

FPGA-SoC Implementation of 3D-HEVC Encoder Based on MD-CNN

Nacir Omran¹, Amna Maraoui², Imen Werda³, Hamdi Belgacem⁴

Electronics and Microelectronics Laboratory(EMLab), University of Monastir, Monastir, Tunisia^{1,4}

Networked Objects, Control, and Communication Systems Laboratory(NOCCS), University of Sousse, Tunisia²

Electronic and Information Technology Laboratory(LETI), University of Sfax, Tunisia³

Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse, Tunisia²³

Abstract—Convolutional Neural Networks (CNNs) are the most common deep learning architecture used for video processing enhancement. Particularly, the Multi-Deep Convolutional Neural Network (MD-CNN) model, embedded into the 3D-HEVC encoder, was able to extract the optimal CTU partition structure efficiently in the depth map and successfully substitute the time-consuming rate-distortion optimisation (RDO) full traversal search, reducing the coding complexity heavily. Nevertheless, MD-CNN requires significant memory resources and is highly computationally intensive. Field-Programmable Gate Arrays (FPGAs), particularly the emerging FPGA-SoC technology, are recognised as the most promising platforms for accelerating CNNs due to their high performance, energy efficiency, and reconfigurable nature. In this study, complementary technologies have been combined and deployed to allow high-performance 3D-HEVC video processing. By exploiting the inherent parallelism of FPGA architectures, MD-CNN processing throughput was significantly improved, execution latency was reduced, and energy efficiency was enhanced, while maintaining the coding performance and compression efficiency of the 3D-HEVC video encoder. The experimental results demonstrate that the proposed co-design achieves an on-chip power consumption of 1.891 W.

Keywords—Three-dimensional high efficiency video coding; field-programmable gate array system-on-chip; multi-deep convolutional neural network; hardware-software co-design

I. INTRODUCTION

The Three-Dimensional High Efficiency Video Coding (3D-HEVC)[1] encoder comes with a significant increase in complexity caused by the integration of depth map coding technology. This complexity is primarily triggered by the quad-tree partition of the Intra Coding Units (CU) in the depth map[2]. 3D-HEVC underpins immersive services such as free-viewpoint television and autostereoscopic displays, where an inaccurate depth-map CU partition directly degrades the quality of synthesised virtual views, underscoring the practical importance of reducing this complexity.

Alongside the ordinary texture video, which conveys the scene's colour and luminance information, 3D-HEVC encodes a depth map for each captured view: a greyscale sequence in which each pixel encodes the camera-to-scene distance rather than colour, and which is used at the decoder to synthesise the additional viewpoints required by autostereoscopic and free-viewpoint displays through depth-image-based rendering. Fig. 1 summarises the main functional blocks of the 3D-HEVC encoder targeted in this work, which jointly processes both

signals; the corresponding decoder mirrors this pipeline in reverse, reconstructing the texture and depth pictures from the transmitted partitioning, prediction, and residual information rather than encoding them. Each CTU is first subjected to rate-distortion-optimised quad-tree partitioning, recursively splitting the depth-map block from 64×64 down to 8×8 . The resulting CUs are then predicted either through intra-frame prediction, combining the standard HEVC prediction modes with the depth-specific Wedgelet (DMM-1) and Contour (DMM-4) modelling modes, through inter-frame prediction via motion estimation and compensation from previously reconstructed pictures, or through inter-view prediction, which exploits disparity estimation and block merging across camera views to avoid re-encoding redundant depth and texture information. The resulting prediction residual is transformed, quantised, and entropy-coded into the output bitstream, with depth maps additionally supported by Segment-wise DC Coding (SDC) to preserve sharp depth discontinuities. Reconstructed pictures are subsequently smoothed by a deblocking filter and a Sample Adaptive Offset (SAO) stage before being stored as reference pictures for future prediction. It is the CU-partitioning stage, highlighted in red at the left of Fig. 1, that this study targets for hardware acceleration.

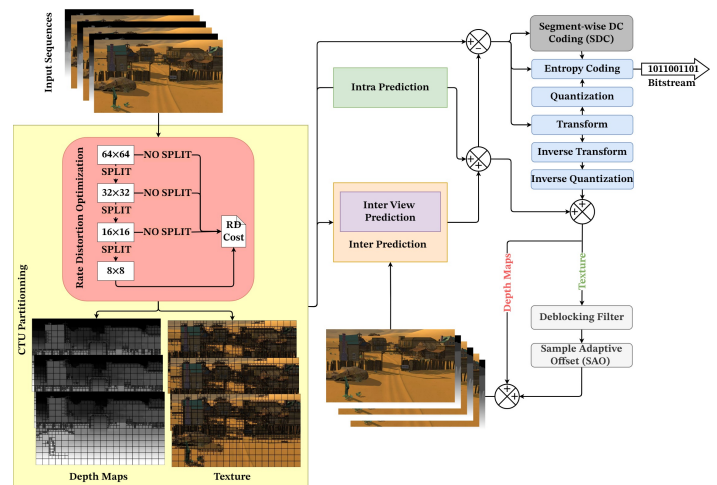


Fig. 1. Overview of the 3D-HEVC encoder architecture.

By means of an efficient CU partition structure prediction, the Multi-Deep Convolutional Neural Network (MD-CNN)

model [3] was able to tackle the issue of the 3D-HEVC encoder's excessive complexity without affecting its robustness, compression efficiency, and video quality. Nevertheless, even if the MD-CNN model exhibited significantly higher levels of accuracy compared to traditional methods[4], it required substantial computational complexity and considerable memory resource consumption. This constraint arises from the large number of parameters involved in the various layer operations (convolution, pooling, etc.). This limitation poses a computational challenge for central processing units (CPUs) and leads to significantly high energy consumption.

To overcome the accelerated evolution in multimedia technologies, various accelerator platforms have been developed in recent years, including application-specific integrated circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs). These platforms aim to optimise the throughput of Convolutional Neural Networks (CNNs) to achieve high performance while ensuring energy efficiency. Graphics processing units (GPUs) are high-performance platforms used during the training phase of CNNs. They can provide high throughput as well as a large memory bandwidth, but at the cost of considerable energy consumption. ASICs provide high throughput with reduced energy consumption but require considerable development time and costs compared to other alternatives. FPGA-based accelerators are effective solutions, particularly due to their rapid prototyping capability, energy efficiency, flexibility, and reconfigurability. It becomes mandatory to deploy MD-CNN learning techniques on FPGA targets to meet the requirements of real-time applications. In this context, this study proposes a hardware-software architecture based on an MD-CNN accelerator for 3D-HEVC CU partition. The main contributions of this study are summarised as follows:

- We propose to accelerate the MD-CNN corresponding to a 64×64 , 32×32 , and 16×16 CU partition for the 3D-HEVC encoder at the All-Intra configuration.
- We design Intellectual Property (IP) hardware and software cores for each MD-CNN convolutional and fully connected layer with Vivado High-Level Synthesis (HLS).
- We create the MD-CNN hardware-software co-design implementation in Xilinx Vivado by importing the convolutional and fully connected IP cores.

The remainder of this study is organised as follows: An overview of the 3D-HEVC standard is presented in Section III. Section IV provides the MD-CNN architecture and performance background. Section V exposes the MD-CNN deployment steps on the FPGA-based System-on-Chip (SoC) platform. The experimental results are discussed in Section VI. Finally, Section VII concludes the study.

II. RELATED WORK

Unlike standard High Efficiency Video Coding (HEVC) [5], 3D-HEVC adds a depth map encoding each object's distance from the camera, among the resulting depth-coding tools, CU partitioning is the most computationally intensive [2]. Most 3D-HEVC optimisation has targeted inter-coding [6], [7], while intra-coding proposals focus on intra-prediction mode selection [8], [9] or CU partitioning, the latter reviewed below by category.

A. Algorithmic CU-Partition Optimisation

Heuristic methods rely on fixed thresholds or rate-distortion (RD) cost/statistical rules: [4] combines tensor features with statistical analysis, [10] uses an empirically tuned coding tree unit (CTU) threshold, and [11] applies possibilistic clustering. Machine-learning methods replace hand-set rules with trained decision trees, using mined encoder context attributes [12] or Gini-value thresholding [13]. Deep-learning methods learn features directly from data: [14] classifies depth-map edges with a CNN but does not report a Bjøntegaard Delta Bitrate (BDBR) or time-reduction figure, while our own MD-CNN model [3] predicts the full 64/32/16 partition hierarchy end-to-end and is benchmarked quantitatively in Table VI. Unlike the present work, all of the methods above remain purely software-level.

