Displays 80 (2023) 102545

Contents lists available at ScienceDirect

Displays

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

Paired decision trees for fast intra decision in H.266/VVC☆

Shengrong Wen, Gongchun Ding, Dandan Ding *

School of Information Science and Technology, Hangzhou Normal University, Hangzhou, 311121, China

ARTICLE INFO

Keywords:
Versatile video coding
Multi-type tree partitioning
Decision tree
Indeterminate Space

ABSTRACT

The Versatile Video Coding (H.266/VVC) has demonstrated significant improvements to its predecessor High-Efficiency Video Coding (H.265/HEVC) in terms of compression efficiency. One of its advancements is attributed to the introduction of the QuadTree with nested Multi-type Tree (QTMT). However, the accompanying computing complexity challenges practical applications. To this end, this paper proposes a fast encoding algorithm to accelerate the encoding process of H.266/VVC while maintaining its compression efficiency as much as possible. Essentially, the Coding Unit (CU) partition can be modeled as a classification problem, and thus conventional machine learning methods such as the decision tree can be used to solve it. Previous decision tree-based works generally use one decision tree to dyadically select one partition from a set of candidate partition modes, easily leading to a local optimum. In contrast, this work models the classification from a new viewpoint and devises Paired Decision Trees (PDT) to tackle it. Specifically, the first decision tree determines whether to divide the current CU or not, and the second decision tree further determines whether to divide the CU horizontally or vertically. However, inaccurate results in the first decision tree can be propagated to the second one, resulting in error accumulation. To address this issue, we introduce the Indeterminate Space to both decision trees and perform Rate-Distortion Optimization (RDO) on samples falling within this space. Experimental results show that compared with H.266/VVC reference software VTM-15.0, the proposed PDT method saves 52.86% encoding time on average while maintaining a small BDBR increase of only 1.36%, which significantly outperforms previous works.

1. Introduction

With the rapid development of information technology, digital video has become an indispensable part of our daily lives. At the same time, the rise of social networks and the popularity of mobile devices have also stimulated the development of the digital video field. Now, video dissemination is ubiquitous all over the world, such as short videos that are popular among young people and video calls made by friends using social software [1,2]. These videos are all compressed by prevalent video coding standards like H.264/AVC and H.265/HEVC. H.265/HEVC [3] is the advanced video coding standard developed by the Joint Collaborative Team on Video Coding (JCT-VC) in 2013. Compared with its predecessor H.264/AVC [4], H.265/HEVC has significantly improved the coding efficiency, saving almost 50% bitrate at the same video quality. To meet the growing market demand, the Joint Video Experts Team (JVET) further developed Versatile Video Coding (H.266/VVC) [5], which achieves about 40% coding efficiency over H.265/HEVC under the same perceptual quality. This great coding efficiency is attained by introducing many new coding tools, such as QuadTree with nested Multi-type Tree (QTMT), Context-Adaptive Binary Arithmetic Coding (CABAC), extended intra-prediction angle modes, etc., at the cost of enormous computational complexity. As investigated in [6], the complexity of H.266/VVC is 18 times higher than that of H.265/HEVC.

As the core technique in video coding, the partition process of Coding Units (CUs) accounts for 91% of the entire encoding time [7]. In H.265/HEVC, the partition of a CU refers to deciding whether or not to perform the quadtree partition. However, H.266/VVC devises QTMT, which additionally adds four new partitioning modes to H.265/HEVC. As shown in Fig. 2(a), from left to right are No Split (NS), QuadTree (QT), Binary Tree Vertically (BTV), Binary Tree Horizontally (BTH), Ternary Tree Vertically (TTV), and Ternary Tree Horizontally (TTH). It is obvious that the use of more partition structures enables more detailed and reasonable partitions, but meanwhile, finding the most suitable partitioning structure for each CU becomes an exponential problem with a power of 6, as there are at most six partition candidates for each CU. As the number of CUs increases, associated computational complexity will also be exponentially increased. As exemplified in Fig. 2(b), a Coding Tree Unit (CTU) with the size of 128 × 128 is finally

☆ This paper was recommended for publication by Prof Guangtao Zhai.

* Corresponding author.
E-mail address: DandanDing@hznu.edu.cn (D. Ding).

https://doi.org/10.1016/j.displa.2023.102545
Received 10 August 2023; Received in revised form 18 September 2023; Accepted 25 September 2023
Available online 5 October 2023

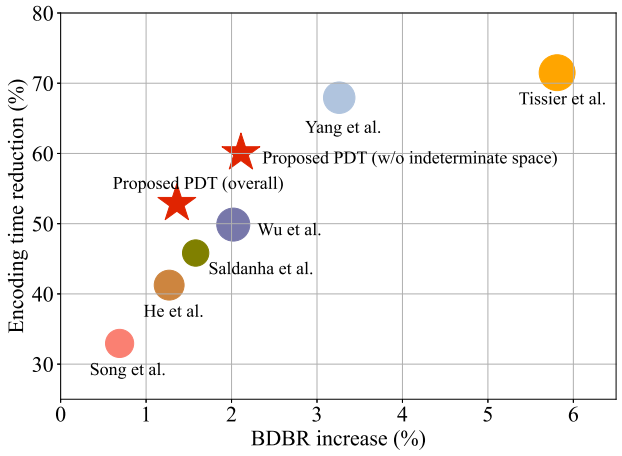


Fig. 1. BDBR loss versus time reduction of different methods. We reproduce all compared methods for comparison except that results of Tissier et al. are obtained from their open-source code and models.

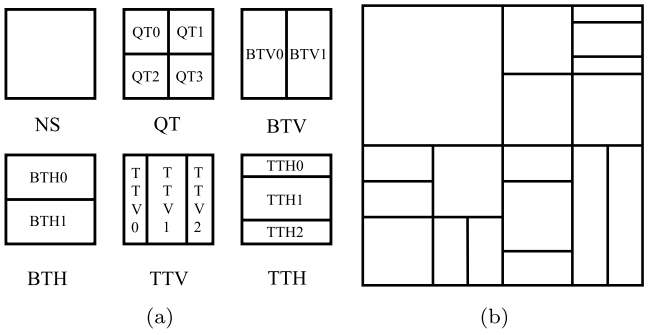


Fig. 2. (a) Illustration of Multi-type tree partition in H.266/VVC, including no split (NS), QuadTree (QT), Binary Tree Vertically (BTV), Binary Tree Horizontally (BTH), Ternary Tree Vertically (TTV), and Ternary Tree Horizontally (TTH). (b) An example of QTMT partition on a 128 × 128 CTU of H.266/VVC.

partitioned into dozens of CUs. Therefore, fast partition algorithms are highly desired to accelerate the encoding process while maintaining the coding performance as much as possible.

In this regard, this work focuses on the fast CU partition for intra coding of H.266/VVC. Typically, the partition problem is modeled as a classification problem, which can be solved by machine learning methods such as Support Vector Machines (SVM), Decision Tree (DT), and Random Forest (RF). Taking the DT that has been widely used in previous research for example, as depicted in Fig. 3, a partition mode is eliminated every time until the best partition pattern is obtained [8]. This process not only triggers a long inference time but also easily leads to a local optimum since errors in previous steps will be propagated and accumulated, which accordingly decreases the encoding performance. To overcome this issue, we propose a Paired Decision Trees (PDT)-based method that remodels the binary classification procedure to shorten the decision process. Additionally, we introduce an Indeterminate Space to handle difficult-to-decide samples and avoid local optima. An overview of our results compared with state-of-the-art methods is illustrated in Fig. 1.

