VCNPU: An Algorithm-Hardware Co-Optimized Framework for Accelerating Neural Video Compression

Siyu Zhang, Wendong Mao, *Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*— Video compression is essential for storing and transmitting video content. Real-time decoding is indispensable for delivering a seamless user experience. Neural video compression (NVC) integrates traditional coding techniques with deep learning, resulting in impressive compression efficiency. However, the real-time deployment of advanced NVC models encounters challenges due to their high complexity and extensive off-chip memory access. This article presents a novel NVC accelerator, called video compression neural processing unit (VCNPU), via an algorithm-hardware co-design framework. First, at the algorithmic level, a reparameterizable video compression network (RepVCN) is proposed to aggregate multiscale features and boost video compression quality. RepVCN can be equivalently transformed into a streamlined structure without extra computations after training. Second, a mask-sharing pruning strategy is proposed to compress RepVCN in the fast transform domain. It effectively prevents the destruction of sparse patterns caused by model simplification, maintaining the model capacity. Third, at the hardware level, a reconfigurable sparse computing module is designed to flexibly support sparse fast convolutions and deconvolutions of the compact RepVCN. Besides, a hybrid layer fusion pipeline is advocated to reduce off-chip data communication caused by extensive motion and residual features. Finally, based on the joint optimization of computation and communication, our VCNPU is constructed to realize adaptive adjustments of various decoding qualities and is implemented under TSMC 28-nm CMOS technology. Extensive experiments demonstrate that our RepVCN provides superior coding quality over other video compression baselines. Meanwhile, our VCNPU achieves 6.7× improvements in throughput, 2.9× in area efficiency, and 4× in energy efficiency compared to prior video processors.**

***Index Terms*— Fast algorithm, hardware accelerator, neural video compression (NVC), pruning, reparameterization.**

I. Introduction

VIDEO has emerged as the predominant medium in the digital media era, serving a multitude of applications, including entertainment, productivity, and security. During the past two decades, several hybrid video coding standards, including H.264/AVC [1] and H.265/HEVC [2], have been developed to achieve compact video representation. However, their backbones include several independently handcrafted components, making it challenging to optimize the framework from an overall perspective. Meanwhile, the growing number of coding options incorporated into the new standards has substantially increased complexity. In contrast, learning-based video compression methods employ nonlinear transformations to lower the entropy of latent variables and enable end-to-end optimizations to achieve outstanding rate-distortion (RD) performance. Thus, integrating deep learning technologies, such as deep neural networks (DNNs), into traditional video compression frameworks holds great promise.

Recently, there has been a flourishing emergence of novel end-to-end neural video compression (NVC) approaches. Lu et al. [3], [4] pioneered the use of auto-encoder models to perform analysis transforms and synthesis transforms for motion and residual. To further enhance the accuracy of motion estimation and compensation, deformable compensation [5] was introduced, shifting NVC systems from pixel to latent space. Subsequently, several well-crafted variants have emerged to significantly boost compression efficiency, such as residual coding-based techniques [6], [7], [8], [9], conditional coding-based techniques [10], [11] among others [12], [13]. Unlike plain DNNs oriented to classification [14] and segmentation tasks [15], deploying NVC models in real-time presents new challenges: 1) their structures involve various branches and intricate connections, resulting in irregular data dependencies and complicated memory access patterns; 2) diverse operations, including deconvolutions (DeConvs) and deformable convolutions (DfConvs), incur serious memory conflicts and computational imbalances; 3) implementing efficient model compression strategies is difficult due to the sensitivity of pixel processing tasks to high sparsity; and 4) nowadays, the demand for real-time processing of high-definition (HD) video streams, such as 1280×720 and 1920×1080, with diverse compression rates at the edge is increasing, placing tremendous pressure on computation and transmission. These stringent requirements for memory and computation reflect the importance of developing solutions for model optimizations and efficient deployments in NVC.

Numerous researchers have focused on optimizing and accelerating certain operations within DNNs, such as convolution (Conv), DeConv, and DfConv, but there has been relatively limited work dedicated to NVC. Several model compression methods, such as pruning [16], quantization [17],

Received 1 July 2024; revised 3 October 2024 and 12 November 2024; accepted 6 December 2024. Date of publication 17 December 2024; date of current version 24 March 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4400600 and in part by the Shenzhen Science and Technology Program under Grant 2023A007. *(Corresponding authors: Wendong Mao; Zhongfeng Wang.)*

Siyu Zhang is with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: syzhang@smail.nju.edu.cn).

Wendong Mao is with the School of Integrated Circuits, Sun Yat-sen University, Shenzhen 518107, China (e-mail: maowd@mail.sysu.edu.cn).

Zhongfeng Wang is with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China, and also with the School of Integrated Circuits, Sun Yat-sen University, Shenzhen 518107, China (e-mail: zfwang@nju.edu.cn).

Digital Object Identifier 10.1109/TVLSI.2024.3515113

fast algorithm [18], and reparameterization [19], aim at reducing the computational complexity and memory requirements of convolutional neural networks (CNNs). For instance, Liu et al. [20] developed a sparse-driven dataflow for dual-side sparse CNNs. Li et al. [21] proposed an algorithm-hardware co-designed method to perform Convs separately on individual blocks. For DeConv, Chang et al. [22] converted DeConvs to Convs by a load balance-aware method, effectively relieving overlapped partial sums. Mao et al. [23] proposed a DeConv-tailored fast algorithm (FTA) and a dedicated accelerator, significantly reducing the computational complexity of DeConvs. For DfConv, Zhang et al. [24] simplified the offset generation and deployed a depthwise DfConv on edge devices. However, directly applying these techniques to accelerate NVC models presents serious issues as follows.

1) Current DNN accelerators [25], [26], [27] face difficulties in managing the intricate data dependencies of NVC models with complex topologies.
2) Most DNN accelerators [28], [29] primarily optimize dataflow and resource configurations tailored for certain operations. The differences in patch size, computational principles, and memory access patterns make it difficult to simultaneously support all kinds of operations within NVC models.
3) Extensive off-chip data communication for intermediate features, such as motion and residual information, leads to substantial power consumption and transmission burdens.
4) Achieving diverse levels of NVC quality requires accessing different model parameters from off-chip memory, which incurs expensive bandwidth and memory resource overhead.

Addressing these issues requires comprehensive research to enhance the performance and deployability of NVC on resource-limited devices.

In real-world applications, cloud servers afford sufficient computational resources to encode HD video streams at reasonable frame rates. Numerous users need to repeatedly execute video decoding on various mobile devices with limited hardware resources, which recover visualized video streams with different bit rates based on the actual network bandwidth available. This inherent asymmetry between encoding and decoding highlights the high demand for real-time video decoding on resource-limited devices. Therefore, we propose an algorithm-hardware co-optimized framework for NVC. The main contributions of this article are outlined as follows.

1) At the algorithmic level, a reparameterizable video compression network (RepVCN) is presented to achieve high-quality video compression by capturing multi-scale correlations based on reparameterizable fast Convs and DeConvs. Moreover, the complex multibranch RepVCN can be equivalently converted into a compact inference-time structure without incurring extra computations.
2) A mask-sharing pruning strategy is proposed to remove redundant computations in reparameterizable fast Convs and DeConvs. It avoids disrupting the sparse patterns caused by the equivalent simplification of RepVCN, preserving the model's capacity. In addition, a lightweight quality modulating layer (QML) is designed to achieve diverse compression qualities while reducing training costs.
3) At the hardware level, to save computation resources, a reconfigurable sparse computing module is developed to flexibly support sparse Convs and DeConvs. Besides, a hybrid layer fusion pipeline is incorporated to reduce off-chip data communication and facilitate the reuse of on-chip memory for large motion and residual features.
4) Based on the joint optimization of computation and communication, a video compression neural processing unit (VCNPU) is constructed to realize real-time video decoding, implemented under TSMC 28-nm CMOS technology. Experimental results demonstrate that the proposed method delivers superior compression performance compared to other NVC models and outperforms previous accelerators in terms of hardware efficiency.

The rest of this article is organized as follows. Section II provides a detailed introduction to previous works. Section III outlines the specifics of RepVCN. The overall hardware architecture, computing modules, and dataflow are discussed in Section IV. Section V exhibits evaluations regarding algorithm accuracy and hardware efficiency. Finally, the conclusion is drawn in Section VI.

II. Related Work

A. Neural Video Compression

Traditional video codecs, such as H.264/AVC [1] and H.265/HEVC [2], rely on block-based hybrid frameworks to achieve intraframe and interframe predictions. However, these hand-designed solutions limit the joint optimization of independent components within the hybrid frameworks. Meanwhile, they commonly employ linear transformations, which may be inadequate for effectively reducing the entropy of the variables to be compressed. In contrast, learning-based NVC methods utilize nonlinear transformations and end-to-end optimizations, significantly enhancing the video compression efficiency. For instance, Lu et al. [3] first proposed an end-to-end deep video compression model by integrating DNNs with traditional frameworks. To further boost the accuracy of optical flow estimation and motion compensation, a feature-space video compression network [5] was developed by introducing DfConvs. Subsequent research replaced the residual coding with conditional coding [10], using feature-domain contexts as conditions to improve compression efficiency. To reduce the algorithmic complexity of NVC decoders, cgSlimDecoder [30] automatically allocated optimal widths for different components with the assistance of slimmable neural networks.

More recently, Le et al. [31] designed a mobile-friendly NVC model and first ran it on a commercial mobile phone, highlighting the potential for deploying NVC tasks at the edge. However, focusing solely on algorithmic optimizations may not fully leverage the potential for accelerating NVC on hardware devices with limited resources. Thus, investigating algorithm and hardware co-optimization for NVC tasks can enhance the achievement of high-performance video compression and reconstruction on resource-constrained devices.

