

# In-Loop Filtering Using Learned Look-Up Tables for Video Coding

Zhuoyuan Li, Jiacheng Li, Yao Li, Jialin Li, Li Li, *Senior Member, IEEE*, Dong Liu, *Senior Member, IEEE*, and Feng Wu, *Fellow, IEEE*

**Abstract**—In-loop filtering (ILF) is a key technology in video coding standards to reduce artifacts and enhance visual quality. Recently, neural network-based ILF schemes have achieved remarkable coding gains, emerging as a powerful candidate for next-generation video coding standards. However, the use of deep neural networks (DNN) brings significant computational and time complexity or high demands for dedicated hardware, making it challenging for general use. To address this limitation, we study a practical ILF solution by adopting look-up tables (LUTs). After training a DNN with a restricted reference range for ILF, all possible inputs are traversed, and the output values of the DNN are cached into LUTs. During the coding process, the filtering process is performed by simply retrieving the filtered pixel through locating the input pixels and interpolating between the cached values, instead of relying on heavy inference computations. In this paper, we propose a universal LUT-based ILF framework, termed LUT-ILF++. First, we introduce the cooperation of multiple kinds of filtering LUTs and propose a series of customized indexing mechanisms to enable better filtering reference perception with limited storage consumption. Second, we propose the cross-component indexing mechanism to enable the filtering of different color components jointly. Third, in order to make our solution practical for coding uses, we propose the LUT compaction scheme to enable the LUT pruning, achieving a lower storage cost of the entire solution. The proposed framework is implemented in the Versatile Video Coding reference software. Experimental results show that the proposed framework achieves on average 0.82%/2.97%/1.63% and 0.85%/4.11%/2.06% bitrate reduction for common test sequences, under the all-intra and random-access configurations, respectively. Compared to DNN-based solutions, our proposed solution has much lower time complexity and storage cost.

**Index Terms**—Deep neural network, in-loop filtering, look-up table, Versatile Video Coding, video coding.

## I. INTRODUCTION

In-loop filtering (ILF) has been widely adopted in advanced video coding standards, such as H.265/HEVC [1], H.266/VVC [2], [3], AV1 and AV2 [4], [5]. To enhance the objective and subjective reconstructed quality of decoded frames, various hand-crafted in-loop filters make a major contribution to these standards and play a key role in the hybrid coding framework, such as deblocking filter (DBF) [3], [6], sample adaptive offset (SAO) [3], [7], [8], and adaptive loop filtering (ALF) [3], [9], [10], etc. Recently, deep neural network (DNN) based coding tools (e.g., ILF [11]–[21], intra/inter prediction [22]–[26], reference frame generation [27], sampling [28]–[30], etc.) have been rapidly developed for next-generation video coding standards [31]. Notably, neural network-based ILF (NNLF) has

The authors are with the MOE Key Laboratory of Brain-Inspired Intelligent Perception and Cognition, University of Science and Technology of China, Hefei 230093, China (e-mail: {zhuoyuanli, jclee, mrliyao, jialin.li}@mail.ustc.edu.cn, {lil1, dongliu, fengwu}@ustc.edu.cn). (Corresponding Author: Dong Liu.)

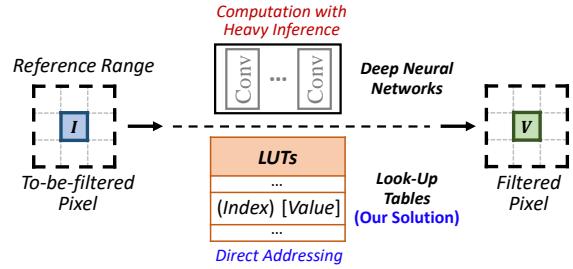


Fig. 1. Illustration of the concept of different deployment manners of DNN-based coding tools in the codec.

emerged as a particularly promising tool, achieving significant progress in advanced standardization activities, such as neural network-based video coding (NNVC) [28] and AV2 [5]. These DNN-based ILF tools leverage data-driven capabilities to learn effective filters and adaptive filtering strategies, surpassing the hand-crafted filtering techniques. However, their significant computational/time complexity, along with high demands for dedicated hardware, pose challenges for practical applications.

To improve their practicality, a series of optimization solutions have been proposed in previous research to facilitate the integration of these DNN-based models into codecs for practical application and deployment, such as lightweight network designs [32]–[35], model ensembling [36]–[40], low-complexity neural operators [41]–[43], dimensionality reduction [44]–[47], computational complexity re-allocation [15], [48]–[50], re-parameterization [42], etc. Particularly, in the standardization activities (such as NNVC [28]), different target computational complexity (operation point) configurations of NNLN, including Very Low Operation Point (VLOP) 1~3 [51]–[53], Low Operation Point (LOP) 1~5 [44], [54]–[57], High Operation Point (HOP) 1~5 [36], [56]–[61], have been introduced and investigated to provide different trade-offs between compression efficiency and computational resource consumption, which allows for flexible adaptation to diverse deployment scenarios. Although the above schemes can reduce and control the network complexity and further improve model efficiency to some extent, the inherently heavy inference computation burden in codecs remains unavoidable.

To address this limitation, inspired by our previous studies in image restoration tasks [62]–[64], we rethink the efficient practical deployment of these DNN-based coding tools in video coding. In our solution, as shown in Fig. 1, we study a look-up table (LUT)-based approach, and our basic concept is to adopt the look-up operation of LUT to replace the heavy computational process of DNN inference in the coding process. To achieve this goal, the DNN is first trained with a restricted reference range for ILF. Then, the output values of the trained DNN are cached into a LUT by traversing

all possible inputs. In the coding process, the filtered pixel is obtained by locating the combination of the to-be-filtered pixel and its reference pixels in the LUT, and interpolating among the cached key values. By introducing a minimal possible storage consumption of LUTs in the codec, we aim to achieve a good trade-off between computational/time complexity and coding performance for practical use, while also being friendly for hardware acceleration with far fewer floating-point operations.

Building upon this basic concept, in our preliminary work [65], we propose a basic **LUT-based in-loop filtering** framework, termed **LUT-ILF**. Inspired by the design principle of typical hand-crafted and DNN-based in-loop filters, we combine the LUT attributes and attempt two explorations: (1) *Filtering Reference*. As a crucial design factor of ILF, the utilization and selection of reference pixels that are related to the to-be-filtered pixel play a significant role in capturing local structures and reducing artifacts. To achieve a good filtering goal while avoiding the bursty growth of LUT storage size as the dimension of indexing entries increases in the reference expansion, a series of customized indexing mechanisms across LUTs is proposed to handle this challenge by linearly stacking multiple LUTs for the access of more reference pixels. (2) *Storage Constraint*. As a crucial factor of LUT for practical use, the storage cost is essential to ensure its feasibility. To achieve the controllable storage cost, customized LUT training and cache strategies are proposed to constrain the storage of each LUT with its entry dimension strictly. The potential of the basic framework has been verified across the different scales of complexity in our preliminary work.

In this paper, we further investigate and address the existing bottlenecks and propose a universal framework, termed **LUT-ILF++**. Specifically, we tackle three bottlenecks: (1) *Filtering Reference Perception*. Due to the constrained cache dimension of index entries of LUT for limited storage cost, it becomes challenging to perceive more effective reference information as the filtering reference range expands, impacting the realization of the optimal filtering goal. (2) *Multi-component Reference Collaboration*. The independent filtering process of each chroma component overlooks the inherent correlations among the different components, impacting the performance of the chroma-component filtering while bringing additional complexity. (3) *Storage Overhead*. Although the customized LUT training and cache strategies can control the storage cost, it is difficult to optimize storage usage based on the specific demands of the filtering goal, impacting its practicality and limiting its flexibility in deployment. To overcome the above existing bottlenecks, we put forward LUT-ILF++. The contributions of this paper are summarized as follows:

- We devise a universal framework, termed LUT-ILF++, for efficient in-loop filtering by activating the multiple functions and cooperative manners of LUTs. Our solution explores a new and more practical way for neural network-based coding tools.
- To overcome the bottleneck of filtering reference perception, we introduce the cooperation of multiple filtering LUTs with customized indexing mechanisms, and design a series of cooperative manners to construct the entire

filtering process, enabling efficient reference modeling.

- To address the cross-component filtering, we propose a cross-component indexing mechanism to enable the collaborative filtering of different components.
- We observe the usage statistics of cached pixel relationships of LUT among different reference ranges. Based on the relationship, we introduce a LUT compaction scheme and propose a LUT pruning strategy with a separable indexing mechanism to enable a lower storage cost of the whole filtering framework in practical use.

## II. PRELIMINARY

In this section, first, we introduce our motivation based on the related works, and retrospect the basic solution of our proposed LUT-based ILF solution in the preliminary work [65]. Second, we analyze the bottlenecks of the LUT-based solution and propose potential directions for further improvement.

### A. Motivation and Basic Solution

LUT has been extensively adopted as an efficient mapping operator, playing a crucial role in the practical applications of image processing tasks, such as photo enhancement [66] and color manipulation [67]–[69]. A LUT comprises a set of discrete index-value pairs, where the indices serve as inputs and the pre-computed values provide the outputs of a complicated function or a series of imaging computations during inference. Their compact structure can be stored in on-device storage, and supports the low-latency, high-throughput execution by just retrieving pre-computed values from the LUT in memory. Recently, with the rapid development of DNNs and their remarkable performance across various image processing tasks, a series of works [62]–[64], [70] have tentatively explored the deployment of DNNs for different image processing tasks more practically, leveraging the efficient structure of LUTs and building the mapping relationships to cache DNN models into the LUTs. These works have verified that the LUT-based solution can preserve decent performance while significantly reducing time and computational complexity, and improving deployment efficiency on the device.

Motivated by the efficiency of LUT structures in imaging tasks and the significant progress of DNN-based coding tools in advanced standardization activities [11]–[22], [27]–[30], we revisit the practical application issue of these DNN-based coding tools in video coding. In our solution, we study an efficient and practical LUT-based coding tool, and adopt the “*space-trading-for-time-and-computation*” strategy that trades a moderate storage cost of LUTs to replace the huge computational overhead of DNN, thereby reducing the computational and time complexity in practical use. We attempt to apply it to the typical ILF tool by caching the filtering mapping of high-performance NNLF. As shown in Fig. 2, our concept is to adopt the look-up operation (direct addressing) of LUT to replace the heavy computational process of DNN inference of NNLF in the coding process. To achieve the filtering goal, our basic solution comprises four stages to achieve the whole filtering process.

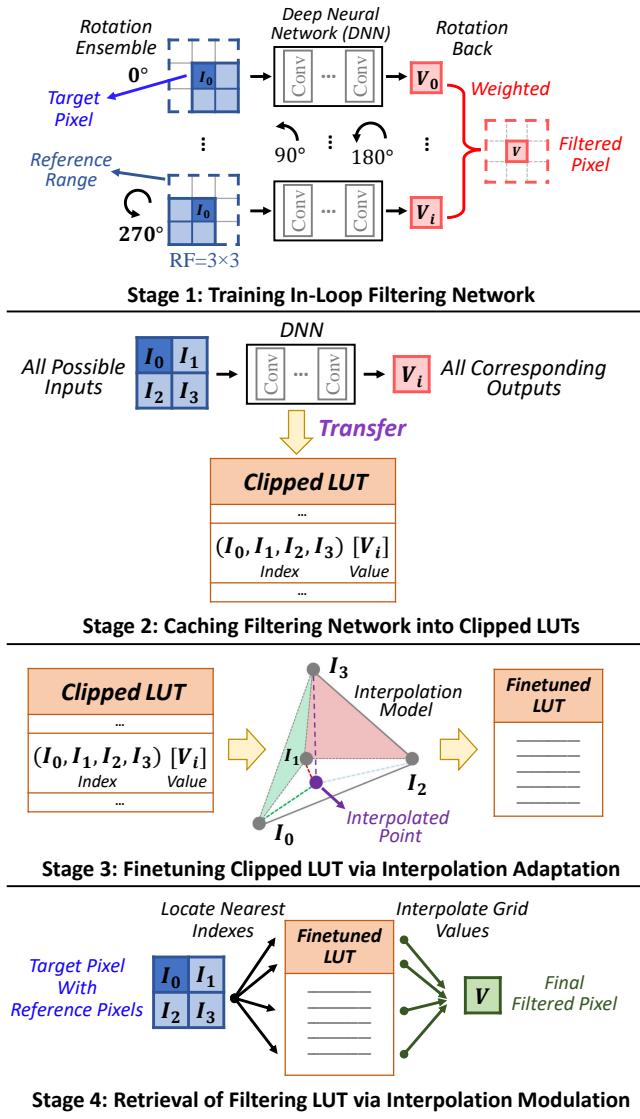


Fig. 2. Overview of the proposed basic LUT-based ILF solution.

1) *Training in-loop filtering network:* First, the filtering network is trained with a restricted reference range/receptive field in an end-to-end manner. As a crucial factor of ILF, the effective relationship modeling between the target (to-be-filtered) pixel and its reference pixels plays a significant role in capturing local structures and reducing artifacts. Unlike training a regular filtering DNN to perceive the pixel relationship of an input patch simply, the LUT-based solution requires caching the complex pixel relationships learned by the network into discrete indexing entries of LUT, which directly impacts on the storage consumption. Due to the storage size of LUT growing exponentially as the dimension of indexing entries (i.e., target pixel with reference pixels) increases, we take the  $2\times 2$  reference range (4D LUT) as the basic unit, and constrain its relationship up to 4 pixels, which can control the cache entries only depending on a small range of input values. For training, the target pixel ( $I_0$ ) with three surrounding reference pixels (solid line) serves as the input to the network. To enlarge the reference range, the rotation ensemble trick is used to cover the  $3\times 3$  reference range (dotted line). The final output value (filtered pixel) is averaged by all outputs of the

four rotation inputs ( $V_0 \sim V_3$ ). During training, the filtered and original pixels form a pair, which is supervised by the mean-squared-error loss.

2) *Caching filtering network into clipped LUTs:* Second, with the filtering network being trained, the filtering LUT (4D) is transferred and cached from the output values of the network via traversing all possible inputs (target pixel with reference pixels,  $[0 \sim 255][0 \sim 255][0 \sim 255][0 \sim 255]$  for uint8 case of input), as shown in Fig. 2. Note that the storage of LUT with a large input/output range will bring heavy storage cost, for example, the full storage size of 4D LUT is  $256^4 \times 1 \times 8$  bit = 4 GB,  $2^{32}$  bins for possible input value, 1 for 8-bit output value. To avoid the heavy storage cost, the indexing entries of the full LUT are uniformly sampled and stored into the 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 pixels. For a clipped LUT, the available indexing entries are reduced to  $[0, 16, \dots, 240, 255][0, 16, \dots, 240, 255][0, 16, \dots, 240, 255][0, 16, \dots, 240, 255]$ , and the size is decreased to  $17^4 \times 1 \times 8$  bit = 81.56 KB.

3) *Finetuning clipped LUT via interpolation adaptation:* Direct sampling of indexing entries can obviously restrict the rapid growth of storage cost, but the non-sampled indexing entries will cause the indexing drift in the retrieval process. For non-sampled entries, we introduce the interpolation model to estimate the drifting entries, such as the trilinear and 4-simplex model, and the interpolation process is performed to calculate the final filtered pixel by locating the nearest neighbor indices (MSBs) of query indices (pixels) and weighting the cached values of neighbor indices during LUT retrieval. To further compensate for the degradation of non-sampled indexing entries, the finetuning of the clipped LUT is further performed to compensate for the adaptation of the uniform sampling and the interpolation model, facilitating the interpolation of the final retrieved filtered pixel of non-sampled indices from the nearest sampled indices. In finetuning, the cached values of the clipped LUT are activated as the trainable parameters and finetuned by the same setting of filtering network training.

4) *Retrieving of filtering LUT via interpolation modulation:* During the retrieval of the filtering LUT in the ILF process, the MSBs of the input pixels ( $I_0, I_1, I_2, I_3$ ) are used to locate the nearest indices in the 4D clipped LUT. The corresponding output values, along with the least significant bits (LSBs) of input pixels, are then fed into a linear interpolation model to modulate the final filtered pixel. For the interpolation scheme of clipped LUT in our framework, we follow the same model as [66], [70], and use the trilinear/4-simplex interpolation model for 3D/4D LUT in the whole filtering framework.

## B. Bottlenecks

Based on our preliminary solution [65], we re-examine the desirable filtering techniques in emerging coding standards, and we identify three bottlenecks of our proposed LUT-based solution in achieving a practical yet effective alternative.

1) *Filtering Reference Perception:* Reference perception modeling has been verified as a crucial factor of filtering

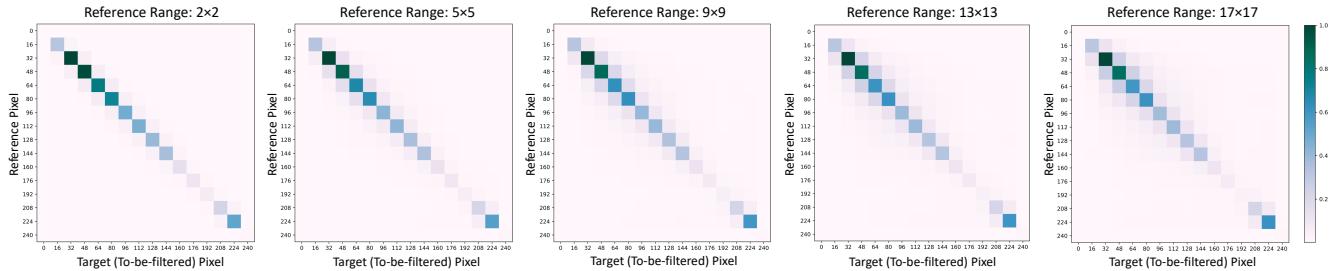


Fig. 3. Pixel value distributions, obtained from pre-filtering frames under All Intra (AI) configuration with QPs of 22, 27, 32, 37, 42, 47 within  $2 \times 2$ ,  $5 \times 5$ ,  $13 \times 13$ ,  $17 \times 17$  reference ranges on the VVC common test sequences [71], [72]. The joint distributions between the to-be-filtered pixel and its neighboring pixels (considering the MSB precision) located at 1, 4, 8, 12, and 16 pixels away in the bottom, right, and bottom-right positions are illustrated. The color scale ranges from white (low frequency) to dark blue (high frequency), representing the normalized occurrence frequency.

performance, such as the diamond filter shape with a  $7 \times 7$  reference range in ALF [3], and the flexible neural operators with a wide reference range (receptive field) in DNN-based ILFs [28]. In the proposed LUT-based ILF solution, the adoption of the “*space-trading-for-time-and-computation*” strategy enables the modeling capability of reference perception to be directly translated into the cached indexing entries of pixel relationships in the LUT, thereby avoiding the online computation of complex filtering functions. However, due to the exhaustive relationship of pixels, the storage size of a single LUT grows exponentially with respect to the increasing input dimension (the number of to-be-filtered and reference pixels). The storage size of a single LUT with the *int8* precision of input/cache values can be formulated as,

$$MS = (2^{8-q} + 1)^n \times V \times 8 \text{ bit}, \quad (1)$$

where  $MS$ ,  $q$ ,  $n$ ,  $V$  denote the abbreviation of storage size, the sampling interval of LUT, the cached dimension of the LUT, and the cached value number of each indexing entry in the LUT, respectively. For instance, a fully 4D LUT ( $q = 0$ ,  $n = 4$ ,  $V = 1$ ) requires 64 GB, while extending to a 5D LUT under the same settings would demand over 16 TB; a clipped 4D LUT ( $q = 4$ ,  $n = 4$ ,  $V = 1$ ) requires 81.56 KB, while extending to a 5D LUT would demand over 1 GB. The exponential growth makes it infeasible and impractical to improve the reference perception by increasing the dimension of a single LUT. To address this limitation, our preliminary framework [65] has proposed that the reference perception can be enlarged to  $9 \times 9$ ,  $13 \times 13$  through the ensemble trick of look-up operations or linearly stacked LUT-driven strategies. However, despite these improvements, the reference perception capability remains significantly limited when compared to that of advanced hand-crafted or DNN-based filtering schemes.