The main contributions of this work are as follows:

- We remodel the binary-classification procedure in H.266/VVC CU partition and devise Paired Decision Trees, termed PDT, to resolve it. Specifically, the first decision tree determines whether to divide the current CU or not, and the second decision tree makes a subsequent determination on whether the current CU should be split horizontally or vertically.

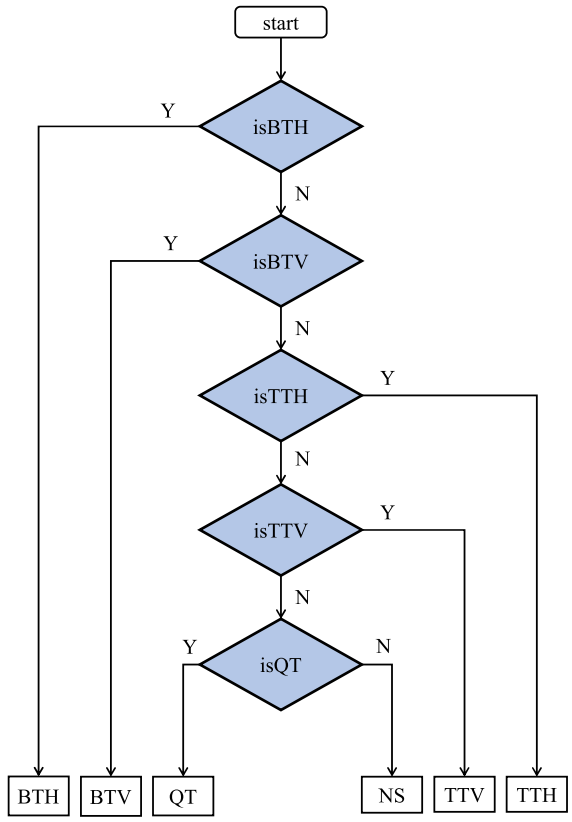


Fig. 3. Typical decision flow using the decision tree where the partition mode is eliminated dyadically until reaching the last one.

- Considering that the wrong decision in the former DT will affect the subsequent DT, we propose the Indeterminate Space where Rate-Distortion Optimization (RDO) is fully applied to these difficult-to-decide samples. As such, error propagation and accumulation can be effectively relieved and the coding performance is maintained.
- All block sizes under the same QP value share the same PDT. Extensive experimental results demonstrate that the PDT method brings in 60.08% encoding time reduction with 2.11% BDBR loss; the additional introduction of Indeterminate Space results in 52.86% runtime savings with only 1.36% BDBR increase, which significantly surpasses state-of-the-arts.

The remaining parts of this paper are arranged as follows. Section 2 introduces the related works. Section 3 describes our proposed method in detail. Section 4 presents our experimental results and analysis. Section 5 concludes this work.

2. Related works

Up to now, lots of studies have been devoted to developing fast encoding algorithms for video encoders [9–11]. They can be roughly divided into three categories: (1) the traditional correlation-based method [12–14] that mainly utilizes features like variance, contrast, and gradient to detect textures and then predicts partitioning modes by setting thresholds and other techniques, (2) the machine learning-based method [15–20] that sets feature vectors, trains sVM, DT, or RF models offline, and finally inherits the models into video codecs for a fast decision, and (3) the neural network-based method [21–26] that predicts the probability of each partition mode and selects the one having the highest probability as the final mode. In the following, we will detail these three categories of methods.

For the **traditional correlation-based method**, Fan et al. [27] proposed a fast intra-frame partitioning algorithm based on the variance value and Sobel operator and set three thresholds to determine the CU partitioning mode based on the relationship between the thresholds and features, which saved 49.27% encoding time with 1.63% BDBR increase for VTM-7.0. Song et al. [28] used the Sum of the Mean Absolute Deviation (SMAD) of the sub-blocks to measure the texture complexities of vertical and horizontal directions. Their method performed well in terms of performance loss but did not save too much time.

Regarding the **machine learning-based method**, typical solutions include DT, SVM, and RF. Wu et al. [29] proposed a fast CU partitioning algorithm based on SVM. They extracted QP, variance, and gradient as features, and then transformed the CU partitioning problem into two binary classification problems: split or not, horizontal partition or vertical partition, saving encoding time ranging from 30.78% to 63.16% with 1.10% to 2.71% BDBR increase in VTM-10.0. Zhang et al. [30] proposed an improved DAG-SVM classifier model for fast CU partitioning. Yang et al. [8] proposed an algorithm that used DTs to accelerate CU partitioning. They set a confidence level at the leaf node of each DT. When the confidence level was less than 90%, they chose not to trust the DT result and instead performed RDO operation. Tahir et al. [31] proposed an online and offline combination of RF to perform fast encoding. Amestoy et al. [32] used RF to determine whether to perform QT or BTV/BTH partitioning and proposed an Uncertainty Zone where RDO was involved when RF cannot distinguish between QT and BTV/BTH. Later, they further proposed a lightweight and adjustable QTMT partition method based on RF [33]. He et al. [34] divided CUs into three categories: simple, complex, and fuzzy according to their texture structure, and RF was used to predict the partition model for simple and complex CUs while split or not for fuzzy CUs. In addition, other machine learning algorithms such as Markov Random Field [35] and Bayes decision [36,37] were also used to implement fast encoding algorithms.

Concerning the **neural network-based method**, Li et al. [25] designed a multi-stage exit Convolutional Neural Networks (CNN) model with an early exit mechanism to determine CU partitioning, which reduced the encoding time of VTM-7.0 by 44.65%–66.88% with a BDBR loss of 1.322%–3.188%. Zhang et al. [38] used a Global Convolutional Network (GCN) to accelerate CU partitioning, but the CNN model was called by every CU for decision-making, which may increase inference time.

3. Proposed method

3.1. Proposed paired decision trees method

Fig. 4 presents the overall flow of our proposed PDT method. First, we determine whether the current CU size needs the fast partitioning algorithm or not. Notice that we perform the proposed PDT on selective block sizes. Using the fast algorithm on some block sizes, such as 64 × 64, will lead to noticeable loss and therefore is not acceptable. On the other hand, not all blocks have six partition candidates, e.g., CU of size 4 × 8 only has BTH and NS modes and takes up limited time. As a result, we apply the proposed method on six block sizes, including 32 ×32, 32 × 16, 16 × 32, 16 × 16, 16× 8, and 8 × 16.

Then, we resort to the first decision tree to obtain the partition probability of the current CU. If the probability is between T_1 and T_2, we will perform RDO to decide its partition status; if the probability is smaller than T_1, the mode NS will be selected, i.e., this CU will not be partitioned (split); otherwise, this CU will be partitioned and continue to go through the second decision tree for texture direction determination. Similarly, if the probability is between T_3 and T_4, RDO will be applied; if the probability is less than T_3, we judge that the current CU will be vertically partitioned; otherwise, we judge that the current CU will be horizontally partitioned.

