Received 6 September 2023, accepted 29 October 2023, date of publication 2 November 2023, date of current version 9 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329717

APPLIED RESEARCH

Speed Up VVC Intra-Coding by Learned Models and Feature Statistics

JIANN-JONE CHEN[1], **(Member, IEEE), YEH-GUAN CHOU**[2], **AND CHI-SHIUN JIANG**[3]
[1]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan
[2]Sunplus Tech Company Ltd., Hsinchu 30076, Taiwan
[3]Telecommunication Laboratories, Chunghwa Telecom Company Ltd., Taoyuan 326402, Taiwan

Corresponding author: Jiann-Jone Chen (jjchen@mail.ntust.edu.tw)

This work was supported in part by the Taiwan National Science and Technology Council (NSTC) under Grant NSTC 112-2221-E-011-113.

ABSTRACT The newest video coding standard, Versatile Video Coding (VVC), adopts a QTMT, quad-tree plus multi-type tree (MTT), block partition structure and improves the compression performance by about 30%∼50%, compared with the previous High-Efficiency Video Coding (HEVC) one, at the cost of higher time complexity. To make practical video communication applications feasible, we have to reduce the high time complexity resulting from an exhaustive rate-distortion optimization (RDO) search procedure. We proposed to predict Coding Unit (CU) split modes by a learned model whose input comprises neighboring line pixels and quantization parameters (QP). In addition, we set thresholds based on statistical image features and coding behaviors to eliminate unnecessary coding operations in critical coding control modules. Experiments showed that, compared with the default VVC intra-coding, the proposed method saves 46.73% of encoding time, with Bjøntegaard Delta Bit Rate (BDBR) increment of 1.16%. After retraining the learned model with a specified QP, the time reduction rate can achieve 51.79%, and the BDBR slightly increases to 2.07%. The proposed speedup coding scheme effectively reduced the VVC time complexity to a large extent.

INDEX TERMS Versatile video coding (H.266/VVC), coding unit split mode, convolutional neural networks, intra-coding acceleration.

I. INTRODUCTION

A. 266/VVC

The H.266/VVC [1] outperforms the HEVC/H.265 [2] 30%∼50% in compression efficiency. It supports 8K video coding and upgrades the service quality of applications like augmented reality (AR), immersive virtual reality (VR), and 360° video. However, the VVC encoding complexity is 31 times HEVC [3]. It needs to reduce the VVC coding complexity for practical applications. The VVC intra-frame coding time is longer than the inter-frame but can provide a high-quality reconstructed reference frame for all others in one GOP. We focus on designing speedup coding methods for the VVC intra-frame coder. VVC adopts a more flexible QTMT coding mode to yield better compression performances than the previous QTBT. In addition, several powerful coding tools are used in the intra prediction (IP) process to improve further the coding performance, such that the time complexity increases. The CCLM method assumes that reconstructed luma pixel values can help predict chroma ones through a linear model so that compressing chroma samples can be more efficient. Previous research [4] shows that different VVC test models (VTM) spend 80%∼90% of time processing luma samples under all-intra (AI) coding mode. We proposed to reduce the encoding time complexity for luma pixels under acceptable quality degradation.

To utilize spatial correlation, the VVC refers to reconstructed neighboring CUs to perform IP and encodes the residual by entropy coding methods. For a $CU_{W\times H}$, the encoder will refer to W pixels from the upper and the upper-right block, respectively, and the left H pixels are obtained in a similar way, as shown in Fig. 1, in which there are 2×(W+H)+1 reference pixels in total. The angle intra-prediction (AIP) has been increased from 35 in HEVC

The associate editor coordinating the review of this manuscript and approving it for publication was Gangyi Jiang.

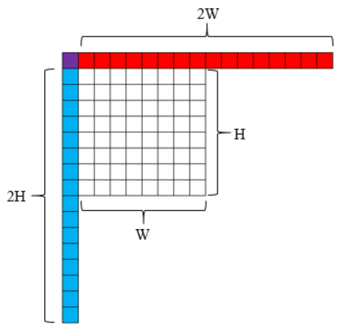


FIGURE 1. $2\times(W+H)+1$ reference pixels from neighboring blocks.

to 67 in VVC to better fit image textures. Due to variable CU shapes in the MTT mode, VVC provides 28 wide AIP (WAIP) modes [5] for unsymmetric angle prediction. With the enhanced AIP, the most probable mode (MPM) in HEVC is increased to 6 in VVC to maintain good MPM performance. In addition to the enhanced coding mode, VVC adopts multiple reference lines (MRL) [6], [7], Matrix weighted intra prediction (MIP) [8], [9], Intra Sub-Partitions (ISP) [10], and 4-tap interpolation filter to improve coding performance. Extending the line-based IP mode, an adaptive intra-subpartition method [11] improves the coding efficiency based on fixed number subpartitions, which yields 0.95% of bitrate savings.

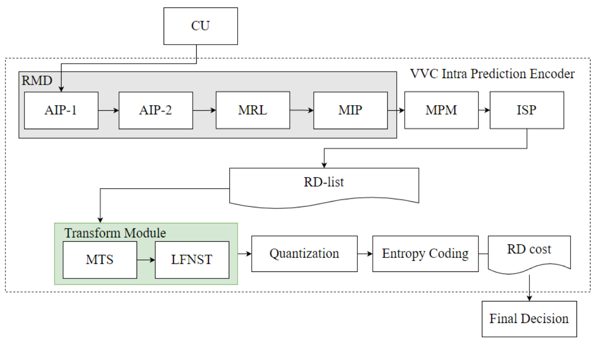


FIGURE 2. Intra Prediction processing flow modified from [12].

The IP coding control is shown in Fig. 2, modified from concurrent processing [12] to sequential. When reference pixels are ready for a CU, the coder will find 2∼3 best, depending on CU size, from the 35 prediction angles of HEVC, and then test the left and right to the best ones to refine the prediction. Whether to use MRL and MIP depends on the CU size and its location. If MPM does not comprise the selected prediction angle, the candidate list will include this selected one together with the MPM, and the ISP mode if available. The best mode can be determined by selecting the one with the minimum rate-distortion cost, RDcost, among modes in the candidate list can be obtained after performing a multiple transform selection (MTS), a low-frequency non-separable transform (LFNST) [13], and entropy coding.

The high time complexity to yield the best RDcost (RDC) when encoding a CU can be reduced by only calculating the RDC of the final candidate mode selected through a low complex rough mode decision (RMD) process. The RMD adds the bitrates with the summed–p absolute transform difference (SATD) after applying a Hadamard transform to the residual:

$$RMD = STAD + \lambda \times bitrate_{mode}, \quad (1)$$

in which λ varies according to different QPs. The RDC is calculated by adding the CU coding error, measured by sum of square differences (SSD), with a specific mode, and the bitrate scaled by the λ:

$$RDC = SSD + \lambda \times bitrate. \quad (2)$$

B. NEURAL NETWORK MODEL

Convolutional neural network (CNN) has been proven efficient in image processing/classification applications. For specific applications, the deep learning process helps find the optimal convolution kernels which act as sliding windows to extract features from an input image yield an output feature map [14]. Activation functions enable CNNs solving non-linear problems, such as image pixels with non-linear relationships. Different types of activation functions, such as Sigmoid linear unit (SiLU), rectified linear unit (ReLU), and leaky rectified linear unit (LReLU), are used by different applications. We adopted ReLU and LogSoftmax defined as:

$$ReLU(x_i) = max(0, x_i) \text{ and} \quad (3)$$

$$\hat{y}_i = LogSoftmax(x_i) = \log\left(\frac{exp(x_i)}{\Sigma_j^K exp(x_j)}\right), \quad (4)$$

where x_i denotes the i-th element of a K-dimensional vector, x, which is the output of a feedforward neural network, $\hat{y}_i$ denotes the LogSoftmax output for x_i. The pooling layer helps reduce feature map size to speed up processing without losing important features. It can also eliminate CNN learning models from over-fitting. Max-pooling, mean-pooling, and stochastic pooling are commonly used methods. The stride of pooling is usually fixed. To make input and output consistent in size, the adaptive pooling layer in Pytorch [15] can learn to find the best stride that is usually fixed.

C. RATE-DISTORTION OPTIMIZED MODE

Both HEVC and VVC adopt a tree-like block partition structure for the encoder to search exhaustively for an optimal coding mode for one CU^i, denoted as M^i_{opt}. VVC adopts a 128×128 coding tree unit (CTU). Fig. 3 shows the six block-split modes of the QTMT coding structure. As shown, the MTT partition mode consists of a vertical binary tree (VBT), horizontal binary tree (HBT), vertical ternary tree (VTT), and horizontal ternary tree (HTT). Fig. 4 demonstrates a practical QTMT coding tree structure and the corresponding CTU (128×128) partition example. As shown, the QTMT enables very flexible block partition structures to compress a CTU in a rate-distortion optimization (RDO) way. However, it needs exhaustive searching for the RDO procedure to find the best CU block split mode.

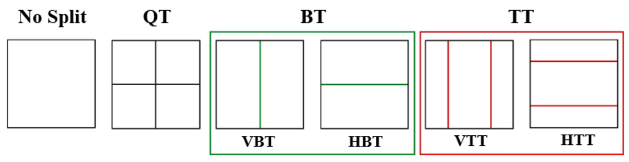


FIGURE 3. The six CU split modes of QTMT [12].

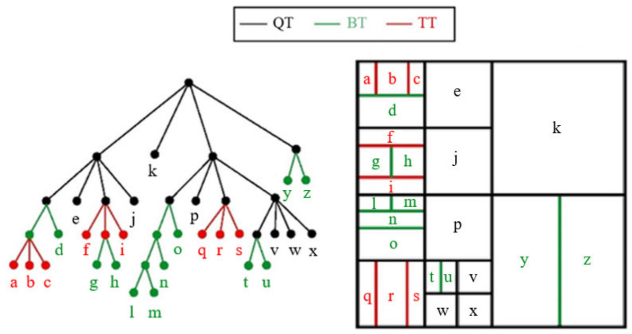


FIGURE 4. Examples of VVC encoding tree (left) and QTMT partition on a $CU_{128\times128}$ (right), modified from [12].