2) *Multi-component Reference Collaboration*: Beyond the direct spatial-wise reference perception of loop filters, various reference perception sources have also been verified to be beneficial for filtering effectiveness, such as the cross-component-guided filtering (e.g., CCALF [3], CCSAO [8] adopted in VVC), and the spatial-wise multi-reference-range-guided filtering (e.g., rich channel-wise interactions across multiple reference ranges in DNN-based filters [28]). Due to the inherent attribute of LUT, the direct relationship extension of multi-component reference information within a single LUT will similarly bring the exponential growth of storage consumption, which constrains the collaborative complementary information perception of the whole filtering process.

3) *Storage Overhead*: In the exhaustive relationship of LUT construction, all possible relationships of pixel values need to be enumerated and stored. Although the design of uniform sampling of the LUT in our basic solution can effectively reduce its storage size by caching only the MSBs of the input, the direct clipping of the LUT ignores the inherent properties of pixel correlations and their actual access relevance of reference perception for the filtering goal, potentially leading to sub-optimal storage compaction. For a simple instance, we observe the retrieved pixel value distributions from the pre-filtering frames under QPs of 22, 27, 32, 37, 42, 47 within different reference ranges on the VVC common test sequences [71], [72]. In Fig. 3, we illustrate the relationships between the to-be-filtered pixel and its neighboring (reference) pixels at MSB precision located at different distances in the bottom, right, and bottom-right directions. We can observe a diagonal phenomenon in the occurrence frequency statistics, which reveals two key limitations of the cached LUT. First, a large number of LUT entries are rarely accessed in practice, due to the high similarity between local reference pixels and the to-be-filtered pixel, which leads to concentrated access within a limited subset of LUT indexing entries. Second, as the reference range increases, the retrieved relationships gradually shift toward pixel pairs with larger value differences, indicating that reference perception imposes varying demands on relationship caching across different reference ranges.

### III. THE FRAMEWORK OF LUT-ILF++

To overcome the existing bottlenecks of our preliminary study [65], we propose a universal framework, LUT-ILF++, for efficient in-loop filtering by enabling multiple LUTs. In LUT-ILF++, we abandon the utilization of a single-LUT paradigm and introduce the concept of multiple cooperative LUTs, as shown in Fig. 4, which activates the multiple functions and cooperative manners of LUTs. Inspired by the design of hand-crafted and DNN-based loop filters, in the whole filtering process, we treat a single LUT as a basic unit and construct the whole filtering architecture by the cooperation of different kinds of LUTs and the link of customized indexing mechanisms. Specifically, as shown in Fig. 4 (1), the framework of LUT-ILF++ is constructed by three kinds of basic LUT units, including spatial-wise, channel-wise, and compacted (\*) reference perceived filtering LUTs, which generalizes the single LUT unit to multiple elementary reference perceived LUT units to establish a LUT-based loop filter flexibly. To promote the collaborative capacity of these basic LUT units, a series

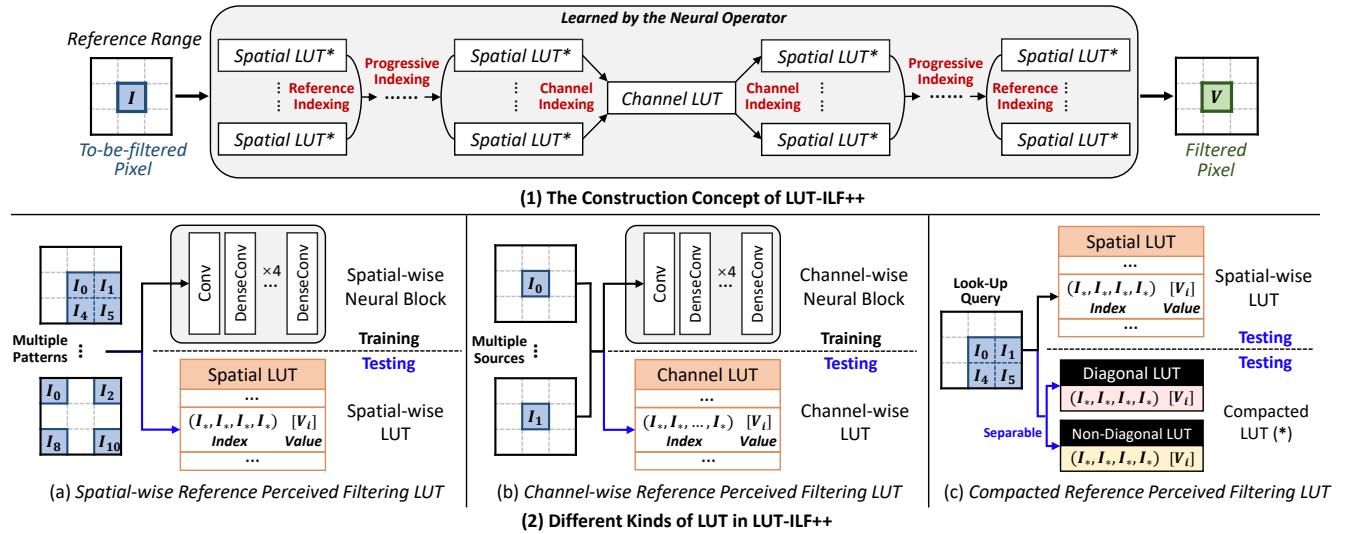


Fig. 4. Overview of LUT-ILF++. (1) The construction concept of LUT-ILF++. The different kinds of LUTs are jointly assembled into a loop filter through coordinated customized indexing mechanisms. The symbol \* denotes that the LUT has been pruned via the LUT compaction scheme, and is inferred through the separable indexing mechanism mentioned in Section VI. (2) Different kinds of LUTs in LUT-ILF++. The single LUT is generalized into multiple kinds of elementary reference perceived LUT units for the establishment of a cooperative LUT-based in-loop filtering framework.

of cooperative manners and customized indexing mechanisms are introduced to coordinate and manage them, including reference indexing, progressive indexing, channel indexing, and separable indexing (\*). It allows each LUT to focus on a specific aspect of reference modeling, achieving effective reference perception with low computation and storage cost.

During training, as shown in Fig. 4, the different kinds of LUTs are reproduced by their corresponding neural blocks, and the blocks are linked through the customized indexing mechanisms to maintain the structural dependencies required for LUT cooperation. These components are jointly assembled into a unified neural network for end-to-end optimization. With the network training completed, the inputs and outputs of each neural block are cached and compacted (\*) into each LUT, and the indexing relationships among neural blocks are retained for LUTs to preserve the cooperative structure learned during training. During inference, all neural computations are replaced by efficient LUT look-up operations. Each LUT independently retrieves the filtering mapping based on its input dimensions, and the indexing relationships ensure the whole filtering pipeline across various LUTs.

In the following sections, we provide a detailed description of three kinds of basic filtering LUT units with their associated cooperative manners and indexing mechanisms, and introduce the whole framework of LUT-ILF++. First, in Section IV, we introduce the spatial and channel-wise filtering LUTs, and explore the cooperation of multiple LUTs through customized indexing mechanisms to establish a better reference perceived filtering paradigm, and put forward the luma filtering framework of LUT-ILF++. Second, in Section V, we introduce the cross-component cooperation of multiple LUTs to enable the filtering of different chroma components jointly, and put forward the chroma filtering framework in LUT-ILF++. Third, in Section VI, we introduce the LUT compaction scheme with its training strategy, and examine its role in balancing filtering performance and storage efficiency.

#### IV. LUT-ILF++ WITH COOPERATION OF MULTIPLE LUTS

In this section, we introduce the cooperative LUT-based filtering scheme that leverages the multiple spatial and channel-wise LUTs to effectively overcome the limitations of filtering reference perception mentioned in Section II.B (1). First, we introduce the design of these basic filtering LUT units with their basic cooperative manners in LUT-ILF++, which enables the LUT-based solution with a large filtering reference range and accurate reference perception in spatial and channel dimensions. Second, we propose the luma filtering framework in LUT-ILF++, and discuss the cooperation of multiple LUTs and the scaling law of the LUT-based ILF framework through a series of spatial and channel-wise modulated explorations.

##### A. Spatial-wise Filtering LUTs with Reference Indexing

Local-context-driven filtering has been widely used in the typical loop filters of video coding standards, such as ALF with a diamond filtering template shape and category-specific filters [9], [10], and SAO with pre-defined offset classes based on pixel statistics [7], [8]. In these schemes, different classification strategies or retrievable approaches of surrounding reference pixels are adopted to capture the local reference features and apply various filtering operations. Inspired by these schemes, we propose the first basic cooperative way among the LUT units, spatial-wise LUTs with reference indexing, which introduces the diverse reference indexing to enlarge the reference range of the to-be-filtered pixel by parallelizing more diverse retrievable patterns to address more reference pixels and capture the rich local structures in the spatial dimension.

Based on the standard single pattern with  $3 \times 3$  reference range of our basic solution (pattern 1 of Fig. 5), we generalize the diverse reference retrievable patterns to support  $5 \times 5$  reference range. As shown in Fig. 5, besides the standard pattern 1, the patterns 2~8 are designed to cover the reference relationships of a  $5 \times 5$  reference range. For the reference modeling in each pattern, we select representative pixel relationships (filtering template shape) among the to-be-filtered

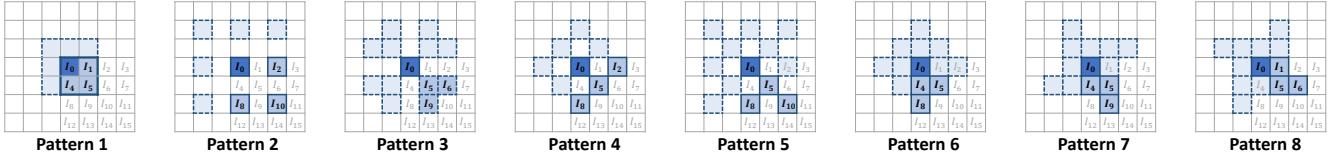


Fig. 5. Illustration of spatial-wise indexing patterns of reference indexing with a large reference range. With the use of pattern 1~8, LUT-ILF++ can address more reference pixel relationships with 5×5 reference range around  $I_0$ . The covered reference pixels with the rotation ensemble are marked with dashed boxes.

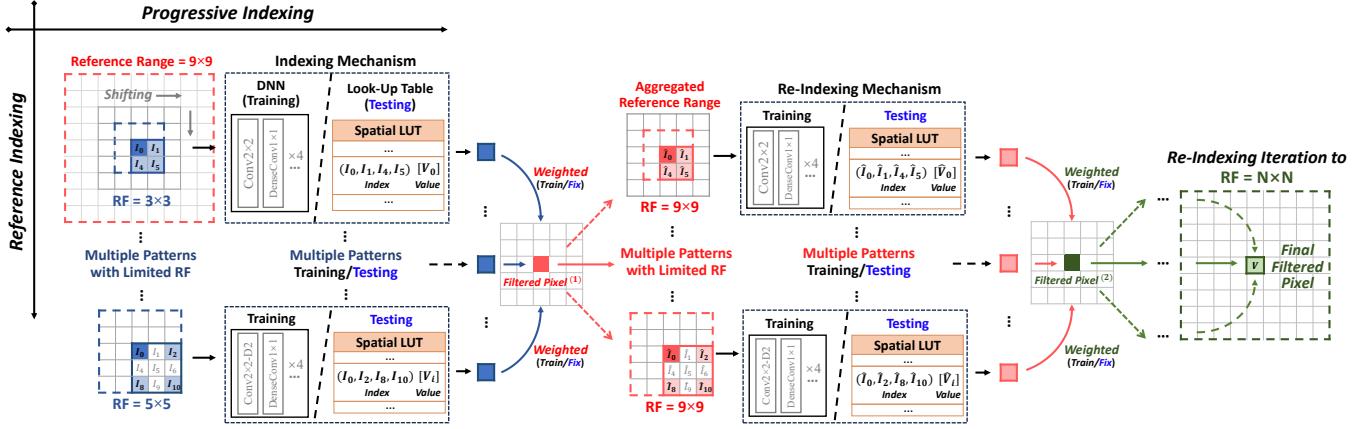


Fig. 6. Illustration of spatial-wise filtering LUTs with reference indexing and progressive indexing mechanisms. The parallel and cascaded spatial-wise neural blocks/LUTs are linked via reference and progressive indexing at the training/testing phase. The covered reference range of each indexing pattern with the rotation ensemble is marked with dashed boxes. The cascaded filtering framework is enabled by iterative re-indexing mechanisms of cascaded spatial-wise LUTs, allowing progressive aggregation of reference information toward an arbitrary target reference range of  $N \times N$  for a to-be-filtered pixel.

and reference pixels under the constraint of maintaining the invariable LUT dimension. In this way, the total size of cached LUTs grows linearly instead of exponentially, and it can be formulated as,

$$MS = R \times (2^{8-q} + 1)^n \times V \times 8 \text{ bit}, \quad (2)$$

where  $R$  denotes the number of the used retrievable patterns,  $n$  denotes the cached dimension of spatial-wise LUT. In training, as shown in Fig. 4 (a) and Fig. 6, the multiple spatial-wise filtering LUTs with different patterns are reproduced by multiple spatial-wise neural blocks, and these blocks are organized into multiple branches to model the different reference relationships in parallel. Note that the first convolution layer of spatial-wise neural block structure is incorporated with the *reshape*, *unfold* operations to support the arbitrary indexing patterns with specified coordinates of reference pixels. In inference, as multiple trained spatial-wise blocks are transferred into cached LUTs, the final filtering result is calculated by indexing and weighting their cached filtering results. The filtering process of the whole reference indexing process can be formulated as,

$$V = (W_1 \times LUT_{p1}[I_0][I_1][I_4][I_5] + \dots + W_n \times LUT_{pn}[\cdot][\cdot][\cdot][\cdot])/n, \quad (3)$$

where  $V$  denotes the filtered pixel value,  $n$  denotes the number of patterns,  $LUT[\cdot]$  denotes the look-up and interpolation operations of LUT retrieval,  $P_n$  denotes the pattern ID,  $W_n$  denotes the weights of different patterns. For the design of the weighting process, the impact of different reference retrievable patterns on the to-be-filtered pixel is considered. The weights of different patterns are designed as trainable parameters and normalized to the range [0, 1] using the *softmax()* function to reflect the confidence of each reference pattern during training.

Once training is completed, these weights are fixed and used by integer operation during inference.

### B. Spatial-wise Filtering LUTs with Progressive Indexing

DNN-based filtering process benefits from the inherent capability of neural networks to progressively modulate the receptive field (RF) through their layer-based architecture [28]. Compared to hand-crafted schemes, DNN-based schemes can perceive the accurate contextual information with the reference range expansion, enabling the modeling of complex reference relationships among local structures. Inspired by its efficient RF modulation, we propose the second cooperative way among the LUT units, spatial-wise LUTs with progressive indexing, which introduces the cascaded filtering LUTs to progressively enlarge the reference range of the to-be-filtered pixel and modulate the local structures in the spatial dimension.

As shown in Fig. 6, building upon the reference indexing pipeline with parallel multiple reference retrievable patterns, we extend the pipeline by adding multiple cascaded LUTs and linking them via the re-indexing mechanisms. The whole cascaded filtering process is supported by iterative look-up operations of cascaded LUTs, enabling progressive aggregation of reference information toward a target  $N \times N$  reference range for a to-be-filtered pixel. Based on (3), the filtering process of progressive indexing can be formulated as,

$$V^{(iter)} = (W_1^{(iter)} \times LUT_{p1}^{(iter)}[\widehat{I}_0][\widehat{I}_1][\widehat{I}_4][\widehat{I}_5] + W_2^{(iter)} \times LUT_{p2}^{(iter)}[\widehat{I}_0][\widehat{I}_2][\widehat{I}_8][\widehat{I}_{10}] + \dots + W_n^{(iter)} \times LUT_{pn}^{(iter)}[\cdot][\cdot][\cdot][\cdot])/n \quad (4)$$

where  $iter$  denotes the current step number of cascaded iteration,  $\widehat{I}$  denotes the output value of previous filtering

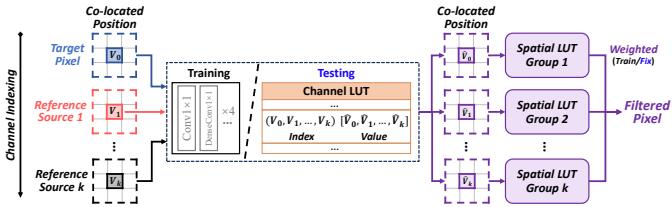


Fig. 7. Illustration of channel-wise LUTs with channel indexing mechanism.

step that serves as the index of the following filtering LUT. As shown in Fig. 6, we take the case of  $\text{iter} = 2$  as an example to detail the progressive indexing process. First, in  $\text{iter} = 1$ , with the filtering of the to-be-filtered pixel by the reference indexing mechanism, the filtered pixel ensembles the local information of a  $5 \times 5$  reference range. By shifting the filtering window in a  $9 \times 9$  reference range, the local information of the  $9 \times 9$  reference range can be aggregated into a  $5 \times 5$  aggregated reference range. In  $\text{iter} = 2$ , the re-indexing mechanism is used to filter the target pixel based on the aggregated reference pixels to incorporate the larger reference range implicitly. The whole process is similar to cascading multiple convolutional layers in a neural network and achieving information aggregation in the feature domain. In this way, the total size of cached LUTs still grows linearly, which can be formulated as,

$$MS = P \times R \times (2^{8-q} + 1)^n \times V \times 8 \text{ bit}, \quad (5)$$

where  $P$  denotes the total step number of progressive indexing to the target reference range. In training, as shown in Fig. 6, each step of progressive indexing is reproduced by the basic multi-branch neural network of reference indexing, and all steps are stacked to form a deep cascaded architecture. Due to the *integer-precision* cache attribute of LUT, the indexing inputs and outputs of LUT are quantized to *integer* precision, whereas network training uses the *floating-point* gradients. To bridge this gap, we apply the straight-through estimator (STE) strategy [73] for cascaded architecture training to reproduce the LUT indexing process. During the forward pass, the outputs of the previous progressive indexing step are quantized to *integer* precision for the follow-up indexing step, while in the backward pass, the gradients are preserved in *floating-point* precision to maintain the end-to-end optimization. In inference, as the deep cascaded architecture is transferred into cached cascaded LUTs, the final filtering result is calculated by step-by-step progressive look-up operations according to Eq. (4).