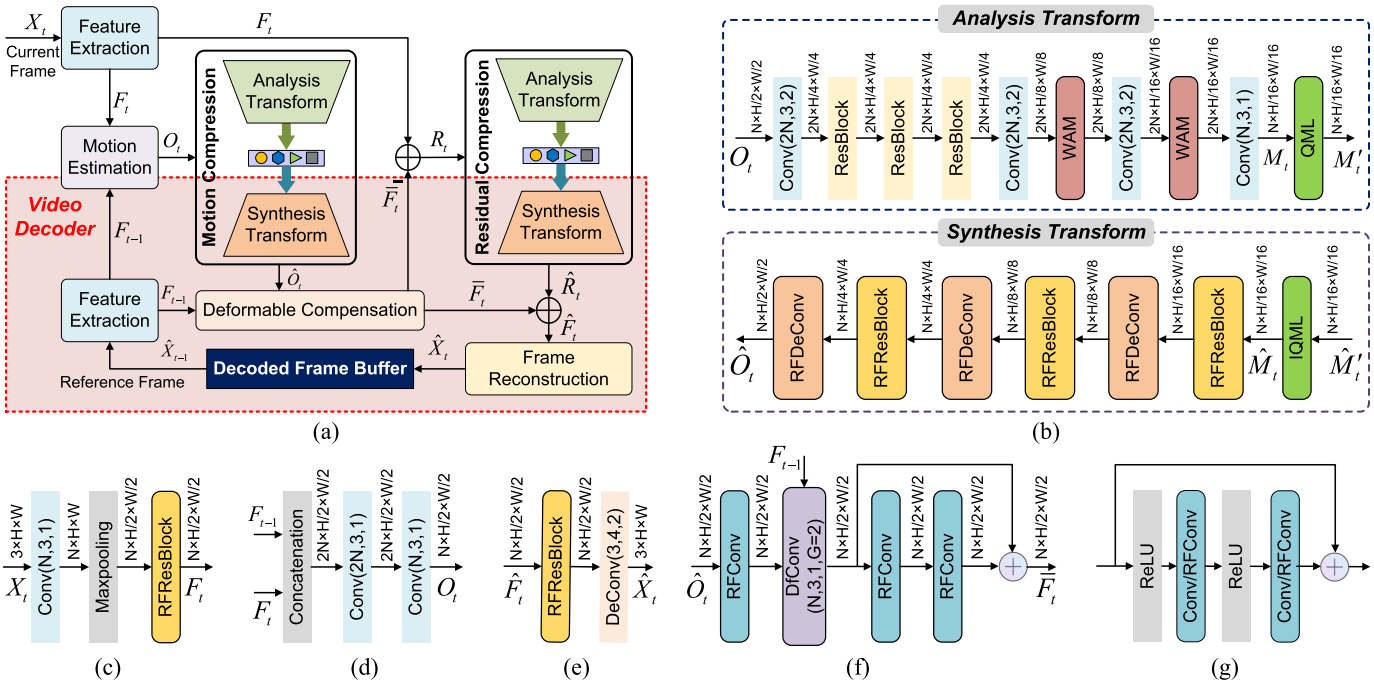

Fig. 1. Overview framework of the RepVCN. (a) Network structure. (b) Analysis and synthesis transform for motion and residual compression. (c) Feature extraction. (d) Motion estimation. (e) Feature reconstruction. (f) Deformable motion compensation. (g) Residual block (ResBlock) or reparameterizable fast ResBlock (RFResBlock). In "Conv(N, k, s)" and "DeConv(N, k, s)," the output channel number, kernel size, and stride of Conv and DeConv are represented as N, $k \times k$, and s, respectively. G of "DfConv(N, K, Str, G)" means the channel dimension of F_{t-1} is divided into G groups in DfConv.

B. Model Compression Algorithm

Typical model compression algorithms include pruning, fast algorithms, and quantization. For example, researchers introduced the Winograd algorithm [18] for fast implementation of small-sized Convs. Andri et al. [32] realized a fully integer inference for performing Winograd Convs on large patches. In low-level tasks, DeConv is a computationally intensive and time-consuming operation. Chang et al. [29] transformed a DeConv into multiple Convs, thereby indirectly utilizing the Winograd algorithm for accelerating the DeConvs.

Model pruning effectively removes unimportant connections to compress memory and computational workload. For instance, DPP [16] developed hardware-oriented sparse structures at various granularities, achieving joint optimization of pruning and weight quantization. Besides, Liu et al. [33] moved the ReLU function into the transform domain, coupling fast algorithms with pruning strategies to extremely minimize computations. SWM [34] was developed for large kernels to stabilize load balance by exploring the sparsity of activations and weights.

Compared to high-level tasks, which are more robust to the loss of fine-grained details, applying an independent model compression algorithm to NVC models often fails to maintain satisfactory video quality due to the strict requirements of pixel-wise predictions. Therefore, combining multiple model compression algorithms is essential for preserving model representational capabilities while reducing computational complexity in NVC tasks.

III. Proposed Novel NVC Model

A. Overview Framework of RepVCN

RepVCN compresses videos in the latent space to accurately model nonrigid motion and produce high-quality frames. As illustrated in Fig. 1(a), to effectively exploit the interframe information, RepVCN takes the video stream $X = \{\ldots, X_{t-1}, X_t, X_{t+1}, \ldots\}$ as the input, and employs spatial-temporal redundancy to compress the current frame X_t based on the previously reconstructed reference frame $\hat{X}_{t-1}$ at any given bit rate. Ultimately, the resulting frame $\hat{X}_t$ is stored in the decoded frame buffer. Note that the RepVCN decoder is indicated by a red dashed box in Fig. 1(a).

1) Feature Extraction: X_t and $\hat{X}_{t-1}$ are first downsampled and transformed from the pixel domain to the latent domain using several stacked Conv layers in Fig. 1(c). Then, the corresponding frame representations F_t and F_{t-1} are generated.

2) Motion Estimation: To analyze the interframe motion relationships, the motion vectors O_t between the adjacent frame features F_t and F_{t-1} are extracted in Fig. 1(d).

3) Motion Compression: As shown in Fig. 1(b), O_t is compressed in a lossy manner using an auto-encoder style network, which consists of an analysis transform and a synthesis transform. The analysis transform compresses O_t into a compact feature M'_t by incorporating several Conv layers and window attention modules (WAMs) [35]. Subsequently, M'_t is quantized and formatted as bitstreams. The synthesis transform decompresses these bitstreams into a motion feature $\hat{O}_t$ using several reparameterizable fast convs (RFConvs) and DeConvs (RFDeConvs).

4) Deformable Compensation: Next, $\hat{O}_t$ is used to compensate for F_{t-1} through a grouped DfConv and several RFConvs in Fig. 1(f). The predicted frame feature $\bar{F}_t$ is produced.

5) Residual Compression: To reduce the prediction error in the spatial domain, the residual feature $R_t = F_t - \bar{F}_t$ is compressed using another auto-encoder style network, analogous to that shown in Fig. 1(b).

6) Feature Reconstruction: By combining the synthesized residual feature $\hat{R}_t$ with $\bar{F}_t$, the reconstructed feature $\hat{F}_t$ is

obtained as $\hat{F}_t = \hat{R}_t + \bar{F}_t$. Finally, $\hat{F}_t$ is transformed back from the latent domain to the pixel domain in Fig. 1(e), and the recovered frame $\hat{X}_t$ is stored in the decoded frame buffer.

B. Reparameterizable Fast Conv and DeConv

Achieving real-time and high-quality NVC on resource-constrained devices is challenging due to the tradeoff between model capacity and algorithm complexity. While enhancing the model capacity can be straightforward by constructing a multibranch topology with multiple paths of various scales and complexities, this approach often results in high computational overhead.

Structural reparameterization [19] improves inference efficiency while preserving multiscale learning capabilities. However, when the kernel stride is smaller than the spatial size, redundant multiplications persist. Given that fast algorithms [18] can effectively address this issue, we combine two seemingly independent and incompatible model compression techniques, reparameterization and fast algorithm, to construct RFConvs and RFDeConvs in the RepVCN decoder. This integration enhances model capacity through the use of asymmetric kernels and split-bypass typologies. This is also the first attempt to extend reparameterization techniques to DeConvs. Due to the high training cost associated with multibranch structures, the RepVCN encoder continues to employ dense and standard Convs. By decoupling the training and inference phases, multibranch RFConvs and RFDeConvs can be equivalently replaced with standard Convs and DeConvs, thereby reducing algorithm complexity.

1) Fast Conv and DeConv: Compared to individually computing each element within output features, the fast algorithm leverages structural similarities to generate output patches, reducing the computational complexity. Inspired by the Conv-tailored fast algorithm (Winograd) [18] and DeConv-tailored fast algorithm (FTA) [23], we represent the fast Conv and DeConv by a general formula with distinct transform matrices

$$\mathbf{V} = \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{G}\mathbf{W}\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A} \tag{1}$$

where $\mathbf{X}$, $\mathbf{W}$, and $\mathbf{V}$ represent the input patch of size $p \times p$, the weight patch of size $k \times k$, and the output patch of size $m \times m$, respectively. $\odot$ is the Hadamard product. The multiplication number is $\mu \times \mu$. $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{G}$ are transform matrices.

2) Reparameterization of the Fast Conv and DeConv: The reparameterization technique enables the decoupling of the training and inference phases. Specifically, during the training phase, considering that weights on the kernel skeleton (the central crisscross position) are typically more critical than those surrounding them, we design horizontal and vertical kernels to enhance the skeleton and extract details at different scales. As illustrated in Fig. 2, the RFConv comprises five branches, including 1×1, 1×3, 3×1, 3×3 Conv kernels, along with an identical mapping. The RFDeConv is a three-branch structure, featuring 2×4, 4×2, and 4×4 DeConv kernels. This topology allows the feature spaces of RFConvs and RFDeConvs to be enhanced by merging multiscale representations

$$\mathbf{V} = \sum_{b=1}^{N_b} \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{G}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A} \tag{2}$$

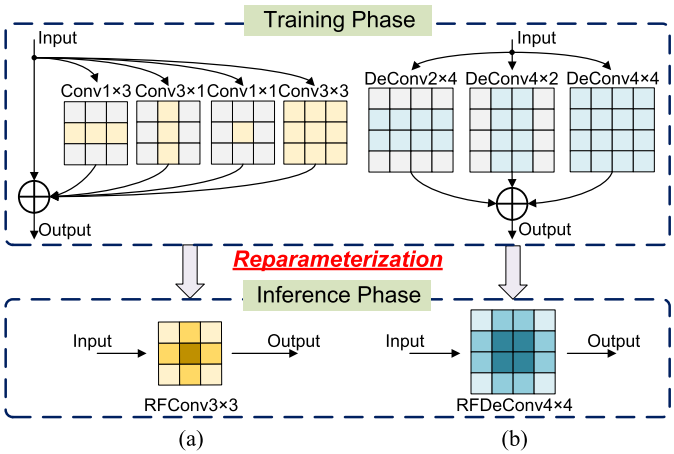

Fig. 2. Descriptions of asymmetric kernels and split-bypass typologies of (a) RFConv and (b) RFDeConv.

where N_b is the branch number. W_b is the bth branch's kernel.

During the inference phase, unlike standard sliding-window computations, asymmetric kernels are transformed into the largest kernel through zero-padding to simplify the complicated structure, as shown in Fig. 2. Consequently, the 2×4 and 4×2 kernels in RFDeConvs can be made compatible with the 4×4 kernels by sharing the same sliding window, and the same applies to RFConvs. With these compatible kernels, multiscale features can be fused using a linear combination

$$\mathbf{V} = \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{G}\sum_{b=1}^{N_b}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A}. \tag{3}$$

In this manner, RepVCN can extract multiscale features during the training phase and is equivalently simplified to standard fast Convs and DeConvs during the inference phase, all without necessitating additional computations.

C. Mask-Sharing Pruning Strategy

To further reduce model complexity, we aim to prune the transform-domain weights of the multibranch RFConvs and RFDeConvs. Many efforts [33], [34] have been dedicated to integrating fast algorithms with pruning techniques to enhance the sparsity in the transform domain. If we directly follow the ideas from these works, we need to generate a specific mask $\mathbf{M}_b$ for each branch to indicate unimportant weights in the transform domain

$$\mathbf{V} = \sum_{b=1}^{N_b} \mathbf{A}^{\mathbf{T}}\left[\left(\mathbf{M}_b \odot \mathbf{G}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}\right) \odot \left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A}. \tag{4}$$

However, independently optimizing $\mathbf{M}_b$ leads to varying sparsity patterns across different branches. The linear combination of all branches may disrupt the sparsity of reparameterized kernels, which is counterproductive. To solve this problem, the mask-sharing pruning strategy is proposed to generate a universal mask $\mathbf{M}$ by predicting the sparse pattern of the streamlined structures in advance. Then, it is broadcasted to each branch, maintaining the same sparse pattern for the equivalent shrink network in the inference phase.