B. Hardware/SoC Implementations

In contrast to the purely algorithmic works above, hardware realisations of CNN-based CU-partition prediction remain scarce. Dehnavi et al. [15] implement a lightweight CNN-based CU depth-decision unit on a Kintex-7 FPGA (XC7K32T) to replace the conventional DCT/RDO search for the standard HEVC intra-encoding chain. The closest available reference targeting an SoC platform is Bouaafia et al. [16], who deploy a multi-IP CNN accelerator on a PYNQ-Z1 FPGA-SoC to predict CU partitioning for standard HEVC inter-prediction (400 ms/CU at 142 MHz), targeting HEVC rather than 3D-HEVC, and inter- rather than intra-prediction. To the best of our knowledge, no prior work reports an SoC-level implementation of intra CU-partition prediction specifically for 3D-HEVC, which is the gap addressed by this study.

III. 3D-HEVC CU PARTITIONING OVERVIEW

3D-HEVC [1] is an extension of the H.265/HEVC [5] video coding standard that supports stereoscopic and multiview video coding. The coding structure of 3D-HEVC is based on HEVC but with some differences to support the additional view and depth information. The basic coding unit in 3D-HEVC is still the CTU, which is a square block of pixels. However, in 3D-HEVC, each CTU consists of multiple slices, each of which corresponds to a particular view. Each slice is encoded independently, allowing for efficient parallelisation of the encoding process. In addition to the traditional intra and inter-prediction modes, 3D-HEVC introduces a new inter-view prediction mode that allows for the prediction of one view from another view, as part of the complete encoder pipeline shown in Fig. 1. The intra-frame coding structure is similar to that of HEVC. The coding process starts with dividing the frame into CTUs of a size 64×64 . Each frame is divided into several non-overlapping CTUs, and each CTU can contain either a single CU or be iteratively divided into several smaller-sized CUs (64×64 , 32×32 , 16×16 and 8×8) based on the quadtree structure. Fig. 3 shows the CTUs in the depth map and their corresponding quadtree division structure. It is worth noting that the optimal partition structure of CTU is obtained by iterative computation. The quadtree partitioning of CTU includes not only the top-down RD-cost calculation but also the bottom-up RD-cost comparison process. For the top-down calculation process, the RD-cost of all coding units at the current depth is calculated in the order of "depth=0,

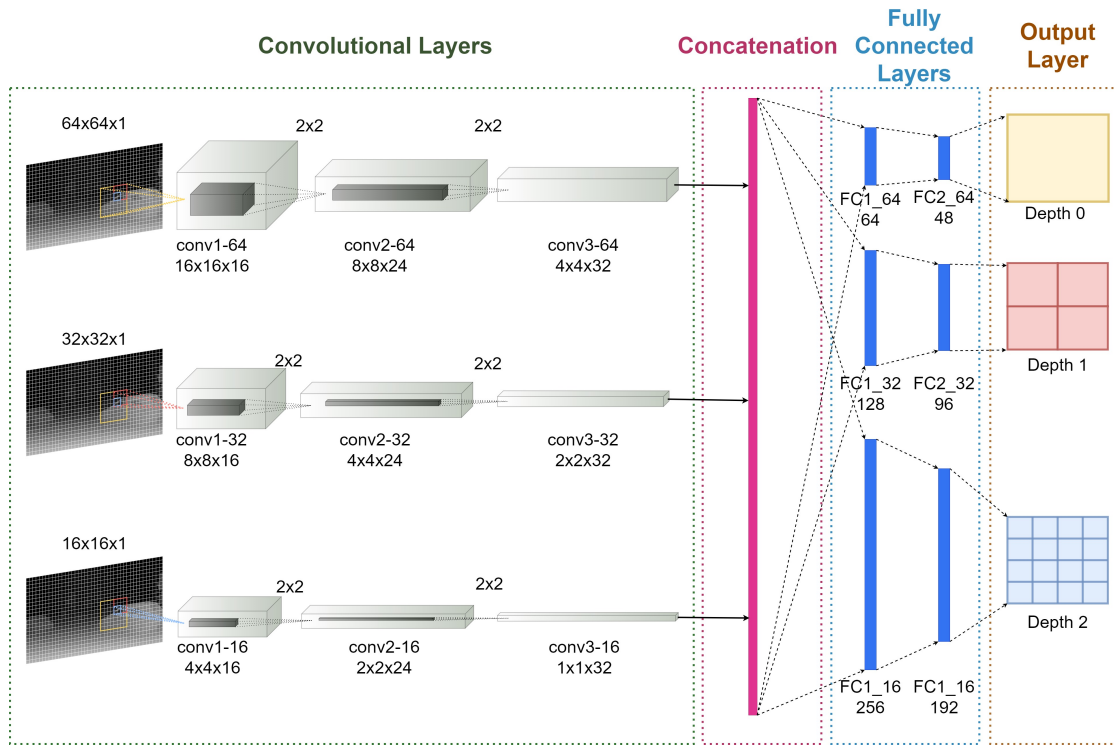


Fig. 2. MD-CNN architecture.

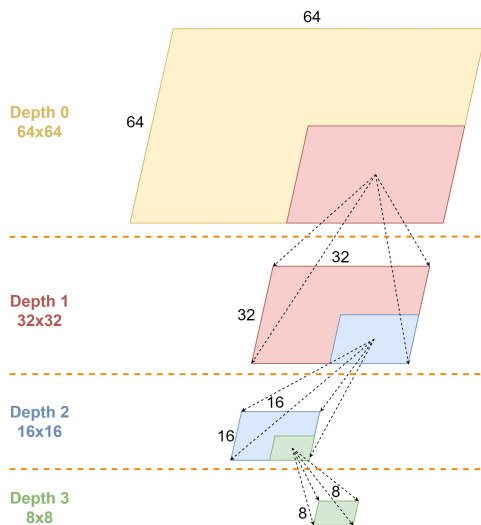


Fig. 3. 3D-HEVC CTU partitioning structure.

depth=1, depth=2, depth=3”, and based on this, the bottom-up comparison process is carried out. If “RD-cost(depth=n) > RD-cost(depth=n+1), n=0, 1, 2,” then the coding unit with “depth=n” needs to be divided, and vice versa, no division is needed. Statistically, the complexity of depth map coding unit division accounts for more than 90% of the complexity of depth map coding[3]. For a CTU of 64x64 size, a total of 85 CU operations, 1935 Sum of Absolute Transformed Difference (SATD) cost operations, and at least 2623 RD-cost operations are required to complete the encoding using the full traversal

mode. Therefore, it is necessary to reduce the complexity of depth map coding unit division in 3D-HEVC.

IV. 3D-HEVC DEPTH MAP CODING UNIT PARTITIONING BASED ON MD-CNN

This section introduces the depth map CU partition based on MD-CNN, and explains the integration of this partitioning optimisation into the 3D-HEVC HTM16.3 [17] reference encoder.

A. MD-CNN partitioning model

To determine CU partitioning decision for a 64×64 depth map CU without exhaustively evaluating every candidate sub-partition through the full rate-distortion optimisation (RDO) process, MD-CNN takes as input the raw 64×64 luma samples of a CU together with the frame-level quantisation parameter (QP) and produces three sets of sigmoid-valued split probabilities corresponding to the 64×64 , 32×32 , and 16×16 partitioning levels of the depth map coding tree, as shown in Fig. 2.

Rather than analysing the CU at a single spatial resolution, the MD-CNN constructs three parallel representations of the same block by average-pooling the original 64×64 patch down to 32×32 and 16×16 resolutions, in addition to using the original 64×64 patch unmodified. This produces three depth-specific branches, denoted 64, 32, and 16, which are processed independently but share an identical set of convolutional weights, allowing the network to learn scale-invariant structural features (edges, depth discontinuities, and texture gradients) that are relevant to partitioning decisions at

TABLE I. MULTI-DEPTH CONVOLUTIONAL BACKBONE (WEIGHTS SHARED ACROSS THE THREE CU DEPTH BRANCHES)

Layer	Input Size	Filters	Kernel Size	Stride	Output Size	Activation
Conv1_64	$64 \times 64 \times 1$	16	4×4	4	$16 \times 16 \times 16$	Leaky ReLU
Conv1_32	$32 \times 32 \times 1$	16	4×4	4	$8 \times 8 \times 16$	Leaky ReLU
Conv1_16	$16 \times 16 \times 1$	16	4×4	4	$4 \times 4 \times 16$	Leaky ReLU
Conv2_64	$16 \times 16 \times 16$	24	2×2	2	$8 \times 8 \times 24$	Leaky ReLU
Conv2_32	$8 \times 8 \times 16$	24	2×2	2	$4 \times 4 \times 24$	Leaky ReLU
Conv2_16	$4 \times 4 \times 16$	24	2×2	2	$2 \times 2 \times 24$	Leaky ReLU
Conv3_64	$8 \times 8 \times 24$	32	2×2	2	$4 \times 4 \times 32$	Leaky ReLU
Conv3_32	$4 \times 4 \times 24$	32	2×2	2	$2 \times 2 \times 32$	Leaky ReLU
Conv3_16	$2 \times 2 \times 24$	32	2×2	2	$1 \times 1 \times 32$	Leaky ReLU

every granularity, while keeping the parameter count of the convolutional stage small.