### C. Channel-wise Filtering LUTs with Channel Indexing

Multiple-component-assisted filtering process benefits from its advantage to exploit complementary information of to-be-filtered pixel from different reference sources in coding process, such as cross-component (luma, chroma, etc.) [3], [8], side-information (prediction, residual, etc.), and multi-reference-range guidance [28]. Compared to the weighted interaction manner of hand-crafted filtering schemes, the flexible high-dimensional channel interaction capability of DNN-based schemes has been verified to provide superior adaptability in modeling the relevance of different reference sources [28], [74]. Inspired by its interaction manner, we propose the

third cooperative way among the LUT units, channel-wise LUTs with channel indexing, which introduces the channel-dimensional caching and indexing to ensemble multiple reference sources.

In contrast to the direct many-to-one mapping relationship of the above spatial-wise LUTs, the correlations among different auxiliary reference sources are more complex than spatial-wise pixel relationships, and the relationship modeling is difficult to represent by a single direct mapping. Therefore, a two-step channel indexing mechanism is introduced to model its complex interactions in the filtering process. As shown in Fig. 7, inspired by the high-dimensional channel modulation process of DNN models, first, a channel LUT is used to perform many-to-many mappings across the target pixel and different reference sources at co-located spatial positions, achieving channel interaction. The number of input and output dimensions is kept identical to avoid potential bottlenecks caused by insufficient information perception in the channel-wise LUT. Second, each output channel is followed by a spatial-wise LUT group, constructed by basic multi-branch LUTs with reference indexing (patterns 1–3), to refine the individual mixed correlations within each channel. The outputs of all channels are then aggregated through the same weighted mechanism as in reference indexing to produce the final filtered pixel. In this way, the total size of cached LUTs with linear growth can be formulated as,

$$MS = (K \times (2^{8-q} + 1)^K + K \times R \times (2^{8-q} + 1)^n) \times V \times 8 \text{ bit}, \quad (6)$$

where  $K$  denotes the cached dimension of channel-wise LUT. In training, as shown in Fig. 7, the  $K$ -dimensional channel-wise LUT is reproduced by a  $1 \times 1$  convolution layer with  $K$  input and output channels, and the reproduction of spatial-wise LUTs is the same as mentioned in Section IV.A. In inference, the final filtering result is calculated by the cooperation of spatial and channel-wise look-up operations and weighted mechanisms.

### D. Cooperation of Multiple LUTs

In the above subsections, basic spatial and channel-wise reference perceived filtering LUTs with their three basic cooperative ways are proposed, which effectively improve reference perception while ensuring constrained storage consumption growth. In this subsection, based on these basic modules, we discuss the cooperation of multiple LUTs in the LUT-based ILF solution, and put forward the luma filtering framework in LUT-ILF++. The bottleneck of filtering reference perception lies in the exponential exhaustive reference relationship manner of LUTs, which constrains the effective expansion of the reference range required for improved filtering performance, as analyzed in Section II.B (1). Based on the aforementioned series of basic reference perceived modules with linearly constrained storage growth, here we revisit how to perceive a larger reference range in the filtering process, and achieve a good trade-off between filtering gains and complexity with the low storage cost by leveraging cooperative multiple LUTs.

As shown in Fig. 8 (a), we adopt the basic reference and progressive indexing architectures as the basic framework, which incorporates patterns 1~3 for each reference

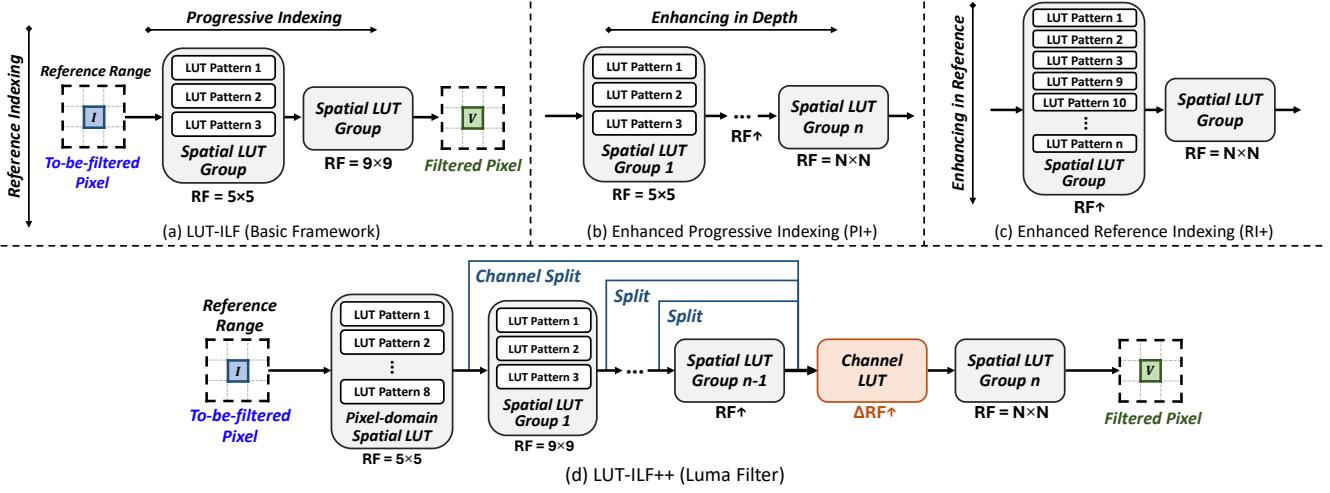


Fig. 8. Overview of the filtering framework of LUT-ILF [65], enhanced variants of LUT-ILF, and the luma filtering framework of LUT-ILF++.

TABLE I  
COMPARISON RESULTS OF LUT-ILF [65], ENHANCED VARIANTS, AND LUT-ILF++ UNDER PSNR METRIC FOR LUMA FILTERING

QP	RF = 9 × 9		RF = 13 × 13			RF = 17 × 17			RF = 21 × 21			RF = 25 × 25		
	LUT-ILF [65]	RI+	PI+	RI+	PI+	LUT-ILF++	RI+	PI+	LUT-ILF++	RI+	PI+	LUT-ILF++	RI+	PI+
22	0.0	+0.037	+0.040	+0.047	+0.043	<b>+0.112</b>	+0.045	+0.050	<b>+0.132</b>	+0.033	+0.057	<b>+0.117</b>		
27	0.0	+0.034	+0.031	+0.042	+0.036	<b>+0.093</b>	+0.038	+0.047	<b>+0.098</b>	+0.031	+0.049	<b>+0.087</b>		
32	0.0	+0.058	+0.053	+0.063	+0.071	<b>+0.081</b>	+0.054	+0.063	<b>+0.088</b>	+0.052	+0.056	<b>+0.071</b>		
37	0.0	+0.018	+0.022	+0.027	+0.032	<b>+0.043</b>	+0.034	+0.030	<b>+0.041</b>	+0.021	+0.024	<b>+0.038</b>		
42	0.0	+0.028	+0.033	+0.032	+0.039	<b>+0.045</b>	+0.037	+0.040	<b>+0.049</b>	+0.030	+0.038	<b>+0.043</b>		
47	0.0	+0.023	+0.019	+0.023	+0.027	<b>+0.034</b>	+0.025	+0.021	<b>+0.029</b>	+0.014	+0.017	<b>+0.023</b>		

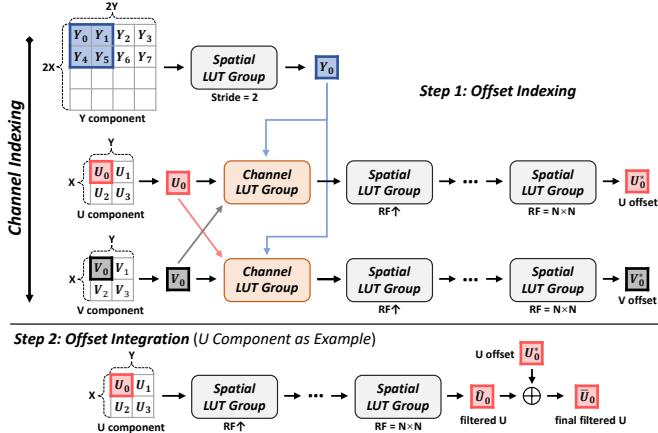


Fig. 9. Overview of the chroma filtering framework of LUT-ILF++.

indexing (RI) mechanism and a two-step cascaded filtering iteration ( $iter = 2$ ) for progressive indexing (PI), enabling a  $9 \times 9$  reference range. The basic framework corresponds to the design of our preliminary framework (LUT-ILF) [65]. To enlarge the reference range further, following our linear-growth establishment principle, the direct additional expansion of retrievable reference patterns and iteration depth can achieve this goal, as shown in Fig. 8 (b) and (c). Motivated by these, we conduct a series of modulated explorations to verify the potential of scaling these modules. The explorations are performed on top of VTM-11.0 with all-intra (AI) setting, and the filter is set into the end step of the ILF process without additional rate-distortion optimization.

As shown in Table I, the comparison results verify the potential of direct scaling of the basic framework's retrievable patterns (RI+) or cascaded iteration depths (PI+) to a larger

target reference range. Although the enhanced variants gain from a larger reference range, the continual expansion of the reference range reaches a bottleneck in accurate reference perception, limiting further noticeable improvements. To ensure the scalability of the cooperative framework, we build upon the basic reference perceived modules and propose a series of scaling strategies that enable its flexible extension. Specifically, as shown in Fig. 8 (d), based on the scaling architecture of PI+ and RI+ to the target reference range, we further improve the cooperative manner of multiple LUTs from two perspectives.

(1) *Pattern allocation of spatial-wise LUTs with reference indexing.* The number of cached pixel-relationship entries is constrained by the restricted set of retrievable patterns, which makes it inevitable to miss some potentially optimal pixel reference relationships as the reference range increases. To mitigate this issue, more reference retrievable patterns (patterns 1~8 in Fig. 5) are allocated into the pixel-domain spatial-wise LUT group to access a broader set of pixel relationships within the proximal reference range, thereby ensuring sufficient perception of the most relevant neighbor reference pixels and maximizing potential optimal dependencies.

(2) *Channel interaction of spatial-wise LUTs with progressive indexing.* As the reference range expands, the distribution of effective reference relationships becomes increasingly sparse. The growth of ineffective relationships in distant references results in the degradation of useful information density, which hinders the effective learning of reference perception during training. To preserve accurate reference perception as the reference range increases, channel split operation and

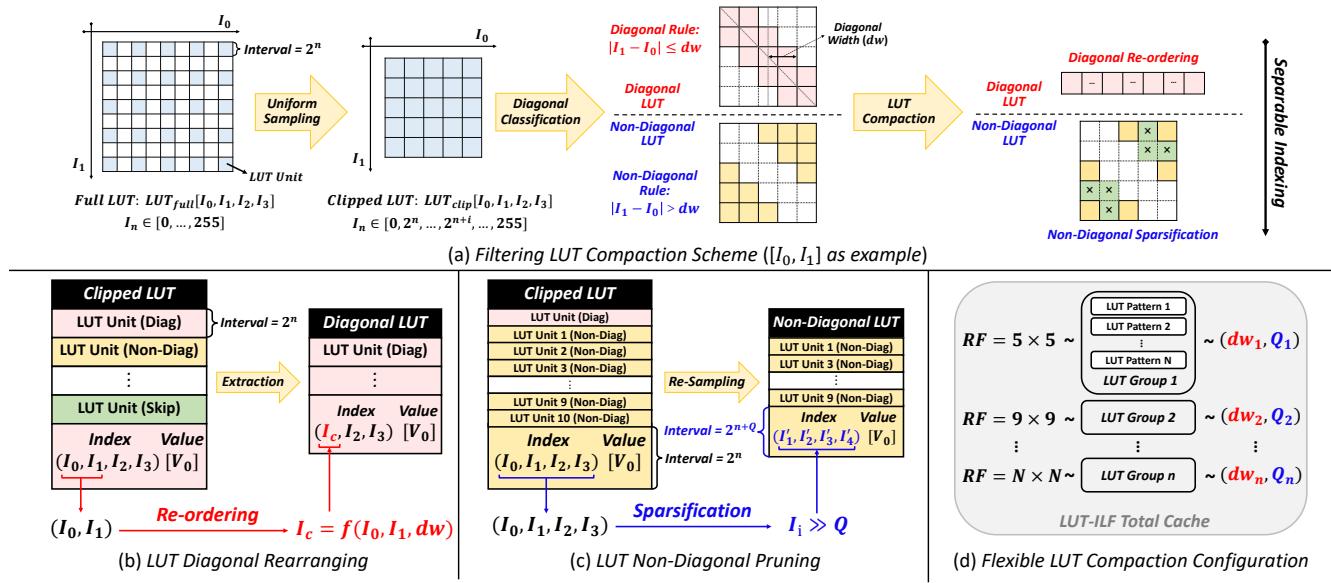


Fig. 10. Overview of LUT compaction scheme in LUT-ILF++.

channel indexing mechanism are used to add channel diversity instead of relying on a purely progressive single-path reference expansion, enabling the additional channel to facilitate the interaction of useful indexing relationships across multiple reference ranges. Meanwhile, the channel interaction also alleviates the non-negligible optimization challenges caused by the integer cache attribute of the LUT-based solution, which helps to avoid the accumulation of errors during the STE strategy-driven training process of the cascaded LUT architecture, thereby improving the convergence stability of neural block reproduction. Note that the channel split operation is performed by doubling the cached value number of each cached indexing entry of the spatial-wise LUT groups. For the neural reproduction of these spatial-wise LUTs, it only requires modifying the output channels of the corresponding convolution layer from 1 to 2. Overall, as shown in Table I, the proposed scaling strategies in LUT-ILF++ (Luma Filter) improve the scalability of the cooperative filtering framework, enabling effective extension to larger target reference ranges.

## V. LUT-ILF++ WITH CROSS-COMPONENT COOPERATION OF MULTIPLE LUTS

In this section, based on the above luma filtering framework of LUT-ILF++, we propose the chroma filtering framework that leverages the interaction ability of channel-wise LUT to effectively overcome the limitation of cross-color-component collaborative filtering mentioned in Section II.B (2). Inspired by the cross-component design of the existing loop filter [3], [8], we propose the cross-component indexing mechanism to exploit the inherent correlations among color components and assist chroma-component filtering, while maintaining low complexity through effective cross-component modeling.

As shown in Fig. 9, the whole cross-component cooperative chroma filtering process is designed into two steps, including cross-component offset indexing and integration. First, channel-wise LUTs are used to exploit the cross-component relationships from the proportionally aligned luma and chroma

information and modulate the corresponding correction offsets for the filtering of U and V components, respectively. Second, cross-component correction offsets are integrated into the main spatial-wise chroma filtering LUT framework via element-wise addition, yielding the final filtered chroma outputs. In this way, the total cached LUT size for chroma filtering still follows a linearly constrained growth, as it is composed of both spatial and channel-wise LUTs with linear cache design. In training, the neural reproduction of spatial and channel-wise LUTs is the same as mentioned in Section IV.A and Section IV.C. In inference, the final chroma filtering result is generated by the cooperation of spatial and channel-wise look-up operations.

## VI. LUT-ILF++ WITH COOPERATION OF COMPACTED LUTS

In this section, based on the above luma and chroma filtering framework of LUT-ILF++, we propose the LUT compaction scheme that leverages the modular attribute of LUT to overcome the limitation of storage overhead mentioned in Section II.B (3). Based on the bottleneck observation of cached pixel relationships between the to-be-filtered pixel and its reference pixels at MSB precision (Fig. 3), we find that the uniform sampling cache design of LUT indexing entries only intuitively reduces storage cost from the perspective of indexing entry correlation, while ignoring the inherent properties of pixel correlations and their actual access concentration of reference perception for the filtering goal, leading to sub-optimal trade-offs. Inspired by these observations, we introduce the LUT compaction scheme into LUT-ILF++, and propose the LUT pruning strategy with a separable indexing mechanism to achieve accurate storage cost reduction according to the reference perception requirements in practical use.

### A. Compacted Filtering LUTs with Separable Indexing

As illustrated in Fig. 3, based on uniform sampling cache design, the diagonal phenomenon of occurrence frequency statistics of accessed pixel value distribution represents the

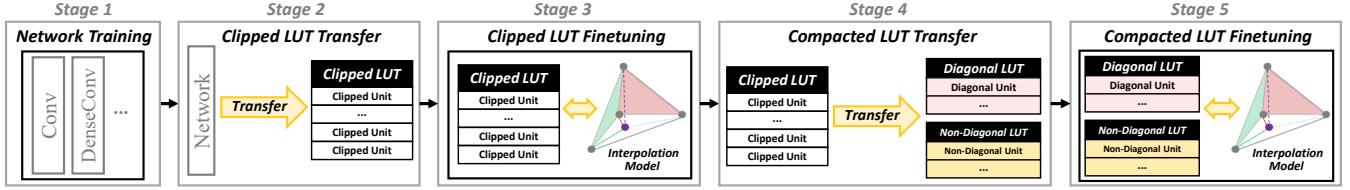


Fig. 11. The training process of compacted LUTs in LUT-ILF++.

actual access requirements of the LUT-based ILF solution in practice. Based on this phenomenon and the LUT modular attribute, we propose the diagonal compact-oriented LUT pruning strategy, which retains the access-concentrated indexing entries and further eliminates the entry redundancy. As shown in Fig. 10 (a), we take the compacted process of the first two dimensions (2D,  $[I_0, I_1]$ ) of a full LUT (4D,  $[I_0, I_1, I_2, I_3]$ ) as an example to detail the proposed strategy, the detailed calculation process is mentioned in the supplementary material.

Starting from a full LUT transferred from a neural block, uniform sampling is applied to roughly reduce redundancy, resulting in a clipped LUT. Based on the clipped LUT, the clipped indexing entries are further extracted and classified into diagonal and non-diagonal entries by judging the diagonal condition rule:  $|I_0 - I_1| \leq dw$ , where  $dw$  stands for diagonal width. For diagonal indexing entries, as shown in Fig. 10 (b), the diagonal re-ordering is used to rearrange them as the diagonal LUT by mapping the LUT coordinate ( $I_c = f_{mapping}(I_0, I_1, dw)$ ), and the diagonal LUT is then stored as a low-dimensional LUT indexed by  $[I_c, I_2, I_3]$ . For non-diagonal indexing entries, as shown in Fig. 10 (c), due to their sparse access concentration in practice, their redundancy is pruned by re-sampling their dimensionality (MSBs) with the allocated sparsification shift ( $Q$ ), and the non-diagonal LUT is then stored with an exponentially reduced storage cost. Note that the flexible LUT compaction configuration is applied to allocate the  $dw, Q$  values to different spatial-wise LUTs corresponding to different reference ranges, as shown in Fig. 10 (d), fitting the diagonal distribution phenomenon with varying diagonal width across different ranges observed in Fig. 3. The LUT compaction scheme can be generalized to multiple dimensions of LUT indexing entries, such as 3D ( $[I_0, I_1, I_2]$ ), 4D ( $[I_0, I_1, I_2, I_3]$ ). For a clipped LUT, based on Eq. (1), the total size of its compacted version can be formulated as,

$$\begin{aligned} LUT_{diag} : MS &= D \times (2^{8-q} + 1)^{n-p} \times V \times 8 \text{ bit} \\ LUT_{non-diag} : MS &= (2^{8-q-Q} + 1)^n \times V \times 8 \text{ bit} \end{aligned} \quad (7)$$

where  $p$  denotes the compacted dimension number of LUT indexing entries,  $D$  denotes the number of diagonal entries determined by  $dw$  and  $p$ .

