In-Loop Filtering via Trained Look-Up Tables

Zhuoyuan Li, Jiacheng Li, Yao Li, Li Li, Dong Liu✉, and Feng Wu
University of Science and Technology of China (USTC), Hefei, Anhui 230027, China
{zhuoyuanli, jclee, mrliyao}@mail.ustc.edu.cn, {lil1, dongeliu, fengwu}@ustc.edu.cn

Abstract*—In-loop filtering (ILF) is a key technology in image/video coding for reducing the artifacts. Recently, neural network-based in-loop filtering methods achieve remarkable coding gains beyond the capability of advanced video coding standards, establishing themselves a promising candidate tool for future standards. However, the utilization of deep neural networks (DNN) brings high computational complexity and raises high demand of dedicated hardware, which is challenging to apply into general use. To address this limitation, we study an efficient in-loop filtering scheme by adopting look-up tables (LUTs). After training a DNN with a predefined reference range for in-loop filtering, we cache the output values of the DNN into a LUT via traversing all possible inputs. In the coding process, the filtered pixel is generated by locating the input pixels (to-be-filtered pixel and reference pixels) and interpolating between the cached values. To further enable larger reference range within the limited LUT storage, we introduce an enhanced indexing mechanism in the filtering process, and a clipping/finetuning mechanism in the training. The proposed method is implemented into the Versatile Video Coding (VVC) reference software, VTM-11.0. Experimental results show that the proposed method, with three different configurations, achieves on average 0.13%∼0.51%, and 0.10%∼0.39% BD-rate reduction under the all-intra (AI) and random-access (RA) configurations respectively. The proposed method incurs only 1%∼8% time increase, an additional computation of 0.13∼0.93 kMAC/pixel, and 164∼1148 KB storage cost for a single model. *Our method has explored a new and more practical approach for neural network-based ILF.

***Index Terms*—In-loop filtering, deep neural network, Look-up Table (LUT), video coding, VVC.**

I. Introduction

In-loop filtering (ILF) has been widely adopted in modern video coding standards, including H.266/VVC [1], AV2 [2]. To promote the reconstruction quality of decoded frame, various complementary filters make a major contribution to these standards and play a key role in hybrid video coding framework, such as deblocking filter (DBF), sample adaptive offset (SAO), adaptive loop filtering (ALF) [3].

Recently, deep neural network-based (DNN) coding tools (e.g. intra prediction, ILF, etc.) have been rapidly developed [4]–[18], and made good progress in some standardization activities, such as neural network-based video coding (NNVC) [7]. The DNN-based tools sufficiently take advantage of data-driven capabilities to better fit the prediction or reconstruction goals. Although these deep tools have made impressive performance, they bring heavy time and computational complexity that makes them difficult to use in practice without high-performance hardware, and this is one of the major obstacles for practical deep tools.

✉: Corresponding author.

To address this limitation, we propose an efficient and practical in-loop filtering scheme by adopting the *Look-up Table* (LUT), which is inspired by explorations in image/video recovery tasks [19]–[23]. The basic idea of the proposed scheme is to adopt the look-up operation (direct addressing) of LUT to replace the inference process of DNN in coding process, which is also friendly for embedded systems to accelerate computation with far fewer floating-point operations. To achieve this goal, we establish a LUT-based in-loop filtering framework (termed ***LUT-ILF***), and introduce a series of LUT-related modules to strengthen its efficiency, including the enhancement of filtering reference range with the limited LUT size (*progressive indexing and reference indexing*, Section III), the optimization of LUT size with limited memory cost (*clipping/finetuning*, Section II), the selection of reference pixels (*learnable weighting*, Section III). Compared to the low/high complexity operation point setting (LOP/HOP) of NNVC-ILF [24]–[26], our ultrafast mode (***LUT-ILF-U***, reference range: 5×5, 0.13 kMACs/pixel, 164 KB), very fast mode (***LUT-ILF-V***, reference range: 9×9, 0.40 kMACs/pixel, 492 KB) and fast mode (***LUT-ILF-F***, reference range: 13×13, 0.93 kMACs/pixel, 1148 KB) provide a series of new trade-off points that show lower time and computational complexity and good performance beyond VVC.

The remainder of this paper is organized as follows. First, we introduce the basic framework (***LUT-ILF***). Second, we introduce the enhanced framework and each module of ***LUT-ILF-U/V/F***. Third, we show the comprehensive evaluation of proposed framework. Finally, we discuss the future work of *LUT-ILF* scheme and put forward future improvements.