To serve variable bitrate (VBR) coding, the coder sets different QP values to yield quality bitstreams. If one designs a learned model to serve the CU mode prediction under one QP it can help the coder provide fixed-quality streams with fewer operations. However, as the coding performance evaluation specifies four QPs, it generally needs to train four models to achieve highly accurate prediction. Li et al. [16] proposed to add a QP Mask layer in the model to distribute the QP to all pixels. Experiments revealed that adopting one learned model to predict CU modes under different QPs is difficult, as the CU mode will change drastically under different QPs. Fig. 9 shows that the CU mode becomes complicated when setting a smaller QP compared to a larger one. In this research, we study how to design a single model to accurately predict CU mode under different QPs.

Reducing the VVC encoding time complexity while keeping good coding quality is the primary topic discussed in this paper. Based on literature studies and experimental studies of the VVC coder, we proposed to design CNN learning models to predict CU split modes to eliminate possible trivial RDO operations, In addition, we set up threshold values based on statistical features of CU samples to terminate the exhaustively searching RDO procedure early. In what follows, Section III discussed the design of the QTMT mode prediction model. The proposed speedup coding scheme is described in Section IV. Section V is the experimental study. Section VI concludes this paper.

II. CU CODING MODE PREDICTION

A. CU SAMPLE UNIT

To train a CNN model, the system has to provide samples with uniformly distributed labels to avoid overfitting. Five videos, **A**_*PeopleOnStreet*(ClassA PeopleOnStreet), **B**_*BasketballDrive*, **C**_*RaceHorses*, **D**_*BasketballPass*, and **E**_*FourPeople* are selected from the list of HEVC common test conditions (CTC) [17] for training, part of which is the same as the VVC CTC. To avoid adopting similar datasets for training and testing, we selected from each video 10 frames which are 8 frames apart. In addition, to increase database diversity, we additionally selected images from a CPIV database (database for the CU Partition of Intra-mode VVC) [16] to collect training samples from videos with various resolutions and diverse scene contents. Videos composed of independent still images can provide more diversified and detailed image content, other than those dependent ones that may lead to blurred regions. Fig. 5 shows both fast-moving balls and black shoes marked by red squares look vague.

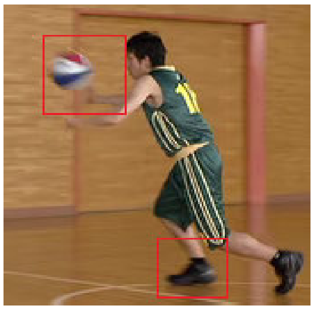


FIGURE 5. Blurred regions of moving objects in video.

To solve the problems of data insufficiency and non-uniformity in training sample collection, we can perform data augmentation operations, such as rotation, noise addition, or partial cutting on data samples. However, data augmentation methods are not suitable to process intra-coded CU samples [18] in that the coder references neighboring block pixels for prediction. The coding mode of one CU would be different after flipping and rotation operations, i.e., one CU may comprise different labels. The model learning process cannot converge under non-coherent label data. For example, the $CU_{64\times64}$ enclosed by a red square in Fig. 6(a) demonstrates totally different partition types from the horizontally flipped counterpart shown in Fig. 6(b), in which the blue-square encloses a $CU_{32\times32}$ whose partition type has been changed from VBT to NO. It's obvious that the coder utilizes different neighboring blocks to perform intra-coding for this CU and yields different coding modes. The model training should comprise neighboring block pixels to predict the intra-coded CU mode. To enable data augmentation for intra-coded CU samples, we designed the model input additionally comprising CU neighboring pixels so that the same CU can provide different label data under different neighboring pixels. Adopting this design, we can construct a training database with uniformly distributed CU labels. Rotating CU samples with 90°, 180°, and 270°, together with their reference pixels fetched from neighboring blocks in rotated images, we obtained a 4 times larger dataset.

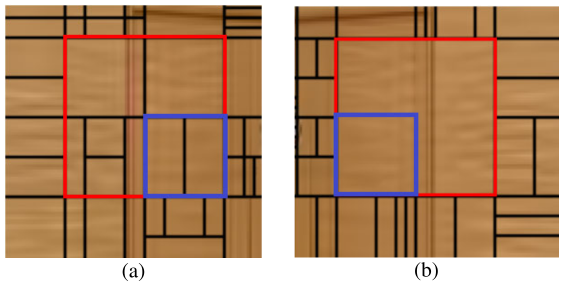


FIGURE 6. Non-coherent optimal coding modes before and after horizontal flipping on a CU (blue box): (a) Original M_{opt}; (b) The M_{opt} after horizontal flipping.

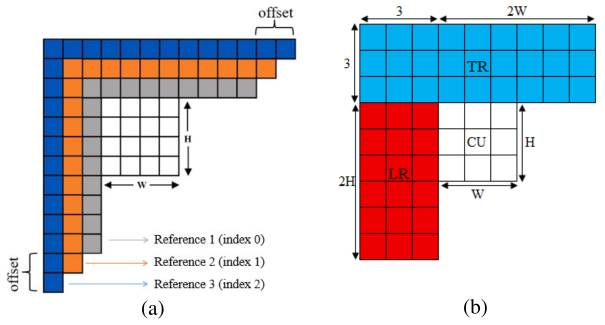


FIGURE 7. Reference pixels: (a) Original MRL Samples; (b) MRL=*TR* ∪ *LR* for model training.

B. CODING MODE HANDLING

The CPIV database program was designed to find only the M^i_{opt}, comprising all CU partition labels, RDcosts for all existing CUs during the RDO process, and can provide as much as 10^9 such training samples. In addition to collecting just the M_{opt}s, we intend to acquire sub-optimal ones that had been exploited during the RDO process. For example, if the ground-truth optimal split mode of a $CU^i_{32\times32}$ is m^i_{opt}=NO, only the label 'NO' can be acquired for training. During the RDO process, all block split modes and their sub-CUs utilized their corresponding reference pixels, and these local optimal coding modes can be included for training as the learning model has to predict split modes of various CU sizes. However, these collected label data are not uniformly distributed because the encoder will not split smaller CUs and only allow QT mode for square CUs. Under this condition, more NO label samples will be acquired than QT. It needs to preprocess the training samples after data augmentation to yield a database with uniformly distributed split modes.

Previous works utilized CU textures and QP to train models to predict the m_{opt} for a CU [18]. However, VVC references neighboring block pixels to perform IP and calculates all RDCs to determine the optimal CU mode. The optimal mode of a CU^i, $m^i_{opt} = m_{opt}(CU^i)$, depends on neighboring pixels, or multiple reference lines (MRL), and can be expressed as $m^i_{opt}(CU^i, MRL^i)$. The MRL should be included in a training sample to comply with the VVC coding procedure. By including neighboring block information [19] for model training, the prediction outperforms that without [20]. However, it needs additional storage and computation loading to train a model. VVC refers to MRLs for intra-prediction and the CNN extracts features from the MRLs to yield better prediction, as shown in Fig. 7(a). Note that the offset pixels are not included for training because they are padded from inner line segments, not from reconstructed images. The referred MRL for model training is shown in Fig. 7(b).

C. DATA STATISTICS

Designing a single CNN model to learn the highly complicated QTMT block split procedure under different QPs is challenging. The CNN model learns general decision rules from training samples with its capacity. As the RDO procedure will test numerous block split modes for the optimal one, it's difficult for the model to yield precise predictions. Although the model helps eliminate exhaustive search, a predicted sub-optimal CU split mode would lead to increased RDCs. To avoid relying only on the learning CNN model, we proposed to find other decision rules in the coding procedure from experiments to early terminate (ET) the RDO process. Two features, standard deviation of pixel values in a CU^i, $\sigma_i = \sigma(CU^i)$, and the RDC to encode a CU, are extracted in the coding procedure to determine whether to ET the RDO process.

The σ_i measures the texture homogeneity of a CU^i, based on which trivial RDO operations can be eliminated if it satisfies the condition, $\sigma_i < T_\sigma$, where T_σ is a pre-determined threshold. As the encoder splits blocks recursively in the QTMT mode, it has to set different T_σs under different QPs and CU split depths comprise those for QT, BT, MT, and average depths. Experiments revealed that, according to the InterQuartile Range (IQR), σ_i would be larger for larger or square CUs under the same depth. When the CU size is larger than 256 × 256, σ_i becomes un-sensitive. Experiments also showed that when (W>H), σ_i is roughly 1.08 times larger. Formulate the above observations as:

$$\hat{T}_\sigma = \sqrt{scale(W \times H) \cdot \left(\frac{QP}{92}\right)} \cdot \log_{10}(depth) \cdot \alpha_{depth}, \quad (5)$$

and

$$T_\sigma = \begin{cases} \hat{T}_\sigma \cdot 1.08, & \text{if } W > H \\ \hat{T}_\sigma/10, & \text{if CU is specific size} \\ \hat{T}_\sigma, & \text{otherwise.} \end{cases} \quad (6)$$

where $scale(Num) = \min(128, Num)$, depth $= \text{QT}_{depth} + \text{MT}_{depth}$, and $\alpha_{depth} = 1.5^{depth}$, all of which have been verified from repetitively extensive experiments.

D. EARLY TERMINATION

For a $CU_{64\times64}$, only QT and NO modes are applicable. When QP is small, it's likely that QT would be applied on a $CU_{64\times64}$. Under this condition, the coding controller can utilize the RDC available in the normal coding procedure to early terminate or skip the coding process through proper thresholds [21]. In comparisons, both CNN model inference and σ_i calculation require extra computations, but the RDC would be available at the corresponding processing stage.

The distributions of split and non-split modes along different RDCs are shown in Fig. 8. As shown, the distribution of NO resembles normal and the speedup control benefits from correctly determining a non-split CU. Experiments revealed that a good compromise can be reached if the threshold is selected such that 60% of NO mode decisions can be covered. Although setting this threshold to determine NO modes would lead to false positive (FP) decision results, i.e., red plots inside the blue ones, the partition depth of optimal modes should be small under this FP condition, i.e., RDC $\leq$ T$_{RDC}$, and the resultant RDC will not deviate too much from the optimal one. Instead of utilizing the Bayes rule to add up a rectified σ. we set the threshold value as:

$$T_{RDC} = \mu_{RDC} + 0.3 \cdot \sigma_{RDC}, \tag{7}$$

which adds 0.3 times of σ to the mean RDC value,

In the formal VTM system, the encoder will calculate the non-split mode RDC of a CU for the next module, *xCheckModeSplit*, to find the optimal coding mode. Our speedup coding scheme (SCS) utilized this ready RDC to check whether RDC $\leq T_{RDC}$ to decide early termination or not, This decision for speedup is different from that using the σ_i. VVC will first infer all possible split modes before the RDO procedure. For example, six split modes are allowed for a CU$_{32\times32}$, The decision condition by T_σ can help turn off some split-mode tests to reduce RDO operations. Using the T_{RDC} can further eliminate some RDO operations after the formal IP procedure. By utilizing the two decision conditions for speedup coding, the probability of burst bitrate increase due to false prediction can be reduced.