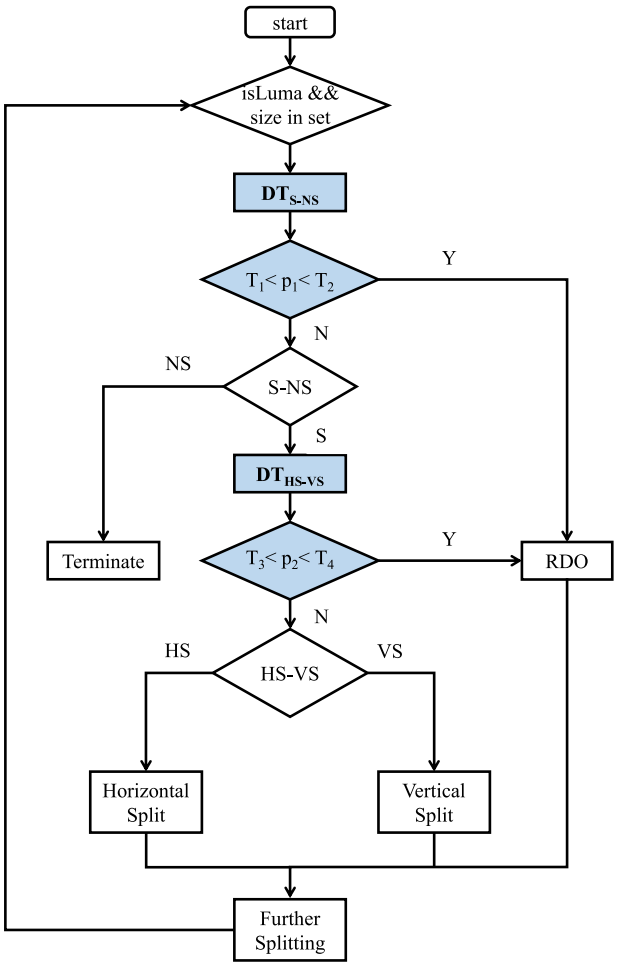


Fig. 4. Flowchart of the proposed fast CU partition decision based on Paired Decision Trees. The first decision tree is responsible for assessing whether the current CU should be divided or not. The second decision tree makes a subsequent determination on whether the CU should be split horizontally or vertically.

3.2. Feature selection

The DT relies on input features to make decisions. Therefore, feature selection is important to the decision results. Since the two DTs used in this work focus on different aspects, i.e., the first DT focuses more on the global texture direction, while the second DT pays more attention to local texture variations, we choose different features for these two trees for adequate processing.

Gradient. Gradient is a feature closely related to CU partition, which is widely used in fast encoding algorithms. In both decision trees, the following features are adopted: the normalized gradients (x_{NZG}), the normalized maximum gradient magnitude (x_{NMG}), the average gradients in the horizontal direction (x_{ARH}), and the average gradients in the vertical direction (x_{ARV}). The Sobel operator is used to calculate the gradient. Corresponding calculation methods of gradients in the vertical direction (G_x) and horizontal direction (G_y) are as follows: Defining

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{1}$$

and

$$S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \tag{2}$$

we can compute

$$G_x = \sum_{i=1}^{H}\sum_{j=1}^{W} L \times S_x \tag{3}$$

and

$$G_y = \sum_{i=1}^{H}\sum_{j=1}^{W} L \times S_y, \tag{4}$$

where S_x and S_y represent Sobel operators in the horizontal and vertical directions, respectively. W and H represent the width and height of the current CU, and L denotes the luminance matrix of the current pixel. Then, the calculation methods of x_{NZG}, x_{NMG}, x_{ARH}, and x_{ARV} are as follows:

$$x_{NZG} = \frac{|G_x| + |G_y|}{W \times H}, \tag{5}$$

$$x_{NMG} = \frac{Max(G_x, G_y)}{W \times H}, \tag{6}$$

$$x_{ARH} = \frac{G_x}{(W-2)\times(H-2)}, \tag{7}$$

$$x_{ARV} = \frac{G_y}{(W-2)\times(H-2)}. \tag{8}$$

Variance. Variance can well reflect the global texture information of a frame. To this end, in the first decision tree, we use the variance of the entire frame as a representative feature, as shown below:

$$Var = \frac{1}{W \times H}\sum_{i=0}^{H}\sum_{j=0}^{W}(p_{ij} - \bar{p})^2, \tag{9}$$

where p_{ij} represents the pixel value at position (i, j), and $\bar{p}$ represents the average pixel value of the current CU.

As for the second decision tree, more detailed texture information is required for a specific direction decision. In this regard, we define Var_H as the variance difference in the horizontal direction and Var_V as the variance difference in the vertical direction. They are calculated as:

$$Var_H = Var_{BTH} + Var_{TTH} \tag{10}$$

and

$$Var_V = Var_{BTV} + Var_{TTV}, \tag{11}$$

where Var_{BTH} represents the variance difference in the horizontal direction under the binary horizontal partition. Similarly, the other three items Var_{BTV}, Var_{TTH}, and Var_{TTV} represent the corresponding differences under other partitioning modes, calculated as follows:

$$Var_{BTH} = |Var_{BTH0} - Var_{BTH1}|, \tag{12}$$

$$Var_{BTV} = |Var_{BTV0} - Var_{BTV1}|, \tag{13}$$

$$\begin{aligned} Var_{TTH} = {} & |Var_{TTH0} - Var_{TTH1}| \\ & + |Var_{TTH0} - Var_{TTH2}| \\ & + |Var_{TTH1} - Var_{TTH2}|, \end{aligned} \tag{14}$$

and

$$\begin{aligned} Var_{TTV} = {} & |Var_{TTV0} - Var_{TTV1}| \\ & + |Var_{TTV0} - Var_{TTV2}| \\ & + |Var_{TTV1} - Var_{TTV2}|. \end{aligned} \tag{15}$$

Contrast. Image contrast refers to the degree of grayscale contrast in an image. The larger the contrast, the clearer the image, and the more it tends to be partitioned. Here, we use the contrast of the entire CU as a feature in the first decision tree, as shown below:

$$C = \sum_{\delta} \delta(i,j)^2 pro_{\delta}(i,j), \tag{16}$$

with $\delta(i, j)$ being the difference between the current pixel i and it surrounding four neighbors j and $pro_{\delta}(i, j)$ denoting the probability of the value $\delta(i, j)$.

As for the local texture information in the second decision tree, it is represented by the contrast C_H and C_V. The calculation of C_H and C_V is similar to Eqs. (10) and (11), which will not be repeated here.

Entropy. Image entropy represents the aggregation feature of image grayscale distribution. Similarly, in the first decision tree, we use the entropy of the entire CU as a feature. The calculation formula of entropy is as follows:

$$E_k = -\sum_{i=0}^{255} pro(i)\log pro(i), \tag{17}$$

where $pro(i)$ represents the probability of grayscale i.

As for the local texture information in the second decision tree, it is represented by the entropy E_H and E_V. The calculation of E_H and E_V is respectively similar to Eqs. (10) and (11).

Block information. Unlike previous works that train a single model for each block size, this work inputs the block size (width and height of the current CU) as features, making all CU sizes share the same PDT models.