II. Basic Framework of LUT-ILF

In this section, we introduce the basic framework of *LUT-ILF*. As shown in Fig. 1, it contains four stages: training filtering network, caching the filtering network into LUT, finetuning of filtering LUT, retrieval of filtering LUT. The cooperation of the above stages realizes the whole filtering process, here we introduce them one by one.

Stage 1: Training Filtering Network. First, due to the size of LUT grows exponentially as the dimension of indexing entries (i.e., target pixel with reference pixels) increases, the lightweight filtering network is trained with the constraint of a small reference range (receptive field, RF) in an end-to-end manner. Here we take the 2×2 reference range (4D LUT) as an example, and the process is shown in stage 1 of Fig. 1, the *target pixel (to-be-filtered/reconstructed pixel, I_0) with three surrounding reference pixels (solid line)* serves as the input to the network. To enlarge the size of RF, the rotation ensemble

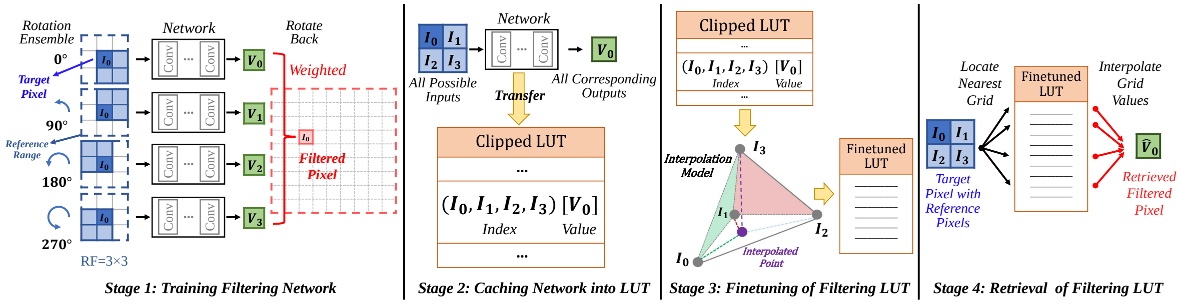

Fig. 1. Illustration of the basic framework of look-up table-based in-loop filtering framework (*LUT-ILF*).

trick is used to cover the 3×3 reference range (dotted line). The final output value (*filtered pixel*) is averaged by all outputs of the 4 rotations ($V_0 \sim V_3$). In training, the filtered and original pixels form a pair, which is supervised by MSE loss.

Stage 2 & Stage 4: Caching Network into LUT & Retrieval of Filtering LUT. Second, with the network being trained, the 4D LUT is transferred and cached from the output values of the network via traversing all possible inputs (*target pixel with reference pixels*, [0∼255][0∼255][0∼255][0∼255] for int8 case of input), as shown in stage 2 of Fig. 1. Note that the storage of LUT with a large input/output range will bring heavy storage cost, for example, the full size of 4D LUT is calculated as $256^4 \times 1 \times 8$ bit = 4096 MB (4 GB), 256^4 bins for possible input value (0∼255), 1 for 8-bit output value. To avoid the heavy storage cost, the indexes of full LUT are uniformly sampled and stored in the small LUT (named *Clipped* LUT), which only caches the output value of the *most significant bits* (MSB) of the input pixel value. In our design, the 8-bit input pixel value is uniformly sampled to 4 MSBs, and the 4 MSBs serve as the initial (nearest) index for the indexing of input pixel. The input/output range of indexing is degraded to [0,16...240,255][0,16...240,255][0,16...240,255][0,16...240, 255], and the size of *Clipped* LUT is calculated as $17^4 \times 1 \times 8$ bit = 81.56 KB. In the retrieval process of finetuned filtering LUT, with the indexing of the MSB of input pixels (I_0, I_1, I_2, I_3) in 4D *Clipped* LUT, the obtained output values of the nearest index and *least significant bits* (LSB) of the input pixels are used to interpolate the final *retrieved filtered pixel* by linear interpolation model. For the interpolation method of *Clipped* LUT, we follow the same model as [19]–[23], and use the 4-Simplex interpolation model.

Stage 3: Finetuning of Filtering LUT. To compensate for the degradation of LUT *Clipping*, the finetuning of *Clipped* LUT is performed to adapt to the uniform sampling and the interpolation model, facilitating the interpolation of the final *retrieved filtered pixel* value of non-sampled indexes of LUT from the nearest sampled indexes of LUT. In finetuning, the values of *Clipped* LUT are activated as the trainable parameters and finetuned by the same setting of filtering network training.