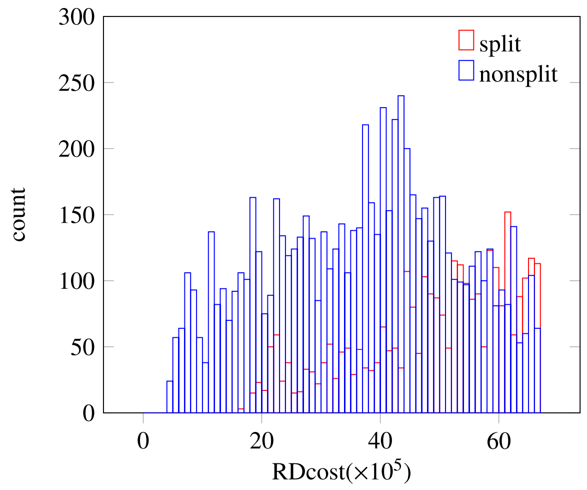


FIGURE 8. Distributions of split and non-split modes along different RDCs.

III. CU MODE PREDICTION

Previous works [21] designed and trained a CNN to predict CU partition modes for speedup coding. Although it achieves good speedup performance under specific conditions, how to design the model and its training strategy to deliver cost-effective speedup performance is still challenging. In this section, we will describe our proposed CNN-based speedup control scheme, training strategy, and fast split-mode decision method.

A. MODEL FRAMEWORK AND TRAINING

Developing an RDO procedure to be non-exhaustive helps reduce the time complexity. However, QTMT is highly complex and difficult to predict precisely. For example, when one CU demonstrates horizontally splittable texture patterns, such as flat edges, we may expect the coder to split the CU by HBT or HTT structure for a lower RDC. However, the optimal coding mode may not be coherent with this subjective evaluation in that the best QTMT mode would be determined based on QP, reference pixels, the parent-block split mode, and so on, which will lead to quite different split modes. Fig. 9 shows that: when VVC quantizes this 128 $\times$ 128 CTU with different QPs, 22, 27, 32, and 37, corresponding final split modes would be different. The CU$_{32\times32}$ block marked with a red frame comprises primarily vertical edges. When QP = 22 and 27, the coder splits the subCU vertically. However, it splits the subCU horizontally when QP = 32 and 37. This example shows that the QTMT split mode is not determined solely by image textures. In addition, the split CU/subCU is not limited to square so it needs specific models for different CU sizes and width/height (W/H) ratios to achieve precise prediction.

This multi-prediction scheme is efficient but with high space and time complexity. To reduce complexity, we can normalize the size of a CU block to act as the input of one learning model. However, preliminary experiments showed that CU texture details would change under different aspect ratios and lead to biased model learning. Li [16] proposed to predict VVC CU modes based on this scheme and designed 19 different learning models for different CU sizes and ratios. To simplify model training, we utilized adaptive pooling to process CUs with different sizes/ratios and yield a fixed-size feature map. As described in section I-B, when the pooling layer resizes the feature map, it can reserve stable features even if some pixel value deviation occurs to achieve the target of utilizing one fixed-size output feature map for different-size inputs. We call this proposed prediction scheme Adaptive CU CNN (ACUCNN).

B. ADAPTIVE CNN

As the QTMT RDO procedure is an exhaustively searching process, it will lead to many possible final CU split modes. How to design a CNN model for accurate CU mode prediction is challenging. In terms of model learning, resizing a CU image block directly before being a CNN input will lead to a biased feature map. For the QTMT structure, the maximum CU size is 128 $\times$ 128, which would be split by QT. The split mode decision process starts from a CU$_{64\times64}$, which generally comprises not so many image details. Imposing

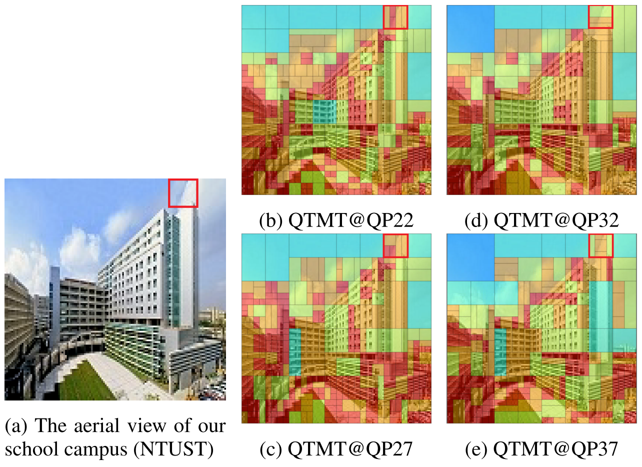


FIGURE 9. Practical QTMT M^i_{opt}s under different QPs: (a) Original image; (b)-(e) QTMT split results when QP = 22, 27, 32, and 37.

pixel interpolation or decimation on this $CU_{64\times64}$ will add extra or lost critical image features and lead to inaccurate model prediction. The adaptive pooling helps eliminate inaccurate model prediction when processing different-size CUs. Previous works [19], [20] showed that adaptive pooling helps improve the prediction accuracy of determining CU split or not. Investigating the adaptive pooling layer reveals: (1) Direct CU resizing is a kind of low-dimensional processing that is operated on one-channel gray pixels or three-channel color pixels; (2) The adaptive pooling performs feature resizing and acts as high-dimensional processing based on multi-channel extracted feature maps, which can pull and reserve original image features after the layer-by-layer deep learning process. Their works justified that feature-based resizing for a model to predict the CU coding mode is better than pixel-based in handling different-size CUs.

Although the MRL consists of three-pixel lines, the IP procedure will reference only one. However, the learning model requires enough image texture to learn the correlation between one CU^i and MRL^i, and three-pixel lines meet the minimum width requirement. Note that the MRL does not include offset pixels because only particular IP directions will reference them. For example, in the IP process, the required reference pixels are not enclosed by a regular rectangle, as shown in Fig. 7(a). Either omitting extra pixels to make reference pixels forming a rectangle block or padding required offset pixels cannot help extract useful features. Chang's work [7] revealed that offset pixels are utilized in the diagonal direction from the lower-left corner to the upper-right, and the one in the reverse direction, i.e., intra-prediction mode (IPM) mode2 and mode66, respectively. They affect only a few IPM modes and can be obtained by extrapolation from neighboring pixels. We proposed to omit these extra pixels for efficient model learning, as shown in Fig. 7(b). To reduce the CNN processing complexity, the upper-left and upper pixel blocks are combined as the top reference (TR), i.e., reference pixels comprise those in the TR and the left reference lines (LR), as shown in Fig. 7(b). The reference pixel samples are the same as in VVC. For the CU in the upper-left corner of one image, the MRL comprises no valid data, and the average value of the CU replaces them to make the model works for all CUs.

Fig. 10 shows the framework of the proposed ACUCNN whose input comprises six parameters, i.e., CU, TR, LR, QP, and (W, H). The first three are image contents provided for feature extraction, the QP Mask module distributes the QP to all pixels, and (W, H) provides the corresponding CU/subCU size information for the model. The Multi-Input Adaptive CNN (MIA-CNN) takes pixels after the QP Mask module as input, each of which will transform pixel values to different scales of feature maps which concatenates the (W, H) as input for the Fully Connected (FC) layers to learn to predict CU split mode. The QP controls the video coding bitrate. As the QP would affect the CU coding mode, it is considered as one input to the FC layer in the prediction model [18], [20], [22]. Others proposed designing independent prediction models for different QPs, increasing the full model size. The QP Mask [16] can act as a miniature model to embed the QP to every pixel before the standard model processing, such that we can design one model to provide accurate CU mode prediction under different QPs. This QP mask normalizes an input QP such that the training process can converge quickly [16].

In the ACUCNN framework, the MIA-CNN dominates the prediction performance. Fig. 11 shows that the MIA-CNN input comprises CU, TR, and LR pixels. Each MIA-CNN can extract image features and resize feature maps through adaptive pooling layers, such that CU, TR, and LR feature map sizes are equal for concatenation. Denote this concatenated feature map as $\mathbb{F}_{concate}$, following which a CNN layer extracts inter-feature correlation inside. We considered that the dominant image feature for the model should be the CU, and the other two are assistants, so the CU dimension is two times TR and LR. In addition to extracting high dimensional features, the $\mathbb{F}_{concate}$ helps reduce model parameters for quick training convergence.

The two MIA-CNN modules provide two different size feature maps, i.e., 32×32 and 16×16, which are concatenated, together with the (W, H) parameters for the FC layers to perform classification. Reasons why selecting 32×32 and 16×16 are: (1) VVC allows only NO and QT modes for a $CU_{64\times64}$ and enables the QTMT from $CU_{32\times32}$, so the maximum allowable QTMT CU size is reserved. Other smaller square CUs will be upscaled to $CU_{32\times32}$s. Although extra features may be generated/extracted from an upscaled $CU_{32\times32}$, it will not miss important features at this scale; (2) All possible QTMT split modes are allowed at a $CU_{32\times32}$ whose scale comprises more critical features for block partitions than others. It can avoid extracting wrong features and improve the model prediction/classification accuracy. Previous works [19], [20] proposed to scale all smaller CUs with rectangle or square shapes to $CU_{32\times32}$ and