Each of the three branches is passed through an identical three-layer convolutional stack, summarised in Table I. The first convolutional layer (Conv1_64, Conv1_32, Conv1_16) applies 16 filters of size 4×4 with a stride of 4, aggressively reducing the spatial resolution while expanding the channel depth. The second convolutional layer (Conv2_64, Conv2_32, Conv2_16) applies 24 filters of size 2×2 with stride 2, and the third convolutional layer (Conv3_64, Conv3_32, Conv3_16) applies 32 filters of size 2×2 with stride 2. Because the three branches operate on inputs of different spatial dimensions (64×64 , 32×32 , and 16×16 , respectively), the output feature-map sizes differ proportionally across the pipeline, shrinking from $16 \times 16 \times 16$, $8 \times 8 \times 16$ and $4 \times 4 \times 16$ after Conv1, down to $4 \times 4 \times 32$, $2 \times 2 \times 32$, and $1 \times 1 \times 32$ after Conv3, for the 64, 32, and 16 branches respectively. A leaky ReLU activation is applied after every convolutional layer to introduce non-linearity while avoiding the “dying ReLU” problem that can arise from the small negative bias terms typical of depth map textures.

After the shared convolutional backbone has processed all three depth branches, the resulting feature maps from Conv2 and Conv3 across all three branches are flattened and concatenated into a single feature vector, as detailed in Table II. Concatenating both the Conv2 and Conv3 outputs, rather than only the deepest layer, allows the network to combine coarse, highly abstracted structural information from Conv3 with the finer-grained spatial detail retained by Conv2, which we found to be particularly beneficial for correctly identifying depth boundaries that influence the partitioning of visually flat or gently sloped regions of the depth map. The resulting concatenated feature vector has a fixed dimensionality of 2688 elements, independent of which decision branch subsequently consumes it, ensuring a consistent and reusable feature representation across all three partitioning levels.

TABLE II. FLATTENED MULTI-DEPTH FEATURE CONCATENATION

Source	Flattened Size
Conv3_64 ($4 \times 4 \times 32$)	512
Conv3_32 ($2 \times 2 \times 32$)	128
Conv3_16 ($1 \times 1 \times 32$)	32
Conv2_64 ($8 \times 8 \times 24$)	1536
Conv2_32 ($4 \times 4 \times 24$)	384
Conv2_16 ($2 \times 2 \times 24$)	96
Total feature vector	2688

The 2688-dimensional fused feature vector is fed into three fully-connected decision branches, corresponding to the

64×64 , 32×32 , and 16×16 partitioning levels, as summarised in Table III. Each branch consists of two hidden fully-connected layers followed by a sigmoid output layer, and the normalised QP value is injected as an additional scalar input at the second hidden layer and again at the output layer of every branch, rather than at the first layer. This design choice reflects the fact that QP primarily modulates the aggressiveness of the partitioning decision (i.e., the decision threshold) rather than the underlying visual structure of the block, and injecting it later in the network allows the convolutional and early fully-connected layers to focus purely on content-derived features.

Critically, the three decision branches are not evaluated unconditionally. The 64×64 branch (FC1_64, FC2_64, FC_out_64) is always executed and produces a single sigmoid probability, out_{64} , indicating the likelihood that the 64×64 CU should be split into four 32×32 sub-CUs. The 32×32 branch (FC1_32, FC2_32, FC_out_32) is only executed if out_{64} exceeds the decision threshold of 0.5, otherwise its four outputs remain at their initialised value of zero, and the corresponding sub-CUs are implicitly treated as not requiring further evaluation. Similarly, the 16×16 branch (FC1_16, FC2_16, FC_out_16), which produces sixteen sigmoid outputs corresponding to the sixteen possible 16×16 sub-CUs, is only executed if at least one of the four 32×32 outputs exceeds the same 0.5 threshold. This conditional, early-terminating structure is what allows the network to skip the majority of its computation for CUs that are unlikely to be split further, which is the central mechanism enabling the proposed method’s speed-up over exhaustive RDO-based partition search, while still allowing full-depth partitioning to be evaluated for CUs containing rich depth detail.

The fully assembled network contains approximately 1.28 million learnable parameters, the majority of which reside in the fully-connected layers of the 16×16 decision branch (FC1_16 alone accounts for $2688 \times 256 = 688,128$ weights), while the shared convolutional backbone contributes fewer than 5,000 parameters in total. This asymmetric parameter distribution reflects the fact that the convolutional layers are reused across all three input depths, whereas each fully-connected decision branch must independently learn the mapping from the shared feature representation to its respective partitioning granularity.

B. Integration of MD-CNN CU Partitioning in 3D-HEVC Encoder

Building on the CNN architecture, the MD-CNN partitioning predictor is integrated into the 3D-HEVC reference software HTM16.3 [17] as a fast CU-depth decision tool. For

TABLE III. HIERARCHICAL, CONDITIONALLY-EXECUTED FC DECISION BRANCHES

Branch	Layer	Input Size	Output Size	Activation	Execution Condition
64×64	FC1_64	2688	64	Leaky ReLU	Always
	FC2_64 (+QP)	64 (+1)	48	Leaky ReLU	Always
	FC_out_64 (+QP)	48 (+1)	1	Sigmoid	Always
32×32	FC1_32	2688	128	Leaky ReLU	$if\ out_{64} > 0.5$
	FC2_32 (+QP)	128 (+1)	96	Leaky ReLU	$if\ out_{64} > 0.5$
	FC_out_32 (+QP)	96 (+1)	4	Sigmoid	$if\ out_{64} > 0.5$
16×16	FC1_16	2688	256	Leaky ReLU	$if\ any\ out_{32,i} > 0.5$
	FC2_16 (+QP)	256 (+1)	192	Leaky ReLU	$if\ any\ out_{32,i} > 0.5$
	FC_out_16 (+QP)	192 (+1)	16	Sigmoid	$if\ any\ out_{32,i} > 0.5$

each 64×64 CTU, the MD-CNN produces a single flattened probability vector of 21 values, as seen in Fig. 4, corresponding to the hierarchical split decision across three depths: one value for the 64×64-level decision, four values for the 32×32 sub-CU decisions, and sixteen values for the 16×16 sub-CU decisions ($1 + 4 + 16 = 21$). In the training/inference pipeline, this vector is formed by concatenating the three network output tensors produced by the model's three fully-connected branches, each terminating in a sigmoid activation and each conditioned on the split decision of the coarser depth above it. This conditional structure, where finer-granularity outputs are only meaningful once a coarser-level split has been signalled, is the same hierarchical dependency later preserved in the hardware (HLS) implementation of the network, where the 32×32 and 16×16 branches are only evaluated once the corresponding higher-level decision exceeds threshold.

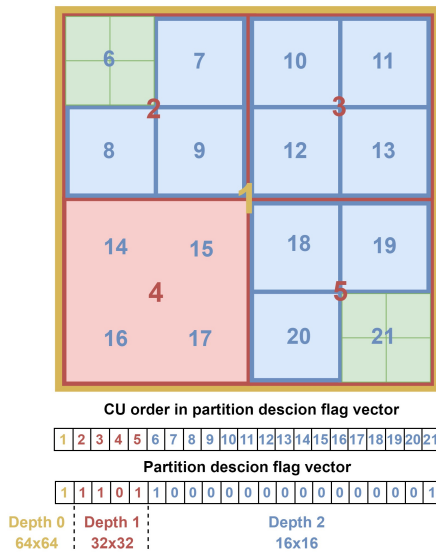


Fig. 4. CU partitioning flag vector.