Specifically, based on (3), the transform-domain weight patch of the simplified RFConv or RFDeConv is represented

as $\mathbf{E} = \mathbf{G}\sum_{i=b}^{N_b}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}$. Note that each element of $\mathbf{E}$ is derived by weighted summation of uncertain elements from $\sum_{i=b}^{N_b}\mathbf{W}_b$. Similarly, for an output patch $\mathbf{V}$, each element is also derived by weighted summation from an uncertain number of $\mathbf{E}$ and input patches $\mathbf{X}$. Given that the contributions of each element $\mathbf{E}_{i,j}$ to $\mathbf{V}_{c,d}$ are not uniform, directly pruning $\mathbf{E}_{i,j}$ based solely on their magnitudes would inevitably ignore their actual importance. Therefore, an factor matrix $\mathbf{Q}$ is introduced to adjust the magnitudes of $\mathbf{E}_{i,j}$

$$\mathbf{Q}_{i,j} = \sqrt{\sum_{\substack{0\leqslant c,d\leqslant m-1,\\ 0\leqslant q,v\leqslant p-1}} \mathbf{H}^2_{c,d,i,j,q,v}} \quad 0\leqslant i,j\leqslant \mu-1 \tag{5}$$

$$\mathbf{H}_{c,d,i,j,q,v} = \mathbf{A}_{i,c}\cdot\mathbf{A}_{j,d}\cdot\mathbf{B}_{q,i}\cdot\mathbf{B}_{v,j}. \tag{6}$$

Then, based on the predefined sparsity ρ, $\tilde{\mathbf{M}}$ is generated

$$\tilde{\mathbf{M}}_{i,j} = \begin{cases} 0, & \mathbf{Q}^2_{i,j}\cdot\mathbf{E}^2_{i,j} < \zeta \\ 1, & \mathbf{Q}^2_{i,j}\cdot\mathbf{E}^2_{i,j} \geqslant \zeta \end{cases} \quad 0\leqslant i,\ j\leqslant \mu-1 \tag{7}$$

where ζ denotes the pruning threshold.

In this manner, our mask-sharing pruning strategy during the training phase can be expressed as follows:

$$\mathbf{V} = \sum_{b=1}^{N_b}\mathbf{A}^{\mathbf{T}}\left[\left(\tilde{\mathbf{M}}\odot\mathbf{G}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}\right)\odot\left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A} \tag{8}$$

and the inference-time sparse RFConvs and RFDeConvs are

$$\mathbf{V} = \mathbf{A}^{\mathbf{T}}\left[\tilde{\mathbf{M}}\odot\left(\mathbf{G}\sum_{b=1}^{N_b}\mathbf{W}_b\mathbf{G}^{\mathbf{T}}\right)\odot\left(\mathbf{B}^{\mathbf{T}}\mathbf{X}\mathbf{B}\right)\right]\mathbf{A}. \tag{9}$$

By applying this fine-grained structural pruning to weight patches, redundant elements can be effectively eliminated, significantly improving the inference efficiency. Furthermore, our pruning strategy unifies the nonzero weight count rather than sparse patterns across patches of different channels, thereby sufficiently preserving the model's representation capacity.

D. QML for Variable Compression Rate

To achieve the tradeoff between video quality and compression rate, the RD criterion $\mathcal{L} = \lambda\cdot\mathcal{D} + \mathcal{R}$ is typically employed to optimize NVC models end-to-end. Here, λ is a hyperparameter that balances the distortion term $\mathcal{D}$ and the compression rate cost term $\mathcal{R}$ for motion and residual features. Previous methods [3], [5] repeatedly tuned λ and fine-tuned the entire NVC model with a new $\mathcal{L}$ to achieve different compression qualities. However, this incurs substantial training costs. When decoding videos of varying quality, this method requires mobile devices to load numerous model parameters to accommodate different network bandwidth, resulting in high off-chip data transmission latency and power consumption.

To solve this problem, we propose a lightweight QML and its inverse transform (IQML) to flexibly adjust latent representations, as shown in Fig. 1(b). For QML, given a compressed feature M_t, the scaling feature M'_t corresponding to the target compression rate is defined as

$$M'_t = f_{\mathrm{QML}}(M_t)$$
$$= \begin{cases} M_t, & \lambda = \lambda_b \\ M_t\cdot\left(1-\sigma_{\mathrm{Sigmoid}}(\theta\cdot M_t+\varphi)\right), & \lambda\neq\lambda_b \end{cases} \tag{10}$$

where λ_b corresponds to the highest compression rate. (θ,φ) are learnable parameters to extract the scaling information.

Symmetrically, IQML (f_{IQML}) is parameterized by another set of learnable parameters (θ',φ')

$$\hat{M}_t = f_{\mathrm{IQML}}(\hat{M}'_t)$$
$$= \begin{cases} \hat{M}'_t, & \lambda = \lambda_b \\ \hat{M}'_t\cdot\left(1+\sigma_{\mathrm{ReLU}}(\theta'\cdot\hat{M}'_t+\varphi')\right), & \lambda\neq\lambda_b. \end{cases} \tag{11}$$

At runtime, we first train the entire RepVCN as a baseline model at the highest compression rate. Then, all parameters in the baseline model are frozen. To adapt RepVCN for lower bit rates, we only fine-tune the QMLs and IQMLs with different λ. This approach allows models at various compression rates to share the same baseline model parameters, which sufficiently saves training costs and reduces memory demands and off-chip data communication for hardware deployments.

IV. NVC Accelerator

A. Overall Hardware Architecture

To achieve video decoding at variable compression rates on resource-limited devices, we propose VCNPU, which features a multicore architecture. As shown in Fig. 3(a), VCNPU is constructed from P NVC cores, a global controller, an IQML parameter buffer, and an IQML parameter selector. The multicore architecture allows for the parallel reconstruction of P video frames at a specific bit rate or provides decoding with P different compression rates for a single frame. Each NVC core consists of a sparse fast transform module (SFTM) and a DfConv processing module (DPM). The SFTM weight buffer, SFTM index buffer, and DPM weight buffer save nonzero weights, mask indices, and DfConv weights, respectively. Since IQML has few parameters, all (θ',φ') can be initially stored in the IQML parameter buffer to support NVC at N_r (where $N_r \geq P$) compression rates. This design enables SFTMs and DFMs from different NVC cores to share the same weights and indices. Compared to loading various parameters from external memory for different compression rates, VCNPU enables adaptive adjustments for different decoding qualities by switching parameters saved in the IQML parameter buffer, thus significantly reducing memory demands and off-chip memory interactions.

1) Sparse Fast Transform Module: As shown in Fig. 3(b), SFTM executes sparse RFConvs and RFDeConvs in the feature extraction, synthesis transform, deformable compensation, and frame reconstruction of the RepVCN decoder, based on the proposed hybrid layer fusion pipeline. The remaining dense Convs and DeConvs located at the head and tail of the RepVCN decoder are also executed using SFTM in the fast transform domain. Note that sparse RFConvs and RFDeConvs are standard Convs and DeConvs that have been simplified and pruned offline. The on-chip memory includes an SFTM input buffer and SFTM output buffer. The SFTM input buffer uses multiplexers to determine whether to store off-chip input activations or the interlayer results of the sparse RFConvs, based on the dependences of RepVCN, which helps reduce the demand for off-chip data communication. The results of

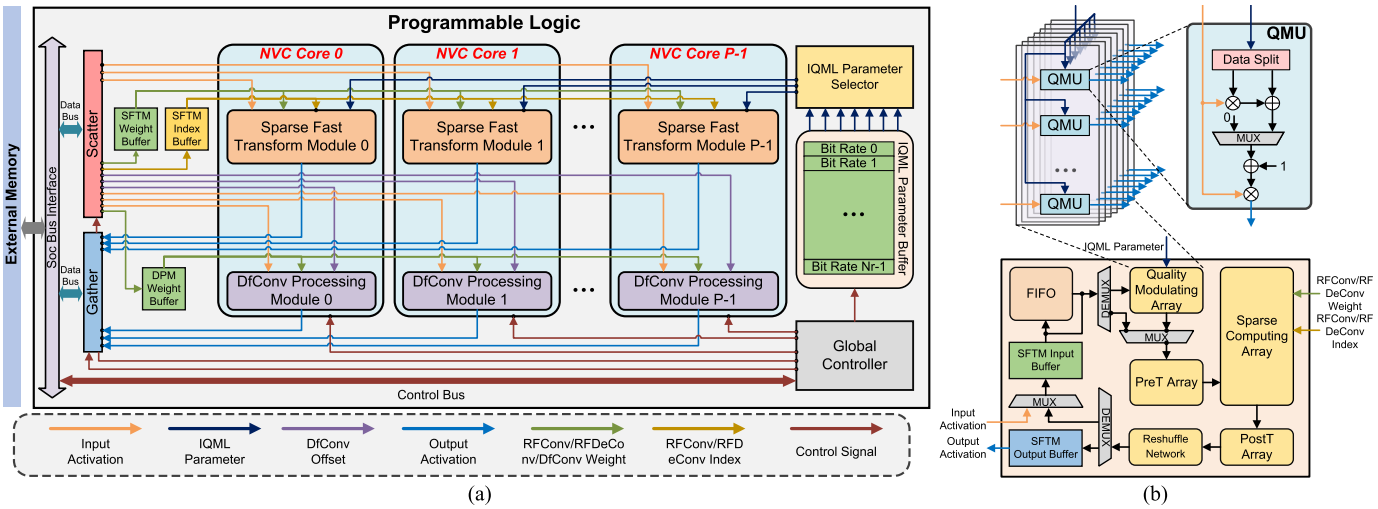

Fig. 3. Overall description of VCNPU. (a) Top-level block diagram of the overall architecture. (b) Details of SFTM.

sparse RFDeConvs are written to the SFTM output buffer and then transferred to external memory. The computing logic of SFTM includes a quality modulating array (QMA), pretransform array (PreTA), posttransform array (PosTA), and sparse computing array (SCA). The QMA maps motion and residual features for any given compression rate and is composed of multiple quality modulating units (QMUs) that perform IQML computations in parallel, following (11). PreTA and PosTA perform domain transforms between spatial and transform domains. The SCA executes Hadamard products between nonzero weights and inputs selected by mask indices. Besides, a first input first output (FIFO) and reshuffle network reorder temporary inputs and outputs before sending them to the subsequent logic.

2) DfConv Processing Module: DPM supports the DfConvs of RepVCN and cooperates with SFTM. It includes an address converter, coefficient generator, bilinear computing array, Conv computing array, and specific on-chip buffers. For future details on DPM, please refer to our previous work [24].