3.3. Decision tree selection

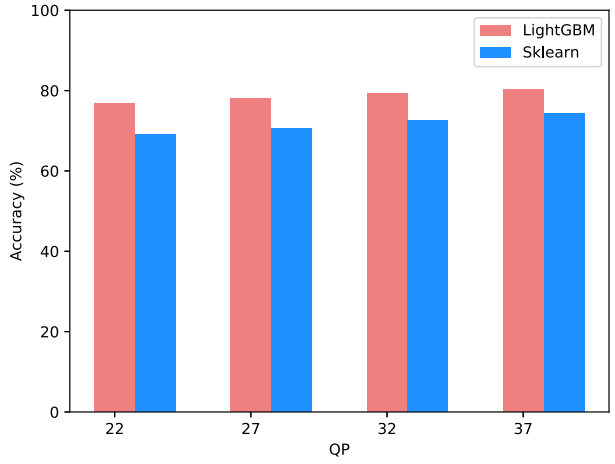
To achieve decent performance, we verify the proposed PDT method on a variety of decision trees, including Matlab R2022 A, Scikit learn (Sklearn) [39], and LightGBM [40]. Since the use of the decision tree provided by Matlab R2022 A results in poor performance, we will not discuss it in this paper.

Then we analyze the performance of Sklearn and LightGBM,[1] in depth. As depicted in Fig. 5 we compare the CU partition accuracy and direction partition accuracy between the two decision trees. It is observed that the decision accuracy of LightGBM is higher than Sklearn by around 6% to 7%. This excellent performance is mainly due to the fact that LightGBM uses the leaf-wise decision tree growth strategy, while Sklearn uses the level-wise growth strategy. As shown in Fig. 6, the level-wise strategy performs unbiased splits on all nodes in each layer and traverses all leavers for splitting. In contrast, the leaf-wise approach selects the node with the maximum splitting gain among all current leaf nodes for splitting, leading to lower complexity and higher accuracy. However, this strategy is more prone to overfitting because of its biased selection. To address this issue, LightGBM adds a maximum depth limit on top of the leaf-wise approach to prevent overfitting while ensuring efficiency. Moreover, LightGBM uses many features that Sklearn does not have to improve its performance, such as histogram and efficient parallel learning.

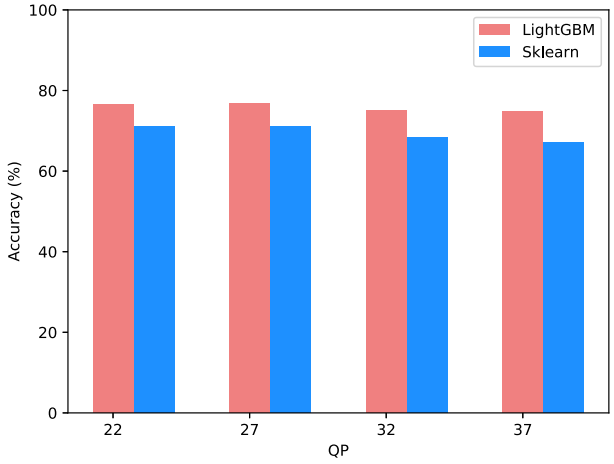
Also, we compare the runtime of Sklearn and LightGBM. Theoretically, the Sklearn decision tree needs to traverse from the root node to the leaf node, i.e., the entire decision tree, to search the category for each sample, leading to $O(logN)$ time complexity, where N indicates the number of samples. Instead, LightGBM uses optimization techniques during prediction, such as histogram-based algorithms and multi-threaded parallel processing, to quickly find the leaf node for each sample, which efficiently reduces the prediction time complexity to $O(1)$. However, the optimization techniques in LightGBM are also time-consuming and thus extend the runtime.

In this work, we collect the inference time of Sklearn and LightGBM in 10 million data for runtime comparison. Results show that they cost comparable runtime. To this end, we adopt the use of LightGBM to build our PDT due to its high accuracy.

[1] https://lightgbm.readthedocs.io/en/latest/index.html.

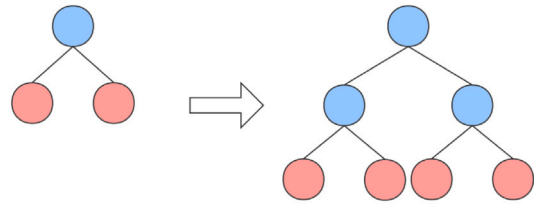


(a) CU partition accuracy

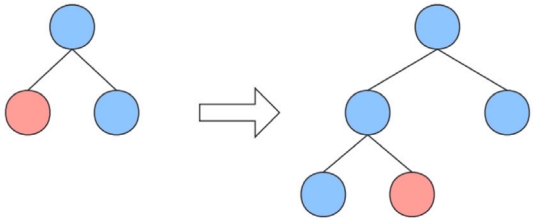


(b) Direction partitioning accuracy

Fig. 5. Comparison between two typical implementation methods (Sklearn vs. LightGBM) of the decision tree on overall accuracy.

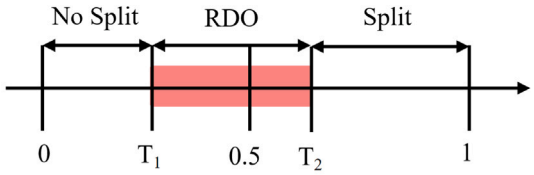


(a) Level-wise decision in Sklearn

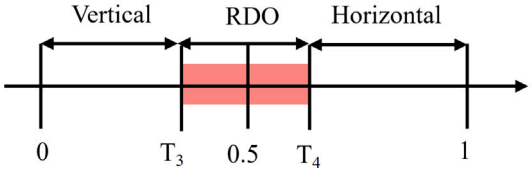


(b) Leaf-wise decision in LightGBM

Fig. 6. Illustration of level-wise and leaf-wise generation strategies in Sklearn and LightGBM decision trees, respectively.

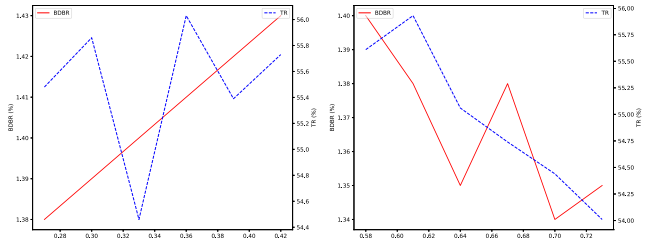


(a) T_1 and T_2 used in the first DT

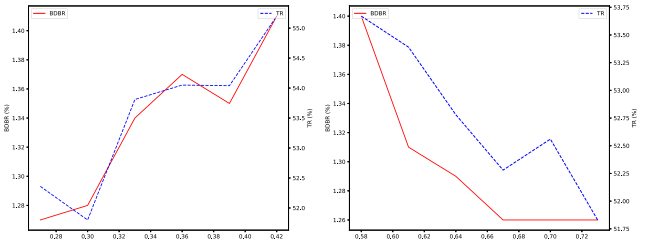


(b) T_3 and T_4 used in the second DT

Fig. 7. Illustration of Indeterminate Space used in this work where two thresholds are set in each decision tree. (a) T_1 and T_2 are used in the first DT to determine No split, Indeterminate Space, or Split. (b) T_3 and T_4 are used in the second DT to determine Vertical Split, Indeterminate Space, or Horizontal Split.



(a) T1 for the first DT (b) T2 for the first DT



(c) T3 for the second DT (d) T4 for the second DT

Fig. 8. Relationship between the threshold values of Indeterminate Space, BDBR increase, and time reduction (TR) at block size 16 × 16 and QP value 22 on all test sequences.