The 21 values within a single CTU's vector are not stored in simple raster order but follow the 3D-HEVC Z-scan (quad-tree) convention: the single 64×64 flag appears first, followed by the four 32×32 quadrant flags in top-left, top-right, bottom-left, bottom-right order, followed by the sixteen 16×16 flags similarly nested in Z-order within each 32×32 quadrant. This ordering arises naturally from how the ground-truth label map is reshaped and progressively pooled at 2× and 4× factors to derive the 32×32 and 64×64 level labels from the finest 16×16 grid, which preserves the spatial quad-tree correspondence

between each 16×16 flag and its parent 32×32 and 64×64 flags. The referenced Fig. 4 illustrates this 21-entry CTU vector layout and the correspondence between depths. While each CTU's internal 21 flags follow Z-scan order, the sequence of CTU vectors written to the output file follows a 3D-HEVC-style raster scan across the frame, proceeding left to right and top to bottom. Each CTU is processed in this order, and its resulting 21-value vector is appended to a per-frame buffer, which is then flattened and written out as a single contiguous binary stream.

C. MD-CNN Model Training Configuration

This subsection details the dataset used, the MD-CNN training procedure, and the training and validation results.

1) *Dataset*: In the absence of a public depth-map database, MD-CNN was trained on high-resolution texture images [18] rather than benchmark sequence excerpts. The HEVC reference encoder HM16.18 [19], invoked internally by the 3D-HEVC encoder HTM16.3 [17] for CU partitioning, was used to encode 2,500 images at multiple resolutions and QPs and derive the corresponding 64×64, 32×32 split-decision labels. The resulting 6,758,040 labelled samples were split, without overlap, into training (2,125 images), validation (125 images), and test (250 images) sets, with split and non-split CUs kept close to balanced (Fig. 8).

2) *Training procedure*: Training was carried out on a Windows 10 Professional (64-bit) workstation equipped with an Intel Core i7-8700 CPU (3.20 GHz), an Nvidia GeForce GTX 1050 Ti GPU, and 16 GB of RAM, using the TensorFlow [20] framework. The network was trained from scratch for 500,000 iterations with mini-batches of 64 samples, using a momentum-based optimiser (momentum = 0.9) with an initial learning rate of 0.01, exponentially decayed to 31.63% of its previous value every 125,000 iterations.

3) *Training and validation results*: Three complementary metrics are used to monitor and assess the network at each of the three decision depths (64×64, 32×32, and 16×16): the classification loss, the classification accuracy, and a tendency indicator quantifying the direction of any misclassification bias.

The loss is the binary cross-entropy between the predicted split probability $\hat{y}_i$ and the ground-truth split label $y_i \in \{0, 1\}$, averaged over the N samples of the branch:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (1)$$

Accuracy is computed from the 2×2 confusion matrix obtained by thresholding $\hat{y}_i$ at 0.5, where n_{gp} is the number of samples with ground-truth class g predicted as class p ($g, p \in \{0, 1\}$; 0 = non-split, 1 = split):

$$\text{Accuracy} = \frac{n_{00} + n_{11}}{n_{00} + n_{01} + n_{10} + n_{11}} \quad (2)$$

To complement accuracy with a measure of directional bias, a tendency indicator based on the log-odds ratio of the confusion matrix is used:

$$T = -\log_{10} \left(\frac{n_{00} n_{10}}{n_{01} n_{11}} \right) \quad (3)$$

with $T = 0$ when the model makes no errors in either direction, and T saturating to ± 100 in the degenerate cases where errors occur exclusively in one direction (avoiding an undefined ratio). A positive T indicates a bias toward over-predicting splits, a negative T indicates a bias toward under-predicting splits, and $T \approx 0$ indicates a symmetric error distribution between the two.

Fig. 5, 6, and 7 report the evolution of the three metrics over training. The loss [Eq. (1)] and accuracy [Eq. (2)] converge within the first 150,000 to 200,000 iterations and remain stable thereafter, with training and validation curves closely overlapping throughout, indicating no overfitting. The tendency metric [Eq. (3)] stabilises more gradually, settling into its final band by approximately 400,000–500,000 iterations. Table IV reports the corresponding average values at 500,000 iterations: the model reaches consistently high accuracy at all three decision depths (78 to 82%), with training and validation values closely matching at every branch, confirming good generalisation. The tendency values remain modest across all branches, well within the metric's ± 100 saturation bounds, indicating no severe split/non-split bias at any depth.

TABLE IV. AVERAGE TRAINING AND VALIDATION METRICS AT 500,000 ITERATIONS

Average value	Branch	Metrics		
		Loss	Accuracy	Tendency
Train	64×64	0,353	0,802	-0,266
	32×32	0,41	0,82	-0,423
	16×16	0,444	0,781	0,124
Validation	64×64	0,365	0,798	-0,091
	32×32	0,412	0,817	-0,423
	16×16	0,443	0,78	0,211

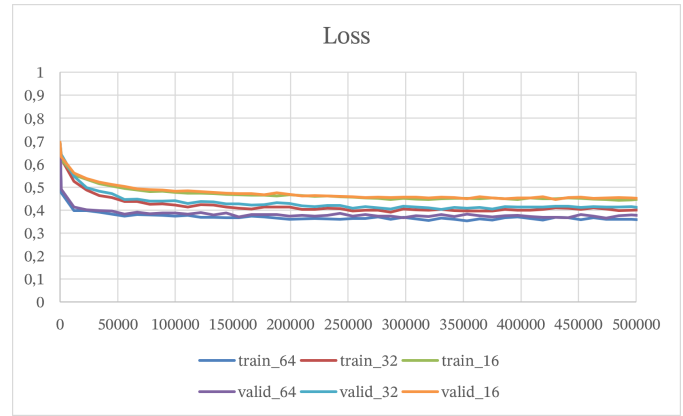


Fig. 5. Training and validation loss.

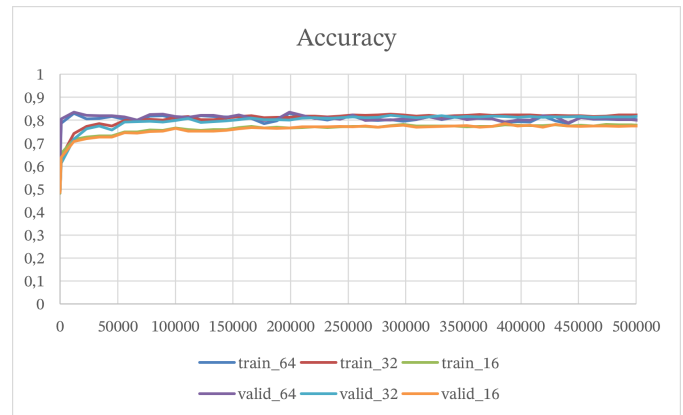


Fig. 6. Training and validation accuracy.

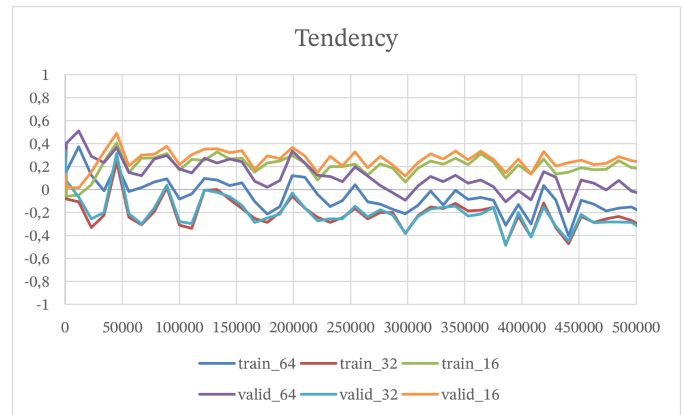


Fig. 7. Training and validation tendency.

D. MD-CNN Performance

This subsection evaluates MD-CNN along three complementary axes: its own rate-distortion and complexity trade-off, the primary measure of whether the model meets its complexity-reduction goal, its standing against representative heuristic and machine-learning baselines, which situates it

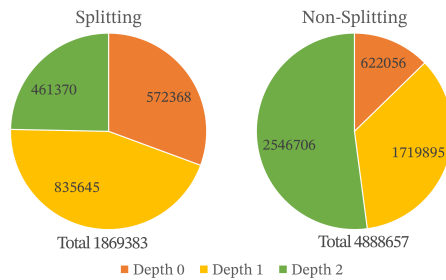


Fig. 8. Distribution of dataset samples.

within the broader literature, and its downstream impact on synthesised view quality, since bitrate and time gains are only meaningful if visual fidelity is preserved.

1) *Rate-distortion and complexity*: Table V reports the proposed method's rate-distortion, noise and time reduction across four QP pairs (25-34, 30-39, 35-42, 40-45), broken down by resolution class.