B. Sparse Fast Transform Module

To simultaneously support the sparse RFConvs and RFDeConvs while minimal hardware resource overhead, we first establish a tile-matching rule to determine appropriate parameter sets and input tile sizes for fast Convs and DeConvs. This ensures that interlayer results can be directly consumed on-chip by subsequent layers, thereby avoiding unnecessary off-chip interactions and result wastage. Next, we construct multiple reconfigurable computing arrays to facilitate parallel domain transforms and dot products. Finally, we design a novel hybrid layer fusion pipeline to dispatch SFTM effectively.

1) Tile-Matching Rule: Cascading two sparse RFConvs and one sparse RFDeConv is a common combination in RepVCN. If interlayer results can be fully consumed on-chip, the off-chip traffic and power consumption of the whole system will significantly drop. Thus, to determine appropriate tile sizes, we first define the parameter sets for fast Convs and DeConvs.

Given the transform order r, original kernel size k, and stride Str, the parameter sets for fast Conv $\{p_c, m_c, \mu_c, \mathrm{Si}_c, \mathrm{Ov}_c\}$ and fast DeConv $\{p_d, m_d, \mu_d, \mathrm{Si}_d, \mathrm{Ov}_d\}$ are constrained by

$$\begin{cases} p_c = k_c + r_c \\ m_c = r_c + 1 \\ \mu_c = k_c + r_c \\ \mathrm{Si}_c = r_c + 1 \\ \mathrm{Ov}_c = k_c - 1 \end{cases}, \quad \begin{cases} p_d = \lceil (k_d + r_d \times \mathrm{Str}_d - 1)/\mathrm{Str}_d \rceil \\ m_d = r_d \times \mathrm{Str}_d \\ \mu_d = k_d + (r_d - 1) \times \mathrm{Str}_d \\ \mathrm{Si}_d = r_d \\ \mathrm{Ov}_d = \lceil (k_d - 1)/\mathrm{Str}_d \rceil \end{cases} \tag{12}$$

where p_c and p_d, m_c and m_d, μ_c and μ_d, Si_c and Si_d, and Ov_c and Ov_d represent input patch size, output patch size, multiplication number, patch stride, and patch overlap of the fast Conv and DeConv, respectively. r_c and r_d measure the reduction in algorithm strength. Note that the original stride of fast Conv is constrained by $\mathrm{Str}_c = 1$.

Row tilling is applied to support the parallel processing of multiple tiles within the feature maps. Let N_{c1}, N_{c2}, and N_d represent the execution times in the row dimension for two sparse RFConvs and one sparse RFDeConv, respectively. The required number of rows Tix_{c1}, Tix_{c2}, and Tix_d for the input tiles are calculated by

$$\begin{cases} Tix_{c1} = p_c + (N_{c1} - 1) \times \mathrm{Si}_c \\ Tix_{c2} = p_c + (N_{c2} - 1) \times \mathrm{Si}_c \\ Tix_d = p_d + (N_d - 1) \times \mathrm{Si}_d \end{cases} \tag{13}$$

where N_{c1}, N_{c2}, and N_d must be positive integers. Therefore, we can derive the number of rows for the output tiles in the three operations: $Tox_{c1} = N_{c1} \times m_c$, $Tox_{c2} = N_{c2} \times m_c$, and $Tox_d = N_d \times m_d$.

To ensure that the outputs of the previous layer meet the triggering requirements of the subsequent layer, we must match the row number of interlayer tiles by constraining $Tox_{c1} = Tix_{c2}$ and $Tox_{c2} = Tix_d$. Besides, since outputs of RFDeConv are transported to external memory, we can derive the execution times N_{c1} and N_{c2} for the two RFConvs when generating $Tox_d = m_d$ rows of the output tile for one RFDeConv (i.e., $Nd = 1$), subject to (12) and (13)

$$\begin{cases} N_{c1} = N_{c2} + (k_c - 1)/(r_c + 1) \\ N_{c2} = \lceil (k_d + r_d \times \mathrm{Str}_d - 1)/(\mathrm{Str}_d \times (r_c + 1)) \rceil. \end{cases} \tag{14}$$

Thus, to execute one complete sparse RFDeConv, we need to transfer Tix_{c1} rows of pixel data to the SFTM Input Buffer, as specified by (13) and (14)

$$Tix_{c1} = \lceil (k_d + r_d \times \mathrm{Str}_d - 1)/\mathrm{Str}_d \rceil + 2 \times k_c - 2. \quad (15)$$

Based on the aforementioned constraints, we set the transform order of the sparse RFConvs and RFDeConvs to $r_c = 1$ and $r_d = 2$. At this point, the transform matrices for the sparse RFConvs are formulated as

$$\mathbf{B}^{\mathbf{T}}_{\mathrm{Conv}} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \quad \mathbf{G}_{\mathrm{Conv}} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad (16)$$

$$\mathbf{A}^{\mathbf{T}}_{\mathrm{Conv}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}. \quad (17)$$

The transform matrices for the sparse RFDeConvs are

$$\mathbf{B}^{\mathbf{T}}_{\mathrm{DeConv}} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}, \quad \mathbf{G}_{\mathrm{DeConv}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (18)$$

$$\mathbf{A}^{\mathbf{T}}_{\mathrm{DeConv}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \quad (19)$$

In this way, a $Tix_{c1} = 8$ rows of input tile is first fed into each SFTM. After being processed continuously by two sparse RFConvs, $Tix_d = 4$ rows of pixel data trigger computations for one sparse RFDeConv. Finally, a $Tox_d = 4$ rows of output tile is produced and stored in the SFTM Output Buffer.

2) Reconfigurable Computing Logic: We develop a series of efficient reconfigurable processing units, including the pretransform unit (PreTU), the posttransform unit (PosTU), and the sparse computing unit (SCU), to handle sparse RFConvs and RFDeConvs within a single SFTM.

We employ multiplexers to flexibly switch the working mode of sparse RFConvs and RFDeConvs based on specific layer configurations. As shown in Fig. 4, each PreTU consists of $\max(p_c, p_d) + \max(\mu_c, \mu_d) = 10$ 1D-PreTUs, which are utilized to transform a $p_c \times p_c = 4 \times 4$ or $p_d \times p_d = 4 \times 4$ input patch $\mathbf{X}$ into the transform domain by $\mathbf{Y} = \mathbf{B^T X B}$. Similarly, each PosTU, comprising $\max(\mu_c, \mu_d) + \max(m_c, m_d) = 10$ 1D-PosTUs, performs the inverse transform $\mathbf{V} = \mathbf{A^T U A}$ to generate an output patch $\mathbf{V}$ of size $m_c \times m_c = 2 \times 2$ or $m_d \times m_d = 4 \times 4$.

To realize data reuse in both spatial and temporal dimensions and enhance hardware efficiency, Pif PreTUs, Pof PosTUs, and Pif $\times$ Pof SCUs are organized as a PreTA, a PosTA, and an SCA, respectively. As shown in Fig. 5, since the weights of sparse RFConvs and RFDeConvs are preprocessed and pruned offline, Weight Regfiles and Index Regfiles store nonzero weight values and mask indices, respectively.

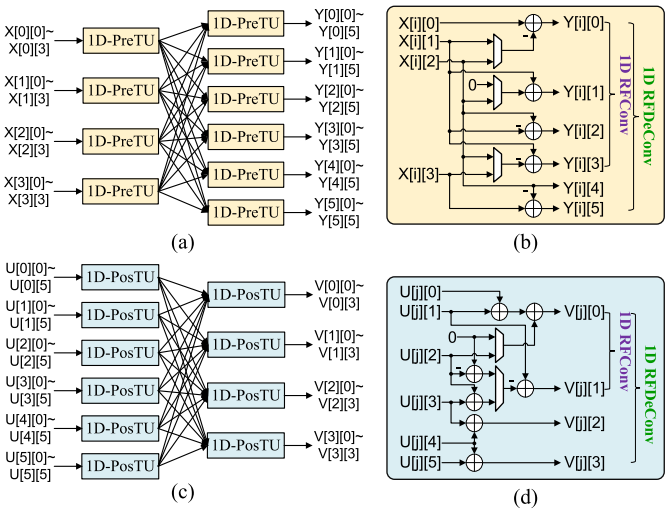

Fig. 4. Illustration of domain transform units. (a) PreTU. (b) Detail of the 1D-PreTU, which is a part of (a). (c) PosTU. (d) Detail of the 1D-PosTU, which is a part of (c).

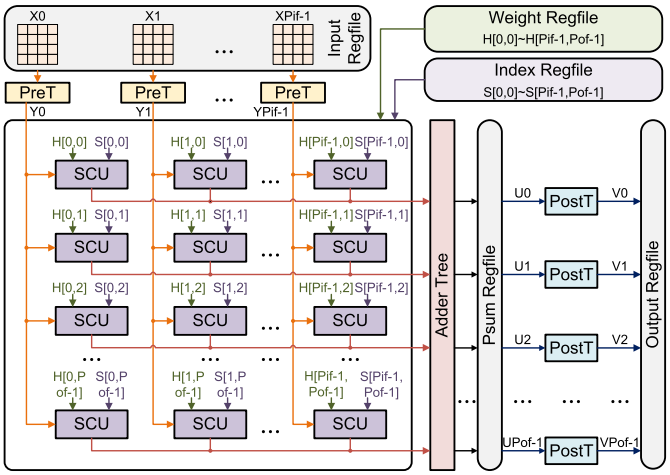

Fig. 5. Description of the micro-architectures of PreA, PosTA, and SCA.

Furthermore, weight patches for different input and output channels share the same nonzero weight count, avoiding load imbalance. Pif input channels and Pof output channels can be unrolled along the row and column directions. Within the SCA, different SCU columns handle multiple input channels, with each input channel shared among various SCU rows. Distinct nonzero weight values and indices are unicast to each SCU. After completing the Hadamard products, adder trees merge the results of the SCUs along the row dimension and send the partial sums (psums) into the Psum Regfiles.

As shown in Fig. 6, each SCU performs a Hadamard product for sparse RFConvs or RFDeConvs. Since these two types of operations require $\mu_c^2 \rho_c = 16\rho_c$ and $\mu_d^2 \rho_d = 36\rho_d$ multiplications, respectively, we enforce the sparsity of RFConvs to be $\rho_c = 37.5\%$ and that of RFDeConvs to be $\rho_d = 50\%$ to maintain the compression quality and balance the workload. One SCU can compute a sparse RFDeConv or process $(\mu_d^2 \rho_d)/(\mu_c^2 \rho_c) = 3$ output channels of a sparse RFConv in parallel. Each SCU is equipped with 18 multipliers.

3) Hybrid Layer Fusion Pipeline: Decoding HD video streams typically involves processing and transmitting extensive motion and residual information. In addition, the RepVCN

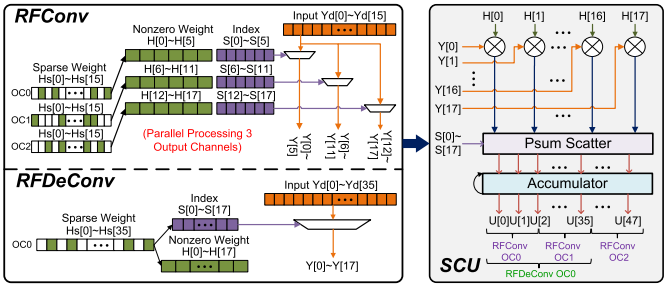

Fig. 6. Details of SCU and its dataflow for sparse RFConvs and RFDeConvs.

