can be used only up to AMD Vitis 2021.1[2]), thus limiting their use and (ii) they rely on a hardware library developed by the authors, tailored to the FPGA implementation tool, which introduces a degree of tool dependency and thus undermining the adaptability and scalability of their approach, and they do not exploit state-of-the-art HBM2 memories. In summary, unlike gemm_hls – which streams matrix B from external memory and depends on a specialized vendor-specific library – our matrix multiplication module is an HLS design which does not use any existing libraries or vendor specific tools/mechanisms that buffers entire matrix B in segments into on-chip BRAM (streaming only A and C) and avoids any vendor-specific IP, thus achieving greater data reuse and portability.

In comparison to those studies, we present in [31] and [32] a methodology which is implemented purely in C with HW-oriented pragmas and it is independent of specific versions of tools and libraries. We, thus, establish a more robust and sustainable solution for enhancing computational performance in HLS designed FPGA systems. Furthermore, our methodology can even outperform the systems presented above since it can be implemented on newer FPGAs, while it is more developer-friendly and it can be seamlessly instantiated in any AMD FPGA (from relatively small to high-end ones).

Moving to the Graphical Processing Units (GPUs) area, several studies have explored the use of cuBLAS for efficient matrix multiplication on GPUs, as well as optimizations for power efficiency and fault tolerance. Authors in [33] provided insights into the anatomy of high-performance General Matrix Multiply (GEMM) operations on GPUs, emphasizing also on fault tolerance mechanisms to ensure reliable computation in the presence of hardware faults. In a similar vein, [34] delves into optimization strategies that enhance the performance of dense matrix multiplication across different GPU architectures, highlighting cuBLAS's adaptability and efficiency. Furthermore, [35] underscores cuBLAS's utility in various linear algebra computations beyond GEMM, demonstrating its comprehensive optimization for diverse matrix operations. These works collectively illustrate cuBLAS's capabilities in maximizing performance and reliability on NVIDIA GPUs.

For AMD GPUs, rocBLAS high-performance library has been developed for matrix operations exploiting the specific architectural features of AMD hardware. Authors in [36] present a robust framework built on rocBLAS that efficiently handles batched matrix multiplications, even with unbalanced input sizes, showcasing rocBLAS's flexibility and efficiency. Additionally, [37] provides a comprehensive evaluation of rocBLAS, emphasizing its strengths in terms of performance, power efficiency, and programmability, particularly when executed on AMD's advanced matrix cores. These studies highlight rocBLAS's effective use of AMD hardware to deliver high performance while maintaining energy efficiency.

While our performance metrics indicate slower computation times compared to GPU-based solutions such as cuBLAS and rocBLAS in state-of-the-art NVIDIA and AMD GPUs accordingly, our approach demonstrates significantly better power efficiency. This makes our approach particularly advantageous in power-constrained environments, where energy consumption is critical.

Moreover, the FPGA systems that are implemented by our design framework can seamlessly utilize several techniques for power/energy reduction. So, dynamic power gating methods such as the ones in [38] are orthogonal to our design approach and can be integrated into the final modules. The same applies for clock gating methods proposed by either other researchers as in [39] or by the FPGA vendors themselves (e.g. [40]) as well as DVFS approaches [41].

3. Background

CNNs have revolutionized the field of computer vision and are now a cornerstone in numerous application domains. The primary building block of a CNN is the convolutional layer, which performs a mathematical operation called convolution.

3.1. Convolutional layer

In a convolutional layer, an input image (or feature map) is convoluted with a set of learnable filters (or kernels) to produce a new feature map. Mathematically, the convolution operation for a single output feature map can be expressed as:

$$C^k_{i,j} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{i+m,j+n} \cdot B^k_{m,n} + b^k \tag{1}$$

where:

- $C^k_{i,j}$ is the output feature map at position (i, j) for the kth filter.
- $A_{i+m,j+n}$ is the input feature map at position $(i + m, j + n)$.
- $B^k_{m,n}$ is the weight of the kth filter at position (m, n).
- b^k is the bias term for the kth filter.
- M and N are the dimensions of the filter.

This operation is repeated for each filter, producing multiple output feature maps that capture different aspects of the input. The power consumption in CNNs is heavily influenced by the number of operations performed, particularly the multiplications and additions in the convolutional layers. Reducing power consumption while maintaining computational efficiency is a significant challenge, especially for real-time applications.

3.2. Matrix multiplication for accelerating convolutional layers

To accelerate the convolutional layer, one effective approach is to transform the convolution operation into a matrix multiplication problem. This transformation leverages the highly optimized algorithms and hardware accelerators available for matrix multiplication, thereby improving computational efficiency. The im2col (image-to-column) transformation is commonly used to achieve this. The im2col process involves unfolding the input feature map into a matrix where each column represents a local region of the input that will be convolved with the filter. This transformation allows the convolution operation to be expressed as a single matrix multiplication:

$$C = B \cdot A_{col} + b \tag{2}$$

where:

- C is the matrix of output feature maps.
- B is the matrix of unfolded filters.
- C_{col} is the matrix of unfolded input feature maps.
- b is the bias matrix.

The transformation of the convolution operations to matrix multiplications offer several advantages:

[2] https://github.com/spcl/gemm_hls/issues/25

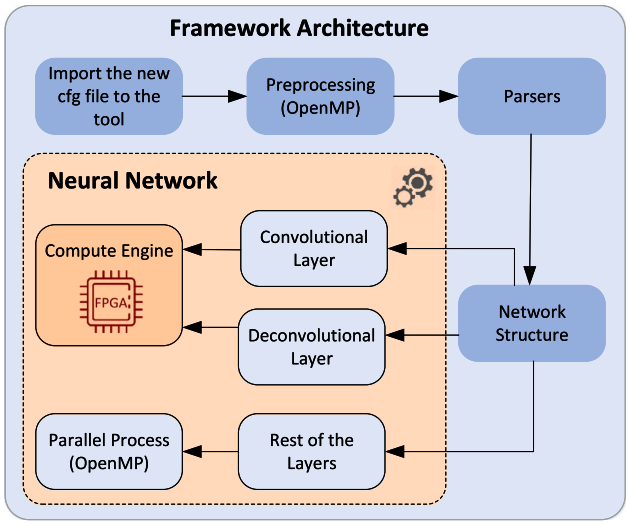


Fig. 1. Architecture of the framework.

- **Parallelism**: Matrix multiplication can be easily parallelized, making it well-suited for hardware acceleration on platforms like FPGAs.
- **Optimization**: There are numerous optimized libraries and techniques for matrix multiplication, which can be leveraged to speed up the convolution operation.
- **Efficiency**: By transforming convolution into a matrix multiplication problem, the number of redundant calculations is reduced, thereby improving computational efficiency.

These advantages make the matrix multiplication approach a powerful technique for accelerating convolutional layers, particularly in parallel hardware implementations such as those in FPGAs.

3.3. Non-quantized CNNs on FPGAs

While quantization techniques, which reduce the precision of the weights and activations, are commonly used to improve the power efficiency of CNNs, they can lead to a loss in accuracy. Non-quantized CNNs, which use full-precision floating-point or fixed-point representations, maintain the original accuracy of the model but pose challenges in terms of power consumption and performance.

Implementing non-quantized CNNs on FPGAs requires careful consideration of the trade-offs between resource allocation and power efficiency. The focus of this work is to develop a framework that leverages the parallelism and reconfigurability of FPGAs to achieve power-efficient execution of non-quantized CNNs without compromising accuracy.

4. Proposed design framework

In this section, we present the architecture of the proposed framework emphasizing on the convolutional kernel and the Software code which is running in the CPU handling the different convolutional layers efficiently.

4.1. Framework architecture

Fig. 1 presents the framework architecture of our tool which handles convolutional and deconvolutional layers within neural networks in an automated manner.

This architecture is represented as a flowchart, emphasizing the sequential and parallel processes involved in the tool's operation. The tool is based on the Darknet framework [42] and is designed to accelerate the processing of convolutional layers while reducing their energy consumption when implemented on heterogeneous systems containing reconfigurable resources.