III. LUT-ILF with Reference, Progressive, Weighted Indexing Mechanism

For the basic framework (*LUT-ILF*), the efficiency of *LUT-ILF* is mainly subject to two aspects. First, the filtering reference range (RF, only 3×3) is limited with the constraint of LUT size, which is verified as an important factor in traditional filtering tools (such as ALF [27] with 7×7 reference range). Second, the selection of reference pixels is very relevant to the filtering (such as the ALF [27] with a diamond shape). To address these limitations, inspired by [20], [22], the ***reference, progressive, weighted indexing mechanism*** is introduced to enhance the above issues. Here, we detail them and serve the *LUT-ILF-V* as an example, the framework is shown in Fig. 3.

Module 1: Reference Indexing. First, the reference pixel range is enlarged to further take advantage of surrounding information for the filtering of target (to-be-filtered) pixel. To avoid the exponential growth in the size of LUT with the dimension, the complementary reference indexing is used to increase the reference range of target pixel by parallelizing more complementary indexing patterns to address more reference pixels and capture the rich local structures. As shown in Fig. 2, it can cover a wide reference range. In *LUT-ILF-V*, besides the standard indexing pattern of *LUT-ILF* (*Pattern 1*, RF=3×3), complementary *Pattern 2* and *Pattern 3* are used to cover the 5×5 reference range. For the patterns of *LUT-ILF-F*, it can cover the 7×7 reference range.

In this way, the total size of cached LUTs grows linearly (3 times a 4D *Clipped* LUT, $3 \times 17^4 \times 1 \times 8$ bit = 244.69 KB), instead of exponentially (the full size of a 25D LUT with an equivalent 5×5 reference range is $256^{(25-4)}$ times a 4D LUT), in a single stage of *reference indexing mechanism*.

Module 2: Progressive Indexing. Second, the reference pixel range of the *to-be-filtered pixel* is further enlarged by introducing the cascaded filtering LUTs with *progressive indexing*. As shown in Fig. 3, in the whole filtering process of *LUT-ILF-V*, the *re-indexing mechanism* is used to link the cascaded framework between multiple 4D LUTs. In the detailed retrieval process of cascaded filtering LUTs, with the filtering of *target pixel* by multiple indexing patterns (5×5 reference range) in stage 1 of *progressive indexing*, the *filtered pixel* of stage 1 contains the local information of 5×5 reference range implicitly. By *shifting* the filtering window in the 9×9 reference range, the local information of 9×9 reference range can be aggregated into a 5×5 *aggregated reference range*. In stage 2 of *progressive indexing*, the *re-indexing mechanism* can be used to filter the *target pixel* in the *aggregated reference pixels* to achieve the larger reference range implicitly. The process of *progressive indexing* is similar to cascading multiple convolutional layers in a neural network and achieving information aggregation in the feature domain.

Above these ways, with the utilization of *reference* and *progressive indexing*, the total size of cached LUTs is linear to its indexing capacity (6 times a *Clipped* 4D LUT, $6 \times 17^4 \times 1 \times 8$

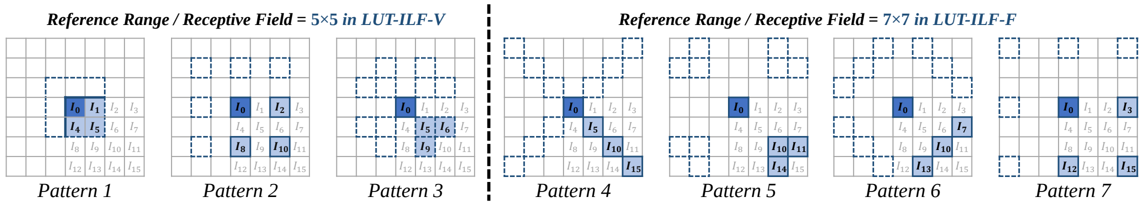

Fig. 2. Illustration of patterns of complementary reference indexing in *LUT-ILF-U* (only *Pattern 1*), *LUT-ILF-V* (*Pattern 1∼3*), and *LUT-ILF-F* (*Pattern 1∼7)*. With the use of proposed indexing patterns, *LUT-ILF* can involve and address more reference pixels. For example, with *Pattern 1∼3*, the 5×5 reference range around I_0 is fully covered in *LUT-ILF-V*. With *Pattern 1∼7*, the 7×7 reference range around I_0 is fully covered in *LUT-ILF-F*. The covered reference pixels with the rotation ensemble trick are marked with dashed boxes.