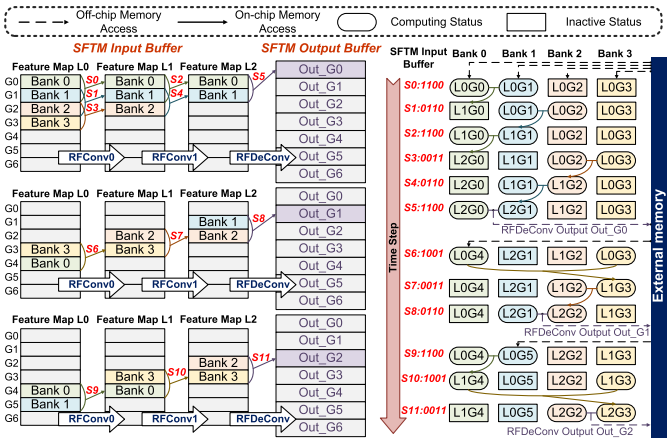

Fig. 7. Runtime scheduling of the hybrid layer fusion pipeline and management of the SFTM Input Buffer. Let Li denotes the ith feature map, Gj means the jth group, and Sk is the kth step. Note that the feature maps (left) correspond directly to the SFTM Input Buffer (right) in terms of color.

decoder comprises various operations with complex computing and memory access patterns, such as RFConvs and RFDeConvs. These challenges collectively limit throughput under constrained interface bandwidth. Frequent off-chip traffic results in significant memory resource overhead and transmission energy. While there have been studies on merged homogeneous convs using layer fusion strategies [21], mismatched data dimensions and computational principles lead to suboptimal performance when processing the RepVCN decoder, which is constructed with interleaved hybrid layers. The tile-matching rule presents opportunities for the interlayer pipeline in our VCNPU. Thus, we customize a novel hybrid layer fusion pipeline to reduce redundant off-chip memory accesses and reuse on-chip memory by partially fusing the computations of sparse RFConvs and RFDeConvs.

In Fig. 7, considering that the output patch size is consistent with the patch stride ($m_c = \mathrm{Si}_c = 2$) for RFConvs, output tiles can be rewritten to the SFTM Input Buffer without affecting the RFConv computations of other tiles. Thus, $m_c = 2$ rows of input data form an input group G, while $m_d = 4$ rows of output data create an output group Out_G. To reuse these overlapping rows, $Tix_{c1} = 8$ rows of input data in the feature map $L0$ are partitioned into four groups and initially transferred to $Tix_{c1}/\mathrm{Si}_c = 4$ banks of the SFTM Input Buffer: Bank0, Bank1, Bank2, and Bank3. Each bank has independent flags, "0" and "1," representing accessing conditions such as computing and inactive status. During the working phase, two banks are in the computing status, while others are in the inactivate status.

Besides, the SFTM weight buffer contains three segments to store the nonzero transform-domain weights for two sparse RFConvs and one RFDeConv. The VCNPU can flexibly control the reading addresses of the SFTM weight buffer based on the current operation type, avoiding the repeated loading of weight data from off-chip memory due to frequent switching between operation types in the hybrid layer fusion pipeline.

Specifically, once the input data for an operation is sufficient, the operation will be prioritized for execution. According to this rule, we can illustrate with Step S0 to S2 as an example.

Step S0 (Status 1100): The input activations from Bank0 and Bank1 are transferred to the computational logic and used in the calculations of RFConv0. Since the first $\mathrm{Si}_c = 2$ rows of input data $L0G0$ do not need to be reused, the $m_c = 2$ rows of output data $L1G0$ are written back to Bank0.

Step S1 (Status 0110): After the computations of the first $p_c = 4$ rows of data ($L0G0$ and $L0G1$) are completed, the data $L0G2$ from Bank2 is combined with the overlapping data $L0G1$ from Bank1, and RFConv0 is executed again on this combined data. Subsequently, the output data $L1G1$ is sent back to Bank1, overwriting the obsolete data $L0G1$.

Step S2 (Status 1100): When the data $L1G0$ and $L1G1$ meet the computational requirements of RFConv1, Bank0 and Bank1 are activated once again to execute RFConv1. Finally, the computation results from $L2G0$ are stored in Bank0, overwriting the first $\mathrm{Si}_c = 2$ rows of data $L1G0$.

Afterward, we execute all the chained computations comprising RFConv0, RFConv1, and RFDeConv according to the aforementioned pipeline. The results of RFDeConvs are sent to the SFTM Output Buffer and transferred to the external memory. In this manner, interlayer results can be consistently stored on the chip and leveraged as inputs for adjacent layers, avoiding configuring specific buffers for each operation. Compared to layer-by-layer and Conv layer fusion strategies, our method significantly economizes massive memory resources and off-chip data communication when processing large feature maps, which is highly beneficial for low-power and real-time NVC. Note that we also apply this method to handle scenarios with only continuous RFConvs in the feature extraction and deformable compensation stages of the RepVCN decoder.

V. Evaluation

A. Experiment Setup

1) Datasets: We apply the Vimeo-90K dataset [36] to optimize our RepVCN. The video sequences are randomly cropped to a resolution of 256 × 256. During the inference stage, we employ the HEVC [2], UVG [37], and MCL-JCV [38] datasets to evaluate the algorithm's effectiveness. The HEVC dataset [2] includes video sequences of various resolutions, such as 1920 × 1080 (Class B), 832 × 480 (Class C), 416 × 240 (Class D), and 1280 × 720 (Class E). The UVG dataset [37] consists of seven high frame rates video sequences, all at 1920 × 1080, while the MCL-JCV dataset [38] contains thirty video clips, also at 1920 × 1080.

2) Algorithm Implementation: RepVCN is implemented by the Pytorch framework on NVIDIA RTX3090 GPUs. During the training phase, we train the model for 30 epochs using the Adam Optimizer. The batch size is set to 4, and the learning rate is configured at $5e^{-5}$. We standardize all channel numbers in the decoding part to $N = 36$. To convert values from floating-point (FP) to fixed-point (FXP) format,

TABLE I

BDBR(%) RESULTS OF TEN BASELINE METHODS, WHERE H.265 IS EMPLOYED AS AN ANCHOR. THE BEST AND SECOND BEST METHODS ARE, RESPECTIVELY, MARKED IN **BOLD** AND UNDERLINED

Metric	BDBR (PSNR) (%) ↓†								BDBR (MS-SSIM) (%) ↓†							
Method	UVG	HEVC					MCL-JCV	Average	UVG	HEVC					MCL-JCV	Average
		Class B	Class C	Class D	Class E	Average				Class B	Class C	Class D	Class E	Average		
H.264 [1]	35.27	28.12	16.94	14.15	44.52	25.93	31.35	30.85	16.97	14.19	8.39	5.13	29.18	14.22	16.18	15.79
DVC [3]	8.45	4.85	27.36	17.05	3.28	13.14	13.94	11.84	13.56	-1.10	-5.88	-12.05	-7.70	-6.68	16.83	7.90
H.265 [2]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DVC++ [4]	-17.80	-18.33	-0.60	-7.23	-16.53	-10.67	-	-14.24	-11.04	-21.61	-19.22	-24.86	-20.66	-21.59	-	-16.31
HLVC [8]	-4.56	-9.72	11.55	-10.80	-	-2.99	-	-3.78	-28.56	-36.32	-21.98	-35.91	-	-31.40	-	-29.98
ECCV'20 [6]	-7.34	-15.92	-3.78	-8.29	-8.73	-9.18	-18.08	-11.53	-29.09	-15.06	-20.35	-23.83	-7.35	-16.65	0.65	-15.03
MLVC [7]	-11.49	-39.00	16.29	-13.29	-20.75	-14.19	-	-12.84	-29.95	-45.19	-20.76	-37.01	-16.55	-29.88	-	-29.91
RaFC [9]	-13.27	-14.91	1.76	-1.77	-19.92	-8.71	-13.71	-11.90	-31.35	-37.51	-26.99	-28.86	-37.83	-32.80	-37.31	-33.82
FVC [5]	-28.71	-23.75	-14.18	-18.39	-16.41	-18.18	-22.48	-23.12	-49.99	-54.68	-46.16	-52.14	-46.06	-49.76	-53.24	-51.00
DCVC [10]	-35.00	-37.96	-14.87	-26.19	-18.09	-24.28	-23.79	-27.69	-48.91	-51.64	-42.82	-50.67	-17.94	-40.77	-50.17	-46.62
RepVCN‡	**-43.64**	**-46.39**	**-18.12**	**-27.28**	**-28.05**	**-29.96**	**-33.15**	**-35.58**	**-54.93**	**-61.05**	**-46.90**	**-54.53**	**-62.29**	**-56.19**	**-56.88**	**-56.00**

†Negative values of BDBR mean bit rate savings, while positive values mean more bit rate consumption.
‡The final results after pruning and quantization are presented here.

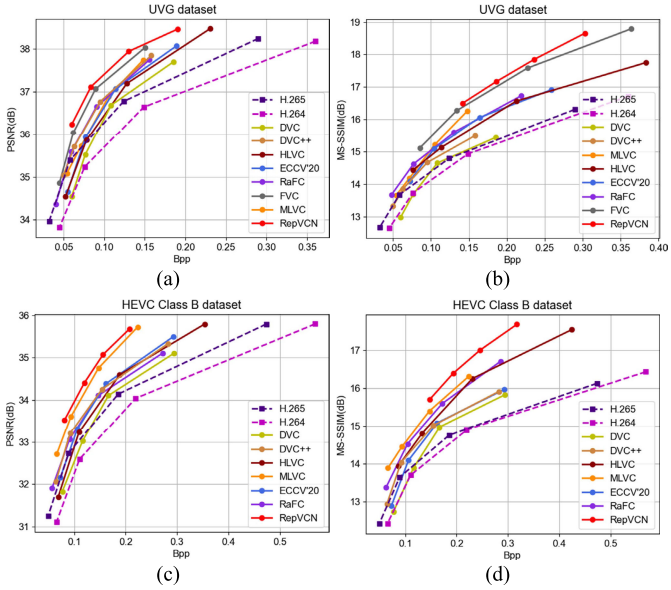

Fig. 8. RD efficiency. (a) RD-curves on UVG dataset (PSNR). (b) RD-curves on UVG dataset (MS-SSIM). (c) RD-curves on HEVC Class B dataset (PSNR). (d) RD-curves on HEVC Class B dataset (MS-SSIM).

we employ posttraining quantization [17] to quantize weights and activations to 16 and 12 bits, respectively. To ensure a fair comparison with related works, no fine-tuning is required during the inference stage for specific test sequences.