3.4. Indeterminate space

As analyzed in the introduction, one disadvantage of DT-based methods is the local optimum problem, particularly in a decision process that uses multiple DTs. The error decision in the former DT will be propagated to the subsequent DTs, resulting in a significant performance decrease. To this end, we introduce the **Indeterminate Space**. Samples difficult to decide typically fall into this Indeterminate Space and we directly go through the RDO process for decision. In this way, the error propagation problem can be effectively alleviated and the coding performance can be maintained.

Specifically, we use two parameters T_1 and T_2 to set the Indeterminate Space for the first decision tree and T_3 and T_4 for the second one. Samples between these two thresholds (within the Indeterminate

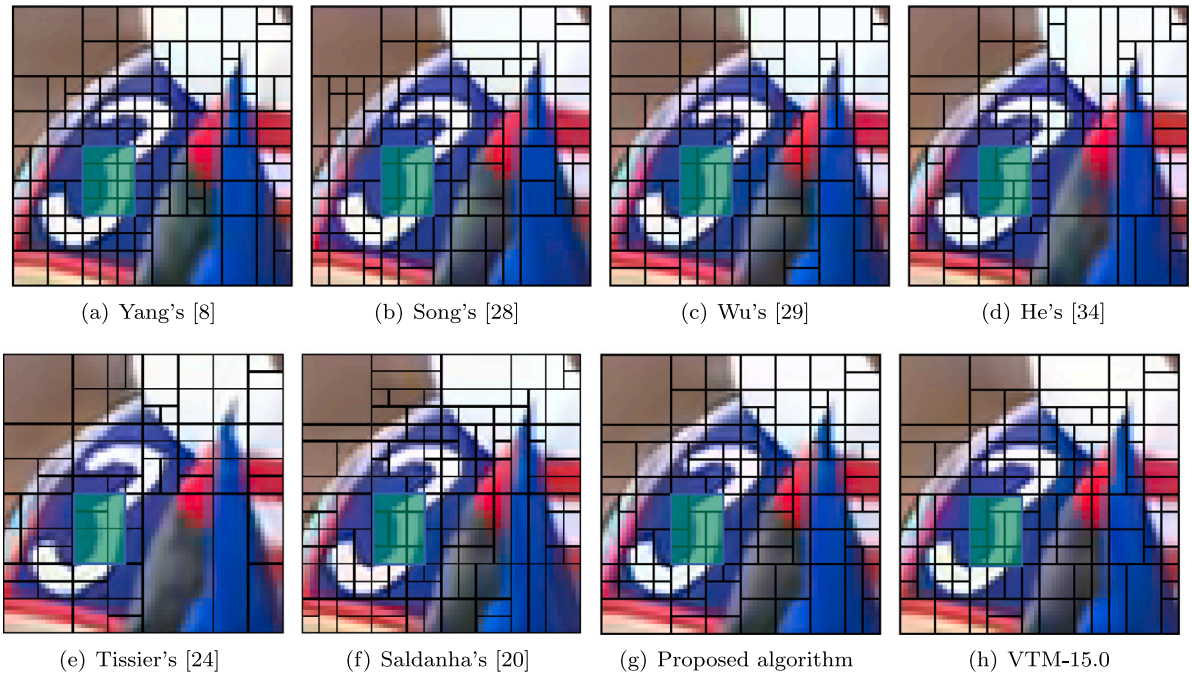


Fig. 9. Visualization of partitioning results on a 64 × 64 block of sequence *RaceHorses*.

Table 1
Threshold Settings in relation to QP values and block sizes.

QP	Threshold	16 × 16 blocks	Other blocks
22	T_1	0.30	0.34
	T_2	0.70	0.46
	T_3	0.39	0.40
	T_4	0.61	0.44
27	T_1	0.32	0.35
	T_2	0.68	0.45
	T_3	0.40	0.40
	T_4	0.60	0.44
32	T_1	0.34	0.36
	T_2	0.66	0.44
	T_3	0.41	0.41
	T_4	0.59	0.43
37	T_1	0.36	0.36
	T_2	0.64	0.43
	T_3	0.42	0.41
	T_4	0.58	0.43

Space) will go through the RDO process, while samples outside will directly follow the results of the decision tree.

The performance of our method is highly associated with the values of these thresholds. We then conduct a comprehensive study on their settings. As shown in Fig. 8, we investigate the BDBR increase and Time Reduction (TR) of all test sequences for 16 × 16 blocks at Quantization Parameter (QP) value 22. As observed, when $T_1 = 0.3$ and $T_2 = 0.7$, the BDBR loss and TR strike a balanced trade-off. We can also find another trade-off point in the figure, which can be set according to practical requirements. Similarly, $T_3 = 0.39$ and $T_4 = 0.61$ are adopted in this work for 16 × 16 blocks.

In our experiments, we find that the threshold settings are basically correlated to not only QP values but also block sizes. After extensive experiments, we finally set these threshold values following Table 1. Notice the block size of 16 × 16 is specifically devised because 16 × 16 blocks are very sensitive to the threshold values.

4. Experiments and analysis

4.1. Settings and conditions

To evaluate the performance of our proposed PDT algorithm, we conducted experiments on the H.266/VVC reference software VTM-15.0. We use 18 common test sequences that are widely used in standard committees to evaluate our coding performance. These sequences cover five classes: Class A (2560 × 1600), Class B (1920 × 1080), Class C (832 × 480), Class D (416 × 720), and Class E (1280 × 720). We select the first frames in *PeopleOnStreet* from Class A, *BasketballDrive* from Class B, *PartyScene* from Class C, *BlowingBubbles* from Class D, and *Johnny* from Class E as our training dataset. We follow the common test conditions of H.266/VVC to evaluate the coding performance. The coding structure is set as All Intra under *encoder_intra_vtm.cfg* configuration. We test our algorithm at QP values 22, 27, 32, and 37 of VTM-15.0 and calculate the averaged Bjøntegaard Delta Bit Rate (BDBR) result. The calculation method of Time Reduction (TR) is calculated as follows:

$$TR(\%) = \frac{1}{4}\sum_{i\in Q}\frac{T_r(i) - T_p(i)}{T_r(i)} \times 100\%, \tag{18}$$

where $T_r(i)$ and $T_p(i)$ represent the total encoding time of VTM-15.0 and our proposed method, respectively, and Q represents the set of QPs $\{22, 27, 32, 37\}$. The proposed method is applied to block size 32 ×32, 32 × 16, 16 × 32, 16 × 16, 16× 8, and 8 × 16. Notice that all block sizes under the same QP value share the same PDT models, which is different from previous works that generally train a single DT model for each block size.

Our experiments are conducted on an Intel (R) Core (TM) i7-8700K CPU 3.70 GHz using the Microsoft Visual Studio C++ 2019 compiler and the Microsoft open-source LightGBM.

4.2. Performance

Overall Performance. The overall performance of our proposed method is presented in Table 2. As seen, using the proposed PDT (w/o Intermediate Space) leads to 60.08% encoding time saving and 2.11% BDBR loss. The introduction of Intermediate Space finally leads to 52.86% encoding time reduction with only 1.36% BDBR increase.

Table 2
Performance comparison of the proposed algorithm and state-of-the-arts.