TABLE V. MD-CNN PERFORMANCE RESULTS

Sequences	QP pairs	MD-CNN		
		BDBR%	Δ Y-PSNR%	Δ time%
Average 1024×768	QP 25 -34	-0.02	-0.03	-46.18
	QP 30-39	0.62	-0.02	-48.69
	QP 35-42	-0.50	-0.02	-50.03
	QP 40-45	0.42	-0.02	-51.12
	Average	0.13	-0.02	-49.00
Average 1920×1088	QP 25 -34	-0.77	-0.02	-45.95
	QP 30-39	-1.13	-0.03	-45.83
	QP 35-42	-0.73	-0.04	-46.11
	QP 40-45	-0.43	-0.04	-47.12
	Average	-0.77	-0.04	-46.25
Average	QP 25 -34	-0.40	-0.03	-46.07
	QP 30-39	-0.26	-0.03	-47.26
	QP 35-42	-0.62	-0.03	-48.07
	QP 40-45	-0.01	-0.03	-49.12
	Average	-0.32	-0.03	-47.63

At low resolution (1024×768), BDBR fluctuates between small positive and negative values across QP pairs, ranging from an improvement of -0.50% at QP(35-42) to a penalty of 0.62% at QP(30-39), with the class average landing at a modest 0.13% increase. Time reduction, by contrast, is both larger and steadily improving with QP: -46.18% at QP(25-34) rising to -51.12% at QP(40-45), suggesting the model's complexity savings grow slightly more effective as quantisation coarsens.

At high resolution (1920×1088), the trend is more consistent: BDBR is negative at every QP pair, from -0.43% at QP(40-45) to -1.13% at QP(30-39), giving a class average of -0.77%, a net coding-efficiency gain rather than a trade-off. Time reduction here is comparatively flat across QP pairs (-45.83% to -47.12%), a narrower spread than at low resolution.

Quality loss, measured via the luma Peak Signal-to-Noise Ratio (Y-PSNR), stays small and stable across both resolution classes and all QP pairs, ranging from -0.017% to -0.044%, with no clear trend tied to QP or resolution, indicating that the bitrate and time gains are not being bought at a meaningful cost to reconstructed video quality.

Averaged across both resolutions, BDBR is negative at every QP pair (-0.01% to -0.62%), giving an overall aver-

age of -0.32%, while time reduction increases monotonically from -46.07% at QP(25-34) to -49.12% at QP(40-45), for an overall average of -47.63%. This QP-wise monotonicity in time savings, combined with consistently negligible PSNR loss, indicates that the proposed method's complexity reduction scales predictably with quantisation level without degrading its already-favourable bitrate behaviour.

2) *Comparison with prior methods*: To situate these results against other CU-partition prediction methods, Table VI compares BDBR and time reduction (TR) by resolution class against one representative heuristic method [11] and one representative machine-learning method [13]. At standard definition, the proposed method's BDBR (0.57%) is higher than both baselines, trading a larger rate penalty for the highest time reduction (-50.42%) of the three. At high definition, the trend reverses: the proposed method is the only one of the three to reach a negative BDBR (-0.79%), an actual coding-efficiency gain that neither [11] nor [13] achieve, though its time reduction (-47.23%) is exceeded by [13]'s -68.17%. Averaged across both classes, the proposed method trails [13] in time reduction (-48.29% versus -59.33%) but is the only one with a net bitrate improvement (-0.34%) rather than a penalty.

3) *Synthesised view quality*: To further assess the impact of the proposed MD-CNN-based fast CU partition algorithm on video quality beyond the BDBR and Y-PSNR analysis above, we evaluated the quality of the virtual views synthesised from the reconstructed depth maps. This evaluation focuses on the two sequences that exhibited the least favourable performance among the tested benchmarking set, namely Poznan Hall2 (1920 × 1088) and Newspaper (1024 × 768), both encoded at the QP couple (25-34). For each sequence, a virtual viewpoint is synthesised through Depth-Image-Based Rendering (DIBR) using the reconstructed texture and depth information produced by our method, and the resulting frames are compared against the corresponding reference views using four complementary metrics: the Structural Similarity Index (SSIM), the Edge Mean Absolute Error (Edge MAE), the Error Rate, and the Peak Signal-to-Noise Ratio (PSNR). SSIM and PSNR capture the overall structural and pixel-level fidelity of the synthesised view, while Edge MAE and the Error Rate are more sensitive to the geometric distortions and rendering artefacts that arise around depth discontinuities and object boundaries.

As reported in Table VII, and despite being identified as the two worst-performing sequences of the benchmarking set, both Poznan Hall2 and Newspaper achieve high synthesised view quality across the QP couple (25-34), with average SSIM values of 0.99 for both sequences, average PSNR values above 42 dB, and low average Error Rates of 0.01. To visually confirm this, Fig. 9 shows, for each sequence, the original and synthesised views at their respective lowest-SSIM frame, together with a per-pixel difference map. A high-contrast colour scale is used for the difference map so that even minor deviations are made visible, despite this, the discrepancies remain confined to a few sparse, low-intensity regions along moving edges (e.g. the reader's arm and the newspaper in Newspaper, and the walking figure on the staircase in Poznan Hall2). In both cases the synthesised view is perceptually indistinguishable from the original, with no visible artefacts, confirming that even the worst-case frames retain high visual fidelity.

TABLE VI. BDBR AND TIME REDUCTION COMPARISON GROUPED BY RESOLUTION CLASS

Resolution	[11] (Heuristic)		[13] (Machine Learning)		Proposed MD-CNN [3] (Deep Learning)	
	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)
1024×768	0.10	-32.60	0.06	-41.67	0.57	-50.42
1920×1088	0.26	-44.97	0.16	-68.17	-0.79	-47.23
Average	0.20	-40.85	0.13	-59.33	-0.34	-48.29



(a) Newspaper—Frame 60, worst-case SSIM = 0.98377



(b) Poznan Hall 2—Frame 69, worst-case SSIM = 0.98927

Fig. 9. Original view, synthesised view, and per-pixel difference map (high-contrast scale) at the worst-case SSIM frame for each sequence. The synthesised views are perceptually indistinguishable from the originals, with no visible artefacts.

TABLE VII. AVERAGE SYNTHESISED VIEW QUALITY RESULTS, QP (25-34)

Metric	Poznan Hall2	Newspaper
SSIM	0.99	0.99
Edge MAE	2.10	2.02
Error Rate	0.01	0.01
PSNR (dB)	46.46	42.41

These results confirm that the proposed MD-CNN-based fast CU partition algorithm reduces encoding complexity without introducing perceptually significant degradation in the rendered virtual views.

V. FPGA ARCHITECTURE AND IMPLEMENTATION

As discussed in Section II, Bouaafia et al. [16] implement a multi-IP hardware-software co-design for CNN-based CU-partition prediction in HEVC inter-prediction. This layer-by-layer decomposition comes at the cost of increased latency, as reflected in their reported 400 ms processing time per image. Motivated by this trade-off, we aim to design a more integrated hardware architecture for our MD-CNN model by consolidating the entire network into a single IP core rather than instantiating and interconnecting multiple independent accelerators. Our goal is to reduce the architectural and communication overhead inherent to multi-IP designs, simplify system integration on the FPGA, and ultimately achieve lower end-to-end latency while still preserving competitive power and resource efficiency.

The proposed MD-CNN was translated into a hardware accelerator and deployed on a Xilinx Zynq-7000 SoC, specifically the xc7z020 device, using Vivado HLS to generate the compute core and Vivado IP Integrator to compose it into a complete embedded system. The resulting system, shown in Fig. 10, is built from five principal hardware blocks: the Zynq Processing System, the custom IP convolutional neural network accelerator net_CNN_Hw_0, two Advanced eXtensible Interface (AXI) interconnect blocks that route control and data traffic between them, and a processor system reset block that generates the synchronised reset signals used throughout the design. This section describes how these blocks are organised and communicate with one another, the internal design of the accelerator core itself, and the resulting resource utilisation and timing performance obtained after synthesis and implementation.

A. Processing System-Programmable Logic Co-Design

The Zynq-7000 family combines, on a single die, a hardened Processing System containing a dual-core ARM Cortex-A9 processor, a Double Data Rate (DDR) memory controller, and system clocking and reset infrastructure, with reconfigurable Programmable Logic in which custom digital hardware can be instantiated. This Processing System-Programmable Logic (PS-PL) architecture allows a conventional embedded software application, running on the ARM cores, to offload computationally intensive tasks to dedicated hardware implemented in the programmable logic fabric, while retaining the

flexibility of general-purpose software for control, orchestration, and higher-level video coding logic. In the proposed system, this role is filled by the Zynq Processing System block, which supplies the platform's clocking, reset, and memory access infrastructure and issues the commands that drive the accelerator.