During inference, the separable indexing mechanism (Fig. 4) is introduced to guide the query of indexing values of the to-be-filtered and reference pixels, where the diagonal condition rule determines which sub-LUT is accessed.

### B. Compacted LUTs with Cascaded Training Strategy

To achieve efficient resource utilization of the whole solution while maintaining coding gains, the cascaded LUT

training strategy is proposed and established on the basic LUT training and fine-tuning mentioned in Section II.A (1) and (3), enabling multiple cooperative LUTs to be effectively learned from data. The training step is shown in Fig. 11, which contains five stages. Due to the potential performance degradation caused by the iterative transfer process of DNNs, clipped and compacted LUTs, the fine-tuning stage with interpolation adaptation is used to bridge these transformations and ensure that the impact on the coding gain remains minimal. In the training stages, after each transformation of the filtering LUT, a fine-tuning stage is cascaded to fine-tune the collaboration between the LUT and the interpolation model, enabling the substitution of the original DNNs for practical use effectively.

## VII. EXPERIMENTS

### A. Experimental Settings

In the experiments, the VVC reference software VTM-11.0 is used as the baseline, which is consistent with the anchor version adopted in NNVC [28]. The codec adopts the configuration of All Intra (AI) and Random Access (RA) according to the VVC Common Test Condition (CTC) [71]. The test sequences, including Classes A to E with different resolutions, are tested as specified in [71], [72]. For each test sequence, quantization parameter (QP) values are set to 22, 27, 32, 37, 42, 47, and Bjontegaard Delta-rate (BD-rate) [75] is used as an objective metric to evaluate coding performance. The BVI-DVC [76] and DIV2K [77] are used as the training datasets.

For the complexity evaluated metrics, time complexity, computational complexity, estimated energy cost ( $pJ/\text{pixel}$  [78]–[80]), and storage cost (KB) are evaluated. For time complexity, the codec time is tested on a CPU model Intel Core i7-11700, and both serial and parallel implementations of LUT-ILF++ are tested. For computational complexity, since in the LUT-based ILF solution, the number of addition and multiplication operations are significantly imbalanced, and in our implementations, multiplication operations are minimized to improve the hardware efficiency and friendliness of LUT-ILF++. Therefore, we report the total number of multiplication and addition operations (Ops) for the comparison of different schemes. In addition, energy cost is used to evaluate the practical complexity of operations according to [78]–[80]. For a single addition operation, the operation of  $\text{int8}/\text{int16}/\text{int32}/\text{float32}$  corresponds to  $0.03/0.05/0.1/0.9 \text{ pJ}$ . For a single multiplication operation, the operation of  $\text{int8}/\text{int16}/\text{int32}/\text{float32}$  corresponds to  $0.2/0.65/3.1/3.7 \text{ pJ}$ .

### B. The Construction Settings of LUT-ILF++

1) *Luma Filter Setting:* As mentioned in Section IV.D, based on the modulated exploration experiments in Table I,

TABLE II  
BD-RATE AND COMPLEXITY RESULTS OF LUT-ILF++, AND COMPARISON RESULTS WITH OTHER IN-LOOP FILTERING SCHEMES UNDER ALL INTRA (AI) CONFIGURATION

All Intra (AI) Configuration									
Schemes	Sequence	Y BD-Rate	U BD-Rate	V BD-Rate	Computational Complexity <sup>3</sup>	Storage Cost <sup>3</sup>	Energy Cost <sup>3</sup> (pJ/pixel)	Time Complexity (Serial)	Time Complexity (Parallel)
NNVC-VLOP1 <sup>1</sup> (VTM-11.0)	Overall	-3.66%	-4.68%	-4.54%	10.2K Ops/pixel	50 KB (int16) 80 KB (float32)	3.57K (int16) 23.46K (float)	105%/1951%	—
NNVC-LOP2 <sup>1</sup> (VTM-11.0)	Overall	-4.84%	-10.12%	-10.31%	32.4K Ops/pixel	130 KB (int16) 229 KB (float32)	11.34K (int16) 74.52K (float)	107%/3983%	—
NNVC-HOP3 <sup>1</sup> (VTM-11.0)	Overall	-9.80%	-8.22%	-9.79%	954.0K Ops/pixel	2828 KB (int16) 5595 KB (float32)	333.90K (int16) 2194.20K (float)	173%/55495%	—
LUT-ILF-U <sup>2</sup> (VTM-11.0)	Overall	-0.08%	-0.27%	-0.11%	0.30K Ops/pixel	492 KB (int8)	0.27K	103%/147%	102%/106%
LUT-ILF-V <sup>2</sup> (VTM-11.0)	Overall	-0.32%	-0.51%	-0.26%	0.83K Ops/pixel	1476 KB (int8)	0.75K	105%/198%	104%/111%
LUT-ILF-F <sup>2</sup> (VTM-11.0)	Overall	-0.47%	-0.95%	-0.64%	1.91K Ops/pixel	3444 KB (int8)	1.69K	105%/223%	105%/119%
<b>LUT-ILF++ (VTM-11.0)</b>	Class A1	-0.54%	-2.21%	-1.80%	3.59K Ops/pixel (Y) 0.82K Ops/pixel (U) 0.82K Ops/pixel (V)	330 KB (Y) 241 KB (U) 241 KB (V)	3.01K (Y) 0.62K (U) 0.62K (V)	110%/275%	107%/188%
	Class A2	-0.73%	-2.59%	-1.14%				111%/232%	109%/173%
	Class B	-0.72%	-3.11%	-1.74%				107%/183%	103%/136%
	Class C	-0.70%	-3.90%	-1.57%				106%/172%	105%/121%
	Class D	-1.23%	-3.40%	-2.24%				102%/142%	99%/118%
	Class E	-0.86%	-2.03%	-1.06%				112%/233%	107%/142%
	<b>Overall</b>	<b>-0.82%</b>	<b>-2.97%</b>	<b>-1.63%</b>	<b>5.23K Ops/pixel (Total)</b>	<b>812 KB (Total)</b>	<b>4.26K (Total)</b>	<b>108%/207%</b>	<b>105%/146%</b>

<sup>1</sup> The results of BD-rate, time complexity, computational complexity, storage cost (int/float model) are cited from [52] (VLOP), [54] (LOP) / [36] (HOP), JVET AHG report [56] and open-sourced repository.

<sup>2</sup> As our preliminary work (LUT-ILF [65]) does not perform chroma-component filtering, we extend its luma architecture to the chroma and retrain the model for performance and complexity comparison. Due to the integration of chroma-component filtering in LUT-ILF-U/V/F, multi-component joint RDO is adopted for filter decision, and the performance of the Y component of LUT-ILF [65] is also re-evaluated.

<sup>3</sup> The computational complexity, storage cost, and energy cost of both luma and chroma models are jointly evaluated and reported for LUT-ILF++ and the comparative schemes.

the target reference range  $17 \times 17$  is selected as the optimal filtering perception and a better trade-off setting for the luma filter of LUT-ILF++ (Fig. 8 (d)). The inference architecture is composed of spatial-wise LUT groups, each corresponding to a distinct reference range size ( $5 \times 5$  with Patterns 1~8, and  $9 \times 9$ ,  $13 \times 13$ ,  $17 \times 17$  with Patterns 1~3), and the channel split and interaction are applied to all LUT groups before the penultimate reference range scale to perform channel interaction. The training architecture is reproduced by standard convolution and dense connected convolution layers.

2) *Chroma Filter Setting*: As mentioned in Section V, the chroma filtering framework of LUT-ILF++ is adopted for the filtering of each chroma component. Due to the inherent resolution proportion between luma and chroma components, the target reference range  $13 \times 13$  is selected as a better trade-off setting. The inference/training architecture is also composed/reproduced in the same manner as the luma.

3) *Filter Compaction Setting*: For the filtering frameworks of both luma and chroma filters, the flexible LUT compaction configuration is applied to minimize the storage cost of each spatial-wise LUT of the entire framework according to the diagonal phenomenon observed in Fig. 3. For luma, based on the uniform sampling cache design of each LUT with 4D dimensions, all dimensions are compacted in a unified manner. Specifically, the  $dw$  is set to 2, 2, 3, 3 for each diagonal LUT corresponding to the reference ranges of  $5 \times 5$ ,  $9 \times 9$ ,  $13 \times 13$ ,  $17 \times 17$ , respectively, and  $Q$  is set to 1 for each non-diagonal LUT. For chroma, the  $dw$  and  $Q$  are set to 2 and 1.

4) *Rare-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 at the coding tree unit (CTU) level to indicate the use of the proposed

filter. The flag is determined by the rate-distortion (RD) cost function that  $J = SSD + \lambda \times R_{flag}$ , where  $R_{flag}$  denotes the rate of decision flag in CABAC-based rate estimation,  $SSD$  denotes the sum of squared differences (SSD) between the reconstructed and filtering result, and the decision is jointly optimized across the luma and chroma components.

5) *Training and Finetuning Strategy*: For the training and finetuning of LUT-ILF++, we adopt the five-step cascaded training strategy mentioned in Section VI.B, with the loss function using MSE. For stages 1 and 3, the learning rate (lr) follows a cosine annealing schedule between 1e-3 and 1e-4. For stage 5, the lr is fixed at 1e-4. For the training iterations, stage 1 adopts 400,000, while stages 3/5 adopt 20,000 each.

### C. Performance

First, we show the coding performance and other evaluated metrics of LUT-ILF++ integrated into the VVC reference software VTM-11.0. Second, we comprehensively compare LUT-ILF++ with advanced filtering schemes to verify its good trade-offs and practicability.

#### 1) Overall Performance Under Common Test Conditions:

The experimental results are shown in Tables II and III. We can see that our proposed LUT-ILF++ can achieve, on average, 0.82%/0.85% (Y), 2.97%/4.11% (U), 1.63%/2.06% (V) BD-rate reduction on VTM-11.0 under AI and RA configurations. Beyond our preliminary framework (LUT-ILF-U, V, F [65]), we observe that the improved framework, LUT-ILF++, offers a better trade-off point between performance and efficiency, making the LUT-based ILF solution more universal and cost-effective in practice. Specifically, LUT-ILF++ achieves a significant performance gain with only slight increases in computational and time complexity, while enabling lower storage cost and supporting chroma-component filtering.

TABLE III

BD-RATE AND COMPLEXITY RESULTS OF LUT-ILF++, AND COMPARISON RESULTS WITH OTHER IN-LOOP FILTERING SCHEMES UNDER RANDOM ACCESS (RA) CONFIGURATION

Random Access (RA) Configuration									
Schemes	Sequence	Y BD-Rate	U BD-Rate	V BD-Rate	Computational Complexity <sup>3</sup>	Storage Cost <sup>3</sup>	Energy Cost <sup>3</sup> (pJ/pixel)	Time Complexity (Serial)	Time Complexity (Parallel)
NNVC-VLOP1 <sup>1</sup> (VTM-11.0)	Overall	-3.69%	-4.83%	-4.03%	10.2K Ops/pixel	50 KB (int16) 80 KB (float)	3.57K (int16) 23.46K (float)	107%/3625%	–
NNVC-LOP2 <sup>1</sup> (VTM-11.0)	Overall	-5.33%	-12.26%	-11.15%	32.4K Ops/pixel	130 KB (int16) 229 KB (float)	11.34K (int16) 74.52K (float)	110%/7538%	–
NNVC-HOP3 <sup>1</sup> (VTM-11.0)	Overall	-12.39%	-13.14%	-13.44%	954.0K Ops/pixel	2828 KB (int16) 5595 KB (float)	333.90K (int16) 2194.20K (float)	234%/113729%	–
LUT-ILF-U <sup>2</sup> (VTM-11.0)	Overall	-0.07%	-0.17%	-0.08%	0.30K Ops/pixel	492 KB (int8)	0.27K	102%/164%	103%/110%
LUT-ILF-V <sup>2</sup> (VTM-11.0)	Overall	-0.26%	-0.47%	-0.22%	0.83K Ops/pixel	1476 KB (int8)	0.75K	105%/196%	104%/123%
LUT-ILF-F <sup>2</sup> (VTM-11.0)	Overall	-0.35%	-0.65%	-0.44%	1.91K Ops/pixel	3444 KB (int8)	1.69K	106%/218%	106%/131%
LUT-ILF++ (VTM-11.0)	Class A1	-0.77%	-2.91%	-1.57%	3.59K Ops/pixel (Y) 0.82K Ops/pixel (U) 0.82K Ops/pixel (V)	330 KB (Y) 241 KB (U) 241 KB (V)	3.01K (Y) 0.62K (U) 0.62K (V)	112%/244%	110%/184%
	Class A2	-0.86%	-4.03%	-0.95%				109%/202%	108%/159%
	Class B	-0.75%	-4.65%	-2.44%				110%/197%	108%/140%
	Class C	-0.54%	-5.42%	-2.21%				107%/168%	106%/127%
	Class D	-1.18%	-4.47%	-2.99%				105%/147%	103%/119%
	Class E	-0.95%	-2.26%	-1.57%				111%/207%	107%/164%
	Overall	<b>-0.85%</b>	<b>-4.11%</b>	<b>-2.06%</b>	<b>5.23K Ops/pixel (Total)</b>	<b>812 KB (Total)</b>	<b>4.26K (Total)</b>	<b>109%/194%</b>	<b>107%/149%</b>

Note: The indication of table notes <sup>1,2,3</sup> is the same as in Table II.

TABLE IV  
ABLATION STUDY ON LUMA FILTER OF LUT-ILF++

Varients	Y BD-Rate (AI/RA)	Computational Complexity (Ops/pixel)	Storage Cost (w/ compaction)	Energy Cost (pJ/pixel)
w/o Pattern Allocation (PA)	-0.63%/-0.55%	2.66K (Y)	247 KB (Y)	2.22K (Y)
w/o Channel Interaction (CI)	-0.29%/-0.36%	1.96K (Y)	189 KB (Y)	1.40K (Y)
LUT-ILF++	<b>-0.82%/-0.85%</b>	<b>3.59K (Y)</b>	<b>330 KB (Y)</b>	<b>3.01K (Y)</b>

2) *Comparisons with Advanced Schemes:* The comparison results with advanced filtering schemes are also shown in Tables II and III. We provide a comprehensive comparison with the different complexity configurations of advanced NNLF in NNVC [28]. For the quantitative comparisons, the computational complexity and decoding time complexity of the LUT-based ILF solution (LUT-ILF-U/V/F, LUT-ILF++) are  $33 \times \sim 3180 \times$  and  $18 \times \sim 1034 \times$  lower than that of NNLF, and the LUT-based schemes also show good performance potential. In addition, compared to the heavy computational process of DNN inference of NNLF, the LUT-based solution achieves superior hardware deployment friendliness due to its storage-and-lookup inference paradigm, which is easily deployable on hardware architectures with fixed-point arithmetic implementations. Although LUT-based schemes generally require necessary storage, in LUT-ILF++, we verify that the modular attributes of LUTs can support flexible storage allocation, which effectively controls the storage cost and further enhances feasibility in practice.

#### D. Performance Analysis

1) *Ablation Study:* The ablation study is shown in Tables IV, V, VI, where a series of variants are introduced to verify the effectiveness of each core module.

TABLE V  
ABLATION STUDY ON CHROMA FILTER OF LUT-ILF++

Varients	U BD-Rate (AI/RA)	V BD-Rate (AI/RA)	Computational Complexity (Ops/pixel)	Storage Cost (w/ compaction)	Energy Cost (pJ/pixel)
w/o Offset Indexing	-2.13%/-2.81%	-0.97%/-1.06%	0.56K (U/V)	166 KB (U/V)	0.44K
w/o Offset Integration	-0.77%/-1.23%	-0.59%/-0.67%	0.21K (U/V)	76 KB (U/V)	0.19K
LUT-ILF++	<b>-2.97%/-4.11%</b>	<b>-1.63%/-2.06%</b>	<b>0.82K (U/V)</b>	<b>241 KB (U/V)</b>	<b>0.62K (U/V)</b>

For LUT-ILF++ (luma filter), in Table I, the impact of reference range and channel interaction (CI) has been validated with PSNR metric, here we further verify the effectiveness of pattern allocation (PA) and CI with BD-rate metric under the fixed target reference range  $17 \times 17$ , as shown in Table IV. For w/o PA, all spatial LUT groups adopt the same reference indexing patterns (Patterns 1~3 of Fig. 5). For w/o CI, LUT-ILF++ is degraded to the PI+ (Fig. 8 (b)) by removing the channel split and indexing mechanism. The comparison results show that both sub-modules lead to significant performance gains with slight increases in computational and storage costs.

For LUT-ILF++ (chroma filter), we verify the effectiveness of offset indexing and integration step under the fixed target reference range  $13 \times 13$ . For w/o offset indexing, the chroma-component filtering adopts the architecture of offset indexing (step 1 of Fig. 9) to directly filter the chroma component instead of the offset reconstruction. For w/o offset integration, the architecture is also degraded to the PI+ (Fig. 8 (b)). The comparison results show that the offset-driven cross-component indexing mechanism can effectively assist chroma-component filtering with lower complexity.