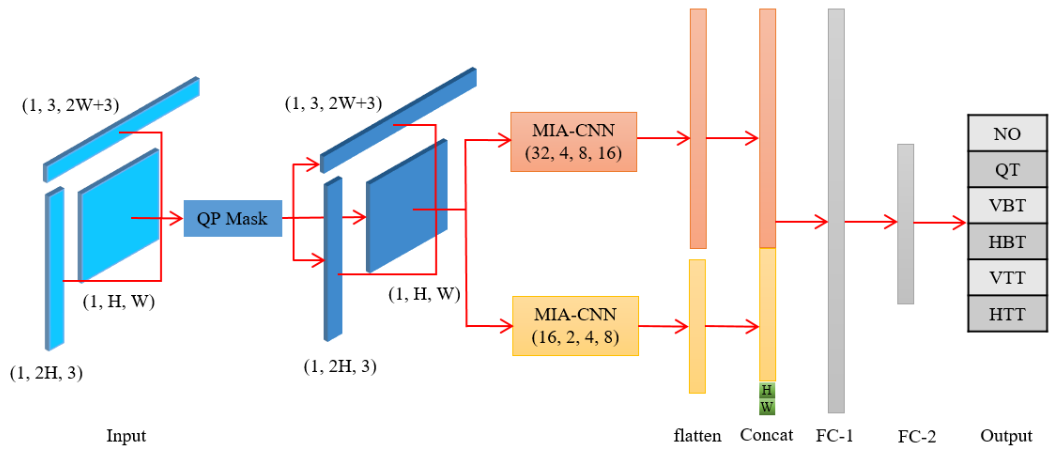


FIGURE 10. **The ACUCNN system framework.**

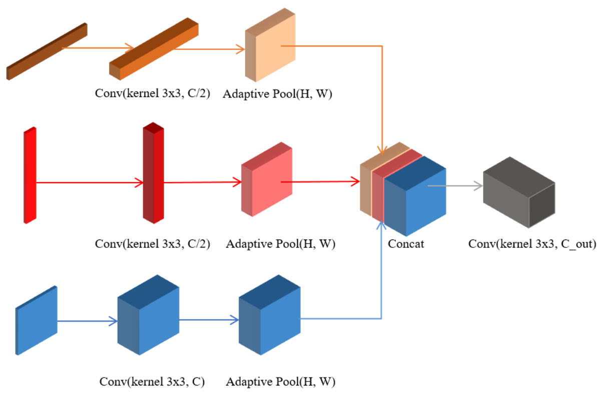


FIGURE 11. **The MIA-CNN framework.**

then to CU$_{16\times16}$ before passing to the CNN and the FC layers. Both works designed prediction models to determine whether to split a CU to bypass the RDO procedure for the NO mode. Regarding network architecture, we adopted a broad neural network model based on the Inception V1 [23] instead of the deeper [19], [20]. Adopting a broad network framework for the MIA-CNN, features with different orientations can be extracted at each layer. It allows the network to extract different scale/sparsity features for model adaptability, and a feature concatenation module provides non-linear attributes. As one CU$_{32\times32}$ represents only a tiny part of an image, it will not comprise very complicated image textures. Experiments showed that when adopting only one MIA-CNN for CU$_{32\times32}$ in the ACUCNN, the model training process yields only 50% of accuracy. The training can converge well by adding another MIA-CNN for CU$_{16\times16}$.

C. MODEL TRAINING

In section II-B, we discussed how to collect and preprocess the dataset for the training model. We selected five videos from the HEVC CTC list. To avoid selecting duplicate videos, we chose eighty frames with ten frames apart from each video. For database augmentation, we rotate frames of these five videos by 90°, 180°, and 270°, respectively, to obtain a four times larger database. Allowing data augmentation in this manner is the most crucial feature of the proposed model because its input consists of CU neighboring pixels, which validates data augmentation. In addition, the CPIV database, composed of individual images, is also included for training, and the training database comprises no duplicated videos and can be fully utilized. From this database, we can obtain several hundred million CUs, from which we can re-select and categorize to provide more uniformly distributed training samples in terms of tags for efficient model training. We randomly pick/discard CUs in this dataset so each category/split-mode contains 200,000 samples. The QTMT mode specifies six split modes for a CU$_{32\times32}$, and the VTM specifies four QPs in general. Under this setup, the training dataset comprises 4.8 million CU samples. During model training, the prediction accuracy will reach nearly 70%, and the loss goes stable after 100$\sim$200 epochs, no matter what the optimizer is, i.e., SGD, RMSprop, or Adam. We selected the Adam optimizer because it helps the training process converge quickly. In addition, we set both learning rate (l_r) and regularization (l_2) to 10^{-4}.

In addition to adopting the Negative log-likelihood Loss (L_{NLL}) for multi-classification applications, the loss function also comprises a modified RDC loss function [16], denoted as L_{RDC}. When the CCS performs RDO operations on the CPIV database, the RDC of a split mode not yet tested is set to be MAX_DOUBLE. For example, it will skip the QT test if it had performed MT for speedup. This skip operation will set the default RDC for QT to MAX_DOUBLE, which means the encoded bitrate of this CU by QT would be very large. However, this setting will lead to exploding gradients during the backpropagation (BP) process and prevent the model training from convergence. In model training, the MAX_DOUBLE is set to ten times the maximum RDC of a CU so that the BP process works efficiently. When combining both loss functions, the L_{RDC} is scaled by a β to match the range of L_{NLL}. The L_{NLL} is calculated by:

$$L_{NLL} = -\frac{1}{N}\sum_{n=1}^{N} y_{k(n)} \cdot \log(\hat{y}_{k(n)}), \tag{8}$$

where N is the batch size, $y = (y_1, y_2, \cdots, y_K)^T$ is the one-hot encoding ground-truth label vector, $k(n)$ is the index of correct label of the n-th sample and $y_{k(n)}$=1, $\hat{y}_{k(n)}$ denotes the corresponding output value of $LogSoftMax(x_{k(n)})$, i.e., the confidence score of the inferred coding mode $k(n)$.

Our experiments indicated that the L_{NLL} is far less than the RDC found to be larger than 10^5. It needs to scale the L_{RDC} to be comparable with the L_{NLL} to prevent the latter from being trivial. The scaling operation is achieved by dividing all the RDC loss values, l_ns, with the maximum one in that batch, i.e., $\max\{l_n\}_{n=1,2,\cdots,N}$, and then multiplying each one with the L_{NLL}, as shown in eq. 9:

$$l_n = \left(\frac{r_n}{r_{min}} - 1\right), \text{ and}$$
$$L_{RDC} = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{l_n}{l_{N,max} + \varepsilon}\right) \cdot L_{NLL}, \tag{9}$$

where r_n denotes the practical RDC corresponding to the inferred coding mode for the n-th sample, r_{min} is the smallest one among all RDCs when encoding a CU^n, l_n is the loss corresponding to an RDC for the n-th sample, $l_{N,max}$ is the maximum one among all l_ns in a batch. When the model prediction accuracy increases, it is possible that all the inferred coding modes are true optimal, and the $l_{N,max}$ will be 0 under this condition. We add a small value, $\varepsilon = 10^{-20}$, to avoid dividing by zero. Combine L_{NLL} and the L_{RDC} with a weight β set to be 0.8 for balance.

For most classification applications, model prediction accuracy is a rule of thumb when designing and training a learning model, and the system can do nothing for a wrong classification result. However, things are different when utilizing a multi-classification model to perform video coding mode prediction. If the predicted coding mode for one CU^i happens to be the worst, it will enlarge the RDC vastly. Nevertheless, the RDO procedure will optimally split the entire CU based on the predicted split mode to yield the smallest RDC, i.e., a constrained optimal CU mode. This observation suggests that when the model cannot guarantee the inference is m_i^{opt} for one CU^i, it should try to yield sub-optimal. It is essential to include the loss, L_{RDC}, for the learning model to yield a sub-optimal coding mode when it fails to yield the optimal one. The loss function is:

$$L = L_{NLL} + \beta \cdot L_{RDC}. \tag{10}$$

During the QTMT speedup coding process, the single model has to determine split modes for CUs with different sizes under different QPs. Denote the set of different CU sizes as $Z = \{z_j\}$ and different quantization parameters as $Q = \{q_i\}$ for easy description. To train such a unified model, we design the learning process to repeat selecting one q_i randomly and selecting one z_j randomly and sequentially under a fixed q_i. When it finishes retrieving all CU samples with size z_j quantized with q_i for training, i.e., repeats randomly selecting one CU with size z_j under a fixed q_i in one Batch, the learning process moves on to the next Batch, i.e., randomly select one q_i again. Although we designed the unified model to predict split modes of different-size CUs, we keep the size of CU samples unchanged in each batch, under which random selection helps avoid overfitting. We describe the model training process by a program-like algorithm below, in which the outermost while loop acts as an epoch and the innermost as a batch:

Algorithm Control Steps of Model Training in One Epoch

```
Input:
  CU($z_j, q_i$): the set of CU samples with size $z_j$ and quantized with $q_i$
  DB$_{CU}$={$CU(z_j, q_i)$}: the entire training dataset
Result: A learned model trained by $DB_{CU}$

Model training steps:
  while $Q \neq \emptyset$ do
      randomly select one $q_i \in Q$;
      $Q = Q \setminus q_i$.
      $Z_t = Z$;
      while $Z_t \neq \emptyset$ do
          randomly select one $z_i \in Z_t$;
          $Z_t = Z_t \setminus z_i$;
          while $CU(z_j, q_i) \neq \emptyset$ do
              randomly select one $CU_k \in CU(z_j, q_i)$;
              $CU(z_j, q_i) = CU(z_j, q_i) \setminus CU_k$;
              train the model with $CU_k$;
          end while
      end while
  end while
```

IV. SPEEDUP CODING SCHEME

In the VVC coding framework, processing blocks are functionally connected and dependent. We can design decision functions at suitable processing points and utilize prediction models to achieve good speedup performances. The overall speedup coding scheme, abbreviated as **ACUCNN_SCS**, is first introduced. The decision function and the thresholding policy are then presented. The ACUCNN model is discussed at last.

A. ACUCNN_SCS CONTROL FLOW

In VVC, a *xCompressCU* function performs a coding mode decision that decides whether the CU is to be Intra- or Inter-coded. In AI coding, the coder must determine whether to split the CU. If it determines NO split, it will perform the IP procedure with this CU size to yield the RDC. Otherwise, it will execute the *xCheckModeSplit* function to split the CU into subCUs with allowable modes and complete the xCompressCU function for each subCU to check its RDC. When no allowable splitting mode exists, the encoder traces back all subCU RDCs to select the optimal one. The QTMT coding mode is executed through this time-consuming exhaustive search process to determine the m^i_{opt}, in which the *xCompressCU* and the *xCheckModeSplit* functions call each other iteratively.