VI. EXPERIMENTAL RESULTS

A. Processing System-Programmable Logic Interconnection

As seen in Fig. 10, the trained MD-CNN itself is implemented entirely within the programmable logic as the `net_CNN_Hw_0` accelerator core, while the Zynq Processing System is responsible for supplying input CU data and QP values, triggering the accelerator, and consuming its partitioning decisions once complete. Communication between the two domains is carried out over two logically separate interfaces, each routed through its own AXI interconnect block. The first interconnect, `ps7_0_axi_periph`, connects the Zynq Processing System's general-purpose master interface to a lightweight control interface exposed by `net_CNN_Hw_0`, allowing the ARM processor to write to and read from a small set of control registers on the accelerator, enabling software to start an inference operation and subsequently poll for or be interrupted upon its completion. The second interconnect, `axi_interconnect_0`, connects a separate, wider data interface on `net_CNN_Hw_0` to one of the Zynq Processing System's dedicated high-performance ports, providing a direct path into the shared DDR memory that the accelerator can use to read and write data on its own, without requiring the processor to relay each transfer. This separation into two independently interconnected paths is important given the scale of the network, since well over a million trained weight values, together with the input image data and QP scalar, must be streamed into `net_CNN_Hw_0` for every single inference, and the output partitioning probabilities must subsequently be written back for the software to consume, keeping this high-volume traffic on its own interconnect and port prevents it from contending with the much lighter control traffic carried by `ps7_0_axi_periph`.

All four of these blocks, together with `net_CNN_Hw_0` itself, operate from a single clock generated by the Zynq Processing System and distributed throughout the programmable logic. Synchronisation of these blocks on startup or software-triggered reset is handled by the processor system reset block, which takes the Zynq Processing System's fabric reset output and produces the individual, synchronised reset signals consumed by both AXI interconnects and by `net_CNN_Hw_0`, ensuring that every block in the design begins operating from a consistent, known state.

B. Accelerator Core Design

Internally, the `net_CNN_Hw_0` accelerator follows a straightforward load-compute-store organisation. At the beginning of each invocation, the entire 64×64 input patch and its associated QP value are transferred from external memory into local on-chip memory, and all of the convolutional and fully-connected weights required for the current computation are likewise brought on-chip as needed. Every intermediate result produced during inference, including the pooled multi-scale representations, the outputs of each convolutional stage,

the fused 2688-element feature vector, and the activations of every fully-connected layer, is kept entirely within on-chip memory for the remainder of the computation, only the final split-decision probabilities are written back out to external memory once inference completes. This buffering strategy avoids repeated, high-latency round trips to external DDR memory for data that is reused multiple times within a single inference pass, at the cost of dedicating a meaningful fraction of the device's on-chip memory resources to hold these buffers, as reflected in the utilisation results reported below.

All arithmetic throughout the accelerator, including the convolutional filtering, the fully-connected matrix vector operations, and the activation functions, is carried out in single-precision (32-bit) floating-point representation, matching the numerical format used during the original software-based training of the network and avoiding the additional design effort and potential accuracy loss associated with a fixed-point quantisation stage. To make this floating-point computation efficient in hardware, the convolutional and fully-connected layers are structured so that their inner arithmetic loops can be pipelined, allowing a new set of multiply-accumulate operations to be issued on nearly every clock cycle rather than waiting for previous operations to fully complete, and the intermediate feature buffers used within the convolutional stages are organised so that several filter channels can be accessed in parallel rather than one at a time. Together, these choices allow the design to sustain a high computational throughput despite the relatively modest amount of dedicated arithmetic hardware available on the target device.

A further architectural characteristic of `net_CNN_Hw_0`, inherited directly from the hierarchical structure of the network itself, is that the early-termination behaviour of the algorithm is realised as genuine conditional hardware execution rather than as a purely software-level shortcut applied afterwards. The stage responsible for evaluating 32×32 partitioning decisions is only activated in hardware when the 64×64 stage's output indicates that further splitting is likely, and the stage responsible for the sixteen 16×16 decisions is, in turn, only activated when at least one of the four 32×32 outputs crosses its own decision threshold. When a given stage is not activated, the corresponding portion of the pipeline performs no arithmetic and initiates no memory transfers for the associated weights, rather than simply masking or discarding its result afterwards. Because the 32×32 and 16×16 decision stages account for the overwhelming majority of the network's fully-connected weights, this conditional activation substantially reduces both the number of clock cycles consumed and the volume of data fetched from external memory for the large fraction of coding units that do not require full-depth evaluation, directly mirroring, in hardware, the same early-termination principle that motivates the network's hierarchical software design.

Finally, because `net_CNN_Hw_0` performs all computation in pipelined hardware rather than in a general-purpose floating-point software environment, care was taken in the design of the activation functions to avoid numerical instabilities that could otherwise arise, such as overflow in the exponential term of the sigmoid function for large-magnitude inputs. The sigmoid activation used throughout the fully-connected decision layers is implemented in a numerically stable form that avoids such overflow across the full expected input range, and additional

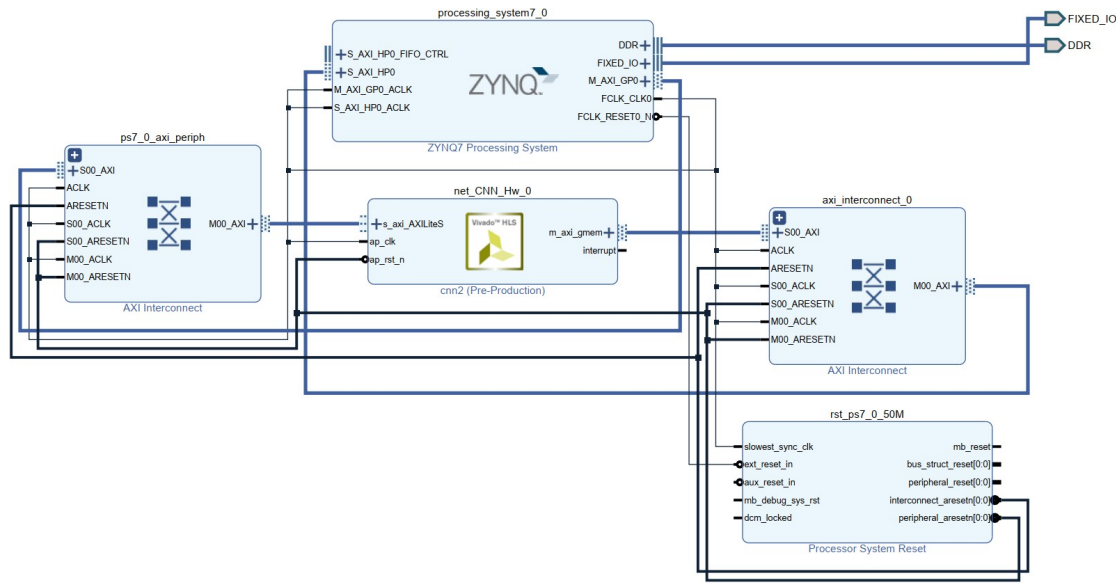


Fig. 10. Hardware–software co-design.

safeguards are included after each fully-connected layer to prevent invalid or out-of-range intermediate values from propagating through the remainder of the pipeline.

C. Hardware Implementation Performance Results

This subsection reports the post-implementation results of the accelerator across six complementary aspects: the achieved timing and latency, which determine whether the design meets its throughput target, the on-chip resource utilisation, which reveals how HLS allocated logic and arithmetic hardware, the power consumption, the primary cost of any embedded deployment, the resulting energy efficiency relative to a general-purpose PC, the associated carbon footprint, and a direct comparison with the closest reference FPGA-SoC design.

1) *Timing and latency*: Table VIII reports the timing and latency figures obtained from HLS synthesis and from the final placed-and-routed design. The design was synthesised against a 19.000 ns target clock period and closed timing at an achieved post-implementation clock period of 18.477 ns, corresponding to a 54.1 MHz operating frequency and a slack of roughly 2.7% over the constraint. The estimated clock period reported by HLS synthesis (18.032 ns) differs from the post-implementation figure by approximately 2.5%, a modest and expected degradation attributable to routing and placement effects introduced after logic synthesis, the fact that this gap remains small indicates that the HLS-level timing estimate was a reliable predictor of the final hardware behaviour for this design.