For the LUT compaction scheme, based on the diagonal cache statistic observation (Fig. 3), we analyze the impact of different compaction elements, including compacted dimen-

TABLE VI  
ABLATION STUDY ON FILTER COMPACTION IN LUT-ILF++

Setting	Compaction Dimension	<i>dw</i>	<i>Q</i>	Storage Cost	Y BD-Rate (AI/RA)
DNN	—	—	—	—	-0.87%/-0.93%
Clipped	—	—	—	2785 KB (Y)	-0.83%/-0.88%
	2D ( $[I_0, I_1]$ )	2	1	996 KB (Y)	-0.75%/-0.79%
	3D ( $[I_0, I_1, I_2]$ )	2	1	449 KB (Y)	-0.70%/-0.69%
	4D ( $[I_0, I_1, I_2, I_3]$ )	2	1	298 KB (Y)	-0.71%/-0.75%
Compaction	4D ( $[I_0, I_1, I_2, I_3]$ )	2	2	101 KB (Y)	-0.66%/-0.62%
	4D ( $[I_0, I_1, I_2, I_3]$ )	3	1	390 KB (Y)	-0.85%/-0.90%
	4D ( $[I_0, I_1, I_2, I_3]$ )	3	2	193 KB (Y)	-0.75%/-0.82%
	4D ( $[I_0, I_1, I_2, I_3]$ )	4	1	535 KB (Y)	-0.89%/-0.87%
LUT-ILF++	4D ( $[I_0, I_1, I_2, I_3]$ )	2~3 (flexible)	1	330 KB (Y)	<b>-0.82%/-0.85%</b>

TABLE VII  
BD-RATE RESULTS OF LUT-ILF++ AT LOW-BITRATE POINTS

Schemes	Y BD-Rate (AI/RA)	Computational Complexity (Ops/pixel)	Storage Cost	Energy Cost (pJ/pixel)
LUT-ILF-V	-0.74%/-0.39%	0.55K (Y)	492 KB (Y)	0.49K (Y)
LUT-ILF-F	-1.03%/-0.57%	1.27K (Y)	1148 KB (Y)	1.12K (Y)
LUT-ILF++	<b>-1.49%/-1.21%</b>	<b>3.59K (Y)</b>	<b>330 KB (Y)</b>	<b>3.01K (Y)</b>

sion, diagonal width (*dw*), and sparsification shift (*Q*). As shown in Fig. VI, starting from the full LUT, the variants of regular clipped LUTs and various compacted LUTs with different settings are introduced to validate the effectiveness of the LUT pruning strategy with a separable indexing mechanism in balancing performance and efficiency. The results verify that the proposed pruning strategy effectively aligns with actual cache access requirements to optimize storage cost while preserving high performance. Furthermore, the flexible compaction configuration achieves a more fine-grained storage optimization tailored to different reference ranges.

2) *Low-Bitrate Points Exploration*: To further explore the potential of LUT-ILF++, we evaluate it at low bitrate points (QP 27~47) and compare it with the preliminary framework, as shown in Table VII. The results verify its strong potential.

## VIII. CONCLUSION

In this paper, we rethink the practical deployment problem of emerging DNN-based coding tools in video coding. In our solution, we propose an efficient look-up table-based approach that does not rely on high-performance computing resources, and attempt to apply it to a typical in-loop filtering module. To achieve a favorable trade-off between performance and complexity for practical use, a series of LUT-oriented filtering designs are proposed to realize multiple filtering functionalities, including reference perception, cross-component filtering, etc, enabling a universal LUT-based filtering framework. The experimental results of the proposed LUT-ILF++ demonstrate that it can effectively achieve a good filtering goal while maintaining lower complexity in advanced VVC, providing a new practical way for neural network-based video coding tools. For future work, we plan to explore two directions: (1) we will further explore a unified filtering framework improved by meta-information (partition, motion vector, etc), aiming to integrate more auxiliary cues for improved filtering; (2) we aim to generalize the proposed solution to other coding tools,

including fractional-pixel motion estimation [81], reference picture resampling [82], etc.

## REFERENCES

- [1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
- [2] 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.
- [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] J. Han, B. Li, D. Mukherjee, C.-H. Chiang, A. Grange, C. Chen, H. Su, S. Parker, S. Deng, U. Joshi *et al.*, “A technical overview of AV1,” *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1435–1462, 2021.
- [5] AOM, “AOMedia Video 1, 2 (AV1, AV2),” <https://aomedia.org/>, 2015.
- [6] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, “Adaptive deblocking filter,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 614–619, 2003.
- [7] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, “Sample adaptive offset in the HEVC standard,” *IEEE Transactions on Circuits and Systems for Video technology*, vol. 22, no. 12, pp. 1755–1764, 2012.
- [8] C.-W. Kuo, X. Xiu, Y.-W. Chen, H.-J. Jhu, W. Chen, N. Yan, and X. Wang, “Cross-component sample adaptive offset,” in *2022 Data Compression Conference (DCC)*. IEEE, 2022, pp. 359–368.
- [9] C.-Y. Tsai, C.-Y. Chen, T. Yamakage, I. S. Chong, Y.-W. Huang, C.-M. Fu, T. Itoh, T. Watanabe, T. Chujo, 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.
- [10] X. Meng, J. Zhang, C. Jia, X. Zhang, S. Wang, and S. Ma, “Optimized adaptive loop filter in versatile video coding,” in *2021 Data Compression Conference (DCC)*. IEEE, 2021, pp. 359–359.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] ———, “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.
- [15] H. Man, H. Wang, R. Lu, Z. Wan, X. Fan, and D. Zhao, “Content-aware dynamic in-loop filter with adjustable complexity for VVC intra coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, 2025.
- [16] C. Jia, S. Wang, X. Zhang, S. Wang, J. Liu, S. Pu, and S. Ma, “Content-aware convolutional neural network for in-loop filtering in high efficiency video coding,” *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3343–3356, 2019.
- [17] D. Wang, S. Xia, W. Yang, and J. Liu, “Combining progressive rethinking and collaborative learning: A deep framework for in-loop filtering,” *IEEE Transactions on Image Processing*, vol. 30, pp. 4198–4211, 2021.
- [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] “Convolutional neural networks-based in-loop filter,” *JVT-T0088*, 2020.
- [20] “EE1-1.6: RDO considering deep in-loop filtering,” *JVT-AB0068*, 2022.
- [21] “Deep in-loop filter with additional input information,” *JVT-AC0177*, 2022.
- [22] 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*, vol. 17, no. 3, pp. 1–23, 2021.
- [23] Z. Li, Z. Yuan, L. Li, D. Liu, X. Tang, and F. Wu, “Object segmentation-assisted inter prediction for versatile video coding,” *IEEE Transactions on Broadcasting*, 2024.

- [24] X. Feng, L. Li, D. Liu, and F. Wu, "Efficient partition map prediction via token sparsification for fast VVC intra coding," in *2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2024, pp. 1–6.
- [25] Z. Li, J. Liao, C. Tang, H. Zhang, Y. Li, Y. Bian, X. Sheng, X. Feng, Y. Li, C. Gao *et al.*, "USTC-TD: A test dataset and benchmark for image and video coding in 2020s," *IEEE Transactions on Multimedia*, 2025.
- [26] X. Feng, Z. Li, L. Li, D. Liu, and F. Wu, "Partition map-based fast block partitioning for VVC inter coding," *arXiv preprint arXiv:2504.18398*, 2025.
- [27] J. Jia, Y. Zhang, H. Zhu, Z. Chen, Z. Liu, X. Xu, and S. Liu, "Deep reference frame generation method for VVC inter prediction enhancement," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 5, pp. 3111–3124, 2023.
- [28] 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.
- [29] 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.
- [30] C. Lin, Y. Li, J. Li, K. Zhang, and L. Zhang, "Low complexity super resolution for resampling-based video coding," in *2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025, pp. 1–5.
- [31] 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.
- [32] "Simplified feature extraction for hop," *JVET-AF0181*, 2023.
- [33] "Complexity reduction of NN-based loop-filters," *JVET-AF0206*, 2023.
- [34] "EE1-1.1: LOP candidate with Low Complexity TB and new BB structure," *JVET-AH0050*, 2024.
- [35] "EE1-1.4: Reduced complexity input feature extraction for LOP and VLOP," *JVET-AJ0066*, 2024.
- [36] "EE1-1.1: Report on training with HOP architecture change for EE1-0 (variant 1)," *JVET-AG0174*, 2024.
- [37] Z. Huang, J. Sun, X. Guo, and M. Shang, "Adaptive deep reinforcement learning-based in-loop filter for VVC," *IEEE Transactions on Image Processing*, vol. 30, pp. 5439–5451, 2021.
- [38] "EE1-related: Complexity reduction of NN in-loop filters through early cropping," *JVET-AH0195*, 2024.
- [39] "EE1-1.3: Neural network-based in-loop filters using early cropping," *JVET-AJ0054*, 2024.
- [40] M. Li and W. Ji, "Lightweight multiattention recursive residual CNN-based in-loop filter driven by neuron diversity," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 11, pp. 6996–7008, 2023.
- [41] "EE1-Related: On Low Complexity Operational Point for In-Loop Filtering," *JVET-AG0155*, 2024.
- [42] "AHG11: A Low-Complexity Neural Network Loop Filter based on Partial Convolution and Over-Parameterization," *JVET-AH0077*, 2024.
- [43] "EE1-Related: Combination test of EE1-1.3.5 and multi-scale component of EE1-1.6," *JVET-AD0211*, 2023.
- [44] "AhG11: LOP with inputs transformed," *JVET-AG0069*, 2024.
- [45] "EE1 related: LOP input adjustment with trainable input transform," *JVET-AH0207*, 2024.
- [46] "LOP input adjustment with trainable components," *JVET-AI0173*, 2024.
- [47] D. Liu, J. Ström, M. Damghanian, and P. Wennersten, "NN-based in-loop filtering with inputs transformed," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2024, pp. 3737–3743.
- [48] "EE1-related: Neural-network loop filters in EE1-1.1.2 with further complexity reduction," *JVET-AD0157*, 2023.
- [49] "EE1-2.1: HOP separate models," *JVET-AH0188*, 2024.
- [50] H. Zhang, C. Jung, Y. Liu, and M. Li, "Lightweight cnn-based in-loop filter for VVC intra coding," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2023, pp. 1635–1639.
- [51] "[AHG11] Study on lower-complexity NNL," *JVET-AG0057*, 2024.
- [52] "EE1-5: Study of the NN architecture at Very Low Operational Point," *JVET-AH0051*, 2024.
- [53] "EE1-1.5: VLOP using LOP3 architecture," *JVET-AI0107*, 2024.
- [54] "Unified LOP filter design, training procedure and filter usage," *JVET-AE0281*, 2023.
- [55] "Status of the joint EE1-0 (LOP.2) training," *JVET-AF0043*, 2023.
- [56] "NNVC software development (AHG14)," *JVET-AH0014*, 2024.
- [57] "NNVC software development (AHG14)," *JVET-AI0014*, 2024.
- [58] "BoG report on NN-filter design unification," *JVET-AD0380*, 2023.
- [59] "EE1-2.2: Wide activation HOP model," *JVET-AH00189*, 2024.
- [60] "EE1-2.3: Integer implementation of HOP In-loop filter with Transformer blocks and Attention blocks," *JVET-AH0205*, 2024.
- [61] "On the complexity adjustment of HOP4," *JVET-AI0172*, 2024.
- [62] 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.
- [63] —, "Toward dnn of lut: Learning efficient image restoration with multiple look-up tables," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [64] Y. Li, J. Li, and Z. Xiong, "Look-up table compression for efficient image restoration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26016–26025.
- [65] Z. Li, J. Li, Y. Li, L. Li, D. Liu, and F. Wu, "In-loop filtering via trained look-up tables," in *2024 IEEE International Conference on Visual Communications and Image Processing (VCIP)*. IEEE, 2024, pp. 1–5.
- [66] H. Zeng, J. Cai, L. Li, Z. Cao, and L. Zhang, "Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 2058–2073, 2020.
- [67] M. Pharr and R. Fernando, "Programming technics for high performance graphics and general purpose computation," *Addison-Wesley Professional*, 2005.
- [68] S. J. Kim, H. T. Lin, Z. Lu, S. Süsstrunk, S. Lin, and M. S. Brown, "A new in-camera imaging model for color computer vision and its application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, pp. 2289–2302, 2012.
- [69] R. Mantiuk, S. Daly, and L. Kerofsky, "Display adaptive tone mapping," in *ACM SIGGRAPH 2008 papers*, 2008, pp. 1–10.
- [70] 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.
- [71] K. Suehring and X. Li, "JVET Common Test Conditions and Software Reference Configurations," *JVET document, JVET-G1010*, 2017.
- [72] S. Liu, A. Segall, E. Alshina, and R.-L. Liao, "JVET common test conditions and evaluation procedures for neural network-based video coding technology," *JVET-X2016*, 2021.
- [73] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.
- [74] C. Dong, H. Ma, Z. Li, L. Li, and D. Liu, "Temporal wavelet transform-based low-complexity perceptual quality enhancement of compressed video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 5, pp. 4040–4053, 2023.
- [75] G. Bjontegaard, "Calculation of Average PSNR Differences between RD-curves," *ITU-T VCEG-M33, April, 2001*, 2001.
- [76] 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.
- [77] 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.
- [78] 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. 15648–15657.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] 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.
- [83] R. C. Gonzalez, *Digital image processing*. Pearson education india, 2009.
- [84] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [85] J. M. Kasson, S. I. Nin, W. Plouffe, and J. L. Hafner, "Performing color space conversions with three-dimensional linear interpolation," *Journal of Electronic Imaging (JEI)*, vol. 4, no. 3, pp. 226–250, 1995.

- [86] 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.
- [87] 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.

# Supplementary Material for In-Loop Filtering Using Learned Look-Up Tables for Video Coding

## APPENDIX A SUPPLEMENTARY METHODOLOGY

In this section, we provide supplementary descriptions and implementation details for some modules of LUT-ILF++ that are only briefly introduced in the main text, including LUT interpolation, LUT compaction implementation. By elaborating on the specific procedures and internal workflows, we aim to improve the clarity and reproducibility of LUT-ILF++.

### A. LUT-ILF++ with Diverse Interpolation

Direct uniform sampling of indexing entries of the full filtering LUT is applied to restrict the rapid growth of storage cost in LUT-ILF++, as mentioned in Section II.A (3) and Section VI of the main text. Due to the non-sampled indexing entries will cause the indexing drift in the LUT retrieve process, the interpolation model is introduced to estimate the drifting entries, and the interpolation process is performed to calculate the final filtered pixel by locating the nearest neighbor indexes (most significant bits) of query indexes (to-be-filtered pixel with reference pixels) and weighting the cached values of neighbor indexes during LUT retrieval. In LUT-ILF++, due to the various types of multi-dimensional LUTs constructed to support diverse filtering operations and goals, such as the reference/progressive indexing with 4D spatial-wise LUTs and cross-component offset indexing with 3D channel-wise LUTs, various types of interpolation model are introduced to support their interpolation process with different dimension numbers, respectively. Specifically, a tri-linear interpolation model is adopted for 3D LUTs, and a 4-simplex interpolation model is used for 4D LUTs, corresponding to the dimensionality of the indexing space in each case. Here we detail the usage of these interpolation models in LUT-ILF++.

1) *Trilinear-based LUT Indexing Entry Interpolation:* The core solution of the linear interpolation scheme is to ensemble known sample values to perform linear weighted interpolations based on the combination of linear distances in each dimension, such as bilinear, bicubic [83], and trilinear model [84]. For 3D LUTs, the trilinear interpolation model [66] is adopted to estimate the non-sampled indexing entries, as shown in Fig. 12.

To perform trilinear interpolation in the 3D LUT space, first, a local cube is defined by the eight nearest neighboring indices (grid points) ranging from  $[u, v, w]$  to  $[u+1, v+1, w+1]$ . During the interpolation process, the most significant bits (MSBs) of the query index (to-be-filtered pixel and its two reference pixel values) are used to locate the local cube (nearest neighboring indices) in the 3D LUT space. Specifically, given the

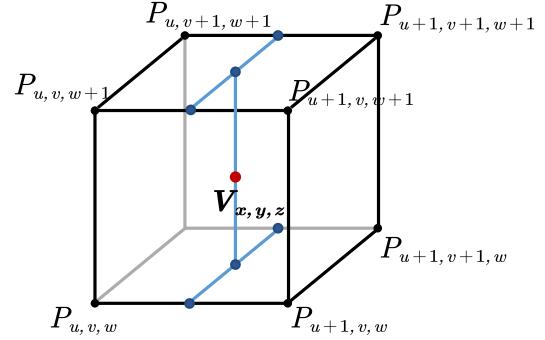


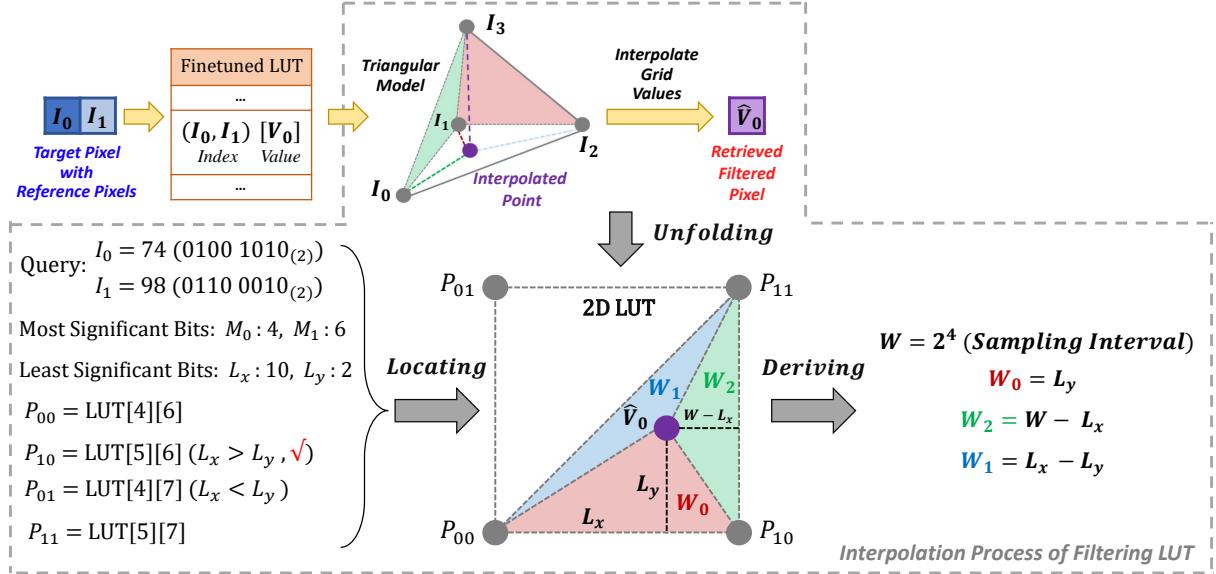
Fig. 12. Illustration of the trilinear interpolation model of a 3D LUT.

target-reference pixel triplet  $(x, y, z)$ , the corresponding index coordinates can be computed as  $u = x \gg L$ ,  $v = y \gg L$ , and  $w = z \gg L$ , where  $L$  is the length of the least significant bits (LSBs). These indices serve as the origin of the interpolation cube, from which the eight surrounding entries are retrieved for weighted averaging. Second, the interpolation weights along each axis are computed based on the fractional offset between the query pixel pair index and its corresponding nearest neighboring indices. These offsets are implicitly encoded in the least significant bits (LSBs), and indicate the relative position of the query index within the interpolation cube. Specifically,  $d_x$ ,  $d_y$ , and  $d_z$  represent the normalized distances from the nearest neighboring indices to the query index along the  $x$ ,  $y$ , and  $z$  dimensions, respectively. These values are then used to derive the trilinear interpolation weights. Using the cached index values of the eight nearest neighboring indices within the local cube, the final filtered (interpolated) value of the target pixel is computed as a weighted sum:

$$\begin{aligned} V_{x,y,z} = & (1-d_x)(1-d_y)(1-d_z)P_{u,v,w} + d_x(1-d_y)(1-d_z)P_{u+1,v,w} \\ & + (1-d_x)d_y(1-d_z)P_{u,v+1,w} + d_xd_y(1-d_z)P_{u+1,v+1,w} \\ & + (1-d_x)(1-d_y)d_zP_{u,v,w+1} + d_x(1-d_y)d_zP_{u+1,v,w+1} \\ & + (1-d_x)d_yd_zP_{u,v+1,w+1} + d_xd_yd_zP_{u+1,v+1,w+1}, \end{aligned} \quad (8)$$

where  $V_{x,y,z}$  denotes the final interpolated (filtered) value of query index  $[x, y, z]$ , and  $P_{u,v,w}$  denotes the cached index value of the  $[u, v, w]$  in the filtering LUT. The interpolation weights  $d_x = \frac{x-(u\ll L)}{1\ll L}$ ,  $d_y = \frac{y-(v\ll L)}{1\ll L}$ , and  $d_z = \frac{z-(w\ll L)}{1\ll L}$  are the normalized distances between the query index and the nearest neighboring indices along the  $x$ ,  $y$ , and  $z$  dimensions, respectively, where  $L$  denotes the number of LSBs.