Fig. 12 shows the control flow of the proposed ACUCNN_SCS. In default VVC configuration, *encoder_intra_vtm*, the maximum CU size that can perform QTMT coding is $|CU_{32\times32}| = 32 \times 32$. The system calculates the σ_i and compares it with the predefined threshold, T_σ. If a CU^i demonstrates simple or smooth image texture, it needs no further splitting, and the default VTM IP coding procedure would process the CU. Otherwise, if the CU^i is square, i.e., W==H, the coder will utilize the ACUCNN to predict the split mode such that the VTM system only needs to perform RDO operations for fewer candidate modes. Following the coding control track, when the CU^i is determined to NoSplit, the VTM uses the xCheckRDCostIntra to perform IP coding to yield the RDC. If ($RDC<T_{RDC}$), it needs no further splitting. Otherwise, the VTM executes the xCheckModeSplit procedure before determining the best mode.

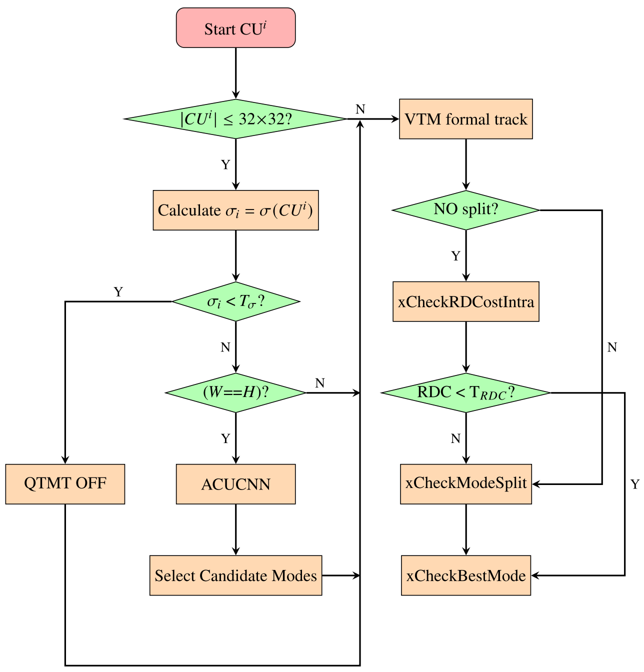


FIGURE 12. **The control flow of the proposed ACUCNN_SCS.**

B. THRESHOLDING POLICY

In the proposed fast encoding method, we set up two threshold values, i.e., T_σ for σ_i and T_{RDC} for RDC, to skip unnecessary RDO operations. In addition, the ACUCNN helps reduce the candidate modes. In designing these speedup modules, we have to consider their efficiencies and limitations at the same time. For example, when we emphasize too much speedup coding efficiency, the BDBR will increase badly, i.e., severe quality degradation. In addition, we still have to handle the tradeoff between speedup efficiency and the extra time complexity of executing these modules. Not all modules constantly contribute appreciable time-savings. For example, when one module determines to split the CU, it must perform RDO operations for all possible split modes, including those operations for all corresponding subCUs. Although the encoder may save the IP operation time for this CU size, the speedup efficiency may be degraded due to the extra module execution time. This example tells us that we must carefully justify the tradeoff between speedup efficiency and performance degradation and select proper threshold values from experiments.

The T_σ can only be applied to CUs with sizes smaller than 32×32. In VVC, the QTMT mode can apply on CUs with sizes smaller than 32×32 under which all split modes are allowed. We design this thresholding module starting from this CU scale for efficient time-saving. For a $\text{CU}_{64\times64}$, it needs more calculation time to compute the $\sigma(\text{CU}_{64\times64})$ and may not contribute to good enough speedup efficiency. In addition, only QT and NO modes are allowed for a $\text{CU}_{64\times64}$ and the probability of selecting the NO mode becomes smaller for smaller QPs. Under this condition, although the model can correctly predict the CU to be NO split, this extra prediction time becomes trivial and leads to lower time-saving rates. The threshold, T_{RDC}, is set to determine whether to split a $\text{CU}_{64\times64}$. We can set up matching thresholds for CUs with different sizes to speed up the coding process. Unlike model prediction or other parameters, such as σ_i and edge histograms, that require extra computation increasing the cost for speedup control, the RDC for a CU of any size is available in the default coding control flow. It costs only one neglectable comparison operation to achieve the speedup control. However, using RDC alone is not efficient enough to help determine splitting or not, which contributes little to speeding up performance. Experiments revealed that adopting this threshold-based split decision method for larger $\text{CU}_{64\times64}$s yields a compromised speedup performance.

C. ACUCNN MODEL

The ACUCNN model is invoked when $\sigma_i \geq T_\sigma$. It extracts higher dimensional features to predict the CU mode in a deeper sense. Experiments showed that it could accurately predict square CUs's mode, though the model can work well for different CU shapes. When we set the ACUCNN to predict split modes of all CU shapes, take the **D**_*BasketballPass* as an example, the time-saving rate is as high as 50.92% but with 10.47% of BDBD increment. This result tells that utilizing the ACUCNN without proper constraints will conflict with the design target of the speedup coding method. The degraded video quality and slightly degraded prediction accuracy other than square CUs/subCUs lead to BDBR increment, which conflicts with the design target of the speedup coding method. For one thing, we will not appreciate the high speedup coding with severe quality degradation; for another, all split modes are allowed for a square CU, and model prediction can yield the most cost-effective time-saving performance. In addition to adopting model prediction, we proposed to set a confidence threshold, T_{conf}, and a tolerance factor,

τ, modified from a multi-thresholding method [16] to accommodate proper candidate modes to improve the quality under speedup coding. In the multi-thresholding method [16], it sets different threshold values under different block split depths to eliminate severe quality degradation of false prediction. We proposed to design one model to predict split modes for CUs with different sizes and under different QPs, based on which only one tolerance factor, τ, has to be specified. The system adjusts T_{conf} and τ concerning σ_i to prevent possible burst-increasing BDBR.

In a classification model, the softmax layer generates a confidence score, $\hat{y}_i$, for each output candidate. The system selects the candidate with the highest confidence score, $\hat{y}_k$ in eq. 11, as the final output. The role of confidence scores in a classification model resembles the accuracy during model training. When the accuracy is around 50%, the output acts as a random guess, and we can reverse the outcome when the accuracy is less than 50%. The above rules for a two-class model are not applicable to multi-classification. We set a threshold for confidence scores, T_{conf}, for the speedup scheme and the coding system to adapt to practical coding applications. Setting a higher T_{conf} will control the model to select candidates strictly and result in a lower encoding bitrate but lower speedup performances. On the contrary, it can achieve better speedup performances but a higher bitrate. Similarly, we can correctly set the τ to make a good compromisation between quality and speedup performances [16]. In a multi-classification application, the prediction result is the candidate mode k whose $\hat{y}_k$ is maximum, i.e.,

$$k = \arg\max_i \{\hat{y}_i\}_{i=1,2,\cdots,K}, \tag{11}$$

and there's un-correlated between $\hat{y}_k$ and other candidates. This uncorrelation assumption is not applicable to video coding mode prediction, in that the true M_{opt} may come from the candidate with the second or the third highest confidence score. We proposed to utilize the τ to enable fault tolerance for the model. Besides the candidate mode k, others with $\hat{y}_m > \tau \cdot \hat{y}_k$ are also included in the final candidate set, i.e.,

$$S_{opt} = \{m | \hat{y}_m \geq \tau \cdot \hat{y}_k\}. \tag{12}$$

In practical coding, adopting fault tolerance for inferred results may conflict with the purpose of setting T_{conf}, which controls the fidelity of mode prediction accuracy. Setting $\tau \leq 1$ helps increase the probability that the model output can hit the *true* optimal one. We expect the $\hat{y}_k$ to be much larger than the others. If not, we should consider candidate modes S_{opt} in the final RDO procedure.

Fig. 13 shows the bitrate and encoding time under different τs. The encoding time and bitrate are 1615.882 secs. and 115.034 KB, respectively, when encoding the **D**_*BasketballPass* with the default VTM-14 at QP = 37. As shown, setting different τs becomes trivial for larger T_{conf}s. By setting a larger T_{conf}, the model will select fewer candidate modes for the final RDO procedure, regardless of

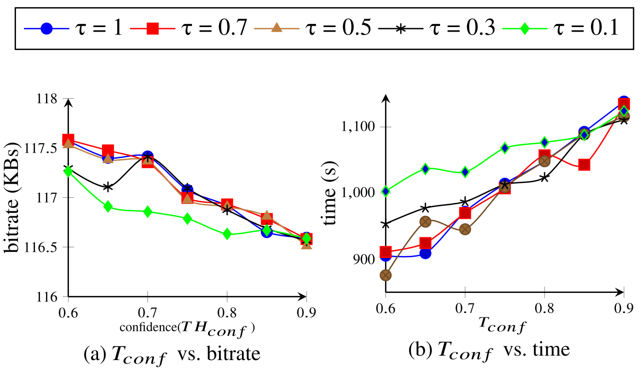


FIGURE 13. The model speedup efficiency analysis: (a) Confidence threshold (T_{conf}) vs. bitrate; (b) T_{conf} vs. time.

the τ value. We selected from experiments the T_{conf} range as [0.6, 0.7], whose actual value depends on σ_i as shown in eq. 13,

$$T_{conf} = \min\left\{0.7, \frac{(60 + \sigma_i - T_\sigma)}{100}\right\}, \tag{13}$$

under which $\tau \in [0.1, 0.5]$ can help speed up coding with tolerable errors. Eq. 13 shows that a high complex CU^i will yield a high T_{conf}, under which a smaller τ lets the coder include more split modes in the RDO procedure to avoid severe quality degradation. On the contrary, low complex CUs require fewer splitting operations, and the τ should be larger to prevent over-splitting for time-saving. Experiments demonstrated that τ can be determined based on the following equation:

$$\tau = -4 \cdot T_{conf} + 2.9. \tag{14}$$

V. EXPERIMENTAL STUDY

Table 1 is the experimental setup in which the VTM 14.0 is configured by default, excluding the QP value. The CTC [24] to verify the VVC encoding efficiency differs from HEVC. Classes **A1** and **A2** (3840 × 2160) replaced the original Class **A** (2500 × 1600) and some new videos were added in Class **B**. Test videos comprise 22 videos from Classes **A1** to **E**. BD-PSNR [25] and BDBR are used to measure the video coding performance in the rate-distortion sense. In this research, we proposed to reduce the VVC time complexity under

TABLE 1. System setup.