Class	Sequence	Yang's [8] (DT)		Song's [28] (Threshold)		Wu's [29] (SVM)		He's [34] (RF)		Tissier's [24] (CNN+LightGBM)		Li [25] (CNN)		Saldanha's [20] (LightGBM)		Feng's [26] (CNN)		Proposed PDT (w/o Indeterminate Space)		Proposed PDT (Overall)	
		BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)	BDBR (%)	TR (%)
A	PeopleOnStreet	4.18	72.52	0.68	32.00	2.26	52.98	1.16	41.72	5.01	77.06	–	–	1.20	45.89	–	-	2.54	64.05	1.47	53.86
	Traffic	4.61	67.87	0.74	34.13	2.35	52.35	1.46	42.84	5.65	72.63	–	–	1.50	47.73	–	–	2.70	63.78	1.74	54.41
B	BasketballDrive	4.43	69.34	1.13	36.41	2.95	48.08	1.32	49.41	3.60	73.22	1.64	52.03	2.74	47.31	2.89	73.33	2.42	62.45	1.89	51.89
	BQTerrace	2.85	64.36	0.57	31.93	1.51	49.18	0.89	39.65	5.03	73.55	1.11	45.61	0.91	38.29	2.49	72.22	1.62	58.59	1.11	49.05
	Cactus	4.04	68.36	0.76	34.97	2.26	52.94	1.41	45.41	5.18	75.67	1.12	49.30	1.53	46.98	2.39	76.52	2.51	63.79	1.52	54.02
	Kimono	1.73	68.27	0.55	35.51	1.17	50.58	0.96	49.26	1.51	69.58	–	–	2.73	55.32	–	–	1.47	64.96	0.91	56.75
	ParkScene	3.47	69.07	0.62	32.82	1.75	52.93	1.17	45.49	4.13	77.92	–	–	1.23	45.18	–	–	1.93	63.71	1.04	52.48
C	BasketballDrill	4.87	63.77	1.01	28.45	3.54	52.87	2.23	46.55	12.30	70.02	1.62	39.29	1.87	43.52	4.03	69.41	3.93	60.55	2.38	50.73
	BQMall	2.84	62.43	0.54	34.62	1.73	50.28	1.18	37.05	6.95	71.67	1.17	49.75	1.27	42.46	2.52	72.22	1.77	60.18	1.17	52.14
	PartyScene	1.49	57.07	0.32	30.64	1.03	49.91	0.51	33.09	4.90	68.59	0.61	45.19	0.38	34.05	1.51	72.06	1.22	58.11	0.74	47.90
	RaceHorsesC	3.31	64.13	0.61	32.27	1.91	53.58	0.97	38.91	4.48	70.58	0.96	46.44	1.02	45.20	2.00	72.89	2.00	62.75	1.21	52.25
D	BasketballPass	2.62	60.07	0.87	37.73	1.87	45.58	1.43	41.92	7.65	69.32	1.40	44.45	2.16	42.46	2.23	58.50	1.66	56.85	1.22	49.80
	BlowingBubbles	2.32	58.81	0.60	32.79	1.69	49.11	1.02	36.85	6.33	68.60	0.92	41.55	0.74	39.49	1.19	53.91	1.87	54.52	1.09	46.76
	BQSquare	1.55	54.86	0.23	28.93	0.75	39.92	0.49	32.24	5.17	68.67	0.74	44.48	0.20	26.82	1.32	54.75	0.96	48.89	0.55	40.44
	RaceHorses	2.30	57.62	0.63	30.84	1.86	49.59	0.90	35.47	5.42	65.53	1.20	41.64	0.63	39.12	1.87	46.52	1.77	56.76	1.15	48.07
E	FourPeople	4.27	68.66	0.85	33.76	2.64	49.72	1.79	40.38	7.26	75.23	1.33	49.87	2.01	47.75	2.75	76.63	2.63	60.94	1.71	52.61
	Johnny	4.25	67.21	0.96	32.29	2.77	48.26	2.02	45.08	7.34	72.51	2.32	48.15	3.37	46.56	3.94	74.87	2.63	60.48	1.90	52.15
	KristenAndSara	3.54	65.76	0.69	32.30	2.41	47.53	1.88	43.88	6.62	71.49	1.76	50.46	3.04	48.07	3.08	74.87	2.35	59.47	1.65	51.23
	Average	**3.26**	**67.94**	**0.69**	**32.95**	**2.02**	**49.87**	**1.27**	**41.25**	**5.81**	**71.50**	**1.28**	**46.30**	**1.58**	**45.82**	**2.44**	**67.76**	**2.11**	**60.08**	**1.36**	**52.86**

Comparison with State-of-the-arts. In addition, we implement several prevalent fast intra partition methods for comparison, including Yang's [8] (DT-based method), Song's [28] (traditional threshold-based method), Wu's [29] (SVM-based method), He's [34] (RF-based method), and Saldanha's [20] (LightGBM-based method). For fair comparisons, we reproduce them in VTM-15.0 under exactly the same training and testing conditions as ours. Moreover, we compare with state-of-the-art CNN-based solutions including Tissier's [24], Li's [25], and Feng's [26]. For Tissier's work, we utilize the code and models released by its website[2] to reproduce results. For the other two CNN-based methods, since not all their code and models are unavailable at this moment, we directly use results in their papers for comparison.

As shown in Table 2, Yang's method [8] saves encoding time as large as 67.94%, but the loss of coding efficiency is also increased to 3.26%. Song's method [28] achieved the least loss (0.69%) and the time saving is also very limited (only 32.95%). This occurs because their method is only applied to 16×16 and 32×32 blocks. For Wu's method [29], it saves 49.87% encoding time with as large as 2.02% loss. He's method [34] optimizes the loss to 1.27% but the time saving is decreased to 41.25%. Relatively, the proposed PDT is more advanced: with comparable BDBR loss to Wu's (2.11% vs. 2.02%), we save more encoding time (60.08% vs. 49.87%); it is a similar case when compared with He's method: with the similar BDBR loss (1.36% vs. 1.27%), the proposed PDT saves more time (52.86% vs. 41.25%). In comparison to Saldanha's method [20], our method saves more encoding time (52.86% vs. 45.82%) but a smaller BDBR loss (1.36% vs. 1.58%).

Concerning the CNN-based solutions, although Tissier et al. [24] reduce 71.50% processing time, the resulting BDBR loss is as large as 5.81%. The BDBR loss of Feng's [26] is also very high. Compared with Li's work [25], our method leads to higher time reduction but a smaller BDBR increase, e.g., in most Class C and D sequences.

Feature Importance. Furthermore, the important degree of features used in the two decision trees is assessed by their LightGBM models, including gradient, variance, contrast, entropy, and block information (width and height), as presented in Fig. 10. It is observed that features such as block information, gradient, and variance are of crucial importance in both decision trees. Additionally, the contrast feature exhibits a higher level of significance in the first decision tree compared to the second one.

Partition Visualization. Fig. 9 visualizes the partition results of different methods on the same 64×64 block of sequence *RaceHorses*. It is observed that our partition results are highly close to that of VTM-15.0, which is consistent with our performance. Yang's [8], Wu's [29], Tissier's, and Saldanha's methods exhibit quite different partition results from VTM-15.0, demonstrating their inaccuracy in the decision process. He's [34] partition is not aligned with VTM-15.0 in regions with vivid textures, e.g., the number "3" in the visualized frame. Notice that the partition results of Song's [28] algorithm are also very similar to that of VTM-15.0 because they only predict CU partition for 16×16 and 32×32 blocks.