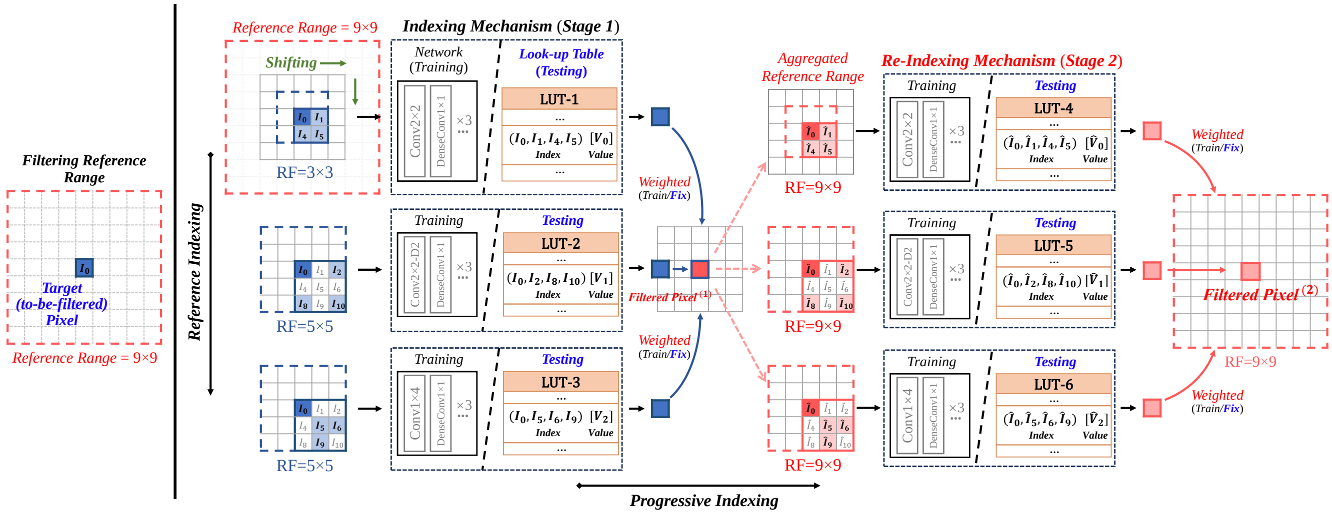

Fig. 3. Illustration of the *LUT-ILF-V* framework, it contains two parts. On the left, the input (to-be-filtered) pixel with the filtering reference range is shown; On the right, the process of *LUT-ILF-V* is shown, the parallel and cascaded networks/LUTs are performed with *reference* and *progressive indexing* at the training/testing. The covered reference range of each pattern with the rotation trick is marked with dashed boxes. For training, the convolution of each pattern can be implemented with standard convolutions and *unfold/reshape* operations. The Conv2×2-D2 denotes the convolutional layer with a dilation size of 2.

bit = 489.38 KB), instead of exponentially (the full size of an 81D LUT with an equivalent 9×9 reference range is $256^{(81-4)}$ times a 4D LUT), in the whole process of very fast setting of *LUT-ILF* (*LUT-ILF-V*). For the ultrafast setting (*LUT-ILF-U*), the total size of cached LUTs is 0.33× *LUT-ILF-V*'s size. For the fast setting (*LUT-ILF-F*), the total size of cached LUTs is 2.3× *LUT-ILF-V*'s size, instead of exponentially (the full size of a 169D LUT with an equivalent 13×13 reference range is $256^{(169-4)}$ times a 4D LUT).

Module 3: Learnable Weighting. Third, with the extension of reference range, the impact of reference pixels on the *target pixel* should be considered. Instead of the direct average of *filtered pixel* of different indexing patterns, the weights of different indexing patterns are activated as the trained parameters and normalized to [0, 1] with the *softmax*() function to adaptively fit the importance of different reference pixels in the training of filtering network. At the test time, the weights of different patterns are fixed and used by integer operation.