1. **Import new cfg file to the tool**: In the initial step the user imports a configuration (cfg) file into the tool. This file likely contains specifications and parameters necessary for setting up the neural network layers.
2. **Preprocessing (OpenMP)**: Following the import, the tool performs preprocessing; this stage leverages OpenMP, a parallel programming model, to enhance the efficiency of preprocessing tasks, which could involve data normalization, augmentation, or other preparatory operations.
3. **Parsers**: The parsed data from the preprocessing stage are handled by parsers, which likely convert the preprocessed data into a format suitable for further processing and analysis.
4. **Network Structure**: The parsed data feed into the network structure component; this component organizes the neural network's architecture, defining the layers and their connections based on the cfg file's specifications.
5. **Layer Segmentation**: The neural network structure is then segmented into three distinct pathways:
    - **Convolutional Layer**: This pathway handles convolutional layers specifically designed for feature extraction. The processed data from these layers are forwarded to the Compute Engine (i.e. FPGA Kernel) for accelerated processing.
    - **Deconvolutional Layer**: This pathway deals with deconvolutional layers, often used in tasks such as image reconstruction. Similar to the convolutional layer, it utilizes the Compute Engine (FPGA Kernel) for efficient computation.
    - **Rest of the Layers**: This pathway encompasses the remaining layers that do not fall under convolutional or deconvolutional categories. These layers are processed using parallel processing in the CPU facilitated by OpenMP.

By using the Darknet framework [42] as basis for our work, we take advantage of its flexibility and very wide user base. Moreover, by utilizing the configuration file, the framework allows us to seamlessly integrate and experiment with various CNN architectures without the need for extensive reprogramming.

As mentioned above, all SW processes are parallelized with the use of OpenMP in order to take advantage of all the cores in any multi-core CPU and get the best possible performance.

4.2. Host-kernel interaction

Fig. 2 provides a detailed flowchart illustrating the interaction between the host and the tasks executed on the FPGA (i.e. FPGA kernels) during the processing of convolutional and deconvolutional layers. This dual functionality necessitates a more complex control structure within the kernel to differentiate between the two types of operations. Unlike the straightforward matrix multiplication kernel, presented in Section 4.3, which focuses solely on dense matrix operations, this implementation includes conditional logic to switch between convolutional and deconvolutional computations. This is achieved through additional parameters and control flags that guide the kernel's behavior based on the specific layer being processed.

In terms of computational flow, the convolutional operation involves sliding a set of filters over the input feature map to produce output feature maps. This sliding window mechanism requires efficient handling of overlapping regions of the input data, which is significantly different from the row-wise and column-wise multiplications characteristic of traditional matrix multiplication.

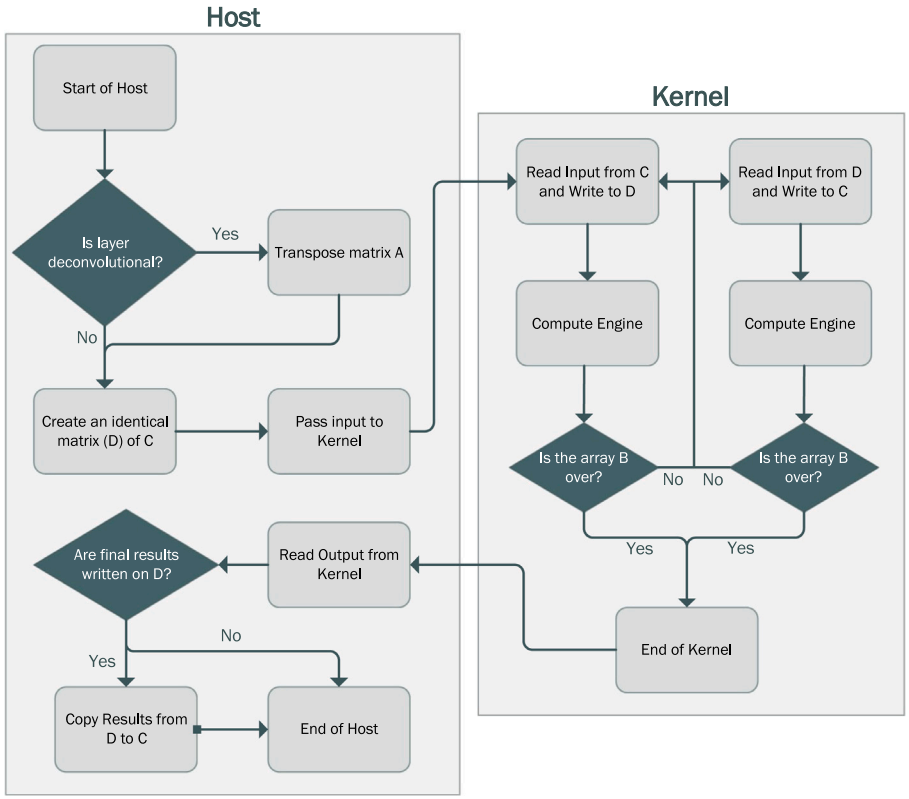


Fig. 2. Architecture of the convolution layer computation.

Moreover, the deconvolutional operation, which essentially performs a reverse convolution, introduces additional complexity in terms of data distribution and accumulation. We manage to addresses these challenges by implementing specialized data handling routines that ensure correct alignment and summation of overlapping regions in the output feature maps. This is very different from the straightforward accumulation of products in matrix multiplication thus requiring a more sophisticated approach to manage the expanded spatial dimensions of the deconvolutional output.

A more detailed analysis of Fig. 2 is presented below:

1. **Host to Kernel Operations**:
   - The host process begins and determines whether the current layer is deconvolutional; if it is, matrix A is transposed to facilitate efficient computation. If the layer is convolutional no intervention on matrix A is needed.
   - An identical matrix D of C is created, and the input is passed to the FPGA kernel. In our implementation, matrix D serves as an internal buffer corresponding to the output matrix C, handling the intermediate read-write operations during the pipelined dataflow execution; so one buffer is used for reading partial sums while the other is updated, allowing continuous streaming of data without stalling the computational pipeline.

2. **FPGA Kernel Operations**:
   - Data flows from the host to the kernel, where it undergoes processing by the Compute Engine.
   - The kernel starts by reading the input data from matrices C and D. In order to exploit the FPGA parallel resources and maximize the pipeline efficiency we use 2 matrices; in each iteration we read from one matrix and write the results to the other matrix.
   - The Compute Engine processes the input data; for convolutional and deconvolutional layers, this typically involves performing mathematical operations such as filtering and feature mapping.
   - Then it reads Data from array C and writes the results to array D.
   - The kernel continuously checks if the array B is completely processed. If not it continues by reading input from D and write output in C and so on.
   - Once array B is fully processed, the kernel ends its operation and sends the results back to the host.
   - Processed data are checked for completeness and returned to the host.
   - The host either ends its operation or copies results based on predefined conditions, ensuring that the final processed data are correctly formatted and stored.

3. **FPGA Kernel to Host Operations**:
   - After kernel execution, the output feature maps are retrieved from the FPGA's global memory and transferred back to the host.
   - Finally, the host checks the matrix in which the final results were written; if the final results are written to matrix D, it copies them to matrix C, concluding the host operations. Otherwise, no action is performed since the final results are already written in matrix C.

The clear separation of tasks and conditional checks ensures that the system operates efficiently and effectively, handling various types of CNN layers in different NN architectures. Based on the Darknet framework, our proposed extensions allow for the seamless acceleration

of the convolutional layers of a wide range of different CNNs, achieving a fast and power-efficient implementation.

In summary, the convolutional kernel in addition to the matrix multiplication kernel presented in Section 4.3 it also implements the required padding and striding while supporting multiple channels. The host code is responsible for preparing data, managing kernel execution, and handling results, ensuring efficient operation and minimal data transfer overhead.

4.3. Implementation of matrix multiplication algorithm

4.3.1. Reference implementation

Our first FPGA-implemented Kernel handles mainly the matrix multiplication tasks which is the cornerstone in virtually every CNN implementation. In order to demonstrate its efficiency, we start with a basic reference matrix multiplication implementation, in which we adopt a simple effective pipeline approach aiming at maximizing the computational efficiency on the targeted FPGAs. This design encompasses the most widely used algorithm for efficient execution of dense matrix multiplication:

```
# Input: Matrices A[M][K], B[K][N]
# Output: Matrix C[M][N] = A * B (accumulated)
for m from 0 to M-1 do
  for k from 0 to K-1 do
    for n from 0 to N-1 do
#pragma HLS pipeline
      C[m, n] <- C[m, n] + A[m, k] * B[k, n]
```

Listing 1: Naive implementation of general matrix multiplication

The algorithm comprises of nested loops that meticulously traverse matrices A, B, and C, and calculate the dot product of a row from matrix A with a corresponding column from matrix B. The key element for the HW implementation is the High Level Synthesis pipeline directive, which allows the execution of one multiplication and one addition in a single clock cycle achieving a relatively high throughput and performance. However, the main drawback of this method is the excessive memory accesses to external DRAMs & HBMs (i.e. in total M*N*K accesses in all arrays) preventing further parallelization.