2) *4-simplex-based LUT Indexing Entry Interpolation:* The formulation of the above linear interpolation scheme can be directly extended to high-dimensional spaces; however, it may face challenges in terms of computational complexity and adaptability to irregularly distributed data in high-dimensional



spaces. To achieve this goal with lower computational complexity, we utilize the well-established simplex interpolation model [85] for a high-dimensional LUT interpolation process, which follows the same model and interpolation scheme used in LUT-based image processing tasks [62], [63], [70], [86], [87]. The core concept is to perform interpolation within a simplex. For an interpolation process in an  $N$ -dimensional space,  $N + 1$  known query indices are identified from the nearest neighboring indices to construct a simplex that encloses the target index (to-be-filtered pixel and its three reference pixel values). The barycentric coordinates of the target index within the simplex are then calculated to determine the interpolation weights relative to these  $N + 1$  vertices.

To detail the 4-simplex interpolation process of 4D LUT in LUT-ILF++, we serve the 2D LUT with the triangular interpolation process as a simple equivalent case, as shown in Fig. 13, from which the 4D interpolation process naturally extends by following the same interpolation principle. In the 2D interpolation process, first, for the query indexing of the target pixel and its one adjacent reference pixel values, four nearest neighboring query indices are retrieved:  $P_{00}$ ,  $P_{01}$ ,  $P_{10}$ , and  $P_{11}$ . Second, among them, three vertices that form a triangle enclosing the target index are selected for interpolation. As the example shown in Fig. 13, the interpolation falls within the bottom right triangle in the case where the LSBs of the reference pixel value are smaller than those of the target to-be-filtered pixel, i.e.,  $L_x > L_y$ . According to the barycentric coordinates of the target index, the interpolation weights can be easily derived as:  $w_0 = L_y$ ,  $w_1 = L_x - L_y$ ,  $w_2 = W - L_x$ , and  $W$  denotes the sampling interval. Finally, the interpolation value can be derived as the weighted sum as follows:

$$\widehat{V}_0 = (w_0 \times P_{11} + w_1 \times P_{10} + w_2 \times P_{00}). \quad (9)$$

Based on the above 2D interpolation case, the 4-simplex interpolation process of 4D LUT can be naturally extended in a similar manner, with the establishment of interpolation simplex selection conditions. The rule of vertices selection can be characterized by the relative magnitudes of the LSB values

TABLE VIII  
SUMMARY RULES OF 4-SIMPLEX INTERPOLATION MODEL  
(FOLLOWED BY [62], [63], [70], [86], [87].)

Condition	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$O_1$	$O_2$	$O_3$
$L_x > L_y > L_z > L_t$	$W - L_x$	$L_x - L_y$	$L_y - L_z$	$L_z - L_t$	$L_t$	$P_{1000}$	$P_{1100}$	$P_{1110}$
$L_x > L_y > L_t > L_z$	$W - L_x$	$L_x - L_y$	$L_y - L_t$	$L_t - L_z$	$L_z$	$P_{1000}$	$P_{1100}$	$P_{1110}$
$L_x > L_t > L_y > L_z$	$W - L_x$	$L_x - L_t$	$L_t - L_y$	$L_y - L_z$	$L_z$	$P_{1000}$	$P_{1001}$	$P_{1101}$
$L_t > L_x > L_y > L_z$	$W - L_t$	$L_t - L_x$	$L_x - L_y$	$L_y - L_z$	$L_z$	$P_{0001}$	$P_{1001}$	$P_{1101}$
$L_x > L_y > L_z > L_t$	$W - L_x$	$L_x - L_y$	$L_y - L_z$	$L_z - L_t$	$L_t$	$P_{1000}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_y > L_t$	$W - L_x$	$L_x - L_z$	$L_z - L_y$	$L_y - L_t$	$L_t$	$P_{1000}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{1000}$	$P_{1010}$	$P_{1101}$
$L_t > L_z > L_x > L_y$	$W - L_t$	$L_t - L_z$	$L_z - L_x$	$L_x - L_y$	$L_y$	$P_{0100}$	$P_{1100}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0100}$	$P_{1100}$	$P_{1110}$
$L_y > L_z > L_x > L_t$	$W - L_y$	$L_y - L_z$	$L_z - L_x$	$L_x - L_t$	$L_t$	$P_{0010}$	$P_{1100}$	$P_{1110}$
$L_y > L_z > L_t > L_x$	$W - L_y$	$L_y - L_z$	$L_z - L_t$	$L_t - L_x$	$L_x$	$P_{0010}$	$P_{1001}$	$P_{1110}$
$L_y > L_x > L_t > L_z$	$W - L_y$	$L_y - L_x$	$L_x - L_t$	$L_t - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_y > L_x > L_t$	$W - L_z$	$L_z - L_y$	$L_y - L_x$	$L_x - L_t$	$L_t$	$P_{0010}$	$P_{1001}$	$P_{1110}$
$L_z > L_x > L_t > L_y$	$W - L_z$	$L_z - L_x$	$L_x - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_x > L_y > L_t$	$W - L_z$	$L_z - L_x$	$L_x - L_y$	$L_y - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_x > L_z > L_y$	$W - L_t$	$L_t - L_x$	$L_x - L_z$	$L_z - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_x > L_y > L_t$	$W - L_z$	$L_z - L_x$	$L_x - L_y$	$L_y - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_y > L_t > L_x$	$W - L_z$	$L_z - L_y$	$L_y - L_t$	$L_t - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_x > L_y$	$W - L_z$	$L_z - L_t$	$L_t - L_x$	$L_x - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_x > L_y > L_z$	$W - L_t$	$L_t - L_x$	$L_x - L_y$	$L_y - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_y > L_z > L_t$	$W - L_x$	$L_x - L_y$	$L_y - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L_x > L_z > L_t$	$W - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_x > L_z > L_t > L_y$	$W - L_x$	$L_x - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_z > L_t > L_y > L_x$	$W - L_z$	$L_z - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_t > L_y > L_x > L_z$	$W - L_t$	$L_t - L_y$	$L_y - L_x$	$L_x - L_z$	$L_z$	$P_{0010}$	$P_{1010}$	$P_{1110}$
$L_y > L$								

This corresponds to selecting the vertices  $P_{1000}$ ,  $P_{1100}$ , and  $P_{1101}$ . Another two vertices  $P_{0000}$ , and  $P_{1111}$  are fixed for all cases. The corresponding weights can be expressed as  $W - L_x$ ,  $L_x - L_y$ ,  $L_y - L_t$ ,  $L_t - L_z$ , and  $L_z$ . The summary rules of simplex selection and the interpolation weights of 4-simplex interpolation model are shown in Table VIII.

### B. The Compaction Implementation of LUT-ILF++

In this subsection, we detail the LUT pruning strategy of the LUT compaction scheme in LUT-ILF++, including the LUT diagonal re-ordering and non-diagonal pruning. Here we also take the first two dimensions (2D,  $[I_0, I_1]$ ) as an example to detail them, as shown in Fig.10 of the main text.

1) *LUT Diagonal Rearranging*: Based on the judgment of the diagonal condition rule ( $|I_0 - I_1| \leq \text{diagonal width } (dw)$ ) in the clipped LUT, the clipped indexing entries are retrieved and classified into diagonal and non-diagonal indexing entries. For diagonal indexing entries, the diagonal re-ordering is used to rearrange them as the diagonal LUT by mapping the LUT coordinate ( $I_c = f_{\text{mapping}}(I_0, I_1, dw)$ ), and the diagonal LUT is then stored as a low-dimensional LUT indexed by  $[I_c, I_2, I_3]$ . In the mapping process, the total number of diagonal indexing entries ( $D$ ) can be calculated as,

$$D = (2 \times dw + 1) \times L - dw \times (dw + 1), \quad (12)$$

among the compacted dimensions in 2D space case, and the first two dimensions of 4D LUT are compacted into one dimension by mapping the index  $[I_0, I_1]$  to compacted index  $I_c$ . The indexing relationship conversion between the index  $[I_0, I_1]$  and compacted index  $I_c$  can be formulated as,

$$I_c = f_{\text{mapping}}(I_0, I_1, dw) = I_1 \times (2 \times dw + 1) + r - 1, \quad (13)$$

where  $L$  indicates the size of each dimension,  $r$  indicates the relative distance between  $I_0$  and  $I_1$ , and it can be calculated as,

$$r = I_0 - I_1 + dw, \quad (0 \leq r \leq 2 \times dw). \quad (14)$$

Through the above mapping manner, the low-dimensional 3D diagonal LUT is compacted from the 4D clipped LUT, and the final index  $[I_c, I_2, I_3]$  is used to retrieve the low-dimensional diagonal LUT. In general use, the LUT re-ordering process can be easily generalized to multiple dimensions of LUT indexing entries.

2) *LUT Non-Diagonal Pruning*: For non-diagonal indexing entries, inspired by the observation of their sparse access concentration in practice (Fig.3 of the main text), their redundancy is pruned by re-sampling their dimensionality (MSBs) with the allocated sparsification shift ( $Q$ ), achieving exponentially reduced storage cost. Starting from the 4D clipped LUT with storage dimension  $(2^{8-q} + 1) \times (2^{8-q} + 1) \times (2^{8-q} + 1) \times (2^{8-q} + 1)$ , as shown in Fig.10 and mentioned in Section II.B (1) of the main text, the whole clipped LUT is further directly re-sampled and sparsified its dimensionality (MSBs) to obtain the non-diagonal LUT with storage dimension  $(2^{8-q-Q} + 1) \times (2^{8-q-Q} + 1) \times (2^{8-q-Q} + 1) \times (2^{8-q-Q} + 1)$ . Note that some cached indexing entries of diagonal LUT are overlapped and cached into non-diagonal LUT to predict the values that do not follow the diagonal condition but are close to the diagonal boundary.

## APPENDIX B SUPPLEMENTARY EXPERIMENTS AND ANALYSES

In this section, first, we supplement the detailed experimental results of *LUT-ILF++*, and detailed comparison results of each sequence with different LUT-based ILF schemes [65], including the detailed BD-rate results of each sequence on regular and low bitrate points, the filter usage ratio and selection results, and subjective exhibition. Second, we supplement some detailed experimental analyses, including the deep analyses of LUT finetuning of different LUT-based ILF schemes [65] and *LUT-ILF++*, the detailed calculation manner of computational complexity and energy cost, and the deployment discussion of LUT-based ILF solution integrated into the codec.

### A. Detailed Performance of LUT-ILF++ under Common Test Condition on Different Coding Configurations

1) *BD-rate Performance of LUT-ILF++ on Regular Bitrate Points (QP 22~42)*: Supplementing Section VII.C (1) of the main text, the R-D performance and usage ratio of the entire LUT-ILF++ filtering framework on the VVC common test sequences is illustrated in Table IX and XI. Y, U, and V represent the R-D performance gain of the three channels of YUV, respectively. We can see that LUT-ILF++ can achieve, on average, 0.82%/2.97%/1.63% and 0.85%/4.11%/2.06%, and achieve up to 1.49% and 1.59% BD-rate reduction (Y component) with high usage ratio and friendly computational complexity (Table II/III of the main text) on VTM-11.0 for all sequences under AI and RA configurations. The experimental results show that the proposed framework performs better for sequences with rich texture and complex scenes, such as *CatRobot*, *RitualDance*, *BQMall*, and *RaceHorses*. The subjective selection and visual results of different LUT-based ILF schemes [65] are also shown and compared in Fig. 14 and 15, which also represents that the LUT-ILF++ can better handle the complex texture regions.

2) *BD-rate Performance of LUT-ILF++ on Low Bitrate Points (QP 27~47)*: To explore the potential and robustness of the proposed framework, we also test it on low-bitrate points (QP 27~47), as shown in Table X and XII. Compared to the results of regular QP points (Table IX and XI), it demonstrates the powerful potential and comprehensive effectiveness of the proposed framework on a wide range of QP points. Specifically, the proposed LUT-ILF++ can achieve, on average, 1.49%/4.69%/2.55% and 1.21%/4.54%/2.21%, and achieve up to 2.43%/8.54%/6.03% and 2.18%/9.73%/4.82% BD-rate reduction on VTM-11.0 for all sequences under the AI and RA configurations.

3) *Usage Ratio*: To verify the efficiency of LUT-ILF++, we also evaluate its usage ratio and compare with other LUT-based ILF schemes [65], as shown in Table IX and X, which is calculated by,  $\text{Ratio} = N_{\text{test}}/N_{\text{total}}$ , where  $N_{\text{test}}$  indicates the number of coding tree units (CTUs) selected and filtered by the corresponding filter, and  $N_{\text{total}}$  indicates the total number of CTUs. Through the comparison of usage ratio, we can observe that the proposed framework achieves a significantly

TABLE IX

BD-RATE RESULTS OF OUR PROPOSED LUT-ILF++ COMPARED TO VTM-11.0 ON CTC TEST SEQUENCES WITH REGULAR-BITRATE POINTS (QP 22~42), AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING SCHEMES [65] UNDER ALL INTRA (AI) CONFIGURATION

Class	Sequence	All Intra (AI) Configuration (%)											
		LUT-ILF-V [65]		LUT-ILF-F [65]		LUT-ILF++							
	Name	Y	U	V	Ratio	Y	U	V	Ratio	Y	U	V	Ratio
ClassA1 (3840x2160)	Tango2	-0.21%	-0.16%	-0.17%	23.13%	-0.33%	-0.44%	-0.53%	39.15%	-0.67%	-5.10%	-2.77%	41.57%
	FoodMarket4	-0.13%	-0.41%	-0.03%	21.33%	-0.26%	-0.24%	-0.41%	33.67%	-0.49%	-1.01%	-0.68%	30.74%
	Campfire	-0.24%	-0.45%	-0.18%	27.60%	-0.32%	-1.03%	-0.72%	40.33%	-0.47%	-0.52%	-1.95%	47.93%
	Average	<b>-0.19%</b>	<b>-0.34%</b>	<b>-0.13%</b>	<b>24.02%</b>	<b>-0.30%</b>	<b>-0.57%</b>	<b>-0.55%</b>	<b>37.72%</b>	<b>-0.54%</b>	<b>-2.21%</b>	<b>-1.80%</b>	<b>40.08%</b>
ClassA2 (3840x2160)	CatRobot1	-0.41%	-0.61%	-0.53%	33.44%	-0.60%	-0.88%	-0.42%	43.16%	-0.91%	-4.80%	-1.60%	51.07%
	DaylightRoad2	-0.44%	-0.34%	-0.13%	32.13%	-0.57%	-0.82%	-0.49%	46.56%	-0.51%	-2.78%	-1.10%	45.70%
	ParkRunning3	-0.21%	-0.28%	-0.15%	29.73%	-0.42%	-0.46%	-0.34%	42.23%	-0.77%	-0.19%	-0.72%	49.69%
	Average	<b>-0.36%</b>	<b>-0.41%</b>	<b>-0.27%</b>	<b>31.77%</b>	<b>-0.53%</b>	<b>-0.73%</b>	<b>-0.42%</b>	<b>43.98%</b>	<b>-0.73%</b>	<b>-2.59%</b>	<b>-1.14%</b>	<b>48.82%</b>
ClassB (1920x1080)	MarketPlace	-0.25%	-0.11%	-0.03%	31.31%	-0.54%	-1.28%	-0.98%	42.18%	-0.79%	-3.54%	-1.95%	49.96%
	RitualDance	-0.38%	-0.51%	-0.32%	37.32%	-0.52%	-1.21%	-1.18%	43.35%	-0.88%	-2.82%	-2.47%	49.78%
	Cactus	-0.21%	-0.38%	-0.12%	24.16%	-0.31%	-0.81%	-1.01%	32.27%	-0.70%	-3.74%	-1.38%	41.46%
	BasketballDrive	0.04%	-0.13%	0.04%	10.27%	-0.02%	-0.59%	-0.22%	18.94%	-0.71%	-2.56%	-1.08%	27.41%
ClassC (832x480)	BQTerrace	-0.33%	-0.47%	-0.42%	31.84%	-0.27%	-1.19%	-0.76%	33.31%	-0.51%	-2.90%	-1.84%	38.27%
	Average	<b>-0.22%</b>	<b>-0.32%</b>	<b>-0.17%</b>	<b>26.97%</b>	<b>-0.33%</b>	<b>-1.02%</b>	<b>-0.83%</b>	<b>34.01%</b>	<b>-0.72%</b>	<b>-3.11%</b>	<b>-1.74%</b>	<b>41.38%</b>
	BasketballDrill	-0.30%	-0.56%	-0.21%	34.31%	-0.37%	-0.91%	-0.33%	41.12%	-0.81%	-2.78%	-1.83%	50.43%
	BQMall	-0.31%	-0.63%	-0.27%	39.26%	-0.59%	-1.57%	-0.27%	51.35%	-0.92%	-4.56%	-1.74%	63.35%
ClassD (416x240)	PartyScene	-0.14%	-0.41%	-0.17%	35.31%	-0.18%	-1.41%	-0.40%	44.67%	-0.67%	-4.35%	-0.99%	64.90%
	RaceHorsesC	-0.29%	-0.52%	0.06%	37.17%	-0.33%	-0.91%	-0.36%	42.57%	-0.41%	-3.90%	-1.73%	41.73%
	Average	<b>-0.26%</b>	<b>-0.53%</b>	<b>-0.14%</b>	<b>36.51%</b>	<b>-0.37%</b>	<b>-1.20%</b>	<b>-0.34%</b>	<b>44.93%</b>	<b>-0.70%</b>	<b>-3.90%</b>	<b>-1.57%</b>	<b>55.10%</b>
	BasketballPass	-0.46%	-1.12%	-0.42%	42.93%	-0.66%	-1.77%	-1.02%	53.38%	-1.27%	-3.28%	-3.57%	68.08%
ClassE (1280x720)	BQSquare	-0.49%	-0.85%	-0.62%	41.08%	-0.52%	-1.12%	-0.52%	53.22%	-1.34%	-1.63%	-1.80%	69.58%
	BlowingBubbles	-0.30%	-0.33%	-0.22%	48.53%	-0.41%	-1.05%	-0.87%	51.77%	-0.83%	-3.45%	-0.63%	66.92%
	RaceHorses	-0.71%	-1.02%	-0.96%	55.28%	-0.97%	-1.19%	-0.79%	61.63%	-1.49%	-5.24%	-2.94%	74.99%
	Average	<b>-0.49%</b>	<b>-0.83%</b>	<b>-0.56%</b>	<b>46.96%</b>	<b>-0.64%</b>	<b>-1.28%</b>	<b>-0.80%</b>	<b>55.01%</b>	<b>-1.23%</b>	<b>-3.40%</b>	<b>-2.24%</b>	<b>69.89%</b>
ClassE (1280x720)	FourPeople	-0.44%	-0.92%	-0.66%	33.74%	-0.59%	-1.03%	-0.72%	46.61%	-1.00%	-1.89%	-0.80%	47.41%
	Johnny	-0.29%	-0.20%	-0.16%	16.64%	-0.54%	-0.83%	-1.07%	23.88%	-0.77%	-2.14%	-1.47%	36.27%
	KristenAndSara	-0.42%	-0.77%	-0.35%	26.63%	-0.61%	-0.88%	-0.89%	37.65%	-0.81%	-2.06%	-0.90%	41.17%
	Average	<b>-0.38%</b>	<b>-0.63%</b>	<b>-0.39%</b>	<b>25.67%</b>	<b>-0.58%</b>	<b>-0.91%</b>	<b>-0.89%</b>	<b>36.04%</b>	<b>-0.86%</b>	<b>-2.03%</b>	<b>-1.06%</b>	<b>41.62%</b>
<i>Overall</i>		<b>-0.32%</b>	<b>-0.51%</b>	<b>-0.26%</b>	<b>31.98%</b>	<b>-0.47%</b>	<b>-0.95%</b>	<b>-0.64%</b>	<b>41.95%</b>	<b>-0.82%</b>	<b>-2.97%</b>	<b>-1.63%</b>	<b>49.50%</b>