Summary: General Retrieval Formula. Finally, we formulate the retrieval of filtering LUT with utilization of *clipping, reference, progressive, weighted indexing mechanism* in the whole process of *LUT-ILF*. In stage 1 of *LUT-ILF*, for the *target pixel* I_0 *with surrounding reference pixels*, the *filtered pixel* can be addressed and calculated by

$$Filtered\ Pixel^{(1)} = (W_1^{(1)} \times LUT_{*p1}^{(1)}[I_0][I_1][I_4][I_5] + W_2^{(1)} \times LUT_{*p2}^{(1)} [I_0][I_2][I_8][I_{10}] + \cdots + W_n^{(1)} \times LUT_{*pn}^{(1)}[\cdot][\cdot][\cdot][\cdot]\cdots)/n \tag{1}$$

where (1) denotes the stage number of LUT, n denotes the number of indexing patterns, $LUT_*[\cdot]$ denotes the look-up and interpolation process of LUT retrieval, Pn denotes the pattern ID, W_n denotes the weights of different indexing patterns.

In stage 2, the final *filtered pixel* can be addressed and calculated by

$$Filtered\ Pixel^{(2)} = (W_1^{(2)} \times LUT_{*p1}^{(2)}[\widehat{I}_0][\widehat{I}_1][\widehat{I}_4][\widehat{I}_5] + W_2^{(2)} \times LUT_{*p2}^{(2)}[\widehat{I}_0][\widehat{I}_2][\widehat{I}_8][\widehat{I}_{10}] + \cdots + W_n^{(2)} \times LUT_{*pn}^{(2)}[\widehat{\cdot}][\widehat{\cdot}][\widehat{\cdot}][\widehat{\cdot}]\cdots)/n \tag{2}$$

where the value $(\widehat{I})$ denotes the output value of the previous filtering stage that serves as the index of the following LUT.

IV. Rate-Distortion-Optimization of LUT-ILF

For the integration of *LUT-ILF* into the filtering process of VVC (DBF, SAO, ALF), we set it at the end of all filtering processes, and the decision flag of *LUT-ILF* is signaled in the Coding Tree Unit (CTU) level to indicate the use of proposed method. The flag is determined by the rate-distortion (RD) cost function that $J = SSD + \lambda \times R_{flag}$, where R_{flag} denotes the rates of decision flag in CABAC-based rate estimation, SSD denotes the sum of squared differences (SSD) between the reconstructed result and filtering result of *LUT-ILF*.

V. Experiment

In our experiment, the VVC reference software VTM-11.0 is used as the baseline. The codec adopts the configuration of all intra (AI) and random access (RA) according to the VVC Common Test Condition (CTC). The test sequences from classes A to E with different resolutions are tested as specified in [31], [32]. For each test sequence, quantization parameter

TABLE I

BD-RATE AND DIFFERENT COMPLEXITY RESULTS OF PROPOSED METHOD, AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING METHODS UNDER AI AND RA CONFIGURATIONS

Methods	BD-Rate (AI)	BD-Rate (RA)	Computational Complexity	Storage Cost	Energy Cost[2]	Time Complexity (enc/dec, CPU)
NNVC-LOP[1] (VTM-11.0)	-4.61%∼-4.78%	-5.20%∼-5.37%	17.0 kMACs/pixel	129.98 KB (*int16*)	11900 *pJ* (*int16*)	108%/4717%∼109%/4724% (AI)
				228.33 KB (*float*)	78200 *pJ* (*float*)	114%/8274%∼114%/8322% (RA)
NNVC-HOP[1] (VTM-11.0)	-7.79%∼-7.91%	-10.12%∼-10.31%	477.0 kMACs/pixel	2826.2 KB (*int16*)	333900 *pJ* (*int16*)	133%/24372%∼276%/134057% (AI)
				7444.5 KB (*float*)	2194200 *pJ* (*float*)	159%/43509%∼399%/227720% (RA)
***LUT-ILF-U* (VTM-11.0)**	**-0.13%**	**-0.10%**	**0.13 kMACs/pixel**	**164 KB (*int8*)[3]**	**180.2 *pJ***	**101%/102% (AI), 101%/105% (RA)**
***LUT-ILF-V* (VTM-11.0)**	**-0.34%**	**-0.27%**	**0.40 kMACs/pixel**	**492 KB (*int8*)**	**497.2 *pJ***	**102%/103% (AI), 103%/106% (RA)**
***LUT-ILF-F* (VTM-11.0)**	**-0.51%**	**-0.39%**	**0.93 kMACs/pixel**	**1148 KB (*int8*)**	**1163.25 *pJ***	**102%/106% (AI), 104%/108% (RA)**

[1] The results of BD-rate, time complexity, computational complexity, storage cost (int/float model) are cited from [26] (LOP) / [24], [25] (HOP) and open-sourced repository.