To bridge the gap between the naive baseline and our fully optimized dataflow architecture, we also apply vectorization, not as a novel contribution, but simply as an intermediate transformation that reduces external memory accesses and prepares the design for the subsequent optimizations.

In traditional FPGA matrix multiplication implementations, the frequent accesses to external memories have been a performance bottleneck since each DRAM access adds latency and consumes valuable energy. In our approach, we propose the use of 512 bit vector elements to dramatically reduce the number of separate external memory accesses and take advantage of the wide interfaces of FPGAs to multiple DRAMs or HBMs. This reduction is achieved by aggregating multiple data points into a single, wide uint512_t element, effectively consolidating data transfers and parallel computations as illustrated in Listing 2 (single precision numbers). Fewer external accesses translate into lower latency and less contention for memory resources, resulting in higher overall performance and energy efficiency. The use of uint512_t elements results in improved FPGA on chip memory (Block RAM-BRAM) utilization since by using wider data elements we increase the utilization of each BRAM memory thus maximizing the on-chip data storage.

```
# Input: A[M][K], B[K][N]
# Output: C[M][N]
# DATAWIDTH = 512 bits, DATATYPE_SIZE = 32 bits (single
    precision)
# VECTOR = DATAWIDTH / DATATYPE_SIZE (number of elements
    per packed vector)

N_vec <- N / VECTOR
K_vec <- K              # conceptual; vectorization
    applied along N

for m from 0 to M-1 do
  for k from 0 to K-1 do
    scalar_a <- A[m, k]                          # M*K DRAM
        Accesses
    for n_vec from 0 to N_vec-1 do
#pragma HLS pipeline
      // 3*M*K*N/VECTOR DRAM Accesses
      vec_b <- load_vector_B(k, n_vec)
      vec_c <- load_vector_C(m, n_vec)

      for v from 0 to VECTOR-1 do
#pragma HLS UNROLL
        vec_c[v]  <- vec_c[v] + scalar_a * vec_b[v]

      store_vector_C(m, n_vec, vec_c)
```

Listing 2: Matrix multiplication vectorization

The optimization of the data handling is done through the following functions: **uint_to_float** and **float_to_uint**. These functions seamlessly pack and unpack data between uint512_t and floating-point representations (single, double and half precision) in one clock cycle. To further improve our efficiency, both functions utilize the #pragma HLS INLINE directive, which instructs the compiler to inline the code thus reducing the control overhead and develop a more efficient data path.

In addition, modern FPGAs are connected to multiple memory banks with dedicated channels (e.g., multiple DRAM modules or HBM lanes) in order to increase the external memory bandwidth. In order to take advantage of those multiple lanes/modules we insert the appropriate HLS directives that split data transfers into a parameterizable number of memory banks/lanes so as to take full advantage of the bandwidth available in each of the memory interfaces.

4.3.2. Innovative optimization flow

Our matrix multiplication accelerator is a fully custom design built without the use of any existing library and tools, with a dataflow architecture that avoids any reliance on vendor-specific IP cores or libraries. In contrast to gemm_hls [30] – which streams the B matrix from off-chip memory and utilizes a specialized vendor-tied hardware library – we buffer the entire matrix B in segments into on-chip BRAM and stream only matrix A and the result of matrix C through the pipeline. In practice, BUFF_K and BUFF_N are selected so that the largest feasible part of matrix B fits in BRAM, thereby maximizing data reuse and minimizing DRAM transactions within each tile. This strategy significantly reduces external memory traffic and access contention by enabling extensive reuse of B's data on-chip (whereas gemm_hls must continually re-fetch B from memory for each computation pass). As presented in Fig. 3, we apply HLS optimizations (e.g., loop unrolling, pipelining, dataflow) via pragmas in plain C code without leveraging any proprietary IP cores or vendor-specific libraries. This vendor-agnostic approach not only highlights the originality of our design but also ensures its portability across different FPGA platforms, making it different from prior solutions like gemm_hls. This optimization flow exploits FPGA-specific architectural features—such as BRAM banking, BRAM partitioning aligned to unrolled compute units, and concurrent load/compute/store hardware pipelines—which are not present in GPUs with fixed memory hierarchies.

Moreover, as illustrated in Fig. 4, we utilize the internal BRAMs in conjunction with HLS streams, to optimize also the on-chip memory access patterns and further increase the computational efficiency. We

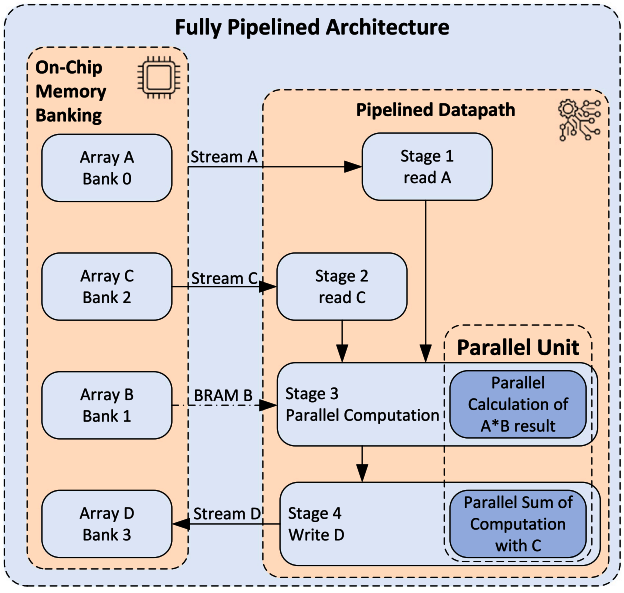


Fig. 3. Fully pipelined architecture.

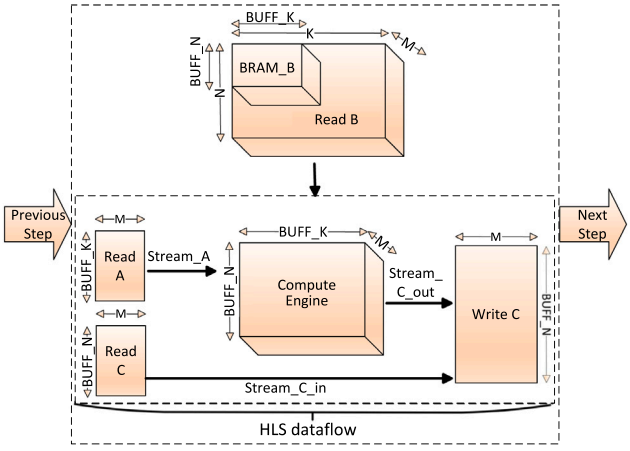


Fig. 4. Architecture of innovative optimization flow.

allocate BRAM resources for matrix BRAM_B and use HLS streams for matrices A and C. Stream_A and Stream_C_in are used to read matrices A and C respectively from DRAM/HBM and Stream_C_out to transfer the data from the internal computation modules back to DRAMs/HBMs.

The aforementioned outcome leads to a significant decrease in the effective latency associated with off-chip memory retrieval and guarantees the accessibility of data for computational purposes, thereby diminishing idle time caused by external memory latency. Specifically, streams enable a continuous flow of data between processing elements without the need for intermediary storage in BRAMs, minimizing the latency and resource overhead. By using streams, data are transferred directly between the producer and consumer processes facilitation pipelined execution and enhancing parallelism. This direct transfer mechanism reduces the utilization of BRAMs for temporary storage, and thus leads to more efficient resource utilization, enabling us to utilize a larger part of array B in our compute engine. Moreover, it reduces the latency of writing and reading from BRAMs.