TABLE X

BD-RATE RESULTS OF OUR PROPOSED LUT-ILF++ COMPARED TO VTM-11.0 ON CTC TEST SEQUENCES WITH LOW-BITRATE POINTS (QP 27~47), AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING SCHEMES [65] UNDER ALL INTRA (AI) CONFIGURATION

Class	Sequence	All Intra (AI) Configuration (%)							
		LUT-ILF-V [65]		LUT-ILF-F [65]		LUT-ILF++			
	Name	Y	Ratio	Y	Ratio	Y	U	V	Ratio
ClassA1 (3840x2160)	Tango2	-0.61%	47.05%	-0.95%	65.72%	-1.32%	-6.67%	-2.26%	63.16%
	FoodMarket4	-0.21%	40.35%	-0.55%	53.97%	-0.84%	-1.42%	-0.80%	48.46%
	Campfire	-0.64%	44.45%	-0.84%	63.07%	-1.38%	-0.79%	-2.97%	59.61%
	Average	<b>-0.49%</b>	<b>43.95%</b>	<b>-0.78%</b>	<b>60.92%</b>	<b>-1.18%</b>	<b>-2.96%</b>	<b>-2.01%</b>	<b>57.07%</b>
ClassA2 (3840x2160)	CatRobot1	-0.98%	52.26%	-1.40%	67.08%	-1.63%	-6.94%	-2.78%	69.73%
	DaylightRoad2	-0.79%	60.50%	-1.12%	64.96%	-1.27%	-3.69%	-1.39%	64.92%
	ParkRunning3	-0.96%	50.75%	-1.20%	65.41%	-1.47%	-0.63%	-0.20%	68.88%
	Average	<b>-0.91%</b>	<b>54.50%</b>	<b>-1.24%</b>	<b>65.82%</b>	<b>-1.45%</b>	<b>-3.75%</b>	<b>-1.45%</b>	<b>67.85%</b>
ClassB (1920x1080)	MarketPlace	-0.68%	51.74%	-1.08%	67.37%	-1.32%	-6.00%	-1.70%	66.92%
	RitualDance	-0.54%	49.10%	-1.02%	65.60%	-1.38%	-4.19%	-3.93%	70.45%
	Cactus	-0.43%	41.77%	-0.73%	55.19%	-1.32%	-7.39%	-2.17%	57.31%
	BasketballDrive	-0.10%	25.65%	-0.24%	38.51%	-1.37%	-3.28%	-1.64%	37.85%
ClassC (832x480)	BQTerrace	-0.66%	44.98%	-0.76%	50.63%	-1.03%	-4.76%	-2.68%	52.98%
	Average	<b>-0.48%</b>	<b>42.65%</b>	<b>-0.76%</b>	<b>55.46%</b>	<b>-1.28%</b>	<b>-5.12%</b>	<b>-2.42%</b>	<b>57.11%</b>
	BasketballDrill	-0.77%	57.23%	-1.00%	62.40%	-1.50%	-3.92%	-2.36%	70.33%
	BQMall	-0.70%	60.61%	-1.02%	73.47%	-1.55%	-7.64%	-2.91%	83.14%
ClassD (416x240)	PartyScene	-0.53%	59.64%	-0.71%	68.09%	-1.23%	-8.54%	-2.08%	84.67%
	RaceHorsesC	-0.55%	47.73%	-0.64%	69.88%	-0.83%	-6.55%	-3.30%	61.62%
	Average	<b>-0.64%</b>	<b>56.30%</b>	<b>-0.84%</b>	<b>68.46%</b>	<b>-1.28%</b>	<b>-6.66%</b>	<b>-2.66%</b>	<b>74.94%</b>
	BasketballPass	-0.97%	64.91%	-1.37%	78.58%	-2.09%	-5.30%	-6.03%	88.08%
ClassD (416x240)	BQSquare	-1.11%	65.08%	-1.41%	79.08%	-2.35%	-2.07%	-4.42%	89.58%
	BlowingBubbles	-0.69%	67.50%	-0.93%	72.91%	-1.47%	-6.00%	-1.45%	86.92%
	RaceHorses	-1.53%	79.25%	-1.87%	84.91%	-2.43%	-7.69%	-4.39%	94.83%
	Average	<b>-1.08%</b>	<b>69.18%</b>	<b>-1.40%</b>	<b>78.87%</b>	<b>-2.09%</b>	<b>-5.26%</b>	<b>-4.07%</b>	<b>89.85%</b>
ClassE (1280x720)	FourPeople	-1.04%	56.50%	-1.47%	71.10%	-1.78%	-3.25%	-1.99%	67.28%
	Johnny	-0.73%	33.14%	-1.12%	46.24%	-1.42%	-3.09%	-2.77%	55.11%
	KristenAndSara	-1.08%	45.24%	-1.36%	59.49%	-1.61%	-3.31%	-1.84%	61.03%
	Average	<b>-0.95%</b>	<b>44.96%</b>	<b>-1.32%</b>	<b>58.94%</b>	<b>-1.60%</b>	<b>-3.22%</b>	<b>-2.20%</b>	<b>61.15%</b>
<i>Overall</i>		<b>-0.74%</b>	<b>51.92%</b>	<b>-1.03%</b>	<b>64.74%</b>	<b>-1.49%</b>	<b>-4.69%</b>	<b>-2.55%</b>	<b>67.99%</b>

TABLE XI

BD-RATE RESULTS OF OUR PROPOSED LUT-ILF++ COMPARED TO VTM-11.0 ON CTC TEST SEQUENCES WITH REGULAR-BITRATE POINTS (QP 22~42), AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING SCHEMES [65] UNDER RANDOM ACCESS (RA) CONFIGURATION

Class	Sequence	Random Access (RA) Configuration (%)								
		LUT-ILF-V [65]			LUT-ILF-F [65]			LUT-ILF++		
	Name	Y	U	V	Y	U	V	Y	U	V
ClassA1 (3840x2160)	Tango2	-0.27%	-0.26%	-0.13%	-0.39%	-0.36%	-0.31%	-0.61%	-6.82%	-1.55%
	FoodMarket4	-0.13%	-0.92%	-0.16%	-0.10%	-0.80%	-0.43%	-0.58%	-1.23%	-0.42%
	Campfire	-0.20%	-0.16%	-0.05%	-0.14%	-0.07%	-0.25%	-1.13%	-0.69%	-2.75%
	Average	<b>-0.20%</b>	<b>-0.45%</b>	<b>-0.12%</b>	<b>-0.21%</b>	<b>-0.41%</b>	<b>-0.33%</b>	<b>-0.77%</b>	<b>-2.91%</b>	<b>-1.57%</b>
ClassA2 (3840x2160)	CatRobot1	-0.30%	-0.71%	-0.51%	-0.49%	-0.74%	-0.28%	-0.90%	-6.12%	-2.34%
	DaylightRoad2	-0.25%	-0.44%	-0.31%	-0.43%	-0.71%	-0.30%	-0.79%	-5.24%	-0.40%
	ParkRunning3	-0.21%	-0.38%	0.07%	-0.13%	-0.30%	-0.12%	-0.90%	-0.72%	-0.10%
	Average	<b>-0.25%</b>	<b>-0.51%</b>	<b>-0.25%</b>	<b>-0.35%</b>	<b>-0.58%</b>	<b>-0.23%</b>	<b>-0.86%</b>	<b>-4.03%</b>	<b>-0.95%</b>
ClassB (1920x1080)	MarketPlace	-0.30%	-0.48%	-0.14%	-0.66%	-0.65%	-0.79%	-0.99%	-5.23%	-2.72%
	RitualDance	-0.23%	-0.38%	-0.04%	-0.20%	-0.39%	-0.74%	-0.92%	-4.83%	-3.26%
	Cactus	-0.25%	-0.42%	-0.07%	-0.37%	-0.57%	-0.88%	-0.86%	-6.90%	-2.48%
	BasketballDrive	-0.10%	-0.25%	-0.11%	-0.19%	-0.04%	-0.55%	-0.45%	-2.48%	-1.65%
ClassC (832x480)	BQTerrace	-0.19%	-0.57%	-0.30%	-0.15%	-0.01%	-0.51%	-0.53%	-3.82%	-2.06%
	Average	<b>-0.21%</b>	<b>-0.42%</b>	<b>-0.14%</b>	<b>-0.31%</b>	<b>-0.33%</b>	<b>-0.70%</b>	<b>-0.75%</b>	<b>-4.65%</b>	<b>-2.44%</b>
	BasketballDrill	-0.23%	-0.66%	-0.19%	-0.25%	-0.75%	-0.11%	-0.45%	-2.61%	-1.71%
	BQMall	-0.13%	-0.73%	-0.04%	-0.27%	-1.41%	-0.35%	-0.75%	-5.82%	-2.06%
ClassD (416x240)	PartyScene	-0.09%	-0.51%	-0.66%	-0.17%	-0.76%	-0.70%	-0.54%	-6.96%	-1.68%
	RaceHorsesC	-0.13%	-0.59%	-0.14%	-0.09%	-1.24%	0.09%	-0.43%	-6.30%	-3.38%
	Average	<b>-0.14%</b>	<b>-0.62%</b>	<b>-0.26%</b>	<b>-0.19%</b>	<b>-1.04%</b>	<b>-0.26%</b>	<b>-0.54%</b>	<b>-5.42%</b>	<b>-2.21%</b>
	BasketballPass	-0.38%	-0.24%	-0.38%	-0.41%	-0.96%	-0.30%	-1.04%	-1.47%	-4.25%
ClassD (416x240)	BQSquare	-0.48%	-0.95%	-0.51%	-0.53%	-1.01%	-0.34%	-1.34%	-1.89%	-2.94%
	BlowingBubbles	-0.22%	-0.66%	-0.65%	-0.38%	-0.89%	-0.57%	-0.77%	-6.33%	-0.44%
	RaceHorses	-0.30%	-0.73%	-0.16%	-0.55%	-1.12%	-0.52%	-1.59%	-8.19%	-4.33%
	Average	<b>-0.35%</b>	<b>-0.65%</b>	<b>-0.41%</b>	<b>-0.46%</b>	<b>-0.99%</b>	<b>-0.44%</b>	<b>-1.18%</b>	<b>-4.47%</b>	<b>-2.99%</b>
ClassE (1280x720)	FourPeople	-0.51%	-0.46%	-0.64%	-0.52%	-0.97%	-0.90%	-1.23%	-2.02%	-1.10%
	Johnny	-0.35%	0.38%	-0.19%	-0.61%	-0.10%	-0.56%	-0.77%	-1.97%	-1.98%
	KristenAndSara	-0.46%	-0.35%	0.53%	-0.73%	-0.43%	-0.41%	-0.84%	-2.79%	-1.62%
	Average	<b>-0.44%</b>	<b>-0.14%</b>	<b>-0.10%</b>	<b>-0.62%</b>	<b>-0.50%</b>	<b>-0.61%</b>	<b>-0.95%</b>	<b>-2.26%</b>	<b>-1.57%</b>
<i>Overall</i>		<b>-0.26%</b>	<b>-0.47%</b>	<b>-0.22%</b>	<b>-0.35%</b>	<b>-0.65%</b>	<b>-0.44%</b>	<b>-0.85%</b>	<b>-4.11%</b>	<b>-2.06%</b>

TABLE XII

BD-RATE RESULTS OF OUR PROPOSED LUT-ILF++ COMPARED TO VTM-11.0 ON CTC TEST SEQUENCES WITH LOW-BITRATE POINTS (QP 27~47), AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING SCHEMES [65] UNDER RANDOM ACCESS (RA) CONFIGURATION

Class	Sequence	Random Access (RA) Configuration (%)								
		LUT-ILF-V [65]			LUT-ILF-F [65]			LUT-ILF++		
	Name	Y	U	V	Y	U	V	Y	U	V
ClassA1 (3840x2160)	Tango2	-0.52%			-0.75%			-0.91%	-7.32%	-1.14%
	FoodMarket4	-0.24%			-0.36%			-0.70%	-1.17%	-0.58%
	Campfire	-0.21%			-0.24%			-1.40%	-1.06%	-2.24%
	Average	<b>-0.33%</b>			<b>-0.45%</b>			<b>-1.00%</b>	<b>-3.19%</b>	<b>-1.32%</b>
ClassA2 (3840x2160)	CatRobot	-0.63%			-0.56%			-1.17%	-6.54%	-3.00%
	DaylightRoad2	-0.28%			-0.94%			-1.06%	-4.67%	-0.13%
	ParkRunning3	-0.23%			-0.40%			-1.35%	-1.03%	-0.20%
	Average	<b>-0.38%</b>			<b>-0.64%</b>			<b>-1.19%</b>	<b>-4.08%</b>	<b>-1.11%</b>
ClassB (1920x1080)	MarketPlace	-0.41%			-0.73%			-1.08%	-4.86%	-2.93%
	RitualDance	-0.26%			-0.40%			-1.05%	-4.76%	-2.90%
	Cactus	-0.24%			-0.44%			-0.94%	-7.55%	-2.36%
	BasketballDrive	-0.49%			-0.30%			-0.82%	-2.28%	-1.39%
ClassC (832x480)	BQTerrace	-0.39%			-0.33%			-1.02%	-4.59%	-2.11%
	Average	<b>-0.36%</b>			<b>-0.44%</b>			<b>-0.98%</b>	<b>-4.81%</b>	<b>-2.34%</b>
	BasketballDrill	-0.42%			-0.66%			-0.76%	-2.28%	-1.80%
	BQMall	-0.23%			-0.63%			-1.06%	-7.08%	-2.85%
ClassD (416x240)	PartyScene	-0.11%			-0.24%			-0.88%	-9.73%	-2.25%
	RaceHorses	-0.38%			-0.39%			-0.99%	-7.20%	-2.76%
	Average	<b>-0.28%</b>			<b>-0.47%</b>			<b>-0.92%</b>	<b>-6.57%</b>	<b>-2.42%</b>
	BasketballPass	-0.52%			-0.74%			-1.40%	-1.98%	-4.42%
ClassD (416x240)	BQSquare	-0.36%			-0.62%			-2.18%	-1.86%	-4.82%
	BlowingBubbles	-0.42%			-0.47%			-1.24%	-7.49%	-0.86%
	RaceHorses	-0.47%			-0.57%			-1.93%	-8.89%	-3.68%
	Average	<b>-0.44%</b>			<b>-0.59%</b>			<b>-1.69%</b>	<b>-5.06%</b>	<b>-3.45%</b>
ClassE (1280x720)	FourPeople	-0.75%			-0.76%			-1.69%	-2.22%	-1.41%
	Johnny	-0.36%			-0.84%			-1.16%	-2.30%	-2.93%
	KristenAndSara	-0.53%			-0.81%			-1.45%	-3.10%	-1.89%
	Average	<b>-0.55%</b>			<b>-0.80%</b>			<b>-1.43%</b>	<b>-2.54%</b>	<b>-2.08%</b>
<i>Overall</i>		<b>-0.39%</b>			<b>-0.57%</b>			<b>-1.21%</b>	<b>-4.54%</b>	<b>-2.21%</b>

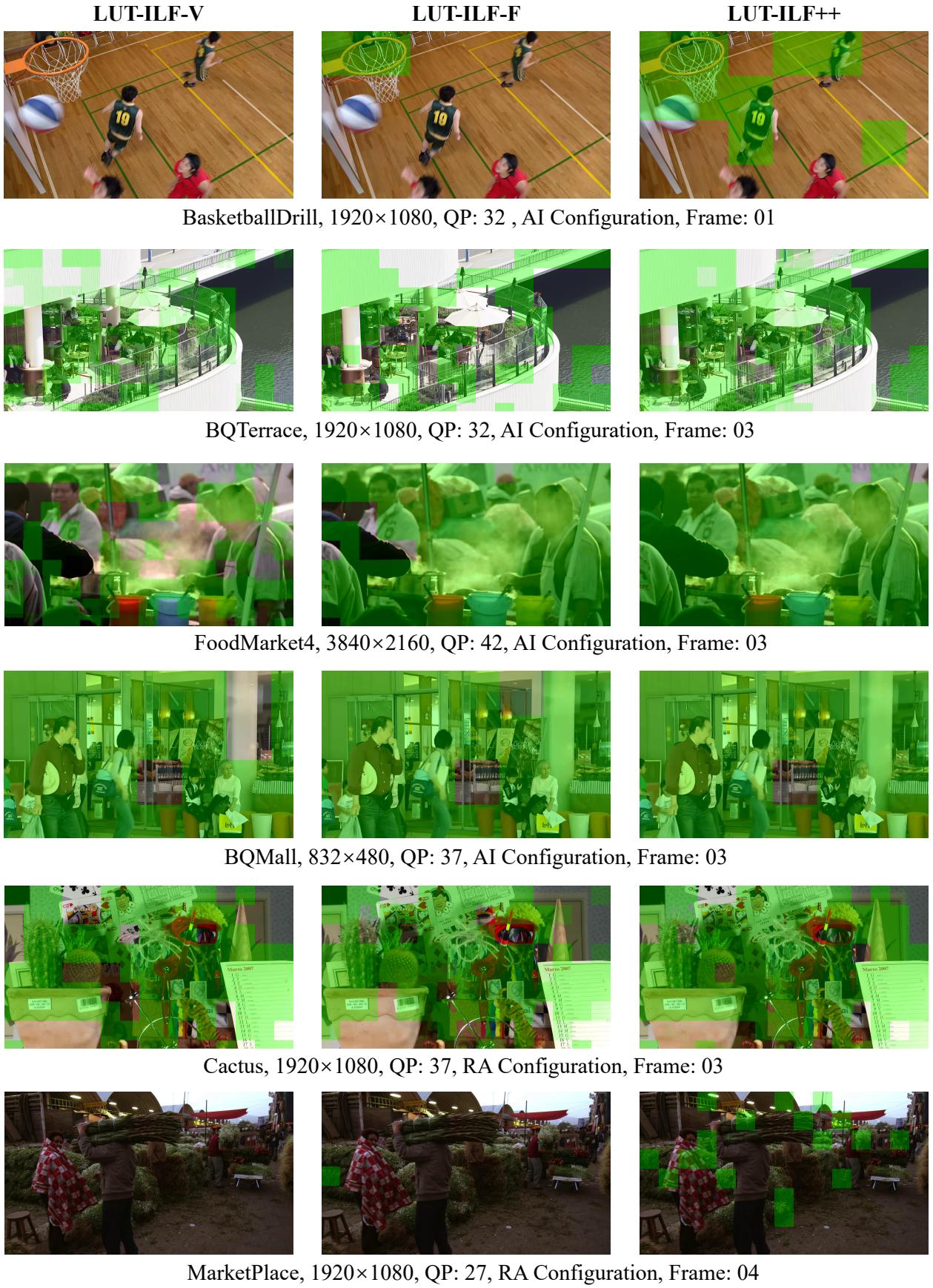


Fig. 14. The filter selection comparison results of different LUT-based ILF schemes [65] (LUT-ILF-V, LUT-ILF-F) and the proposed LUT-ILF++ on several test sequences with rich textures and complex scenes under AI or RA configuration are presented. For each sequence, we select the most significant comparisons from the first five coding frames to demonstrate the advantages of our proposed LUT-ILF++. The green blocks indicate the regions filtered by the corresponding schemes, and a larger area covered in green indicates a higher usage ratio of the corresponding filter.