Hardware and VTM software Setup	
CPU	Intel Core i7-9800X @4.5GHz
RAM	128GB 2400MHz
GPU	NVIDIA GeForce RTX 2080 SUPER
Host OS	Ubuntu 20.04
Docker Image	pytorch:1.9.0-jupyterlab
VTM software	
Reference Software	VTM Version 14.0
Configuration File	encoder_intra_vtm.cfg

negligible quality degradation. Time reduction rate, R_T, for the ACUCNN_SCS is calculated to evaluate its performance:

$$R_T = \frac{1}{4}\sum_{i=1}^{|Q|}\left(\frac{t_{orig}^{q_i} - t_{speedup}^{q_i}}{t_{orig}^{q_i}}\right)\cdot 100\%, \qquad (15)$$

where $q_i \in Q = \{22, 27, 32, 37\}$, $t_{orig}^{q_i}$ and $t_{speedup}^{q_i}$ are encoding times by the VTM and the speedup method quantized with q_i, respectively, and R_T is the average of time-reduction rates for all q_is. Higher R_Ts mean higher speedup coding performances.

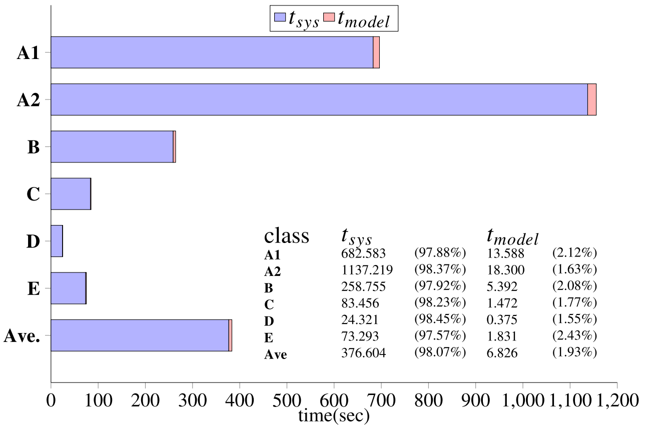
A. SPEEDUP PERFORMANCE

As we select part of training videos from HEVC CTC [17] and some of which appear in the VVC CTC [24], we mark videos with the same name in both CTCs by '*' to evaluate possible over-fitting artifacts. Table 2 shows that R_T is 46.73% with quite a small BDBR increment of 1.16%. Note that the R_T performances are not especially high for training and testing videos, such as **B**_*BasketballDrive*, **C**_*RaceHorsesC*, **D**_*BasketballPass*, and **E**_*FourPeople*, which justified the generalization capability of the learned model.

TABLE 2. Speedup performance compared with default VTM-14.

Class	Test sequence	BDBR (%)	BD-PSNR (dB)	R_T (%)
A1 3840×2160	Campfire	1.278	-0.039	46.209
	FoodMarket4	0.736	-0.024	49.924
	Tango2	0.885	-0.013	55.244
A2 3840×2160	CatRobot	1.325	-0.035	41.475
	DaylightRoad2	1.133	-0.028	45.834
	ParkRunning3	0.641	-0.038	43.765
B 1920×1080	BQTerrace	1.567	-0.091	39.021
	BasketballDrive*	1.320	-0.061	41.522
	Cactus	1.205	-0.058	40.022
	MarketPlace	0.825	-0.033	39.930
	RitualDance	1.188	-0.060	42.767
C 832×480	BQMall	1.244	-0.087	50.274
	BasketballDrill	2.131	-0.101	38.474
	PartyScene	0.720	-0.056	37.466
	RaceHorsesC*	0.809	-0.056	43.472
D 416×240	BQSquare	1.298	-0.109	51.405
	BasketballPass*	0.917	-0.064	55.121
	BlowingBubbles	0.576	-0.042	45.674
	RaceHorses	0.721	-0.049	51.262
E 1280×720	FourPeople*	1.570	-0.114	58.626
	Johnny	1.680	-0.099	57.486
	KristenAndSara	1.721	-0.119	53.031
	Average	1.159	-0.063	46.727

Regarding the ACUCNN model, as compared to previous research [18], it yields 20% higher prediction accuracy. This improved prediction accuracy helps eliminate more unnecessary RDO operations. In addition to using model prediction, the speedup coding scheme utilizes RDC and σ_i to determine early stop or skip of the CU coding. In the normal video coding process, the coder calculates the RDC to make decisions to yield the best CU coding mode, i.e., it needs no extra computations to get RDC. Besides, the calculation



class	t_{sys}		t_{model}	
A1	682.583	(97.88%)	13.588	(2.12%)
A2	1137.219	(98.37%)	18.300	(1.63%)
B	258.755	(97.92%)	5.392	(2.08%)
C	83.456	(98.23%)	1.472	(1.77%)
D	24.321	(98.45%)	0.375	(1.55%)
E	73.293	(97.57%)	1.831	(2.43%)
Ave	376.604	(98.07%)	6.826	(1.93%)

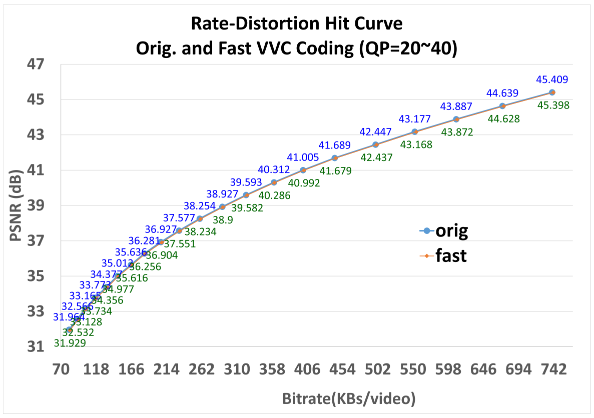
FIGURE 14. Percentage of model prediction time, t_{model}, *w.r.t.* system encoding time, t_{sys}, when encoding one frame.

time of $\sigma(\mathrm{CU}_{32\times32})$ is less than 1ms but it helps much in reducing the time complexity. In contrast, the ACUCNN model needs significantly few numerical operations to predict the coding mode of a CU. Experiments revealed that the model prediction time is less than 2% of the total encoding time. Fig. 14 shows that the percentage of model prediction time is smaller than 2% of an entire one frame's encoding time for all class videos. Aiming to design a lightweight prediction model, we increase the width and shorten the depth of the CNN model and adopt only half of the total channels for convolution layers to essentially reduce the model prediction time. Note that the ACUCNN model assists the VVC coder to speed up coding instead of dominates the coding process. The speedup performance would be better if we can reduce the model prediction frequency, as the model prediction process requires extra computations. In our design, only when the CU/SubCU is square and σ_i is larger than a threshold can it invoke the model prediction. It reduces the coder time complexity with the fewest operations.

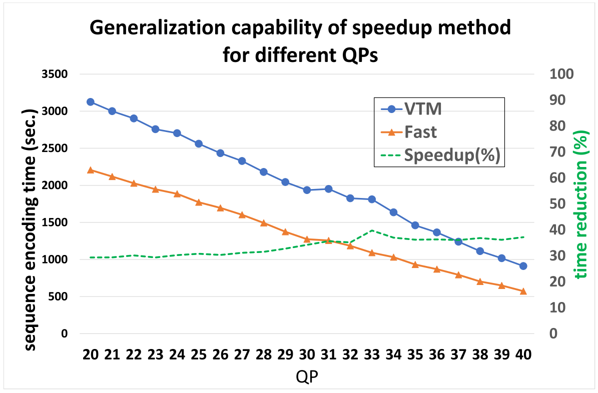
B. VBR CODING

The most prominent feature of the proposed speedup coding scheme is one model can predict CU mode under different QPs. In comparison, previous works designed individual models for different QPs to provide VBR coding. To justify our speedup coding scheme applicable in a VBR coder, we set the QP range from 20 to 40 to verify the generalization capability of our ACUCNN model. Take the **D**_*BasketBallPass*, as an example, the rate-distortion hit-curve and time-saving performance for all QPs are shown in Fig. 15. Fig. 15 (a) shows that the rate-distortion performances of the default VTM and the VTM under speedup control are nearly the same under 28.6% of time reduction rate. Fig. 15 (b) shows the encoding time of the VTM and the speedup VTM under different QPs, which demonstrates the generalization capability of the proposed speedup scheme.

However, the model's robustness to a changing QP implies generalization capability, which can not yield exact mode



(a)



(b)

FIGURE 15. Generalization capability of the proposed speedup coding in VBR coding: (a) rate-distortion plot of original and speedup VTM; (b) Generalization performance of the speedup coding method.

predictions for a specific QP. To speed up the video coding process under a specific QP, we proposed a small-scale model retraining process so that the generalized model can adequately adapt to this QP. As shown in Table 3, the retraining process improves the speedup-coding performance under the same system setup. The R_T is as high as 51.79% with BDBR increment 2.07%, which is acceptable in terms of the highly efficient VVC coding performance. Investigation into the mode prediction process revealed that the ACUCNN-fast scheme could yield more highly confident prediction outcomes, such that the coding controller can perform fewer RDO operations to find the optimal CU splitting mode. As the speedup coding scheme helps reduce the set of candidate CU splitting modes, it can yield a higher time-reduction rate, while it also leads to quality degradation, i.e., higher BDBR. How the speedup control operations affect the coding quality is elaborated with practical CU examples in appendix B. In addition, why adopting the tolerance factor τ in the proposed speedup method and its impact on video quality is explained by practical CU mode in section C.

C. COMPARISONS WITH OTHER SPEEDUP METHODS

In addition to evaluating the performance of the proposed speedup coding scheme we also compare performances with

TABLE 3. Re-trained model speedup performance.