To further reduce the memory access overhead, we strategically reorder the traversal of the matrices. By iterating through K and N outside of the main computation loop, we take more advantage of the caching in BRAMs, which significantly reduces the DRAM/HBM accesses. This reordering guarantees also a more contiguous and efficient retrieval of the data from the external memory/ies. So, all external accesses are cached into BRAMs in 512-bit quantities, minimizing by at least three orders of magnitude the total DRAM accesses (depending on BUFF_K & BUFF_N) as described in Listing 3. By also using HLS streams and the DATAFLOW pragma we can read matrix A from the external memory/ies simultaneously with the actual computations that use the previous data items while, since BRAMs are dual-port, we can also simultaneously read matrix C from the external memory/ies alongside with the writing back of the results to them.

```
# Input: A[M][K], B[K][N]
# Output: C[M][N]
# Tiling factors: BUFF_K, BUFF_N
# On-chip buffers: BRAM_B[BUFF_K][BUFF_N]
# Streams: stream_A, stream_C_in, stream_C_out

for ex_k from 0 to K-1 step BUFF_K do
  for ex_n from 0 to N_vec-1 step BUFF_N do

    //Stage 1: Read tile of B into BRAM
    for k from 0 to BUFF_K-1 do
      for n from 0 to BUFF_N-1 do
#pragma HLS pipeline
        BRAM_B[k, n] <- load_vector_B(ex_k + k, ex_n + n)

#pragma HLS DATAFLOW
    //Stage 2: Stream tiles of A
    for m from 0 to M-1 do
      for k_vec from 0 to BUFF_K_vec-1 do
#pragma HLS pipeline
        stream_A.push( load_vector_A(m, ex_k_vec + k_vec)
            )

    //Stage 2: Stream tiles of C
    for m from 0 to M-1 do
        for n from 0 to BUFF_N-1 do
#pragma HLS pipeline
        stream_C_in.push( load_vector_C(m, ex_n + n) )

    //Stage 3: Compute on-chip using BRAM_B and streams
    for m from 0 to M-1 do
      for k_vec from 0 to BUFF_K_vec-1 do
#pragma HLS pipeline
        //Parallel calculation of MM and accumulation to
            stream_C_out
        //UNROLL_N*UNROLL_K*VECTOR elements in 1 cycle
        for n from 0 to BUFF_N-1 do
            stream_C_out.push( BRAM_C[n] )

    //Stage 4: Write back C tile to external memory
    for m from 0 to M-1 do
      for n from 0 to BUFF_N-1 do
        store_vector_C(m, ex_n + n, vec_c_in + vec_c_out)
```

Listing 3: Innovative Memmory Access

To provide further clarification, as illustrated in Fig. 5, Stream_A is read in uint512_t format in order to reduce latency even further. After that, Stream_A and BRAM_B are fed to the computational part in order to calculate the outcomes. The outcomes for whole BUFF_K dimension are stored in BRAM_C array and then streamed with Stream_C_out in WriteC sized blocks.

As demonstrated in Section 5.3 our approach can leverage both HBMs and DRAMs. Moreover, through the allocation of matrices among the three Sliced Logic Regions (SLRs) of the recent AMD Alveo devices, we can further parallelize the kernel operations, which translate to even better performance and energy efficiency.

Finally, we strategically utilize the *#pragma HLS pipeline* in the matrix multiplication computation tile. This directive plays a critical

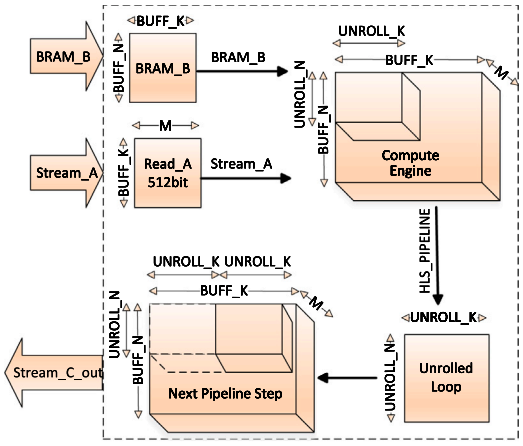


Fig. 5. Architecture of compute engine.

role in enabling fully pipelined processing for the entire tile. In terms of the implemented Synthesizable C code each iteration of the loops becomes a distinct processing step executed within a single clock cycle. In that respect we reduce pipeline stalls, guaranteeing an uninterrupted stream of data to the multiplication and addition hardware, achieving the processing of UNROLL_N*UNROLL_K*VECTOR elements in a **single (1) clock cycle**.

The implemented kernels employ several key HLS optimization directives, including *#pragma HLS INLINE*, *#pragma HLS PIPELINE II=1*, *#pragma HLS UNROLL*, *#pragma HLS DATAFLOW*, *#pragma HLS INTERFACE*, and *#pragma HLS ARRAY_PARTITION*. These pragmas collectively enable fully pipelined computation loops, efficient memory access, and maximum data reuse across the complete datapath. The achievable FPGA clock frequency depends on both the selected device and the selected parallelizing parameters. For instance, on the AMD Alveo U55C the maximum post-implementation clock reaches approximately 270 MHz for high-performance configurations and up to 340 MHz for lower-utilization designs. On the embedded Kria KR260 platform, the corresponding frequency ranges from 200 MHz to 240 MHz depending on the design complexity and resource utilization. Higher unroll factors increase the degree of parallelism but they also lead to more complex datapaths and routing, sometimes resulting in a lower maximum implementation frequency. Conversely, smaller unroll factors allow higher operating frequencies.

One very important characteristics of our design framework is that all those optimization schemes can be used by the CNN designer through only four (4) fully configurable parameters: buffer sizes BUFF_K, BUFF_N as well as loop unrolling parameters UNROLL_N and UNROLL_K. These parameters enable the designer to customize the architecture to fully utilize any FPGA architecture/size and memory technology/topology. Through the careful selection of buffer sizes and unrolling factors, the designer can enhance memory access patterns and computational parallelism, achieving an optimal balance between resource usage, in case there are additional modules that are also placed on the same device, and throughput. The presented approach can also be simulated using the full-system, cycle-accurate COSSIM simulator which can simulate custom co-processors implemented in HLS [43].

5. Experimental results and discussion

5.1. Performance of proposed design and implementation framework

In order to demonstrate the efficiency of our framework in terms of design time, Fig. 6 presents the development time needed to import a new CNN to our framework; more complex neural networks require

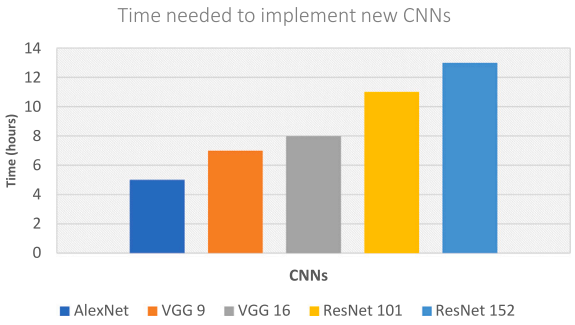


Fig. 6. Development time needed to import new CNNs to the framework.

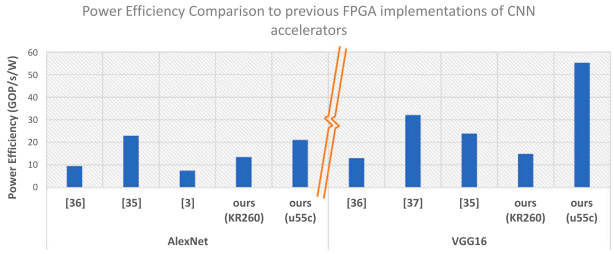


Fig. 7. Power efficiency comparison to previous FPGA implementations of CNN accelerators.

only marginally more development time since the main input is the graphical scheme of the CNN and the values of the different optimization parameters described in Section 4.1. As demonstrated, the design time which is required for all CNNs is less than 13 h of a junior engineer with less than 1.5 years of experience in FPGA design. This proves that our tool allows for very high designer productivity.

Moving to the performance and energy/power consumption of our CNN implementation, Table 1 presents the comparison between our work and the numerous similar systems presented in the related work section. As demonstrated, when compared to similar frameworks that support full precision, the power efficiency is superior when AlexNet is run on [3] and VGG16 is run on [44], while is equivalent when AlexNet is run on [44]. Furthermore, the power efficiency of our solution is significantly higher than that of quantized solutions [45] and [46] running AlexNet and VGG16 respectively. As summarized in Fig. 7, our approach exhibits a significant increase in power efficiency compared to previous FPGA implementations of CNN accelerators.

To ensure the fairest possible comparison across heterogeneous CPU, GPU, and FPGA platforms, it should be stated that all evaluated devices are implemented in similar processes (12,nm for the NVIDIA Jetson AGX Xavier, 14,nm for the Intel Xeon E5-2620v4, and 16,nm for the AMD Alveo U55C, Kria KR260, and NVIDIA GTX 1080), and therefore, are expected to exhibit comparable CMOS efficiency characteristics. Furthermore, differences in thermal design power (TDP) are inherently normalized in our evaluation by using the performance-per-watt metric (GFLOPS/W), which is widely accepted as a technology-agnostic measure for cross-platform energy efficiency comparisons.