TABLE VIII. TIMING AND LATENCY OF THE ACCELERATOR CORE

Metric	Value
Target clock period	19.000 ns
Estimated clock period (HLS)	18.032 ns
Achieved clock period (post-implementation)	18.477 ns
Timing closure	Met
Minimum latency (early termination at 64×64 level)	1,694,545 cycles
Maximum latency (full hierarchical evaluation)	2,804,793 cycles

The substantial gap between the minimum and maximum latency figures, amounting to roughly 1.11 million cycles, is not incidental but follows directly from the early-termination mechanism embedded in the kernel description, whereby two boolean predicates evaluated on the intermediate 64×64-level output determine whether the 32×32 and 16×16 partition branches are executed at all. When both predicates evaluate false, the design short-circuits the corresponding convolutional and fully connected stages, so the reported minimum latency reflects a CU that never leaves the coarsest evaluation level, while the maximum latency reflects a CU that is evaluated hierarchically through every partition depth. A cycle-accounting exercise over the individual loop bounds in the report confirms this reading: the convolutional stages together account for roughly 53% of the worst-case latency, while the three instances of the first fully connected layer, corresponding respectively to the 64-, 32-, and 16-level partition widths, account for the remainder, and the incremental cost of the conditionally skipped 32- and 16-level fully connected layers alone is close in magnitude to the observed minimum-maximum gap. This confirms that the reported spread is a direct hardware consequence of the algorithmic early-termination strategy rather than an artefact of pipeline stalls or memory contention. It is also worth noting architecturally that within each fully connected stage only the innermost neuron-accumulation loop is scheduled with an initiation interval of one, the outer loop over input features is not unrolled or pipelined, so each dense layer executes close to one multiply-accumulate operation per clock cycle regardless of its width, and total latency scales essentially linearly with the product of feature count and neuron count. This sequential accumulation pattern, rather than clock frequency, is the principal factor limiting throughput, and represents the most direct target for future optimisation through loop unrolling or the introduction of dataflow pipelining between the convolutional and fully connected stages, neither of which is currently exploited in the design.

Translating these cycle counts into wall-clock time at the

achieved operating frequency of 54.1 MHz yields a best-case latency of approximately 31.3 ms and a worst-case latency of approximately 51.8 ms per 64×64 coding unit, corresponding to a throughput of roughly 19 to 32 CU decisions per second. This figure is useful in situating the accelerator relative to the requirements of real-time video encoding, where the number of coding-unit partition decisions to be made per second for even moderate resolutions is several orders of magnitude higher, the present design should therefore be understood as a proof-of-concept core suitable for offline or non-real-time evaluation rather than as a drop-in real-time encoding accelerator, a limitation that follows directly from the sequential dense-layer execution pattern described above rather than from any deficiency in the achieved clock frequency.

2) *Resource utilisation*: This section presents the utilisation of the hardware resources mobilised by the co-design method proposed on the PYNQ-Z2 platform. Following the implementation phase, the use of the hardware resources of the proposed design is presented in Table IX.

TABLE IX. POST-SYNTHESIS RESOURCE UTILISATION OF THE ACCELERATOR CORE

Resource	Used	Available	Utilisation
BRAM_18K	93	280	33%
DSP48E	22	220	10%
FF	22,692	106,400	21%
LUT	34,661	53,200	65%

The resources utilised include a global buffer (BUFG), input-output interfaces, digital signal processors (DSP), block RAM (BRAM), flip-flops (FF), and lookup tables (LUT). According to Table IX, the hardware design uses 65% of the LUTs, 21% of the FFs, 33% of the BRAM, and 10% of the DSPs, and it is this pronounced asymmetry between the low DSP occupancy and the high LUT occupancy that most usefully characterises the implementation strategy chosen by HLS. Inspecting the synthesis report at the instance level shows that only a small number of floating-point multiplier and adder cores are instantiated, each requiring two to three DSP48E slices, and that these cores are shared through time-multiplexing across the thousands of multiply-accumulate operations required by the convolutional and fully connected layers rather than being replicated per operation. This resource-binding decision is precisely why DSP occupancy remains as low as 10% despite the arithmetic intensity of the network, the cost of sharing a handful of arithmetic units across so many operations is paid instead in combinational logic, since a large multiplexer network is required at every cycle to steer the correct pair of operands into the shared multiplier and adder and to route the shared result back to the appropriate destination register. This is corroborated by the operation-level breakdown of the synthesis report, in which multiplexer and select operations dominate the instantiated logic alongside the exit-condition comparators generated by the numerous loop bounds of the network's layers, together explaining why LUT utilisation reaches 65% while DSP utilisation remains an order of magnitude lower in relative terms. The low DSP and high LUT profile should therefore be read not as an efficient use of heterogeneous FPGA resources but as the expected signature of aggressive operator sharing, which trades DSP silicon for LUT-based control and steering logic and, combined with

the unpipelined outer accumulation loops discussed above, is consistent with the observed latency profile.

Fig. 11 illustrates the energy consumption of the hardware architecture implemented on the FPGA board. According to the power report, the static power of the proposed system is approximately 0.147 W, while the dynamic power reaches 1.744 W, giving a total on-chip power of approximately 1.891 W and confirming that the design's power budget is dominated by dynamic switching activity rather than by static leakage, consistent with the resource profile of a logic-heavy, moderately clocked accelerator.

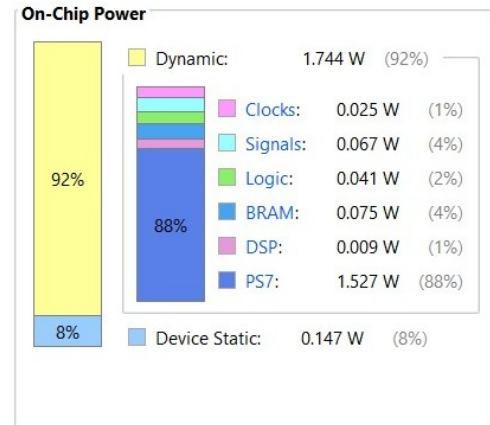


Fig. 11. Power consumption.

3) *Power consumption*: Fig. 11 illustrates the energy consumption of the hardware architecture implemented on the FPGA board. According to the power report, the static power of the proposed system is approximately 0.147 W, while the dynamic power reaches 1.744 W, giving a total on-chip power of approximately 1.891 W and confirming that the design's power budget is dominated by dynamic switching activity rather than by static leakage, consistent with the resource profile of a logic-heavy, moderately clocked accelerator.

4) *Energy efficiency*: Table X compares the average energetic efficiency of the FPGA-based accelerator board against the host PC (GPU, CPU, and peripherals) in terms of frames processed per watt consumed, using the average per-frame latency reported for the accelerator and the measured average per-frame processing time for the PC at each resolution. The board's power figure corresponds to its total on-chip power consumption (static plus dynamic), while the PC's power figure reflects the combined GPU, CPU, and peripheral draw.

TABLE X. AVERAGE ENERGETIC EFFICIENCY COMPARISON BETWEEN THE FPGA BOARD AND THE PC

Platform	Resolution	Power (w)	Avg. fps	Efficiency (fps/W)
Board (FPGA)	1920×1088	1.891	0.047	0.025
Board (FPGA)	1024×768	1.891	0.125	0.066
PC (GPU+CPU+periph.)	1920×1088	143	2.857	0.020
PC (GPU+CPU+periph.)	1024×768	143	3.846	0.027

Using the board's total on-chip power consumption of 1.891 W, the FPGA board remains the more energetically

efficient platform on a frames-per-watt basis at both resolutions, despite its substantially lower raw throughput. At 1920×1088 , the board reaches 0.025 fps/W against 0.020 fps/W for the PC, and at 1024×768 the gap widens further, with the board achieving 0.066 fps/W against 0.027 fps/W for the PC, corresponding to an efficiency advantage of roughly 2.4 times at the lower resolution. This result follows directly from the board's power consumption being nearly two orders of magnitude lower than the PC's (1.891 W versus 143 W), a difference large enough to outweigh the board's lower absolute frame rate. The efficiency advantage is also resolution-dependent: because the number of coding units per frame, and therefore the board's per-frame latency, scales with frame area, the board's relative efficiency improves as resolution decreases, whereas the PC's efficiency scales more moderately since its processing pipeline is not similarly bottlenecked by sequential per-coding-unit evaluation. These results suggest that, although the board remains far from real-time throughput in absolute terms, its very low power envelope already makes it a favourable choice for energy-constrained deployments where frame-rate requirements are modest, and that closing the throughput gap through architectural optimisations such as coding-unit-level parallelism or deeper pipelining would further widen this efficiency advantage in the board's favour.