[2] The energy cost is calculated according to [28]–[30]. For addition, *int8/int16/float32* corresponds to 0.03/0.05/0.9 pJ. For multiplication, the operation of *int8/float16/float32* corresponds to 0.2/1.1/3.7 pJ. Since the multiplication of *int16* is not reported in [29], it is referred to as median of the energy of *int8* and *float16*. For the energy cost of NNVC-ILF, the results are directly calculated by their computational complexity.

[3] The storage cost of a single model of *LUT-ILF* is shown.

TABLE II

ABLATION STUDY OF *LUT-ILF-V/F* UNDER AI CONFIGURATION

Class	w/o *PI*	w/o *LW*	w/o *RDO*	*LUT-ILF-V/F*
A	-0.09% / -0.19%	-0.25% / -0.39%	1.89% / 1.06%	-0.30% / -0.45%
B	-0.08% / -0.15%	-0.19% / -0.30%	2.21% / 1.32%	-0.23% / -0.40%
C	-0.07% / -0.13%	-0.23% / -0.34%	0.64% / 0.31%	-0.29% / -0.39%
D	-0.18% / -0.27%	-0.46% / -0.60%	0.12% / -0.29%	-0.52% / -0.70%
E	-0.13% / -0.21%	-0.34% / -0.59%	2.86% / 1.32%	-0.44% / -0.63%
Avg.	**-0.10% / -0.17%**	**-0.28% / -0.43%**	**1.55% / 0.77%**	**-0.34% / -0.51%**

TABLE III

CTU-LEVEL USAGE RATIO OF *LUT-ILF-V/F* UNDER AI CONFIGURATION

Class	A	B	C	D	E	Avg.
LUT-ILF-V	31.81%	27.13%	39.44%	49.18%	27.07%	**34.41%**
LUT-ILF-F	43.92%	37.88%	48.10%	58.87%	39.13%	**45.31%**

TABLE IV

BD-RATE RESULTS ON LOW-BITRATE POINTS UNDER AI CONFIGURATION

Class	A	B	C	D	E	Avg.
LUT-ILF-V	-0.70%	-0.48%	-0.64%	-1.08%	-0.95%	**-0.74%**
LUT-ILF-F	-1.01%	-0.76%	-0.84%	-1.40%	-1.32%	**-1.03%**

(QP) values are set to 22, 27, 32, 37, 42, and Bjontegaard Delta-rate (BD-rate) [33] is used as an objective metric to evaluate coding performance. For the complexity metrics, *time complexity*, *computational complexity* (kMACs/pixel [32]), *theoretical energy cost* (pJ [28]–[30]), and *storage cost* (KB) are evaluated. For the training setup of *LUT-ILF-U/V/F*, as shown in Fig. 3 , the network is designed as 4 dense convolutions with 1 convolutions in each stage, only the size of the first layer is different to adapt to different shape of pattern, and the BVI-DVC, DIV2K are used as the training datasets [34], [35]. For different QPs, the LUT is trained separately. The experimental results and the comparison with other methods are shown in Table I.

Performance Analysis: From Table I, we can find that the different modes (*ultrafast, very fast, fast*) of our proposed *LUT-ILF* provide a series of new trade-off points between the performance and efficiency for practical applications. For the quantitative comparisons of performance and complexity, the computational complexity and decoding time complexity of *LUT-ILF* are $130\times\sim3600\times$ and $46\times\sim2200\times$ lower than that of popular NN-based ILF methods [24]–[26], and *LUT-ILF* also shows good performance potential.

Ablation Study: To validate the contributions of core modules in our scheme, we conduct the ablation experiments on

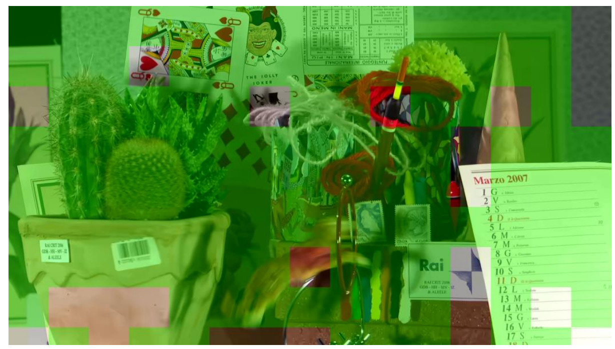

Fig. 4. The selection results of *LUT-ILF-F* of $Cactus$ 1920×1080 on VTM-11.0 (AI configuration, QP:32, POC:29), the green block indicates the block filtered by *LUT-ILF*.

proposed *progressive indexing* (*PI*), *learnable weighting* (*LW*), and the CTU-level RDO (*RDO*), under AI configuration. As shown in Table II, for the comparison of variants and *LUT-ILF*, the results verify the effectiveness of the proposed modules.