Class	Test sequence	BDBR (%)	BD-PSNR (dB)	R_T (%)
A1 3840 × 2160	Campfire	1.982	-0.059	52.451
	FoodMarket4	1.178	-0.038	52.909
	Tango2	1.448	-0.022	59.864
A2 3840 × 2160	CatRobot	1.751	-0.046	43.267
	DaylightRoad2	1.668	-0.038	50.494
	ParkRunning3	0.788	-0.046	48.372
B 1920 × 1080	BQTerrace	2.614	-0.145	47.378
	BasketballDrive*	2.223	-0.101	50.379
	Cactus	2.030	-0.097	46.566
	MarketPlace	1.251	-0.050	49.223
	RitualDance	1.827	-0.091	49.816
C 832 × 480	BQMall	2.472	-0.173	58.548
	BasketballDrill	3.294	-0.155	50.382
	PartyScene	1.851	-0.145	49.327
	RaceHorsesC*	1.729	-0.121	50.403
D 416 × 240	BQSquare	2.907	-0.243	55.850
	BasketballPass*	2.218	-0.153	51.838
	BlowingBubbles	1.668	-0.124	53.273
	RaceHorses	1.927	-0.131	55.081
E 1280 × 720	FourPeople*	2.683	-0.193	57.099
	Johnny	2.927	-0.171	56.599
	KristenAndSara	2.999	-0.205	50.208
	Average	2.065	-0.116	51.788

previous researches. Li [16] proposed to design 19 models that require 2.9 MB space to perform speedup coding and the proposed ACUCNN model needs only 1.2 MB, 41% of Li's model size. As one robust prediction model cannot perform better than 19 smaller ones located in proper control flow locations in any case, we exploited statistically-based decision rules to help bridge this accurate prediction gap. Experiments justified that the proposed small ACUCNN model yields comparable speedup performance with that of the Li [16]. In Li's fast mode, it achieves R_T=63.79% while BDBR is as high as 3.19%. In comparison, our ACUCNN scheme can save 46.73% of encoding time with the smallest increase of BDBR=1.16% among all methods. In addition, among methods that yield $R_T \geq 50\%$, the proposed ACUCNN-fast scheme also demonstrates the best BDBR performance, which is 2.07(51.79%) as compared to that of Li's fast method 2.14(56.72%) and Yang's 2.66(54.30%).

VI. CONCLUSION

We proposed a VVC speedup encoding scheme based on a learned model to predict CU coding mode and a compromised thresholding policy to early terminate/skip in the QTMT coding procedure: (1) We designed an ACUCNN model comprising six inputs: a CU and its neighboring pixels TR and LR, CU size (W, H), and QP, to accurately predict the QTMT coding mode under different CU sizes and QPs, so that the video coding controller can essentially eliminate unnecessary RDO operations. The TR and LR enable the model to learn the intra-prediction algorithm in a deeper sense to improve prediction accuracy. In addition, the model adopts adaptive pooling to predict coding modes of different-size CUs accurately. The most prominent is that data

TABLE 4. Performance comparisons with previous works, Li [16] and Yang [26].

Class	Test sequence	ACUCNN		ACUCNN-fast		Li [16]-fast		Li [16]-medium		Yang [26]	
		BDBR (%)	R_T (%)	BDBR (%)	R_T (%)	BDBR (%)	R_T (%)	BDBR (%)	R_T (%)	BDBR (%)	R_T (%)
A1 3840×2160	Campfire	**1.28**	46.21	1.98	52.45	2.91	**59.87**	2.02	48.69	2.64	50.11
	FoodMarket4	**0.74**	49.92	1.18	52.91	1.95	52.27	1.26	42.24	5.15	**60.49**
	Tango2	**0.89**	55.24	1.45	**59.86**	2.33	49.60	1.52	40.78	1.63	45.28
A2 3840×2160	CatRobot	1.33	41.47	1.75	43.27	3.28	**55.99**	2.16	45.40	**1.13**	47.17
	DaylightRoad2	**1.13**	45.83	1.67	50.49	1.95	**59.37**	1.16	49.35	2.18	52.45
	ParkRunning3	**0.64**	43.76	0.79	48.37	1.76	**53.78**	1.15	41.68	1.25	45.58
B 1920×1080	BQTerrace	1.57	39.02	2.61	47.38	1.79	56.94	**1.11**	45.61	5.25	**58.96**
	BasketballDrive	**1.32**	41.52	2.22	50.38	2.61	**62.00**	1.64	52.03	2.19	55.07
	Cactus	1.2	40.02	2.03	46.57	1.86	60.56	**1.12**	49.30	1.32	**60.68**
	MarketPlace	0.82	39.93	1.25	49.22	1.28	**58.22**	**0.80**	46.63	4.20	57.97
	RitualDance	1.19	42.77	1.83	49.82	1.80	56.75	**1.07**	44.87	3.73	**61.93**
C 832×480	BQMall	1.24	50.27	2.47	58.55	2.05	**60.77**	**1.17**	49.76	2.84	55.04
	BasketballDrill	2.13	38.47	3.29	50.38	2.99	52.62	**1.63**	39.29	4.29	**60.09**
	PartyScene	0.72	37.47	1.85	49.33	1.16	56.44	**0.61**	45.20	2.79	**57.20**
	RaceHorsesC	**0.81**	43.47	1.73	50.40	1.61	**57.89**	0.96	46.45	2.39	54.91
D 416×240	BQSquare	1.3	51.4	2.91	**55.85**	1.33	55.16	**0.74**	44.48	1.37	51.36
	BasketballPass	**0.92**	55.12	2.22	51.84	2.35	**55.21**	1.41	44.45	1.88	54.69
	BlowingBubbles	**0.58**	45.67	1.67	53.27	1.57	**53.40**	0.92	41.56	3.17	52.42
	RaceHorses	**0.72**	51.26	1.93	**55.08**	1.88	53.31	1.20	41.64	1.19	45.47
E 1280×720	FourPeople	1.57	58.63	2.68	57.10	2.20	**59.74**	**1.33**	49.87	1.66	51.51
	Johnny	**1.68**	57.49	2.93	56.60	3.57	**57.88**	2.33	48.15	2.43	57.49
	KristenAndSara	**1.72**	53.03	3.00	50.21	2.74	**60.01**	1.76	50.47	3.78	58.85
	Average	**1.16**	46.726	2.07	51.79	2.14	**56.72**	1.32	45.81	2.66	54.30

augmentation is applicable to the original CU dataset with highly non-uniform label distribution to improve the model training and prediction efficiency. Experiments showed that this design helps increase 20% of prediction accuracy; (2) Though possessing generalization capability, the model may lead to severe quality degradation of reconstructed CU resulting from very few mode prediction errors for outlier CU samples. We proposed a compromised thresholding policy to determine early termination/skip in the QTMT coding procedure to bridge the gap between generalization and specialization of model prediction. Experiments showed that the model prediction spends 1.28% while contributing to 46.73% time saving of the total encoding time and yields only a 1.16% BDBR increment, If we retrain the model for a specific QP, the time saving can be up to 51.79% with a 2.07% BDBR increment. Future research comprises adapting these threshold values for robustness and performing the VVC intra-prediction direction prediction to improve speedup performances. In addition, how to reduce the VVC inter-frame coding complexity is also ongoing.

APPENDIX A
ACRYONYM AND SYMBOL LIST

See Table 5.

APPENDIX B
QUALITY VS. SPEEDUP ANALYSIS

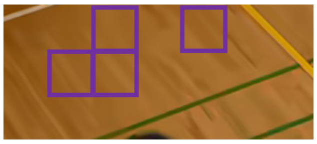
Table 2 shows that **C**_*BasketballDrill* yields a higher BDBR under acceptable speedup rates. This video was captured with a fixed camera and objects inside followed a regular moving pattern. In comparison, both **B**_*BasketballDrive* and **C**_*BQMall* were captured by moving cameras, and objects inside move in irregular ways, yet they exhibit good speedup rates. Additionally, their visual complexity is not as high as **A1**_*FoodMarket4* which has a bustling market atmosphere and smoke-filled scenes, or **C**_*PartyScene* which features children playing and bubbles floating around. These visual observations help little in explaining the poor BDBR performance. We turn to examine the split mode which can be analyzed from mode prediction and the σ_i measure. As the RDC involves complex calculations and has corresponding associations at different levels, it is difficult to analyze it based on the final split mode and excluded for analysis. Fig. 16(a) shows part of a **C**_*BasketballDrill* frame, representing the wooden floor inside the gymnasium. As shown, the four purple-box-marked $\mathrm{CU}_{32\times32}$s demonstrate no complex texture. However, Fig. 16(b) shows that only one CU kept unsplit in the ground truth split mode. In the proposed method, the four CUs kept unsplit, as shown in Fig. 16(c). Upon closer inspection, we observe that fine textures exist in these wooden floor areas. This example tells us the feature σ_i is not that efficient in measuring block textures. In addition, it does not comprise features of reference pixels. However, in most cases, these kinds of CUs are not deeply split, which helps reduce the exhaustive search time significantly.

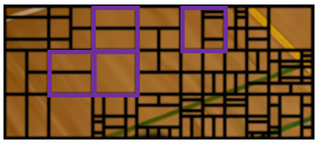
APPENDIX C
SUB-OPTIMALITY AND SELF ADJUSTMENT

Although the system performs model inference on square CUs only, inference errors can affect other CU modes on

TABLE 5. Acryonym and symbol.

Symbol	Meaning
ACUCNN	adaptive CU CNN
MRL	Multiple Reference Lines, MRL=TR∪LR
MIA-CNN	Multi-Input Adaptive CNN
RDC	rate-distortion cost
LR	Left-Row reference pixels of a CU
TR	Top-Row reference pixels of a CU
α_{depth}	scale factor of $\hat{T}_{\sigma}$ for different split depths (eq. 5)
$depth$	depth = QT$_{depth}$ + MT$_{depth}$ (eq. 5)
σ_i	standard deviation of a CU^i
σ_{RDC}	standard deviation of RDC in square CUs (eq. 7)
β	weighting for Loss Functions (eq. 10)
K	number of label classes: K=6 in the VVC application
$k(n)$	the index of the inferred mode for CU^n (eq. 8)
L_{RDC}	loss function in terms of RDC
l_n	loss corresponding to the RDC with mode $\hat{y}_{k(n)}$ (eq. 9)
$l_{N,max}$	the maximum one among all ℓs (eq. 9)
M^i_{opt}	the optimal coding mode of a CU^i
m^i_{opt}	the optimal split mode of a CU^i, $m^i_{opt} \in \{1, 2, \cdots, K\}$
R_T	time reduction rate (eq. 15)
r_m	RDC of the model predicted split mode (eq. 9)
r_{min}	the minimum RDC when splitting one CU (eq. 9)
ε	a very small value: 10^{-20} (eq. 9)
S_{opt}	the set of candidate coding modes with tolerance factor
T_{σ}	threshold of CU standard deviation (eq. 6)
T_{RDC}	threshold of RDC (eq. 7)
T_{conf}	threshold of confidence value (eq. 13)
$t^{q_i}_{orig}$	VTM encoding time with QP=q_i
$t^{q_i}_{speedup}$	speedup VTM encoding time with QP=q_i
t_{model}	the model Inference time (Fig. 14)
t_{sys}	VTM encoding time excluding t_{model} (Fig. 14)
τ	a tolerance factor (eq. 12)
μ_{σ}	mean of σ for square CUs
μ_{RDC}	average of RDC for square CUs (eq. 7)
x	feedforward network output $(x_1, x_2, \cdots, x_K)$
y	one hot encoding ground-truth vector $(y_1, y_2, \cdots, y_K)$
$\hat{y}_i$	the output of LogSoftMax(x_i) (eq. 4)
Q	the set of QPs :$\{q^i\}$ used to describe the **Algorithm**
Z	the set of CU sizes : $\{z_j\}$ used to describe the **Algorithm**