3) Hardware Implementation: To evaluate the effectiveness of our design, VCNPU is implemented in RTL, synthesized using Synopsys Design Compiler (DC), and placed and routed by IC Compiler under TSMC 28-nm technology. The accelerator operates at 400 MHz. To verify the energy efficiency, we employ the switching activity interchange format (SAIF) to estimate power consumption with Synopsys PrimeTime-PX. We apply 4-GB DDR4 DRAM as our external memory to store the weights, indexes, motion vector bitstreams, residual bitstreams, reference frames, recovered frames, and intermediate feature maps. The internal memory bandwidth is set at 12.8 GB/s. For hardware design parameters, our design supports video decoding with $N_r = 4$ compression rates. The numbers of columns and rows for SCA, as well as the number of NVC Core are Pif $= 12$, Pof $= 4$, and $P = 2$, respectively. The sizes of the SFTM input buffer, weight buffer, index buffer, and output buffer are 32, 80, 32, and 32 KB, respectively. The sizes of DPM input buffer, weight buffer, offset buffer, and output buffer are 48, 32, 48, and 42 KB, respectively. These parameters can be flexibly adjusted based on available hardware resources, task complexity, and performance metrics.

B. Algorithm Evaluation

1) Evaluation Metrics: We use bit per pixel (bpp) to measure the average number of bits used for motion and residual coding per pixel in each frame, and PSNR and MS-SSIM [39] to evaluate the distortion between the reconstructed frame and ground-truth. Besides, to assess the performance of different video compression methods, we employ the BDBR (%) metric [40], which represents the average percentage of compression rate savings when compared to baseline algorithms at the same PSNR and MS-SSIM.

2) Setting of the Baselines: To validate the effectiveness of RepVCN, several recently published learning-based video compression methods [3], [4], [5], [6], [7], [8], [9], [10] as well as conventional video compression standards [1], [2] are chosen for comparisons. We set the group-of-pictures (GoP) size to 10 for HEVC datasets and 12 for other datasets to ensure fair comparisons with other works. Besides, we employ $\lambda = \{256, 512, 1024, 2048\}$ for PSNR and $\lambda = \{8, 16, 32, 64\}$ for MS-SSIM to realize the various compression rate points.

3) RD Efficiency: Table I reports the BDBR results across all datasets using H.265 [2] as the anchor method. Note that Table I displays the final results of our RepVCN after reparameterization, pruning, and quantization. RepVCN demonstrates the best video coding performance and significantly outperforms representative video compression methods [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. Specifically, compared to the traditional H.265 standard [2], RepVCN achieves an average bit rate savings of 35.58% across all datasets and average gains of 0.96 dB in compression performance in terms of PSNR. This underscores the effectiveness of integrating deep learning technologies into traditional video compression frameworks. Furthermore, our method outperforms other NVC algorithms [3], [4], [5], [6], [7], [8], [9], [10], indicating

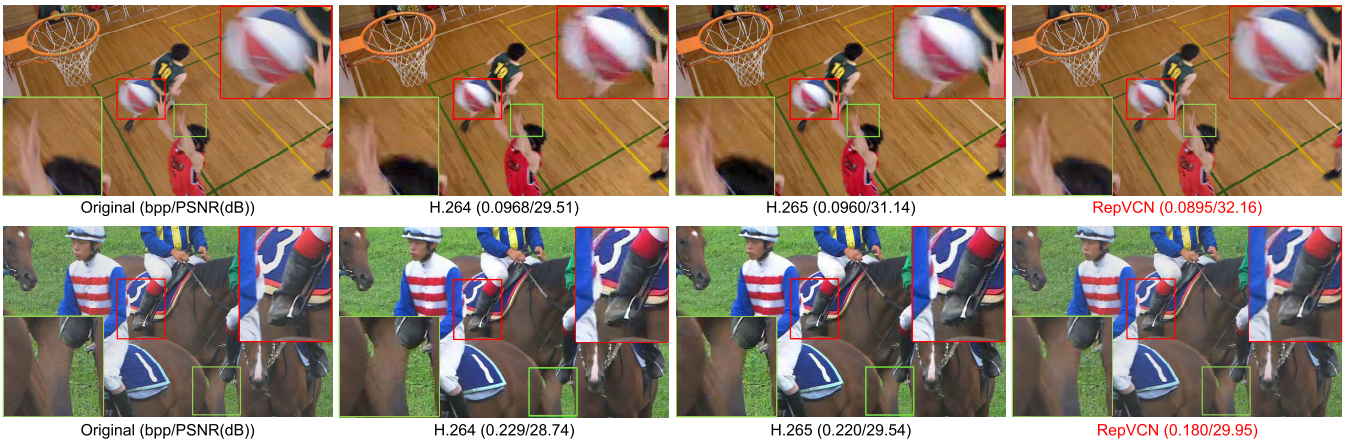

Fig. 9. Decoded video frames of H.264/AVC, H.265/HEVC, and our proposed method. Zoom-in view for better visualization.

that the proposed reparameterization method can efficiently enhance the model capacity of RepVCN.

Fig. 8 illustrates the RD-curves based on PSNR and MS-SSIM metrics. We train RepVCN with $\lambda = 2048$ and $\lambda = 64$ to obtain PSNR and MS-SSIM scores (indicated by the dots in the top right corner) at the highest bit rate. Then, only QMLs and IQMLs are fine-tuned to generate the other result dots shown in Fig. 8. It is evident that our method outperforms other algorithms [1], [2], [3], [4], [5], [6], [7], [8], [9] in RD performance, and these algorithms fine-tune the whole NVC models with different λ. Specifically, RepVCN achieves approximately 0.6 dB gain over MLVC [7] at 0.13 bpp on the UVG dataset in PSNR, with similar trends observed in MS-SSIM. Thus, our design consumes the lowest bit number and training costs at the same compression quality.

4) Visualization of the Decoded Video Frames: We present a visual quality comparison with widely used H.264/AVC [1] and H.265/HEVC [2] in Fig. 9. RepVCN demonstrates superior video quality and objective metrics at lower compression rates, exhibiting significant improvements. For example, in the "Basketball Drill" sequence, which features a moving basketball and hands, our approach achieves noticeably better texture reconstruction, less noise, and fewer blocky artifacts, particularly around motion edges. This highlights RepVCN's ability to extract realistic multiscale texture details.

5) Ablation Analysis: To validate the effectiveness of our reparameterization, mask-sharing pruning, and FXP quantization methods, we present some ablation analyses by evaluating the RD performance under different settings on the UVG dataset. As shown in Fig. 10(a), the pruned RepVCN effectively maintains the compression quality of the dense version with negligible PSNR loss. Following quantization, the FXP version achieves compression efficiency comparable to that of the FP version, facilitating the deployment of RepVCN in various resource-constrained environments.

As shown in Fig. 10(b), the effectiveness of the proposed reparameterization strategy is verified. The blue curve represents the RD performance of the baseline, which substitutes RFConvs and RFDeConvs with standard Convs and DeConvs in our RepVCN. The reparameterization strategy enables RepVCN to extract multiscale motion and residual representations, achieving superior video compression quality by constructing split-bypass topologies.

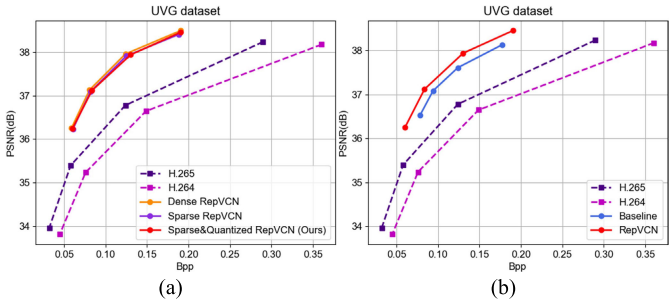

Fig. 10. RD-curves of different settings for RepVCN. (a) Comparisons of two float-point variants (dense and sparse versions) and our proposed methods (sparse and quantized RepVCN). (b) Comparisons of the baseline (directly trained simplified version) and our proposed methods.

We further evaluate the effectiveness of the proposed series of optimization algorithms from the perspective of the tradeoff between computational complexity and compression efficiency. Notably, given that the primary goal is to optimize and deploy the NVC decoder, we present the number of operations (GMACs) at each stage of the RepVCN decoder and its impact on compression efficiency under various settings. As illustrated in Fig. 11, the baseline model represents a dense and complex RepVCN constructed from multibranch Conv and DeConv blocks. By applying a linear combination of these multibranch Conv and DeConv blocks, along with fast algorithms, the overall computational complexity is reduced by 40.7% and 68.2%, respectively, while maintaining the BDBR. In addition, pruning the weights in the transform domain reduces the computational load from 38.48 to 22.36 GMACs. Despite an 81.5% reduction in computational load, the compression efficiency only decreases by 2%. Finally, we perform FXP quantization on the latest model, setting the bit widths of weights and activations to 16 and 12 bits, respectively, resulting in a BDBR loss of merely 0.4%.

6) Impact of GoP Size: Since the GoP size significantly affects the alternation frequency between intraframe and interframe coding, we conduct extensive experiments on the HEVC dataset to explore its impact, as shown in Table II. For the HEVC Class C and Class D datasets, an increase in GoP size leads to a monotonically increasing BDBR(%), indicating that a larger GoP size requires a higher bit rate to maintain comparable video reconstruction quality to that of a smaller GoP size. For the HEVC Class B and Class E

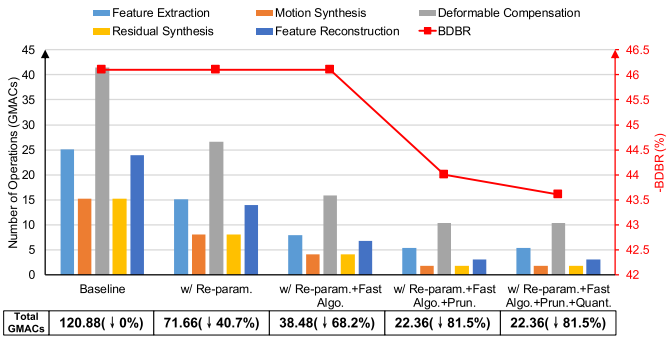

Fig. 11. Number of operations (GMACs) for the RepVCN decoder and BDBR of different settings on the UVG dataset.

TABLE II

BDBR (PSNR) RESULTS OF DIFFERENT GOP SIZE, WHERE THE GOP SIZE OF 10 IS EMPLOYED AS ANCHOR

HEVC Dataset	GoP=5	GoP=10	GoP=20	GoP=25	GoP=50
Class B	6.68%	**0.00%**	0.25%	1.60%	5.17%
Class C	**-1.19%**	0.00%	9.78%	13.97%	28.19%
Class D	**-1.60%**	0.00%	9.94%	13.61%	25.62%
Class E	16.67%	0.00%	**-0.13%**	2.00%	11.3%

TABLE III

BDBR (PSNR) RESULTS OF REPVCN DECODER WHEN WEIGHTS AND ACTIVATIONS ARE QUANTIZED TO DIFFERENT BITS: H.265 AS ANCHOR

	FP	FXP				
	W32A32	W8A8	W12A8	W12A12	W16A12	W16A16
BDBR (%)	-44.10	-37.01	-41.58	-43.04	-43.64	-43.72
Raise (%)	0.00	↑7.09	↑2.52	↑1.06	↑0.46	↑0.38

datasets, increasing the GoP size from the default value to 20 has minimal impact on BDBR, with a slight improvement observed in the HEVC Class E dataset. However, further increasing the GoP size does not yield additional benefits. This is because the increase in GoP size intensifies error propagation, necessitating a reasonable range for the GoP size setting. The high-resolution videos in HEVC Class B and Class E datasets introduce substantial contextual information and are robust to interframe predictive errors, making them suitable for larger GoP sizes.