5) *Carbon footprint*: Beyond raw energetic efficiency, the substantial power gap between the two platforms also translates into a markedly different carbon footprint per unit of operating time. Using the International Energy Agency's (IEA) reported global average electricity grid carbon intensity of approximately 445 g CO₂/kWh in 2024 [21], the FPGA board's total on-chip power consumption of 1.891 W corresponds to an emission rate of roughly 0.000234 g CO₂ per second of operation, compared to approximately 0.0177 g CO₂ per second for the PC at 143 W, a difference of about two orders of magnitude consistent with the power ratio between the two platforms. Although the PC's higher throughput partially offsets this gap on a per-frame basis, the board still emits noticeably less CO₂ per unit of processing time at both resolutions, reinforcing its suitability for energy- and emissions-constrained deployment contexts, particularly as future architectural optimisations narrow the throughput gap without altering its favourable power profile.

6) *Comparison with reference design*: Table XI compares the key performance figures reported by Bouaafia et al. [16] for their PYNQ-Z1-based CNN_1 accelerator against those obtained for our proposed design on the PYNQ-Z2 platform, both targeting the 64×64 -level CU partition prediction for HEVC.

TABLE XI. COMPARISON WITH THE REFERENCE PYNQ-Z1 CNN ACCELERATOR [16]

Metric	Bouaafia et al. [16]	Our Design
Platform	PYNQ-Z1 (XC7Z020)	PYNQ-Z2 (XC7Z020)
PL frequency (MHz)	142	54.1
LUT utilisation (%)	47	65
FF utilisation (%)	26	21
BRAM utilisation (%)	16	33
DSP utilisation (%)	14	10
On-chip power (W)	1.69	1.891
Processing time / CU	400 ms	31.3–51.8 ms

Compared with the reference design, our accelerator op-

erates at a substantially lower clock frequency (54.1 MHz versus 142 MHz), yet achieves a processing time per 64×64 coding unit that is roughly 8 to 13 times faster (31.3–51.8 ms versus 400 ms). This indicates that, cycle efficiency aside, the reference design's higher operating frequency does not translate into a comparable advantage in absolute latency, most likely due to differences in interface overhead and inter-IP data movement between the streaming architecture and the PS in [16]. In terms of resource utilisation, our design consumes more LUTs (65% versus 47%) and more BRAM (33% versus 16%), reflecting the additional control and multiplexing logic associated with the higher degree of arithmetic resource sharing discussed earlier, while using fewer FFs (21% versus 26%) and notably fewer DSPs (10% versus 14%). On-chip power consumption is comparable between the two designs (1.891 W versus 1.69 W), with our design drawing marginally more power, which is consistent with its higher LUT occupancy. Overall, these results suggest that our design trades a modest increase in logic resource usage and power for a considerably lower per-CU processing latency, positioning it favourably relative to the reference work in terms of raw throughput, even though both designs remain well below the requirements of real-time HEVC encoding at typical video resolutions.

D. Discussion

When read together, the results above reflect the same trade-off at two levels. In software, the heuristic and machine-learning methods reviewed in Section II [11], [13] cut RDO calls through fixed or learned decision rules, in hardware, the accelerator cuts clock cycles and memory traffic through the identical rule, so the complexity reduction in Table VI and the latency spread in Table VIII are two views of the same early-termination mechanism. Relative to [16]'s multi-IP design, consolidating the network into a single IP core is what drives the latency advantage rather than clock frequency, which is lower here (54.1 versus 142 MHz), the LUT/DSP imbalance this incurs (65% versus 10%, Table IX) is the resource cost of sharing a handful of floating-point units across the network's weights, and is consistent with the low-power, sub-real-time profile in Table X. Overall, the accelerator is best read as evidence that early-termination CNN inference can be mapped onto genuine conditional hardware execution at a favourable power budget, with throughput the main gap left to close. We note that a broader comparison against other FPGA-based CNN accelerators beyond [16] is limited by the availability of full, citable results for those designs, Section II can be extended with additional hardware baselines when new works become available.

VII. CONCLUSION

In this article, a hardware-software architecture based on a deep convolutional neural network has been proposed for the prediction of coding units in 3D-HEVC on FPGA-SoC. The presented work aims to accelerate convolutional neural networks due to their high computational load. Nevertheless, a hardware IP core for MD-CNN was developed using the Vivado HLS tool. Subsequently, an architecture integrating hardware and software was developed by incorporating hardware IP cores based on the PYNQ-Z2 board. The simulation results revealed that our design achieves an on-chip power

consumption of 1.891 W. Our design requires reduced consumption of hardware resources: 65% of LUTs, 21% of FFs, 33% of BRAMs, and 10% of DSPs. As future work, we plan incorporate additional encoder modules in the hardware implementation, in particular the intra-prediction and transform modules.

REFERENCES

- [1] G. Tech, Y. Chen, K. Müller, J.-R. Ohm, A. Vetro, and Y.-K. Wang, "Overview of the multiview and 3d extensions of high efficiency video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 35–49, 2016.
- [2] G. Sanchez, L. Agostini, and C. Marcon, "Algorithms for efficient and fast 3d-hevc depth map encoding," 2020.
- [3] N. Omran, A. Maraoui, I. Werda, and B. Hamdi, "Fast partition algorithm in depth map intra coding unit based on multi-deep convolution neural network," *Journal of Real-Time Image Processing*, vol. 21, no. 1, p. 23, 2024.
- [4] H. Hamout and A. Elyousfi, "Fast 3d-hevc pu size decision algorithm for depth map intra-video coding," *Journal of Real-Time Image Processing*, vol. 17, no. 5, pp. 1285–1299, 2020.
- [5] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
- [6] S. Bakkouri and A. Elyousfi, "An adaptive cu size decision algorithm based on gradient boosting machines for 3d-hevc inter-coding," *Multimedia Tools and Applications*, pp. 1–19, 2023.
- [7] Z. Zhang, L. Yu, J. Qian, and H. Wang, "Learning-based fast depth inter coding for 3d-hevc via xgboost," in *2022 Data Compression Conference (DCC)*. IEEE, 2022, pp. 43–52.
- [8] W. Song, P. Dai, and Q. Zhang, "Content-adaptive mode decision for low complexity 3d-hevc," *Multimedia Tools and Applications*, pp. 1–16, 2023.
- [9] J. Y. Lee, M. Kang, and S.-h. Park, "Fast depth intra mode decision using machine learning in 3d-hevc," *Available at SSRN 4197680*.
- [10] J.-C. Chiang, K.-K. Peng, C.-C. Wu, C.-Y. Deng, and W.-N. Lie, "Fast intra mode decision and fast cu size decision for depth video coding in 3d-hevc," *Signal Processing: Image Communication*, vol. 71, pp. 13–23, 2019.
- [11] H. Hamout and A. Elyousfi, "A computation complexity reduction of the size decision algorithm in 3d-hevc depth map intracoding," *Advances in Multimedia*, vol. 2022, p. 3507201, 2022.
- [12] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Fast 3d-hevc depth map encoding using machine learning," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 3, pp. 850–861, 2019.
- [13] C.-H. Fu, H. Chen, Y.-L. Chan, S.-H. Tsang, H. Hong, and X. Zhu, "Fast depth intra coding based on decision tree in 3d-hevc," *IEEE Access*, vol. 7, pp. 173 138–173 147, 2019.
- [14] C. Liu, K. Jia, and P. Liu, "Fast depth intra coding based on depth edge classification network in 3d-hevc," *IEEE Transactions on Broadcasting*, vol. 68, no. 1, pp. 97–109, 2021.
- [15] H. Dehnavi, M. Dehnavi, and S. Haghazad Klidbary, "Fcd-cnn: Fpga-based cu depth decision for hevc intra encoder using cnn," *Journal of Real-Time Image Processing*, vol. 21, p. 105, 2024.
- [16] S. Bouaafia, R. Khemiri, S. Messaoud, and F. E. Sayadi, "Deep cnn co-design for hevc cu partition prediction on fpga-soc," *Neural Processing Letters*, vol. 54, no. 4, pp. 3283–3301, 2022.
- [17] ITU/ISO/IEC, "3D-HEVC HTM reference software, version HTM-16.3," 2017, accessed: May 25, 2026. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/branches/HTM-16.3-fixes/
- [18] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, "Raise: A raw images dataset for digital image forensics," in *Proceedings of the 6th ACM multimedia systems conference*, 2015, pp. 219–224.
- [19] ITU/ISO/IEC, "HEVC HM reference software, version HM-16.18," 2017, accessed: May 25, 2026. [Online]. Available: <https://vcgit.hhi.fraunhofer.de/jvet/HM/-/tree/HM-16.18>
- [20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.
- [21] International Energy Agency, "Electricity mid-year update 2025," 2025, accessed 2026. [Online]. Available: <https://www.iea.org/reports/electricity-mid-year-update-2025/emissions-power-generation-co2-emissions-are-plateauing>