(a) The original image texture


(b) The ground-truth split mode

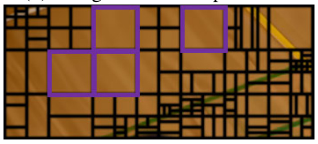
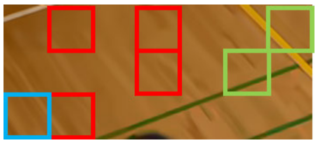
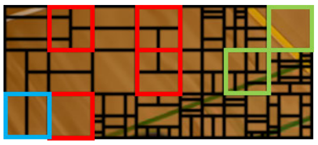

(c) The proposed method split mode

FIGURE 16. Split mode analysis: purple CUs denote $\sigma < T_{\sigma}$.

subsequent processes. In Fig. 17(a), green squares label blocks whose split modes are correctly inferred, reds show partially incorrect inference with a minor impact on the block split mode, and blue CUs are correctly inferred but subsequent coding procedures yield different outcomes. The image textures of green blocks are not overly complex, similar to others with simple textures, but a clear white line crosses the block and allows salient feature extraction. The top-right green box, if the model does not include


(a) **C**_*BasketballDrill* original image


(b) The ground-truth split mode

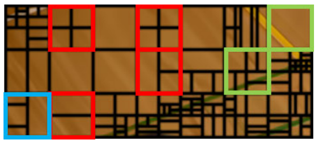

(c) The proposed method split mode

FIGURE 17. Splitmode analysis: Green box represents correct model inference; red box represents slight error; blue box represents automatic compensation by the system.

other reference information, would likely yield a BT split resulting in the line falling on one side of the CU. However, by incorporating reference pixels as input, it is possible to determine that the line is an extension from the left box and thus avoid the split. The red-boxed regions, although yielding different results, yield similar inference outcomes with the ground truth. The blue-boxed examples demonstrate how the model yields varying inferences in subsequent operations. Both the ground truth and proposed methods perform VBT on this CU. However, the ground truth does not perform HTT again on the left CU, while the proposed method does not perform mode inference on this elongated CU. Therefore, this difference arises internally in the coding system and explains why the speedup method yields different model inferences. For example, in Fig. 17(b), the top-left CU$_{32\times32}$ undergoes three HBTs, resulting in an additional segmentation line in the middle CU. As a consequence, based on this sub-optimal CU mode, it will lead to completely different CU split modes for subsequent CUs. By introducing the tolerance factor τ, the system can try different split modes. When there have been errors in the past, even if the model infers the best solution, the system may choose other CU split modes to compensate for the previous judgment errors and their associated RDC.

REFERENCES

[1] B. Bross, J. Chen, S. Liu, and Y.-K. Wang. *Versatile Video Coding Editorial Refinements on Draft 10*, document JVET-T2001, Oct. 2020.

[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] F. Pakdaman, M. A. Adelimanesh, M. Gabbouj, and M. R. Hashemi, "Complexity analysis of next-generation VVC encoding and decoding," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 3134–3138.

[4] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Complexity analysis of VVC intra coding," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 3119–3123.

[5] L. Zhao, X. Zhao, S. Liu, X. Li, J. Lainema, G. Rath, F. Urban, and F. Racapé, "Wide angular intra prediction for versatile video coding," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2019, pp. 53–62.

[6] Y.-J. Chang, H.-J. Jhu, H.-Y. Jiang, L. Zhao, X. Zhao, X. Li, S. Liu, B. Bross, P. Keydel, H. Schwarz, D. Marpe, and T. Wiegand, "Multiple reference line coding for most probable modes in intra prediction," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2019, p. 559.

[7] Y.-J. Chang, "Intra prediction using multiple reference lines for the versatile video coding standard," *Proc. SPIE*, vol. 11137, Sep. 2019, Art. no. 1113716.

[8] M. Schäfer, B. Stallenberger, J. Pfaff, P. Helle, H. Schwarz, D. Marpe, and T. Wiegand, "An affine-linear intra prediction with complexity constraints," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 1089–1093.

[9] J. Pfaff, "Data-driven intra-prediction modes in the development of the versatile video coding standard," *ITU J. ICT Discoveries*, vol. 3, no. 1, pp. 25–32, 2020.

[10] S. De-Luxán-Hernández, V. George, J. Ma, T. Nguyen, H. Schwarz, D. Marpe, and T. Wiegand, "An intra subpartition coding mode for VVC," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 1203–1207.

[11] O. Akbulut and M. Z. Konyar, "Improved intra-subpartition coding mode for versatile video coding," *Signal, Image Video Process.*, vol. 16, no. 5, pp. 1363–1368, Jul. 2022.

[12] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Performance analysis of VVC intra coding," *J. Vis. Commun. Image Represent.*, vol. 79, Aug. 2021, Art. no. 103202.

[13] M. Koo, M. Salehifar, J. Lim, and S.-H. Kim, "Low frequency non-separable transform (LFNST)," in *Proc. Picture Coding Symp. (PCS)*, Nov. 2019, pp. 1–5.

[14] S. Du, "Understanding deep self-attention mechanism in convolution neural networks," AI. Accessed: Nov. 2, 2023. [Online]. Available: https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251

[15] A. Paszke, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019, pp. 1–12.

[16] T. Li, M. Xu, R. Tang, Y. Chen, and Q. Xing, "DeepQTMT: A deep learning approach for fast QTMT-based CU partition of intra-mode VVC," *IEEE Trans. Image Process.*, vol. 30, pp. 5377–5390, 2021.

[17] F. Bossen. *Common Test Conditions and Software Reference Configurations*, document JCTVC-L1100, JCTVC, Geneva, Switzerland, Jan. 2013.

[18] Y.-H. Huang, J.-J. Chen, and Y.-H. Tsai, "Speed up H.266/QTMT intra-coding based on predictions of ResNet and random forest classifier," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2021, pp. 1–6.

[19] J. Zhao, Y. Wang, and Q. Zhang, "Adaptive CU split decision based on deep learning and multifeature fusion for H.266/VVC," *Sci. Program.*, vol. 2020, pp. 1–11, Aug. 2020.

[20] G. Tang, M. Jing, X. Zeng, and Y. Fan, "Adaptive CU split decision with pooling-variable CNN for VVC intra encoding," in *Proc. IEEE Vis. Commun. Image Process. (VCIP)*, Dec. 2019, pp. 1–4.

[21] Q. Zhang, Y. Zhao, B. Jiang, and Q. Wu, "Fast CU partition decision method based on Bayes and improved de-blocking filter for H.266/VVC," *IEEE Access*, vol. 9, pp. 70382–70391, 2021.

[22] Z. Jin, P. An, C. Yang, and L. Shen, "Fast QTBT partition algorithm for intra frame coding through convolutional neural network," *IEEE Access*, vol. 6, pp. 54660–54673, 2018.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[24] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin. *VTM Common Test Condition and Software Reference Configurations for SDR Video*, document JVET-T2010, Teleconference, Oct. 2020.

[25] G. Bjøntegaard, "Improvements of the BD-PSNR model," Berlin, Germany, Tech. Rep. VCEG-AI11, ITU-T SG16/Q6, 2008.

[26] H. Yang, L. Shen, X. Dong, Q. Ding, P. An, and G. Jiang, "Low-complexity CTU partition structure decision and fast intra mode decision for versatile video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 6, pp. 1668–1682, Jun. 2020.

JIANN-JONE CHEN (Member, IEEE) received the B.S.E.E. and M.S.E.E. degrees from National Cheng-Kung University, Tainan, in 1989 and 1991, respectively, and the Ph.D. degree in electronic engineering from National Chiao Tung University, Hsinchu, in 1997. He was a Researcher with the Information and Communications Research Laboratories, Advanced Technology Center, Industrial Technology Research Institute (ITRI), Hsinchu, from 1999 to 2002. He was a Visiting Scientist with the MIT AI Laboratories, from September 2000 to November 2000. Since 2002, he has been with the Department of Electrical Engineering, National Taiwan University of Science and Technology (NTUST), where he is currently an Associate Professor. His research interests include image/video processing, deep image/video compression, image retrieval, and several topics in multimedia communications. His research team won the First Prize in the National College Microcomputer Design, in 2011 and 2012, with a live video streaming system and a real-time cloud video transcoding system. In 2021, he received the IEEE ICCE Excellent Paper Award. He also won the 2022 Best Journal Paper Award from the Taiwan Association of Cloud Computation. His research team won the First Prize in EE-Section in the 2021 National Technical College Thematic Competition. From 2018 to 2023, he conducted several industrial research projects on developing artificial intelligence (AI) assisted application systems, such as lightweight AI module design. He designed AI teaching plans for high-school students for the Taiwan Ministry of Education, in 2020 and 2022.

YEH-GUAN CHOU received the B.S.E.E. and M.S.E.E. degrees from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2020 and 2022, respectively. His experience in designing deep-learning systems for enhancing image compression performances and accelerating video coding helps provide solutions for practical industrial function systems. He is currently a System Engineer with Sunplus Inc., developing software/firmware for automobile electronic devices supporting Carplay and AndriodAuto. His research interests include image/video compression and several deep-learning system designs.

CHI-SHIUN JIANG received the B.S.E.E. and M.S.E.E. degrees from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2020 and 2023, respectively. He interned with HIWIN Inc., from September 2019 to June 2020. He was a Visiting Student with the Kyushu Institute of Technology, from September 2020 to January 2021. He helped design AI-enabled software systems for the 2022 AI education and implementation of the high school student camp. He assisted in an industrial research project on how to build a cloud-based web communication (WebRTC) service platform primarily based on the selective forwarding unit (SFU) architecture. He helped dockerize the original open-source video software and deploy it in the Minikube environment to verify this system's feasibility of Kubernetes (K8S) integration. His research interests include image/video compression and lightweight deep-learning system designs.