[2] https://github.com/Souhailkudo/VTM_intra_CNN_LGBM_patch.

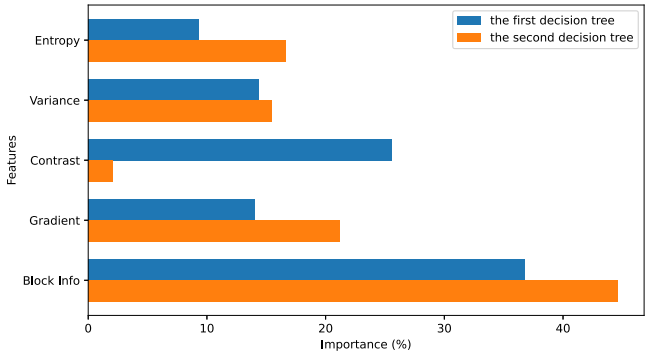


Fig. 10. Importance of various features in the two decision trees.

4.3. Ablation study

Paired Decision Trees. In this ablation study, we conduct experiments on each decision tree to evaluate the coding performance. The results are shown in Table 3. It is observed that using only the first decision tree saves 11.68% runtime with only 0.11% BDBR loss while using only the second decision tree leads to a 40.60% runtime reduction, accompanied by a 0.90% increase in BDBR. Overall, when both decision trees are combined, i.e., the proposed paired decision trees method, it attains an impressive 52.86% reduction in processing time while incurring a modest 1.36% BDBR loss.

Threshold Settings in Indeterminate Space. In addition, we adopt unified thresholds for T_1, T_2, T_3, and T_4 in Fig. 7 regardless of QP values to assess the coding performance. Specifically, the threshold settings of QP 32 (see Table 1) are used to test all QPs. Results in Table 4 demonstrate that the use of unified thresholds has no impact on the runtime. However, it does introduce a minor BDBR increase (from 1.36% to 1.41%). Thus, the use of unified thresholds can be considered for practical simplicity without significantly affecting performance.

5. Conclusion

This paper proposed Paired Decision Trees for H.266/VVC fast intra partition. Existing works generally use several decision tree models

Table 3
Ablation on each decision tree of proposed method.

Class	Sequence	First decision tree		Second decision tree	
		BDBR (%)	TR (%)	BDBR (%)	TR (%)
A	PeopleOnStreet	0.15	12.62	0.94	41.19
	Traffic	0.12	11.49	1.14	41.59
B	BasketballDrive	0.05	11.80	1.37	38.08
	BQTerrace	0.05	9.35	0.80	38.14
	Cactus	0.09	11.55	1.05	42.40
	Kimono	0.06	10.80	0.54	39.74
	ParkScene	0.11	13.41	0.69	39.14
C	BasketballDrill	0.32	14.94	1.58	39.09
	BQMall	0.11	10.02	0.79	44.21
	PartyScene	0.03	7.67	0.51	39.50
	RaceHorsesC	0.03	9.70	0.86	42.24
D	BasketballPass	0.04	12.80	0.81	42.77
	BlowingBubbles	0.11	9.31	0.75	39.23
	BQSquare	0.07	9.52	0.34	32.55
	RaceHorses	0.02	7.51	0.76	39.00
E	FourPeople	0.20	11.96	1.10	42.80
	Johnny	0.21	13.95	1.23	40.99
	KristenAndSara	0.14	13.22	0.98	41.69
	Average	**0.11**	**11.68**	**0.90**	**40.60**

Table 4
Ablation on the threshold setting of indeterminate space.

Class	Sequence	Proposed PDT		Unified thresholds	
		BDBR (%)	TR (%)	BDBR (%)	TR (%)
A	PeopleOnStreet	1.47	53.86	1.65	53.40
	Traffic	1.74	54.41	1.74	53.97
B	BasketballDrive	1.89	51.89	1.81	55.53
	BQTerrace	1.11	49.05	1.10	51.52
	Cactus	1.52	54.02	1.46	54.87
	Kimono	0.91	56.75	0.88	54.93
	ParkScene	1.04	52.48	1.26	52.25
C	BasketballDrill	2.38	50.73	2.73	50.88
	BQMall	1.17	52.14	1.15	52.07
	PartyScene	0.74	47.90	0.73	47.68
	RaceHorsesC	1.21	52.25	1.19	51.79
D	BasketballPass	1.22	49.80	1.22	50.18
	BlowingBubbles	1.09	46.76	1.03	46.75
	BQSquare	0.55	40.44	0.51	41.32
	RaceHorses	1.15	48.07	1.08	47.56
E	FourPeople	1.71	52.61	1.88	55.57
	Johnny	1.90	52.15	2.25	51.85
	KristenAndSara	1.65	51.23	1.66	53.36
	Average	**1.36**	**52.86**	**1.41**	**53.25**

to recursively eliminate the low-probability mode, leaving the last partition mode as the selected one. This process requires calling the decision tree models multiple times, resulting in a long inference time. On the other hand, it easily leads to the local optimum since partition errors in the previous steps will be propagated to the following steps and trigger error accumulation. To address this issue, this work proposed a Paired Decision Trees method, using two decision trees for partition determination. Specifically, we remodel the partition flow to two steps, and assign a decision tree to each step: the first decision tree determines whether to divide the current CU or not, and the second decision tree further determines whether to horizontally or vertically divide the CU. Moreover, we propose the Indeterminate Space to deal with those difficult-to-decide samples, for which RDO is applied for finetuning. Extensive experiments confirm the effectiveness of the proposed method, it saves 52.86% encoding time with only 1.36% BDBR loss compared with the H.266/VVC reference software VTM-15.0, significantly outperforming state-of-the-art works. Our partition results are consistently much closer to that of VTM-15.0.

Currently, the thresholds for Indeterminate Space are manually configured for various block sizes and QP values. The precision of these thresholds could be potentially enhanced by establishing a direct relationship between content features and QP values. This will be studied in our future work.

CRediT authorship contribution statement

Shengrong Wen: Writing – review & editing, Methodology. **Gongchun Ding:** Methodology, Verify and crosscheck results/experiments. **Dandan Ding:** Writing – review & editing, Supervise the research activity, Approve the final manuscript.

Declaration of competing interest

The authors declared that they have no conflicts of interest to this work. We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Data availability

The authors do not have permission to share data.

Acknowledgments

This research was supported by the National Natural Science Foundation of China under Grant 62171174.