Usage Ratio: To verify the efficiency of *LUT-ILF*, we evaluate its usage ratio (Table III), which is calculated by, $Ratio = N_{test}/N_{total}$, where N_{test} indicates the number of filtered CTU, and N_{total} indicates the total number of CTUs. The selection results are also shown in Fig. 4, representing that the *LUT-ILF* can better handle the complex texture regions.

Low-bitrate Points Exploration: To further explore the potential of proposed method, we test our proposed method on low bitrate points (QP 27∼47), as shown in Table IV. The results verify the powerful potential of the proposed method.

VI. CONCLUSION

In this paper, we propose an efficient look-up table-based ILF method, which adopts the strong fitting ability of deep neural networks to model the compact look-up tables for ILF. For practical application, the use of *LUT-ILF* does not need to rely on high-performance hardware and devices. The experimental results of *LUT-ILF* demonstrate it can achieve a good performance with low time/computational complexity in VVC, which provides a new practical way for neural network-based video coding tools in the future. For future work, we will further extend the proposed method to improve the performance of more coding tools, such as the interpolation of fractional-pixel motion estimation [36], [37], reference picture resampling [38], etc.

ACKNOWLEDGMENT

This work was supported in part by the Natural Science Foundation of China under Grant 62021001, and in part by the Fundamental Research Funds for the Central Universities under Grant WK3490000006.

We acknowledge the support of GPU and CPU cluster built by MCC Lab of Information Science and Technology Institution, USTC.

REFERENCES

[1] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, “Overview of the versatile video coding (VVC) standard and its applications,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[2] AOM, “AOMedia Video 1 and 2 (AV1 and AV2),” *https://aomedia.org/*, 2015.

[3] M. Karczewicz, N. Hu, J. Taquet, C.-Y. Chen, K. Misra, K. Andersson, P. Yin, T. Lu, E. François, and J. Chen, “VVC in-loop filters,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3907–3925, 2021.

[4] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, “Deep learning-based video coding: A review and a case study,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–35, 2020.

[5] Y. Dai, D. Liu, and F. Wu, “A convolutional neural network approach for post-processing in HEVC intra coding,” in *MultiMedia Modeling: 23rd International Conference, MMM 2017, Reykjavik, Iceland, January 4-6, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 28–39.

[6] Y. Dai, D. Liu, Z.-J. Zha, and F. Wu, “A CNN-based in-loop filter with CU classification for HEVC,” in *2018 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2018, pp. 1–4.

[7] Y. Li, J. Li, C. Lin, K. Zhang, L. Zhang, F. Galpin, T. Dumas, H. Wang, M. Coban, J. Ström *et al.*, “Designs and implementations in neural network-based video coding,” *arXiv preprint arXiv:2309.05846*, 2023.

[8] Y. Li, L. Zhang, and K. Zhang, “Convolutional neural network based in-loop filter for VVC intra coding,” in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 2104–2108.

[9] Y. Li, Y. Yi, D. Liu, L. Li, Z. Li, and H. Li, “Neural-network-based cross-channel intra prediction,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 17, no. 3, pp. 1–23, 2021.

[10] Y. Li, D. Liu, H. Li, L. Li, F. Wu, H. Zhang, and H. Yang, “Convolutional neural network-based block up-sampling for intra frame coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 9, pp. 2316–2330, 2017.

[11] Y. Li, L. Zhang, K. Zhang, Y. He, and J. Xu, “AHG11: Convolutional neural networks-based in-loop filter,” *Doc. JVET-T0088, Joint Video Exploration Team (JVET)*, 2020.

[12] Y. Li *et al.*, “EE1-1.6: RDO considering deep in-loop filtering,” *JVET-AB0068*, 2015.

[13] ——, “EE1-1.7: Deep in-loop filter with additional input information,” *JVET-AC0177*, 2015.