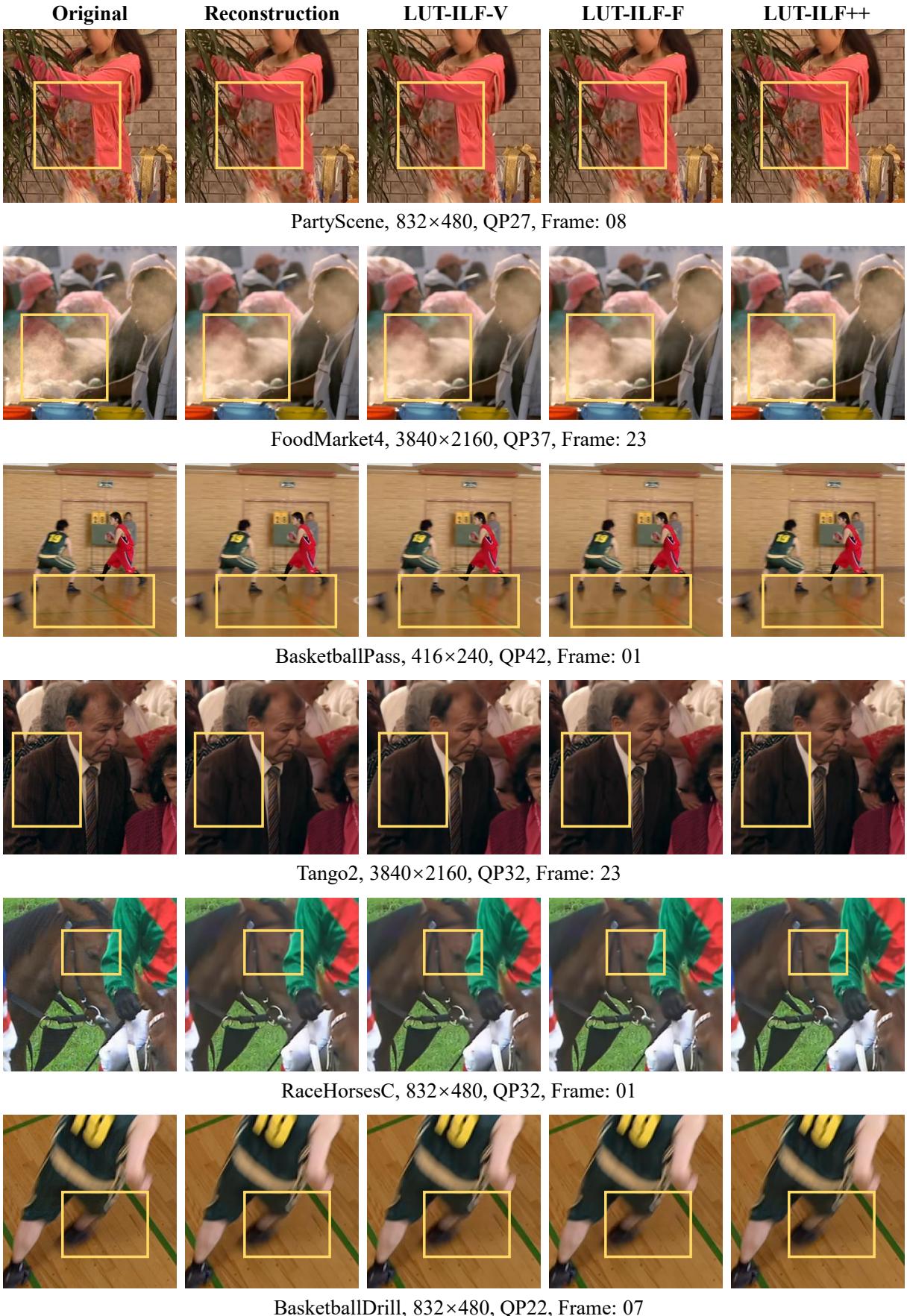


Fig. 15. The subjective comparison results of original frames, reconstructed frames, the filtered results of different LUT-based ILF schemes [65] (LUT-ILF-V, LUT-ILF-F), and the filtered results of the proposed LUT-ILF++ under AI or RA configuration are presented. For each sequence, we select the most significant comparisons to demonstrate the advantages of our proposed LUT-ILF++. The yellow blocks indicate the regions filtered by the corresponding filtering schemes.

TABLE XIII

THE COMPUTATIONAL COMPLEXITY RESULTS AND SPECIFIC OPERATION NUM OF LUMA COMPONENT OF OUR BASIC FILTERING FRAMEWORK (*LUT-ILF-U/V/F*) AND OUR PROPOSED IMPROVED LUMA FILTERING FRAMEWORK (*LUT-ILF++*) ON THE PIXEL (PER PIXEL) AND FRAME LEVEL (A  $1920 \times 1080$  HD FRAME)

<b>Operation</b>	<b>Level</b>	<i>Operation Num of LUT-ILF-U</i>	<i>Operation Num of LUT-ILF-V</i>	<i>Operation Num of LUT-ILF-F</i>	<i>Operation Num of LUT-ILF++ (w/o Compaction)</i>	<i>Operation Num of LUT-ILF++ (w/ Compaction)</i>
int8 Add	Pixel-wise	70	206	478	1001	1553
int8 Multiply		4	4	4	14	14
int32 Add		68	190	446	1107	1107
int32 Multiply		55	152	344	919	919
Total Add		138	396	924	2108	2660
Total Multiply		59	156	348	933	933
int8 Add	Frame-wise	145,152,000	427,161,600	991,180,800	2,075,673,600	3,220,300,800
int8 Multiply		8,294,400	8,294,400	8,294,400	29,030,400	29,030,400
int32 Add		141,004,800	393,984,000	924,825,600	2,295,475,200	2,295,475,200
int32 Multiply		114,048,000	315,187,200	713,318,400	1,905,638,400	1,905,638,400
<b>Total Add</b>		<b>286,156,800</b>	<b>821,145,600</b>	<b>1,916,006,400</b>	<b>4,371,148,800</b>	<b>5,515,776,000</b>
<b>Total Multiply</b>	Frame-wise	<b>122,342,400</b>	<b>323,481,600</b>	<b>721,612,800</b>	<b>1,934,668,800</b>	<b>1,934,668,800</b>
<b>Computational Complexity (Ops/pixel)</b>		<b>0.20</b>	<b>0.55</b>	<b>1.27</b>	<b>3.04</b>	<b>3.59</b>
<b>Energy Cost<sup>1</sup>(pJ/pixel)</b>	<b>Pixel-wise</b>	<b>180.2</b>	<b>497.2</b>	<b>1126.1</b>	<b>2992.43</b>	<b>3008.99</b>

<sup>1</sup> The energy cost is calculated according to [78]–[80] mentioned in Section VII.A of main text.

TABLE XIV

THE COMPUTATIONAL COMPLEXITY RESULTS AND SPECIFIC OPERATION NUM OF LUMA COMPONENT OF OUR ADOPTED 3D/4D CHANNEL LUT AND EACH CHROMA COMPONENT OF OUR PROPOSED IMPROVED CHROMA FILTERING FRAMEWORK (*LUT-ILF++*) ON THE PIXEL (PER PIXEL) AND FRAME LEVEL (A  $1920 \times 1080$  HD FRAME)

<b>Operation</b>	<b>Level</b>	<i>Operation Num of Trilinear-based 3D Channel LUT</i>	<i>Operation Num of 4-simplex-based 4D Channel LUT</i>	<i>Operation Num of LUT-ILF++ (w/o Compaction) on Each Chroma Component</i>	<i>Operation Num of LUT-ILF++ (w/ Compaction) on Each Chroma Component</i>
int8 Add	Pixel-wise	37	21.5	964	1612
int8 Multiply		2	2	20	20
int32 Add		27	28	882	882
int32 Multiply		77	27	761	761
Total Add		64	49.5	1846	2494
Total Multiply		79	29	781	781
int8 Add	Frame-wise	76,723,200	44,582,400	499,737,600	835,660,800
int8 Multiply		4,147,200	4,147,200	10,368,000	10,368,000
int32 Add		55,987,200	58,060,800	457,228,800	457,228,800
int32 Multiply		159,667,200	55,987,200	394,502,400	394,502,400
<b>Total Add</b>	Frame-wise	<b>132,710,400</b>	<b>102,643,200</b>	<b>956,966,400</b>	<b>1,292,889,600</b>
<b>Total Multiply</b>		<b>163,814,400</b>	<b>60,134,400</b>	<b>404,870,400</b>	<b>404,870,400</b>
<b>Computational Complexity (Ops/pixel)</b>	<b>Pixel-wise</b>	<b>0.14</b>	<b>0.08</b>	<b>0.66</b>	<b>0.82</b>
<b>Energy Cost<sup>1</sup>(pJ/pixel)</b>	<b>Pixel-wise</b>	<b>242.91</b>	<b>87.55</b>	<b>620.06</b>	<b>624.92</b>

<sup>1</sup> The energy cost is calculated according to [78]–[80] mentioned in Section VII.A of main text.

TABLE XV  
THE ABLATION STUDY OF LUT FINTUNING OF DIFFERENT LUT-BASED ILF MODELS UNDER ALL INTRA (AI) CONFIGURATION

Schemes	One-Step Finetuning	Two-Step Finetuning	Y BD-rate (%)	Training Iteration	Training Time (days)
LUT-ILF-U (DNN)	—	—	-0.13%	400000	3
LUT-ILF-U (LUT)	✗	—	0.19%	—	—
LUT-ILF-U (LUT)	✓	—	-0.08%	20000	0.5
LUT-ILF-V (DNN)	—	—	-0.36%	400000	17
LUT-ILF-V (LUT)	✗	—	-0.07%	—	—
LUT-ILF-V (LUT)	✓	—	-0.32%	20000	4
LUT-ILF-F (DNN)	—	—	-0.51%	400000	27
LUT-ILF-F (LUT)	✗	—	-0.15%	—	—
LUT-ILF-F (LUT)	✓	—	-0.47%	20000	4
LUT-ILF++ (DNN)	—	—	-0.87%	400000	32
LUT-ILF++ (LUT)	✗	✗	-0.39%	—	—
LUT-ILF++ (LUT)	✓	✗	-0.77%	20000	4
LUT-ILF++ (LUT)	✓	✓	<b>-0.82%</b>	<b>20000</b>	<b>7</b>

higher selection proportion, averaging 49.50%/67.99% on regular/low bitrate points under AI configuration, and achieving up to 74.99%/94.83%, respectively. The results demonstrate its remarkable filtering capability and effectiveness. (Note that the usage ratio of different schemes is not shown and compared under RA configuration, because some schemes, LUT-ILF-U/V/F, disable filtering on some temporal layers to avoid temporal error accumulation, while LUT-ILF++ enables all layers, a direct comparison may be unfair.)

### B. Deep Analyses of Finetuning in LUT-based ILF Solutions

Based on Section II.A (3), Section VI.B and Section VII.B (5) of main text, LUT finetuning serves as a crucial step for transferring the filtering capacity of deep neural network to the practical LUTs. Specifically, the finetuning with interpolation adaptation is used to bridge these transformations and ensure that the impact on the coding gain remains minimal, ensuring that the LUT-based ILF maintains filtering capacity while significantly reducing computational/time complexity.

To further analyze the benefits that finetuning brings to the LUT-based ILF solution, we conduct ablation studies on all LUT-based ILF models to demonstrate the critical importance of these prolonged LUT training, transferring, and finetuning processes. Specifically, for the ablation setting of LUT-ILF-U, LUT-ILF-V and LUT-ILF-F [65], since they only adopt the uniformly sampled pruning and storage strategies, their filtering LUTs only require a single finetuning stage. Thus, the ablation study for these models focuses on evaluating the effect of this single step. In contrast, as detailed in the Section VI.B and Fig.11 of main text, the training of LUT-ILF++ contains a two-step finetuning process: the first step finetunes the LUTs after uniform clipping, and the second step finetunes them again after non-uniform (diagonal-oriented) sampling. Therefore, we perform ablation studies on both finetuning steps to analyze their contributions to the final performance.

Based on Table II of the main text, in Table XV, here we present the comparison results of different LUT-based ILF models at various stages of the above ablation pipelines under all intra (AI) configuration. In detail, these stages are evaluated: (1) reproduced DNN training stage, representing

the baseline performance achieved by the fully trained neural network before any LUT conversion; (2) DNN-to-LUT conversion stage, reflecting the performance degradation caused by directly transferring the DNN into LUT without any finetuning; (3) LUT finetuning stage, the LUTs are finetuned with an interpolation model adaptation to regain lost performance after conversion. As shown in Table XV, for LUT-ILF-U, LUT-ILF-V, and LUT-ILF-F, it can be observed that these models achieve notable performance degradation when directly converting from DNNs to LUTs without finetuning, while a single-step finetuning effectively restores their performance to the DNN baseline with minimal additional training cost. For LUT-ILF++, the results further verify that each finetuning step contributes progressively to performance improvement. These findings confirm that the finetuning strategy plays a critical role in maximizing the filtering effectiveness, especially in LUT-ILF++, ensuring that its compacted LUT representation maintains coding performance comparable to that of the reproduced DNN models.

### C. Detailed Calculation Manner of Computational Complexity and Energy Cost

To clearly show the calculation process of the computational complexity of the proposed framework and facilitate researchers to follow the LUT-based ILF solution, here we detail the specific operation num of the basic framework (LUT-ILF-U/V/F in [65]) and improved framework (LUT-ILF++) on the pixel (per pixel) and frame level (a image/video frame with  $1920 \times 1080$  spatial resolution), as shown in Table XIII. For the extension of this solution, computational complexity can be calculated with the reference of the basic architecture of LUT-ILF-U for relative expansion. For the basic architecture, it is constructed by incorporating pattern 1 of Fig.5 of main text for reference indexing mechanism and a two-step cascaded filtering iteration (iter = 2) for progressive indexing, enabling a  $5 \times 5$  reference range.

As shown in Table XIII, the detailed construction of operation counts, computational complexity, and energy cost for LUT-ILF-U, LUT-ILF-V, LUT-ILF-F, and LUT-ILF++ is presented. Specifically, we report the results of LUT-ILF++ before and after applying the LUT compaction scheme (Section VI of main text). It can be observed that the application of the proposed LUT compaction scheme not only greatly reduces the storage cost of the entire framework but also only brings a slight increase in the number of addition operations while keeping the number of multiplication operations unchanged, demonstrating its excellent hardware friendliness.

As shown in Table XIV, the detailed construction of operation counts, computational complexity, and energy cost of the 3D channel LUT with trilinear interpolation model and the 4D channel LUT with 4-simplex interpolation model is presented. Compared to the trilinear model for 3D LUT, although the 4-simplex model for 4D channel LUT involves a more complex formulation, the adopted interpolation model only requires partial surrounding storage index interpolation, effectively controlling its overall computational complexity. In addition, the detailed construction of the improved framework (LUT-ILF++) for each chroma component is also presented.

TABLE XVI  
THE COMPARISON OF PEAKING MEMORY CONSUMPTION BETWEEN  
VTM DECODER AND DIFFERENT LUT-BASED ILF MODELS

Schemes	Storage Manner	Peak Operating Memory (MB)	Computational Complexity (Ops/pixel)	Storage Cost	Energy Cost ( $pJ/pixel$ )
VTM Decoder	—	275.4MB	—	—	—
LUT-ILF-V	Uniform	53.7 MB	0.83K Ops/pixel	1476 KB	0.75K
LUT-ILF-F	Uniform	59.4 MB	1.91K Ops/pixel	3444 KB	1.69K
<b>LUT-ILF++</b>	Non-uniform	<b>56.3 MB</b>	<b>5.23K Ops/pixel</b>	<b>812 KB</b>	<b>4.26K</b>

Compared to the luma component, we adopt a cross-color-component collaborative scheme for chroma filtering (Section V of main text), which effectively improves performance while consuming fewer computational resources.

#### D. Deployment Discussion of LUT-based ILF Solution Integrated Into Codec

Based on our reported LUT storage consumption of the whole LUT-based ILF solutions (LUT-ILF-U/V/F, and LUT-ILF++) mentioned in Table II and III of main text, here we further verify their practical deployability by measuring the peak memory usage on CPU using *Memray*<sup>1</sup>, respectively. As shown in Table XVI, all LUT-based filtering models with only integer precision operations exhibit significantly lower peak memory consumption compared to the consumption of the entire decoder during runtime, occupying only about one-fifth of the VTM decoder's peak memory, demonstrating that LUT-based ILF solutions impose minimal additional runtime memory demands and are thus favorable for practical codec deployment. Note that the proposed LUT compaction and pruning strategies in LUT-ILF++ not only improve its advantage in reducing storage consumption but also maintain lower runtime storage consumption.

<sup>1</sup>Memray in the media: <https://github.com/bloomberg/memray>