References

[1] X. Cui, Z. Peng, F. Chen, G. Jiang, M. Yu, Perceptual ultra-high definition video coding based on adaptive just noticeable distortion model, Displays 75 (2022) 102301.
[2] H. Zhang, L. Song, W. Gan, R. Xie, Multi-scale-based joint super-resolution and inverse tone-mapping with data synthesis for UHD HDR video, Displays (2023) 102492.
[3] G.J. Sullivan, J.-R. Ohm, W.-J. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1649–1668.
[4] T. Wiegand, G.J. Sullivan, G. Bjontegaard, A. Luthra, Overview of the H.264/AVC video coding standard, IEEE Trans. Circuits Syst. Video Technol. 13 (7) (2003) 560–576.
[5] B. Bross, Versatile Video Coding (Draft 1), JVET-J1001, Joint Video Exploration Team (JVET), San Diego, CA, USA, 2018.
[6] F. Bossen, X. Li, K. Suehring, AHG Report: Test Model Software Development (AHG3), JVET-J0003, Joint Video Exploration Team (JVET), San Diego, CA, USA, 2018.
[7] G. Ding, X. Lin, J. Wang, D. Ding, Accelerating QTMT-based CU partition and intra mode decision for versatile video coding, J. Vis. Commun. Image Represent. 94 (2023) 103832.
[8] H. Yang, L. Shen, X. Dong, Q. Ding, P. An, G. Jiang, Low-complexity CTU partition structure decision and fast intra mode decision for versatile video coding, IEEE Trans. Circuits Syst. Video Technol. 30 (6) (2019) 1668–1682.
[9] Z. Gu, J. Zheng, N. Ling, P. Zhang, Low complexity Bi-Partition mode selection for 3D video depth intra coding, Displays 40 (2015) 2–8.
[10] K. Lee, T.S. Kim, C.E. Rhee, H.-J. Lee, Simplified algorithms for rate-distortion optimization in high efficiency video coding, Displays 40 (2015) 35–44.
[11] D. Jun, A fast coding unit mode decision method based on the mode inheritance of upper coding unit for low-complexity compression of smart contents, Displays 55 (2018) 3–9.
[12] Y.-F. Cen, W.-L. Wang, X.-W. Yao, A fast CU depth decision mechanism for HEVC, Inform. Process. Lett. 115 (9) (2015) 719–724.
[13] L. Shen, Z. Liu, X. Zhang, W. Zhao, Z. Zhang, An effective CU size decision method for HEVC encoders, IEEE Trans. Multimed. 15 (2) (2012) 465–470.
[14] P.-T. Chiang, T.S. Chang, Fast zero block detection and early CU termination for HEVC video coding, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2013, pp. 1640–1643.

[15] X. Shen, L. Yu, J. Chen, Fast coding unit size selection for HEVC based on Bayesian decision rule, in: 2012 Picture Coding Symposium, IEEE, 2012, pp. 453–456.
[16] H.-S. Kim, R.-H. Park, Fast CU partitioning algorithm for HEVC using an online-learning-based Bayesian decision rule, IEEE Trans. Circuits Syst. Video Technol. 26 (1) (2015) 130–138.
[17] W. Liao, Z. Chen, A fast CU partition and mode decision algorithm for HEVC intra coding, Signal Process., Image Commun. 67 (2018) 140–148.
[18] Y. Huang, D. Wang, Y. Sun, B. Hang, A fast intra coding algorithm for HEVC by jointly utilizing naive Bayesian and SVM, Multimedia Tools Appl. 79 (2020) 33957–33971.
[19] G. Correa, P.A. Assuncao, L.V. Agostini, L.A. da Silva Cruz, Fast HEVC encoding decisions using data mining, IEEE Trans. Circuits Syst. Video Technol. 25 (4) (2014) 660–673.
[20] M. Saldanha, G. Sanchez, C. Marcon, L. Agostini, Configurable fast block partitioning for VVC intra coding using light gradient boosting machine, IEEE Trans. Circuits Syst. Video Technol. 32 (6) (2021) 3947–3960.
[21] K. Kim, W.W. Ro, Fast CU depth decision for HEVC using neural networks, IEEE Trans. Circuits Syst. Video Technol. 29 (5) (2018) 1462–1473.
[22] G. Tech, J. Pfaff, H. Schwarz, P. Helle, A. Wieckowski, D. Marpe, T. Wiegand, Fast partitioning for VVC intra-picture encoding with a CNN minimizing the rate-distortion-time cost, in: 2021 Data Compression Conference (DCC), IEEE, 2021, pp. 3–12.
[23] M. Xu, T. Li, Z. Wang, X. Deng, R. Yang, Z. Guan, Reducing complexity of HEVC: A deep learning approach, IEEE Trans. Image Process. 27 (10) (2018) 5044–5059.
[24] A. Tissier, W. Hamidouche, S.B.D. Mdalsi, J. Vanne, F. Galpin, D. Menard, Machine learning based efficient QT-MTT partitioning scheme for VVC intra encoders, IEEE Trans. Circuits Syst. Video Technol. (2023).
[25] T. Li, M. Xu, R. Tang, Y. Chen, Q. Xing, DeepQTMT: A deep learning approach for fast QTMT-based CU partition of intra-mode VVC, IEEE Trans. Image Process. 30 (2021) 5377–5390.
[26] A. Feng, K. Liu, D. Liu, L. Li, F. Wu, Partition map prediction for fast block partitioning in vvc intra-frame coding, IEEE Trans. Image Process. (2023).
[27] Y. Fan, H. Sun, J. Katto, J. Ming'E, et al., A fast QTMT partition decision strategy for VVC intra prediction, IEEE Access 8 (2020) 107900–107911.
[28] Y. Song, B. Zeng, M. Wang, Z. Deng, An efficient low-complexity block partition scheme for VVC intra coding, J. Real-Time Image Process. (2022) 1–12.
[29] G. Wu, Y. Huang, C. Zhu, L. Song, W. Zhang, SVM based fast CU partitioning algorithm for VVC intra coding, in: 2021 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2021, pp. 1–5.
[30] Q. Zhang, Y. Wang, L. Huang, B. Jiang, X. Wang, Fast CU partition decision for H.266/VVC based on the improved DAG-SVM classifier model, Multimedia Syst. 27 (2021) 1–14.
[31] M. Tahir, I.A. Taj, P.A. Assuncao, M. Asif, Fast video encoding based on random forests, J. Real-Time Image Process. 17 (4) (2020) 1029–1049.
[32] T. Amestoy, A. Mercat, W. Hamidouche, C. Bergeron, D. Menard, Random forest oriented fast QTBT frame partitioning, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 1837–1841.
[33] T. Amestoy, A. Mercat, W. Hamidouche, D. Menard, C. Bergeron, Tunable VVC frame partitioning based on lightweight machine learning, IEEE Trans. Image Process. 29 (2019) 1313–1328.
[34] Q. He, W. Wu, L. Luo, C. Zhu, H. Guo, Random forest based fast CU partition for VVC intra coding, in: 2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), IEEE, 2021, pp. 1–4.
[35] J. Xiong, H. Li, F. Meng, S. Zhu, Q. Wu, B. Zeng, MRF-based fast HEVC inter CU decision with the variance of absolute differences, IEEE Trans. Multimed. 16 (8) (2014) 2141–2153.
[36] T. Fu, H. Zhang, F. Mu, H. Chen, Fast CU partitioning algorithm for H.266/VVC intra-frame coding, in: 2019 IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2019, pp. 55–60.
[37] K. Lim, J. Lee, S. Kim, S. Lee, Fast PU skip and split termination algorithm for HEVC intra prediction, IEEE Trans. Circuits Syst. Video Technol. 25 (8) (2014) 1335–1346.
[38] S. Zhang, S. Feng, J. Chen, C. Zhou, F. Yang, A GCN-based fast CU partition method of intra-mode VVC, J. Vis. Commun. Image Represent. 88 (2022) 103621.
[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[40] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, Adv. Neural Inf. Process. Syst. 30 (2017).