7) Influence of Different Quantization Precision: As presented in Table III, quantizing both weights and activations to 16 bits (W16A16) leads to only a slight increase (↑0.38%) in BDBR compared to the full-precision model. However, when both weights and activations are quantized to 8 bits (W8A8), there is a sharp increase of ↑7.09% in BDBR. This highlights the high quantization sensitivity of low-level vision models which reconstruct each pixel's contents. Further exploration of various bit width settings reveals that W16A12 can achieve compression efficiency comparable to W16A16.

C. Hardware Evaluation

1) Evaluation Metrics: To provide a comprehensive comparison with existing DNN accelerators, several essential metrics that reflect hardware performance are thoroughly considered, including total power (W), throughput (GOPS and Mpixels/s), area (mm^2), frame rate (FPS), and latency (ms). For a fair comparison, we normalize the throughput by area efficiency (GOPS/mm^2) and energy efficiency (GOPS/W and Mpixels/W).

TABLE IV

PERFORMANCE COMPARISON OF VCNPU VERSUS CPU AND GPU IMPLEMENTATIONS. THE IMAGE RESOLUTION IS 1920 × 1080

	CPU	GPU	VCNPU
Platform	Intel i9-9900X	NVIDIA RTX 3090	TSMC 28 nm
Frequency(MHz)	3500	1700	400
Model		RepVCN	
Percision (W-A bits)	FP 32-32	FP 32-32	FXP 16-12
Frame Rate (FPS)	10.42	27.25	38.69
Power (W)	121.21	257.12	1.28
Throughput (GOPS)	1493	3905	5544
Energy Efficiency (Mpixels/W)	0.11	0.25	188.05
Energy Efficiency (GOPS/W)	12.32	15.19	4331.25

2) Comparison With CPU and GPU: To evaluate the potential of VCNPU for efficient neural video decoding, we first compare it with a widely used commercial GPU (NVIDIA RTX 3090) and a CPU (Intel i9-9900X). Both implementations are built on the PyTorch framework. In addition, we use the NVIDIA System Management Interface to monitor the average power consumption of the GPU. We collect the average decoding time of a test sequence constructed from 600 video frames in the UVG dataset [37] using RepVCN. As shown in Table IV, when decoding 1080P videos, our design achieves the highest throughput of 5544 GOPS and a frame rate of 38 FPS, while consuming significantly less power than other implementations. Compared to the commercial GPU, VCNPU achieves a 1.4× speedup when processing video data at the same resolution.

3) Comparison With Related ASIC-Based Accelerators: To the best of our knowledge, this is the first dedicated ASIC-based accelerator for NVC based on algorithm-hardware co-design. The layout and detailed specifications are presented in Fig. 12. While existing research has deployed interframe NVC on Snapdragon 8 chips [31] and FPGAs [43], these studies did not specifically optimize hardware architectures and dataflow, and the lack of hardware performance metrics complicates fair comparisons. Furthermore, existing ASIC-based DNN accelerators are unsuitable for running NVC models due to complex data dependencies, diverse operations, extensive off-chip data communication, and real-time processing requirements of HD video streams. Thus, we provide general comparisons with several well-known DNN accelerators focused on image and video processing tasks [14], [15], [20], [25], [26], [27], [42]. As shown in Table V, compared to the video processor [42], VCNPU improves throughput (Mpixels/s) by 66.9×, throughput (GOPS) by 6.7×, area efficiency by 2.9×, and energy efficiency (GOPS/W) by 4×. In addition, when compared to classical image processors, our design also outperforms them in terms of throughput and area efficiency. Liu et al. [20] achieved dual-side sparsity and dynamic zero skipping for a simple image classification task. Rao et al. [27] employed a lightweight DNN (e.g., nine-layer MLP) for image reconstruction at a resolution of 256 × 256 with low bit width. Thus, the relative simplicity of those tasks makes [20], [27]

TABLE V
GENERAL COMPARISON WITH OTHER RELATED ACCELERATORS. AREA AND ENERGY RESULTS ARE ALL SCALED TO 28 nm TECHNOLOGY [41]

	Task	Process (nm)	Freq. (MHz)	Percision (W-A bits)	SRAM (KB)	#MAC Unit	Throughput (Mpixels/s)	Throughput (GOPS)	Area Effi. (GOPS/mm^2)	Energy Effi. (GOPS/W)
TCAS-I'20 [15]	Image Processor	65	200	FXP 8-8	220.5	240	53.4	96	76	1066.7
JETCAS'20 [25]		65	200	FXP 8-16	572	1152	65.9	232.1	78.1	2578.9
TCAS-I'22 [14]		28	700	FXP 16-16	480	576	-	403	187.4	2160
ISCAS'22 [26]		40	600	-	102	1260	124.4	-	-	-
TCSVT'23 [20]		55	200	FXP 8-8	116	144	-	541	462.4	5410
TCAS-I'23 [27]		40	400	FXP 10-16	1720	4160	5.2	3328	492.3	5942.9
TCAD'22 [42]	Video Processor	65	800	FXP 16-16	512	256	3.6	833	162.1	1096.1
VCNPU		28	400	FXP 16-12	554	4080	240.7	5544	472.7	4331.3

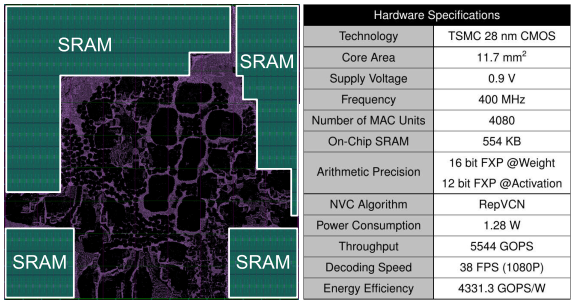

Fig. 12. Layout and hardware specification of the proposed VCNPU.

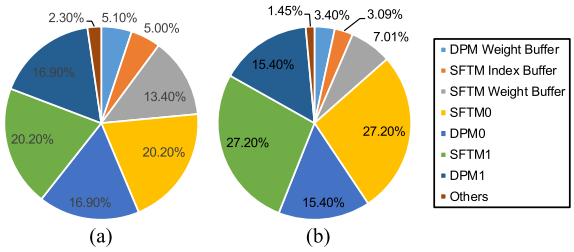

Fig. 13. (a) Area breakdown. (b) Power consumption breakdown.

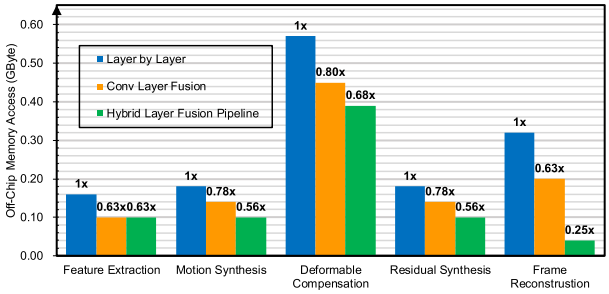

Fig. 14. Comparison of off-chip memory access by different dataflow.

more energy-efficient than our design. However, the advantage in throughput enables VCNPU to exhibit exceptional energy efficiency. Besides, [25] introduced a selective caching-based Conv fusion strategy to reduce off-chip interactions, at the cost of consuming 473.6 KB of SRAM to store as many weights as possible from target networks. [42] utilized MAC units for learning-based decoding of a few reference frames, while traditional decoding methods are executed by logic units for abundant predicted frames stored in SRAM. In contrast, our VCNPU not only reduces off-chip data communication and enhances processing speed, but also significantly conserves on-chip memory resources.

4) Area and Power Breakdown: We provide an area and power consumption breakdown in Fig. 13 to enhance the understanding of our hardware architecture. The VCNPU consists of two NVC Cores, with the majority of area and power consumptions attributed to the SFTM and DPM. Each SFTM, designed for executing sparse RFConvs and RFDeConvs, occupies 20.20% area and 27.20% power. Each DPM, responsible for DfConvs in deformable compensation, brings 16.90% area and 15.40% power consumption.

5) Off-Chip Data Communication of Different Dataflow: To validate the effectiveness of the hybrid layer fusion pipeline, we present a comparison of off-chip memory access (GByte) for different components in the RepVCN decoder. As shown in Fig. 14, the hybrid layer fusion pipeline reduces off-chip memory access for all stages of the RepVCN decoder when processed by our VCNPU. Specifically, compared to the layer-by-layer and Conv layer fusion dataflow [21], our design achieves overall reductions of 45.4% and 25.2% in off-chip memory access, respectively. This indicates that despite the diverse operations and complex topologies of RepVCN, we construct the most common operations in the network into a combination. By aligning the loop dimensions and hardware design variables for heterogeneous operations through the tile-matching rule, we significantly minimize off-chip interactions of intermediate results in the combination, which is highly advantageous for high-resolution video processing scenarios.

6) Processing Speed: To compare the compression efficiency and decoding speed of different video compression designs, we report the results for 1920×1080 video sequences from the UVG dataset [37] in Fig. 15. We directly run FFmpeg tools for the traditional video codecs for H.264 [1] and H.265 [2] on an Intel i9-9900X CPU platform. Our VCNPU operates at $1.3\times$ faster than H.265 while obtaining outstanding visual quality results. Although the low complexity of H.264 allows for the highest decoding speed, its compression efficiency does not reach a satisfactory level. Besides, Fig. 15 shows that our VCNPU is $7.7\times$ faster than the second-fastest NVC design [9], while reducing the BDBR by 30%.

We also present the latency (ms) and frame rate (FPS) when processing different datasets. Each dataset features varying video resolutions while influencing the processing speed

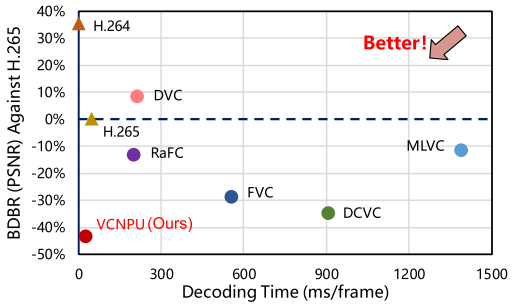

Fig. 15. BDBR (PSNR) and decoding time on the UVG dataset.

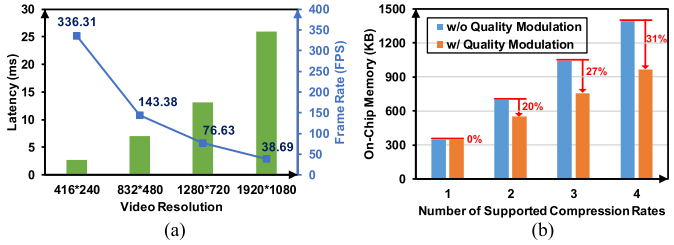

Fig. 16. (a) Frame rates of different video resolutions. (b) Validation of the effectiveness of variable compression rates.

of the VCNPU. As shown in Fig. 16(a), when processing 1920 × 1080 video datasets such as UVG [37], HEVC Class B [2], and MCL-JCV [38], our VCNPU achieves a latency of 25.9 ms and a real-time processing speed of 38.69 FPS.