All reported power results were obtained through the official AMD Xilinx tools. Specifically, both the *Xilinx Power Estimator (XPE)* and the *Power Design Manager (PDM)* were used after the synthesis and implementation stages to compute static and dynamic power consumption.

Finally, Table 2 presents the comparison of SW parallel implementations and our solution. The selected high-end CPU is Intel Xeon E5, which is implemented in a CMOS technology which is marginally better that of the target FPGA (Alveo u55c). Similarly, Table 3 presents the comparison of the performance and power consumption of the parallel SW execution on an embedded multi-core ARM CPU which resides on the same device as our reconfigurable resources (KR260). In both

Table 1
Comparison to previous FPGA implementations of CNN accelerators.

	AlexNet					VGG 16				
	[45]	[44]	[3]	Ours	Ours	[45]	[46]	[44]	Ours	Ours
Device	ZCU102	VC709	7A100T	KR260	u55c	ZCU102	ZCU102	VC709	KR260	u55c
Year	2019	2021	2024	2025	2025	2019	2020	2021	2025	2025
Precision	16b fx	32b fp	32b fp	32b fp	32b fp	16b fx	16b fx	32b fp	32b fp	32b fp
Latency (ms)	–	–	123.87	107	30	–	–	–	1153	150.45
Performance (GOP/s)	223.4	220.0	12.095	20.131	71.8	309.0	495.4	230.1	26.620	204.002
Power Efficiency (GOP/s/W)	**9.466**	**22.9**	**7.478**	**13.421**	**21.118**	**13.093**	**32.169**	**23.9**	**14.789**	**55.136**

Table 2
Comparison of 5 CNNs implementations on Alveo u55c to CPU.

	AlexNet		VGG 9		VGG 16		ResNet 101		ResNet 152	
Device	Intel Xeon E5	Alveo u55c	Intel Xeon E5	Alveo u55c	Intel Xeon E5	Alveo u55c	Intel Xeon E5	Alveo u55c	Intel Xeon E5	Alveo u55c
Latency (sec)	11.6	0.03	8.5	0.328	173	0.150	104	0.285	156.6	0.385
Performance (GOP/s)	0.186	71.8	0.191	4.952	0.177	204	0.189	68.65	0.187	76.139
Power Efficiency (GOP/s/W)	0.0021	21.118	0.0022	1.457	0.002	60	0.0022	20.28	0.0022	22.394

Table 3
Comparison of 5 CNNs implementations on embedded CPU and KR260.

	AlexNet		VGG 9		VGG 16		ResNet 101		ResNet 152	
Device	ARM Cortex-A53	KR260	ARM Cortex-A53	KR260	ARM Cortex-A53	KR260	ARM Cortex-A53	KR260	ARM Cortex-A53	KR260
Latency (sec)	104.40	0.11	76.50	0.33	1557.00	1.15	936.00	2.19	1409.40	2.96
Performance (GOP/s)	0.02	20.13	0.02	4.99	0.02	26.62	0.02	9	0.02	9.94
Power Efficiency (GOP/s/W)	0.0023	13.42	0.0024	3.33	0.0021	17.75	0.0023	6	0.0023	6.62

comparisons our FPGA implementation can trigger significantly better performance.

5.2. Evaluation on a real-world use case

Our overall design and implementation flow has also been verified and evaluated on a real time system which is described in [47]. This underline UAV system is autonomous and employs Red–Green–Blue (RGB) and infrared (thermal) imaging to provide real-time alerts on the condition of power lines. The primary objective of the UAV is to identify and segment power lines from their background using a hybrid approach that integrates RGB and thermal data processing and an advanced CNN which requires full accuracy not only in the training but also in the inference process. This system aims to render the power line inspections in a more efficient, more accurate and safer manner, by minimizing human intervention, which is often time-consuming and hazardous. In a UAV-based power line inspection over the mountainous terrain of Crete, where wildfire risk is high, a full-precision CNN processing fused RGB and thermal camera inputs is required to reliably detect any faults in a single pass, as any repeat UAV flight would be prohibitively costly and dangerous. In the UAV pilot deployment in Crete, the system was required to operate on battery for an entire 5 km autonomous flight, making overall energy consumption a critical constraint for the target application. In detail, our measurements demonstrated that the FPGA achieved approximately 60% higher energy efficiency compared to the GPU. This meant that the GPU-based solution was unable to complete the full 5 km mission in most of the real-world scenarios tested, while the FPGA-based implementation completed the flight reliably.

The proposed system (Fig. 8) offers several significant benefits, including timely and precise fault detection, enhanced personnel safety, and reduced operational costs. By utilizing a custom-made drone platform, the system can perform in situ analyses, which enable immediate identification of faults and their exact locations. This capability is particularly advantageous in challenging terrains where traditional inspection methods are inefficient. The employed automated monitoring of the power lines is based on a specially designed CNN which allows for the processing of large volumes of visual and thermal data, ensuring that potential issues are detected and addressed promptly.

To evaluate the effectiveness of the proposed framework we tested it on the UAV intelligent system and we evaluated the NN in a real-life scenario. The real world experiment involved deploying UAVs equipped with CPUs, FPGAs and GPUs and both RGB and thermal sensors to monitor power lines in various regions of Greece. The diverse terrain and environmental conditions provided a robust testbed for assessing the system's performance.

Fig. 9 illustrates the experimental results when using different configurations: for the on-the-UAV processing we use a KRIA260 FPGA board and a Jetson AGX Xavier GPU board while for the offline case we process the exact same RGB and thermal data on an NVIDIA 1080 GPU and a U55 Alveo FPGA board. We also included the latency and power efficiency metrics regarding the unoptimized memory-accesses on both Kria KR260 and Alveo u55c, so as to highlight the improvements, in both latency and energy consumption, achieved by reducing memory transactions and external transfers, when our novel framework is utilized. The results reveal significant disparities in execution time and power efficiency among the online and offline platforms as expected. In both cases, though, our FPGA implementations stand out for their superior power efficiency, achieving 0.67 GFLOPS/W and 1.72 GFLOPS/W respectively, while maintaining competitive execution times with the GPUs. All GPU results were obtained using standard full-precision PyTorch inference, internally employing highly optimized CUDA libraries such as cuDNN and cuBLAS. The total energy consumption highlights the efficiency gap, with the Kria KR260 requiring 72 J versus 115 J on the Jetson AGX Xavier, and the Alveo u55c consumes only 28.2 J compared to 70J on the NVIDIA GTX 1080. This highlights the advantage of the designed FPGA modules in optimizing energy consumption without compromising on performance. In the online experiment, although the NVIDIA Jetson AGX Xavier offers low execution time, it falls short in power efficiency compared to the embedded FPGA platform, indicating a trade-off between speed and energy consumption.

In the offline experiment, traditional high-end CPUs and GPUs, such as the Xeon E5-2620 and NVIDIA 1080, exhibit higher execution times and lower power efficiency. The experimental results clearly demonstrate that the systems utilizing FPGAs and implemented by our novel, yet very easily used even by non-experienced FPGA developers, design and optimization framework provide a compelling solution for energy-sensitive applications, offering a balanced approach to performance and power efficiency while retaining the accuracy of the original CNN models.