[14] Y. Zhao, S. Wang, K. Lin, M. Lei, C. Jia, S. Wang, and S. Ma, “Towards next generation video coding: from neural network based predictive coding to in-loop filtering,” in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[15] S. Ma, J. Gao, R. Wang, J. Chang, Q. Mao, Z. Huang, and C. Jia, “Overview of intelligent video coding: from model-based to learning-based approaches,” *Visual Intelligence*, vol. 1, no. 1, p. 15, 2023.

[16] Y. Li, L. Zhang, and K. Zhang, “iDAM: Iteratively trained deep in-loop filter with adaptive model selection,” *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 19, no. 1s, pp. 1–22, 2023.

[17] D. Ding, J. Wang, G. Zhen, D. Mukherjee, U. Joshi, and Z. Ma, “Neural adaptive loop filtering for video coding: Exploring multi-hypothesis sample refinement,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 10, pp. 6057–6071, 2023.

[18] B. Kathariya, Z. Li, and G. Van der Auwera, “Joint pixel and frequency feature learning and fusion via channel-wise transformer for high-efficiency learned in-loop filter in VVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 5, pp. 4070–4083, 2023.

[19] Y. Jo and S. J. Kim, “Practical single-image super-resolution using look-up table,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 691–700.

[20] J. Li, C. Chen, Z. Cheng, and Z. Xiong, “Mulut: Cooperating multiple look-up tables for efficient image super-resolution,” in *European conference on computer vision*. Springer, 2022, pp. 238–256.

[21] G. Liu, Y. Ding, M. Li, M. Sun, X. Wen, and B. Wang, “Reconstructed convolution module based look-up tables for efficient image super-resolution,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 12 217–12 226.

[22] J. Li, C. Chen, Z. Cheng, and Z. Xiong, “Toward dnn of luts: Learning efficient image restoration with multiple look-up tables,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[23] G. Yin, Z. Qu, X. Jiang, S. Jiang, Z. Han, N. Zheng, H. Yang, X. Liu, Y. Yang, D. Li *et al.*, “Online streaming video super-resolution with convolutional look-up table,” *IEEE Transactions on Image Processing*, vol. 33, pp. 2305–2317, 2024.

[24] F. Galpin *et al.*, “JVET AHG report: NNVC software development AhG14,” *JVET-AF0014*, 2023.

[25] ——, “AhG11: HOP Full Results,” *JVET-AF0041*, 2023.

[26] D. Rusanovskyy *et al.*, “AHG11: Status of the Joint EE1-0 (LOP.2) Unified Filter Training,” *JVET-AF0043*, 2023.

[27] C.-Y. Tsai, C.-Y. Chen, T. Yamakage, I. S. Chong, Y.-W. Huang, C.-M. Fu, T. Itoh, T. Watanabe, T. Chujoh, M. Karczewicz *et al.*, “Adaptive loop filtering for video coding,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 934–945, 2013.

[28] D. Song, Y. Wang, H. Chen, C. Xu, C. Xu, and D. Tao, “Addersr: Towards energy efficient image super-resolution,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 15 648–15 657.

[29] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[30] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[31] K. Suehring and X. Li, “JVET Common Test Conditions and Software Reference Configurations,” *JVET document, JVET-G1010*, 2017.

[32] S. Liu, A. Segall, E. Alshina, and R.-L. Liao, “JVET common test conditions and evaluation procedures for neural network-based video coding technology,” in *JVET 24th Meeting, document JVET-X2016*, 2021.

[33] G. Bjontegaard, “Calculation of Average PSNR Differences between RD-curves,” *ITU-T VCEG-M33, April, 2001*, 2001.

[34] D. Ma, F. Zhang, and D. R. Bull, “BVI-DVC: A training database for deep video compression,” *IEEE Transactions on Multimedia*, vol. 24, pp. 3847–3858, 2021.

[35] E. Agustsson and R. Timofte, “Ntire 2017 challenge on single image super-resolution: Dataset and study,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 126–135.

[36] N. Yan, D. Liu, H. Li, B. Li, L. Li, and F. Wu, “Convolutional neural network-based fractional-pixel motion compensation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 3, pp. 840–853, 2018.

[37] ——, “Invertibility-driven interpolation filter for video coding,” *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 4912–4925, 2019.

[38] T. Fu, K. Zhang, L. Zhang, S. Wang, and S. Ma, “An efficient framework of reference picture resampling (RPR) for video coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 10, pp. 7107–7119, 2022.