7) Performance of Variable Compression Rates: Finally, we analyze the effectiveness of video quality modulation in VCNPU for supporting variable compression rates from the perspective of memory resource overhead. RepVCN modulates video quality at different compression rates by fine-tuning QML and IQML while freezing other layers. VCNPU takes advantage of this to decode a specific video frame at P different compression rates using P NVC cores. Extensive sharing of network parameters significantly reduces memory footprints for weights and indexes, albeit with a slight computational cost for executing IQML. As shown in Fig. 16(b), our baseline comparison method does not involve quality modulation; instead, the entire network is retrained separately for different compression rates, resulting in multiple sets of parameters. This approach not only incurs high training costs on the cloud side but also increases resource consumption in VCNPU for storing numerous model parameters. In contrast, our VCNPU reduces memory requirements by over 20% when decoding videos at multiple qualities ($P \geq 2$).

VI. Conclusion

In this work, we have developed and validated an efficient accelerator for NVC. At the algorithmic level, a novel RepVCN constructed from RFConvs and RFDeConvs, along with a mask-sharing pruning strategy are proposed to improve NVC performance and reduce algorithmic complexity. At the hardware level, a sparse computing core and a hybrid layer fusion pipeline are developed to flexibly support sparse RFConvs and RFDeConvs, significantly reducing off-chip data communication. Through the co-optimization of computation and communication, our VCNPU is constructed and implemented using TSMC 28 nm CMOS technology. Experimental results demonstrate that our RepVCN provides superior compression quality compared to other methods, while VCNPU achieves significantly better hardware efficiency than prior image and video processors.

References

[1] G. B. T. Wiegand, G. J. Sullivan, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, "DVC: An end-to-end deep video compression framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10998–11007.

[4] G. Lu, X. Zhang, W. Ouyang, L. Chen, Z. Gao, and D. Xu, "An end-to-end learning framework for video compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3292–3308, Oct. 2021.

[5] Z. Hu, G. Lu, and D. Xu, "FVC: A new framework towards deep video compression in feature space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 1502–1511.

[6] G. Lu et al., "Content adaptive and error propagation aware deep video compression," in *Proc. 16th Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, Aug. 2020, pp. 456–472.

[7] J. Lin, D. Liu, H. Li, and F. Wu, "M-LVC: Multiple frames prediction for learned video compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 3543–3551.

[8] R. Yang, F. Mentzer, L. Van Gool, and R. Timofte, "Learning for video compression with hierarchical quality and recurrent enhancement," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6627–6636.

[9] Z. Hu, Z. Chen, D. Xu, G. Lu, W. Ouyang, and S. Gu, "Improving deep video compression by resolution-adaptive flow coding," in *Proc. Eur. Conf. Comput. Vis.*, Nov. 2020, pp. 193–209.

[10] J. Li, B. Li, and Y. Lu, "Deep contextual video compression," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Sep. 2021, pp. 18114–18125.

[11] X. Sheng, J. Li, B. Li, L. Li, D. Liu, and Y. Lu, "Temporal context mining for learned video compression," *IEEE Trans. Multimedia*, vol. 25, pp. 7311–7322, 2022.

[12] F. Mentzer et al., "VCT: A video compression transformer," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Jan. 2022, pp. 1–19.

[13] J. Liu et al., "Conditional entropy coding for efficient video compression," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Nov. 2020, pp. 453–468.

[14] Z. Shao et al., "Memory-efficient CNN accelerator based on interlayer feature map compression," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 2, pp. 668–681, Feb. 2022.

[15] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: An energy-efficient dilated and transposed convolutional neural network processor for region of interest based image segmentation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3471–3483, Oct. 2020.

[16] L. Gonzalez-Carabarin, I. A. M. Huijben, B. Veeling, A. Schmid, and R. J. G. van Sloun, "Dynamic probabilistic pruning: A general framework for hardware-constrained pruning at different granularities," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 733–744, Jun. 2022.

[17] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[18] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4013–4021.

[19] X. Ding, Y. Guo, G. Ding, and J. Han, "ACNet: Strengthening the kernel skeletons for powerful CNN via asymmetric convolution blocks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1911–1920.

[20] M. Liu, C. Zhou, S. Qiu, Y. He, and H. Jiao, "CNN accelerator at the edge with adaptive zero skipping and sparsity-driven data flow," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 12, pp. 7084–7095, Dec. 2023.

[21] G. Li, Z. Liu, F. Li, and J. Cheng, "Block convolution: Toward memory-efficient inference of large-scale CNNs on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 5, pp. 1436–1447, May 2022.

[22] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 1, pp. 281–295, Jan. 2020.

[23] W. Mao, P. Yang, and Z. Wang, "FTA-GAN: A computation-efficient accelerator for GANs with fast transformation algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 6, pp. 2978–2992, Sep. 2023.

[24] S. Zhang, W. Mao, and Z. Wang, "An efficient accelerator based on lightweight deformable 3D-CNN for video super-resolution," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 70, no. 6, pp. 2384–2397, Jun. 2023.

[25] J. Lee, J. Lee, and H.-J. Yoo, "SRNPU: An energy-efficient CNN-based super-resolution processor with tile-based selective super-resolution in mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 320–334, Sep. 2020.

[26] A.-J. Huang, K.-C. Hsu, and T.-S. Chang, "A real time super resolution accelerator with tilted layer fusion," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 2665–2669.

[27] C. Rao, Q. Wu, P. Zhou, J. Yu, Y. Zhang, and X. Lou, "An energy-efficient accelerator for medical image reconstruction from implicit neural representation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 4, pp. 1625–1638, Apr. 2023.

[28] X. Wang, C. Wang, J. Cao, L. Gong, and X. Zhou, "WinoNN: Optimizing FPGA-based convolutional neural network accelerators using sparse Winograd algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4290–4302, Nov. 2020.

[29] J.-W. Chang, S. Ahn, K.-W. Kang, and S.-J. Kang, "Towards design methodology of efficient fast algorithms for accelerating generative adversarial networks on FPGAs," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 283–288.

[30] Z. Hu and D. Xu, "Complexity-guided slimmable decoder for efficient deep video compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 14358–14367.

[31] H. Le et al., "MobileCodec: Neural inter-frame video compression on mobile devices," in *Proc. 13th ACM Multimedia Syst. Conf.*, Jun. 2022, pp. 324–330.

[32] R. Andri, B. Bussolino, A. Cipolletta, L. Cavigelli, and Z. Wang, "Going further with Winograd convolutions: Tap-wise quantization for efficient inference on 4×4 tiles," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 582–598.

[33] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Jan. 2018, pp. 1–10.

[34] D. Wu, X. Fan, W. Cao, and L. Wang, "SWM: A high-performance sparse-winograd matrix multiplication CNN accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 936–949, May 2021.

[35] R. Zou, C. Song, and Z. Zhang, "The devil is in the details: Window-based attention for image compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 17471–17480.

[36] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, "Video enhancement with task-oriented flow," *Int. J. Comput. Vis.*, vol. 127, no. 8, pp. 1106–1125, Feb. 2019.

[37] A. Mercat, M. Viitanen, and J. Vanne, "UVG dataset: 50/120fps 4K sequences for video codec analysis and development," in *Proc. 11th ACM Multimedia Syst. Conf.*, May 2020, pp. 297–302.

[38] H. Wang et al., "MCL-JCV: A JND-based H.264/AVC video quality assessment dataset," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2016, pp. 1509–1513.

[39] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Proc. 37th Asilomar Conf. Signals, Syst. Comput.*, vol. 2, Nov. 2003, pp. 1398–1402.

[40] G. Bjontegaard, "Calculation of average psnr differences between rd-curves," in *Proc. VCEG Meeting (ITU-T SG16 Q. 6)*, 2001, pp. 2–4.

[41] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. JSSC-9, no. 5, pp. 256–268, Oct. 1974.

[42] Y. Wang, Y. Wang, H. Li, and X. Li, "An efficient deep learning accelerator architecture for compressed video analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 9, pp. 2808–2820, Sep. 2022.

[43] C. Jia, X. Hang, W. Liu, S. Wang, and S. Ma, "FPX-NVC: An FPGA-accelerated P-frame based neural video coding system," in *Proc. IEEE Int. Conf. Vis. Commun. Image Process. (VCIP)*, Dec. 2022, p. 1.

Siyu Zhang received the M.S. degree in information and communication engineering from Harbin Institute of Technology, Harbin, China, in 2021. He is currently working toward the Ph.D. degree in information and communication engineering with Nanjing University, Nanjing, China.

His current research interests include deep learning model acceleration and algorithm-hardware co-design for image/video processing.

Mr. Zhang received the IEEE Computer Society Annual Symposium on VLSI 2022 Best Paper Award.

Wendong Mao (Member, IEEE) received the B.S. degree in information engineering from Jilin University, Changchun, China, in 2018, and the Ph.D. degree in information and communication engineering from Nanjing University, Nanjing, China, in 2023.

She was a Visiting Student at the Wangxuan Institute of Computer Technology, Peking University, Beijing, China, in 2019. She is currently an Assistant Professor with the College of Integrated Circuits, Sun Yat-sen University, Shenzhen, China. Her current research interests include image/video processing algorithms, 3-D vision, and VLSI design for deep learning.

Dr. Mao received the IEEE Computer Society Annual Symposium on VLSI 2022 Best Paper Award. She also serves as the reviewer of various journals and conferences, including IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and the IEEE International Symposium on Circuits and Systems.

Zhongfeng Wang (Fellow, IEEE) received the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2000.

He has been working for many years at Nanjing University, Nanjing, China, as a Distinguished Professor, since 2016. Previously, he was at Broadcom Corporation, San Jose, CA, USA, from 2007 to 2016, as a Technical Director. Even earlier, he was at National Semiconductor Corporation, Santa Clara, CA, USA, and Oregon State University, Corvallis, OR, USA, successively. He is currently a Professor and the Head of the School of Integrated Circuits, Sun Yat-sen University, Shenzhen, China. His current research interests are in the area of optimized VLSI design for digital communications and deep learning.

Dr. Wang is a fellow of AAIA. He is a World-Recognized Expert on Low-Power/High-Speed VLSI Design for Signal Processing Systems. He has published over 400 technical papers with numerous best paper awards received from IEEE major conferences and transactions, including the Circuits and Systems Society (CASS) VLSI Transactions Best Paper Award in 2007. He has filed over 100 patent applications with tens of them granted in either USA or China. In the past, he had deeply participated in many commercial chip designs. In addition, he has served as an Associate Editor for multiple transactions of IEEE. Meanwhile, he has contributed significantly to the industrial standards. So far, his technical proposals and key ideas have been adopted by more than 20 international networking standards.