5.3. Results on matrix multiplication algorithm

We have evaluated the efficiency of our open-source, fully parameterizable, purely-C library for dense matrix multiplication when

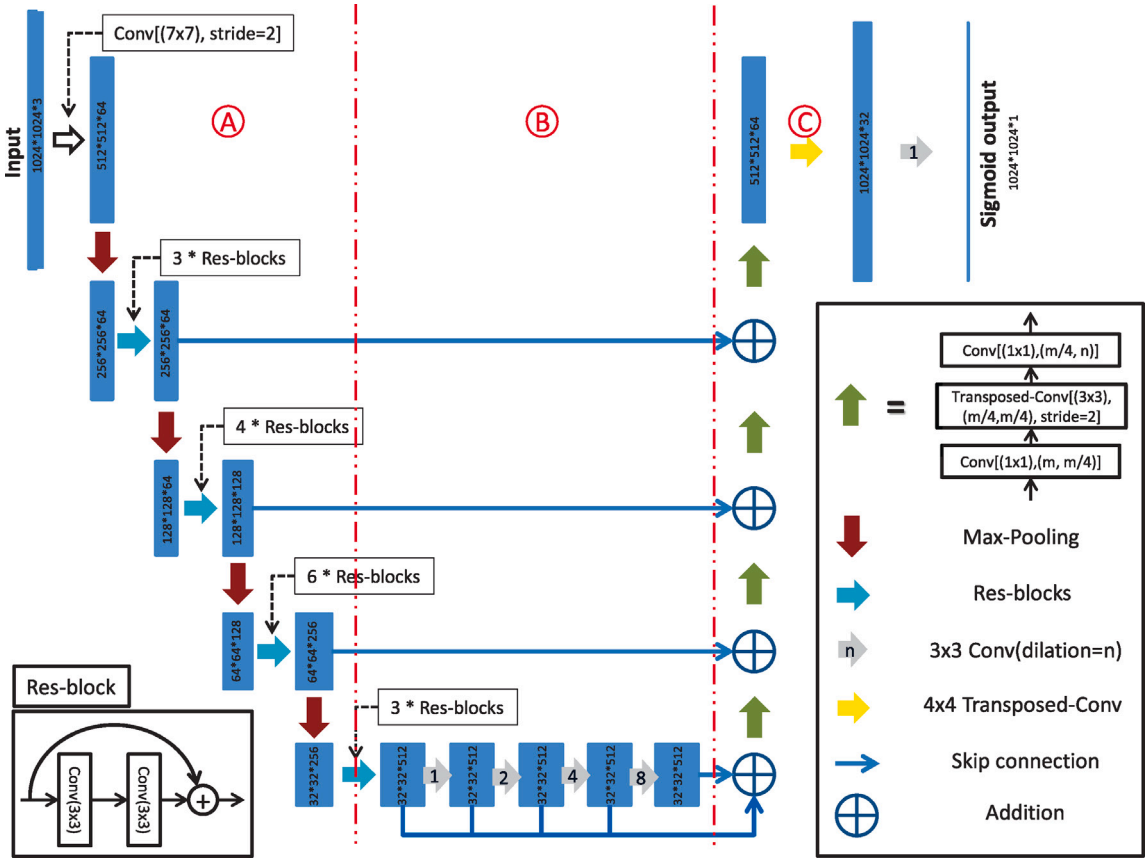


Fig. 8. D-LinkNet architecture.

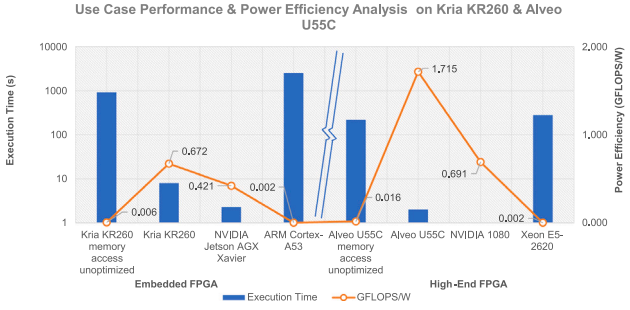


Fig. 9. Use case performance & power efficiency analysis on Kria KR260 & Alveo U55C.

targeting numerous FPGA boards. In order to comprehensively evaluate the efficacy and robustness of our proposed matrix multiplication approach, multiple experimental runs conducted across varying dimensions are presented in Table 4. The experimental results demonstrate that we can achieve high performance in diverse dimensional configurations which highlights the wide applicability of our approach. The maximum performance achieved is 249 GFLOPS for matrix sized of M = 2048, K = 16284 and N = 4096 on a high end FPGA (AMD Alveo U55C) and 24 GFLOPS for matrix sized of M = 512, K = 1024 and N = 2048 on an embedded one (Kria KR260), while the performance is also relatively high in smaller matrix sizes, in contrast to the GPU approaches.

To demonstrate further the effectiveness of the presented approach we compare our fully optimized model when executed on both a high-end FPGA (AMD Alveo U55C) and an embedded one (Kria KR260) with

Table 4
Results with multiple matrix dimensions.

M	K	N	U55 [GFLOPS]	U55 [Exec.]	KR260 [GFLOPS]	KR260 [Exec.]
512	1024	2048	207	0.0104 s	24	0.89 s
1024	2048	4096	215	0.08 s	22.6	0.76 s
1024	4096	2048	219	0.078 s	22.6	0.76 s
2048	2048	2048	215	0.08 s	22.3	0.77 s
2048	4096	16384	229	1.2 s	22	12.5 s
2048	16384	4096	249	1.1 s	22	6.25 s
4096	4096	4096	229	0.6 s	22	12.5 s

(i) the non-optimized reference model, (ii) the parallel execution in a multicore CPU, (iii) the CUDA implementation on an NVIDIA T4 GPU. In all experiments the array dimensions which are M = 2048, K = 4096, and N = 16384 are selected so as to be different from those triggering our peak performance demonstrating the flexibility of our approach. Furthermore, our results obtained from numerous experiments with different dimensions are fully inline with those presented in Fig. 10. As illustrated in this figure, our fully-optimized implementation for the embedded FPGA is two orders of magnitude faster than the reference implementation and 9x times faster compared to the fully parallelized algorithm executed on an embedded ARM 4core CPU (Cortex-A53) which resides, as a hard-IP, in the same chip as the reconfigurable resources; those numbers include all the memory accesses and the external memory technologies and topologies are exactly the same. In terms of total energy, the Alveo u55c consumes 30.5 J, substantially lower than the 91 J measured for the NVIDIA T4 under the same full-precision workload.

Similarly, our fully optimized approach when implemented on the Alveo U55C board is approximately three orders of magnitude faster than the reference implementation and 10× times faster compared to the fully parallelized algorithm executed on an Intel Xeon E5-2620 v4

Table 5
Comparison to previous FPGA implementations.

	Year	Device	Logic. Util. BRAM	Logic. Util. DSPs	Perf. FP32 [GFLOPS]	Perf. FP64 [GFLOPS]	Power Effic. GFLOPS/W	Tool/library independency	Open source
D' Hollander [28]	2016	Zynq-7000	32%	99%	5	–	–	–	X
Guan [48]	2017	Stratix V	67%	17%	100	–	2.92	–	X
gemm_hls [30]	2021	Alveo U200	58%	44%	211	74	8.49	X	✓
This work	2024	Alveo U55C	56%	44%	229	80	9.02	✓	✓
This work	2024	Kria KR260	94%	60%	22	10.1	2.44	✓	✓

Table 6
Hardware cost of widely-used FPGAs (our architecture exploits the resources of each board by customizing the 4 Parameters).

	BRAM	DSPs	FF	LUTs	BUFF_K	BUFF_N	UNROLL_K	UNROLL_N
Artix 7 FPGA AC701	70.55% (515)	90.81% (672)	45.46% (122372)	74.28% (99979)	128	128	4	2
Kintex7 FPGA KC705	57.87% (515)	80.00% (672)	21.92% (89584)	44.99% (91693)	128	128	4	2
Kria KR260	62.15% (179)	53.85% (672)	37.91% (88810)	78.27% (91669)	64	32	4	2
Alveo U200 (each SLR)	98.33% (1416)	85.66% (1953)	35.04% (276134)	57.57% (226884)	256	256	16	3
Alveo U55C (each SLR)	73.81% (992)	86.17% (2592)	41.11% (357329)	64.55% (280522)	256	256	16	4

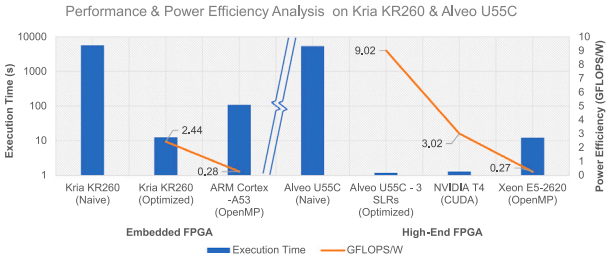


Fig. 10. Performance and power efficiency analysis (FP32).

(8 cores). In order to further compare the overall efficiency of the presented approach with that triggered by the software implementation, the GFLOPS/Watt for each implementation were also measured. Based on our measurements we achieve 34× and 9× higher energy efficiency than the best CPU parallel implementation, in an Alveo U55C and a Kria KR260 respectively. Moreover our design, when implemented on the Alveo, is 3× more power efficient than the CUDA implementation on an NVIDIA T4 GPU which is implemented on a better CMOS technology (12 nm for T4 vs 16 nm for U55c).

Table 5 presents the comparison of our approach with other FPGA-tailored similar systems. Since prior works report results on different FPGA devices, our comparison follows the standard practice of using normalized metrics (e.g., GFLOPS/W) to emphasize architectural efficiency rather than raw device capability. From those, the only open-source widely-used library for dense Matrix Multiplication is the one in [30] so we tried to implement it in our reference modern FPGAs. However, even though it is an open-source library, it has rather limited applicability because it requires AMD Vitis v2021.1 or older as also referred in Section 2. As a result, we implemented it in the largest FPGA supported by this version of the tool we had available (i.e. Alveo U200). In addition, [30] utilizes a specialized hardware library, which also makes it less designer-friendly, than our purely C-based approach. As shown in the table, our design achieves 9.02 GFLOPS/Watt, while gemm_hls' implementation triggers 8.49 GFLOPS/Watt for the same utilization percentage (229 GFLOPS compared to 211 GFLOPS from gemm_hls).[3]

More importantly, our methodology has been developed with a focus on simplicity and wide compatibility, eliminating the need for any specific external libraries and HW implementation tools. The designer-friendliness and the flexibility of our approach enables even non-advanced designers to seamlessly develop matrix-multiplication modules, probably within broader systems (e.g. Convolutional Neural Networks (CNNs) or Deep Neural Networks (DNNs)).

Finally, in order to assess the library adaptability and resource utilization on different FPGA boards, we performed experiments on five platforms (embedded & high-end): Artix 7, Kintex7, Kria KR260, Alveo U200 and Alveo U55c. Table 6 presents the hardware cost for the optimized reconfigurable implementation of our module. Additionally, as a practical guideline, Table 6 provides parameter configurations for a wide range of FPGA devices. Specifically, the unrolling parameters directly correspond to the available DSP resources, while the buffer sizes (BUFF_K and BUFF_N) correspond to the available BRAM resources. These two categories are interconnected, since the BRAM buffers depend on the unrolling factors through the array_partition directive. Users can consult Table 6, compare their device's resource budget to the configurations listed, and select the combination that maximizes utilization on their hardware. Our parametrizable approach is highly effective across all those boards as demonstrated by the high resource utilization and very high performance. By customizing the parameters to the individual architecture of each board, we fully capitalized on the distinctive attributes and capabilities of each FPGA. Due to the high levels of parallelism imposed by the optimizations, the critical factor in most of the cases is the number of DSPs, since more than 80% of them are utilized on most of the boards, except for the Kria KR260, where we achieve a 53% utilization of DSPs due to the limited on-chip resources (i.e. BRAMs) of the FPGA device.

[3] It is very difficult to compare the actual resources in each FPGA since AMD is listing differently the resources for the Alveo U200 and the U55c boards (i.e. CLBs and Registers vs System Logic Cells).

6. Conclusions & future work

The presented work addresses the critical need for efficient CNN implementations in power-constrained environments. The proposed design framework for efficient implementation of non-quantized CNNs in FPGAs, successfully combines high performance with energy efficiency, leveraging the inherent parallelism and reconfigurability of FPGAs. By maintaining full precision in network parameters, the framework achieves high accuracy without compromising significantly on power consumption. Moreover, the proposed design suite can be used by CNN designers with very limited (if at all) experience in HW design since it expands the very widely used DarkNet design flow and requires mainly the selection of 4 optimization parameters in order to create the most optimal implementation for a given CNN, FPGA device and performance and power requirements. The experimental results validate the framework's effectiveness, showing substantial speedups in inference processing and significant reductions in power usage. This work paves the way for more robust and efficient deployment of CNNs in embedded systems and edge computing, offering a viable solution for real-time artificial intelligence applications on the edge.

In future work, we intend to extend the proposed framework with optional quantization and mixed-precision inference support. This addition will enable users to dynamically balance accuracy, computational performance, and power/resource efficiency according to application requirements. Integration of these capabilities is planned for a subsequent publication as part of the ongoing evolution of the framework. We also plan to directly benchmark our framework's designer's productivity, accuracy, resource efficiency, and power consumption, compared to AccDNN, CINVESTAV's tool, and AutoVCoder to more quantitatively assess the trade-offs and benefits of each approach.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Hellenic Foundation for Research and Innovation, Greece (H.F.R.I.) under the "Framework of the National Recovery and Resilience Plan Greece 2.0, funded by the European Union – NextGenerationEU" under grant agreement No 74967 (REDESIGN project).

Data availability

I have shared the code in the manuscript.

References

[1] Zhibo Yang, Yanan Yang, Yunzhe Xue, Frank Y. Shih, Justin Ady, Usman Roshan, Accurate and adversarially robust classification of medical images and ECG time-series with gradient-free trained sign activation neural networks, in: 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM, 2020, pp. 2456–2460.

[2] Belal Jahannia, Jiachi Ye, Salem Altaleb, Nicola Peserico, Navid Asadizanjani, Elham Heidari, Volker Sorger, Hamed Dalir, Low-latency full precision optical convolutional neural network accelerator, 2024, p. 9.

[3] Yuhua Xu, Jie Luo, Wei Sun, Flare: An FPGA-based full precision low power CNN accelerator with reconfigurable structure, Sensors 24 (7) (2024).

[4] Yiwen Xu, Tariq M. Khan, Yang Song, Erik Meijering, Edge deep learning in computer vision and medical diagnostics: A comprehensive survey, Artif. Intell. Rev. 58 (3) (2025) 93.

[5] Chao Chen, Nor Ashidi Mat Isa, Xin Liu, A review of convolutional neural network based methods for medical image classification, Comput. Biol. Med. 185 (2025) 109507.

[6] Paweł Tumialis, Marcel Skierkowski, Jakub Przychodny, Paweł Obszarski, The impact of 8- and 4-bit quantization on the accuracy and silicon area footprint of tiny neural networks, Electronics 14 (1) (2025).

[7] Dorfell Parra, David Escobar Sanabria, Carlos Camargo, A methodology and open-source tools to implement convolutional neural networks quantized with TensorFlow lite on FPGAs, Electronics 12 (20) (2023).

[8] Chunlei Liu, Peng Chen, Bohan Zhuang, Chunhua Shen, Baochang Zhang, Wenrui Ding, SA-BNN: State-aware binary neural network, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 3, 2021, pp. 2091–2099.

[9] Tianqi Chen, Weixiang Xu, Weihan Chen, Peisong Wang, Jian Cheng, Towards efficient and accurate winograd convolution via full quantization, in: Advances in Neural Information Processing Systems, NeurIPS 36, 2023, 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

[10] Yury Mishchenko, Quentin Baptist, Qian Zhao, Tuomas Nurminen, Low-bit quantization and quantization-aware training for small-footprint keyword spotting, in: Proceedings of Interspeech, 2020, pp. 2542–2546.

[11] Shiji Xu, Jue Wang, Yuchen Liang, Yunhe Wang, Q-DETR: An efficient low-bit quantized detection transformer, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2023, pp. 16923–16932.

[12] Yiming Cai, Chuang Gan, Song Han, Zeroq: A novel zero-shot quantization framework, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 13169–13178.

[13] Zhi Qi, Wei Chen, R. Ali Naqvi, Kamran Siddique, Designing deep learning hardware accelerator and efficiency evaluation, Comput. Intell. Neurosci. 2022 (2022) 1291103, Published 13 July 2022.

[14] Cecilia Latotzke, Tim Ciesielski, Tobias Gemmeke, Design of high-throughput mixed-precision CNN accelerators on FPGA, in: 2022 32nd International Conference on Field-Programmable Logic and Applications, FPL, 2022, pp. 358–365.

[15] Zhiqiang Liu, Paul Chow, Jinwei Xu, Jingfei Jiang, Yong Dou, Jie Zhou, A uniform architecture design for accelerating 2D and 3D CNNs on FPGAs, Electronics 8 (1) (2019).

[16] Yingchao He, Di Liu, Chengying Gao, Xiang Cui, Qiang Xu, Efficient CNN computation on FPGAs with off-the-shelf quantization, J. Supercomput. 78 (10) (2022) 13271–13293.

[17] Nermine Ali, Jean-Marc Philippe, Benoit Tain, Philippe Coussy, Generating efficient FPGA-based CNN accelerators from high-level descriptions, J. Signal Process. Syst. 94 (10) (2022) 945–960.

[18] Mengfei Ji, Zaid Al-Ars, Peter Hofstee, Yuchun Chang, Baolin Zhang, FPQNet: Fully pipelined and quantized CNN for ultra-low latency image classification on FPGAs using OpenCAPI, Electronics 12 (19) (2023).

[19] Peng Peng, Mingyu You, Kai Jiang, Youzao Lian, Weisheng Xu, MBFQuant: A multiplier-bitwidth-fixed, mixed-precision quantization method for mobile CNN-based applications, IEEE Trans. Image Process. 32 (2023) 2438–2453.

[20] Jian Xu, Li Wang, Yu Chen, et al., Base-reconfigurable segmented logarithmic quantization and hardware design for deep neural networks, J. Signal Process. Syst. 92 (2020) 1263–1276.

[21] Panagiotis Mousouliotis, Nikolaos Tampouratzis, Ioannis Papaefstathiou, Squeezejet-3: An HLS-based accelerator for edge CNN applications on SoC FPGAs, in: 2023 XXIX International Conference on Information, Communication and Automation Technologies, ICAT, 2023, pp. 1–6.

[22] Nermine Ali, Jean-Marc Philippe, Benoit Tain, Philippe Coussy, Exploration and generation of efficient FPGA-based deep neural network accelerators, in: 2021 IEEE Workshop on Signal Processing Systems, SiPS, 2021, pp. 123–128.

[23] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wenmei Hwu, Deming Chen, AccDNN: An IP-based DNN generator for FPGAs, in: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM, 2018, 210–210.

[24] Miguel Rivera-Acosta, Susana Ortega-Cisneros, Jorge Rivera, Automatic tool for fast generation of custom convolutional neural networks accelerators for FPGA, Electronics 8 (6) (2019).

[25] Mingzhe Gao, Jieru Zhao, Zhe Lin, Wenchao Ding, Xiaofeng Hou, Yu Feng, Chao Li, Minyi Guo, AutoVCoder: A systematic framework for automated verilog code generation using LLMs, in: 2024 IEEE 42nd International Conference on Computer Design, ICCD, 2024, pp. 162–169.

[26] Afzal Ahmad, Muhammad Pasha, Optimizing hardware accelerated general matrix-matrix multiplication for CNNs on FPGAs, IEEE Trans. Circuits Syst. II: Express Briefs PP (2020) 1–1.

[27] Pouya Haghi, Anqi Guo, Tong Geng, Justin Broaddus, Derek Schafer, Anthony Skjellum, Martin Herbordt, A reconfigurable compute-in-the-network FPGA assistant for high-level collective support with distributed matrix multiply case study, in: IEEE Conference on Field Programmable Technology.

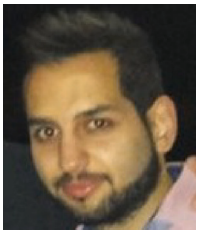
[28] Erik H. D'Hollander, High-level synthesis optimization for blocked floating-point matrix multiplication, SIGARCH Comput. Arch. News 44 (4) (2017) 74–79.

[29] Johannes de Fine Licht, Maciej Besta, Simon Meierhans, Torsten Hoefler, Transformations of high-level synthesis codes for high-performance computing, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1014–1029.

[30] Johannes de Fine Licht, Grzegorz Kwasniewski, Torsten Hoefler, Flexible communication avoiding matrix multiplication on FPGA with high-level synthesis, FPGA '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 244–254.

[31] Angelos Athanasiadis, Nikolaos Tampouratzis, Ioannis Papaefstathiou, Energy-efficient FPGA framework for non-quantized convolutional neural networks, 2025.
[32] Angelos Athanasiadis, Nikolaos Tampouratzis, Ioannis Papaefstathiou, An open-source HLS fully parameterizable matrix multiplication library for AMD FPGAs, WiPiEC J. - Work. Prog. Embed. Comput. J. 10 (2) (2024).
[33] Shixun Wu, Yujia Zhai, Jinyang Liu, Jiajun Huang, Zizhe Jian, Bryan Wong, Zizhong Chen, Anatomy of high-performance GEMM with online fault tolerance on GPUs, in: Proceedings of the 37th International Conference on Supercomputing, ICS '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 360–372.
[34] Jianan Sun, Mingxue Liao, Yongyue Chao, Pin Lv, Accelerate dense matrix multiplication on heterogeneous-gpus, in: 2023 IEEE 29th International Conference on Parallel and Distributed Systems, ICPADS, 2023.
[35] Guixia He, Jiaquan Gao, Jun Wang, Efficient dense matrix2010vector multiplication on GPU, Concurr. Comput.: Pr. Exp. 30 (19) (2018).
[36] Ruimin Wang, Zhiwei Yang, Hao Xu, Lu Lu, A high-performance batched matrix multiplication framework for gpus under unbalanced input distribution, J. Supercomput. 78 (2) (2021) 1741–1758.
[37] G. Schieffer, D. Medeiros, J. Faj, A. Marathe, I. Peng, Characterizing the performance, power efficiency, and programmability of AMD matrix cores, in: IEEE International Symposium on Performance Analysis of Systems and Software, 2024, pp. 1–12.
[38] Hadi Jahanirad, Dynamic power-gating for leakage power reduction in FPGAs, Front. Inf. Technol. Electron. Eng. 24 (4) (2023) 582–598.
[39] Safeen Huda, Muntasir Mallick, Jason H. Anderson, Clock gating architectures for FPGA power reduction, in: 2009 International Conference on Field Programmable Logic and Applications, 2009, pp. 112–118.
[40] Xilinx, Inc., Vivado Design Suite User Guide: Power Analysis and Optimization (UG907), 2024.2 ed., Xilinx, San José, CA, 2024, Version 2024.2.
[41] Shuze Zhao, Ibrahim Ahmed, Carl Lamoureux, Ashraf Lotfi, Vaughn Betz, Olivier Trescases, A universal self-calibrating dynamic voltage and frequency scaling (DVFS) scheme with thermal compensation for energy savings in FPGAs, in: 2016 IEEE Applied Power Electronics Conference and Exposition, APEC, 2016, pp. 1882–1887.
[42] Joseph Redmon, Darknet: Open source neural networks in C, 2013–2016, http://pjreddie.com/darknet/.
[43] Nikolaos Tampouratzis, Ioannis Papaefstathiou, A novel, simulator for heterogeneous cloud systems that incorporate custom hardware accelerators, IEEE Trans. Multi-Scale Comput. Syst. 4 (4) (2018) 565–576.
[44] Jixuan Li, Ka-Fai Un, Wei-Han Yu, Pui-In Mak, Rui P. Martins, An FPGA-based energy-efficient reconfigurable convolutional neural network accelerator for object recognition applications, IEEE Trans. Circuits Syst. II: Express Briefs 68 (9) (2021) 3143–3147.
[45] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, Yun Liang, An efficient hardware accelerator for sparse convolutional neural networks on FPGAs, in: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM, 2019, pp. 17–25.
[46] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, Haibin Shen, An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 28 (9) (2020) 1953–1965.
[47] Aikaterini Tsellou, George Livanos, Dimitris Ramnalis, Vassilis Polychronos, Georgios Plokamakis, Michalis Zervakis, Konstantia Moirogiorgou, A UAV intelligent system for greek power lines monitoring, Sensors 23 (20) (2023).
[48] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, Jason Cong, FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates, 2017, pp. 152–159.

Angelos Athanasiadis graduated from the ECE department of Aristotle University of Thessaloniki (AUTh) in 2019 and since then he has worked in projects ranging from digital logic design on FPGAs to C/C ++ application development for embedded systems. Currently, he is a Ph.D. student in the ECE AUTh department working on algorithm acceleration in FPGAs using HLS tools.



Nikolaos Tampouratzis is an Assistant Professor at Department of Industrial Engineering and Management, at International Hellenic University. He has been involved for more than 9 years as a researcher in numerous European and National competing research programs (H2020, FP7, ARISTEIA) towards the design, development and validation of state-of-the-art technologies in embedded/hardware design and FPGAs.

Ioannis Papaefstathiou is a Professor at the School of Electrical and Computer Engineering at Aristotle University of Thessaloniki and a co-Founder and CEO of EXASCALE PERFORMANCE SYSTEMS – EXAPSYS Plc. He is working in the design and implementation methodologies for systems with tightly coupled design parameters and highly constrained resources.