In-Loop Filtering via Trained Look-Up Tables

Zhuoyuan Li, Jiacheng Li, Yao Li, Li Li,
Dong Liu, and Feng Wu*

Intelligent Visual Data Coding Laboratory (iVC)
University of Science and Technology of China (USTC)

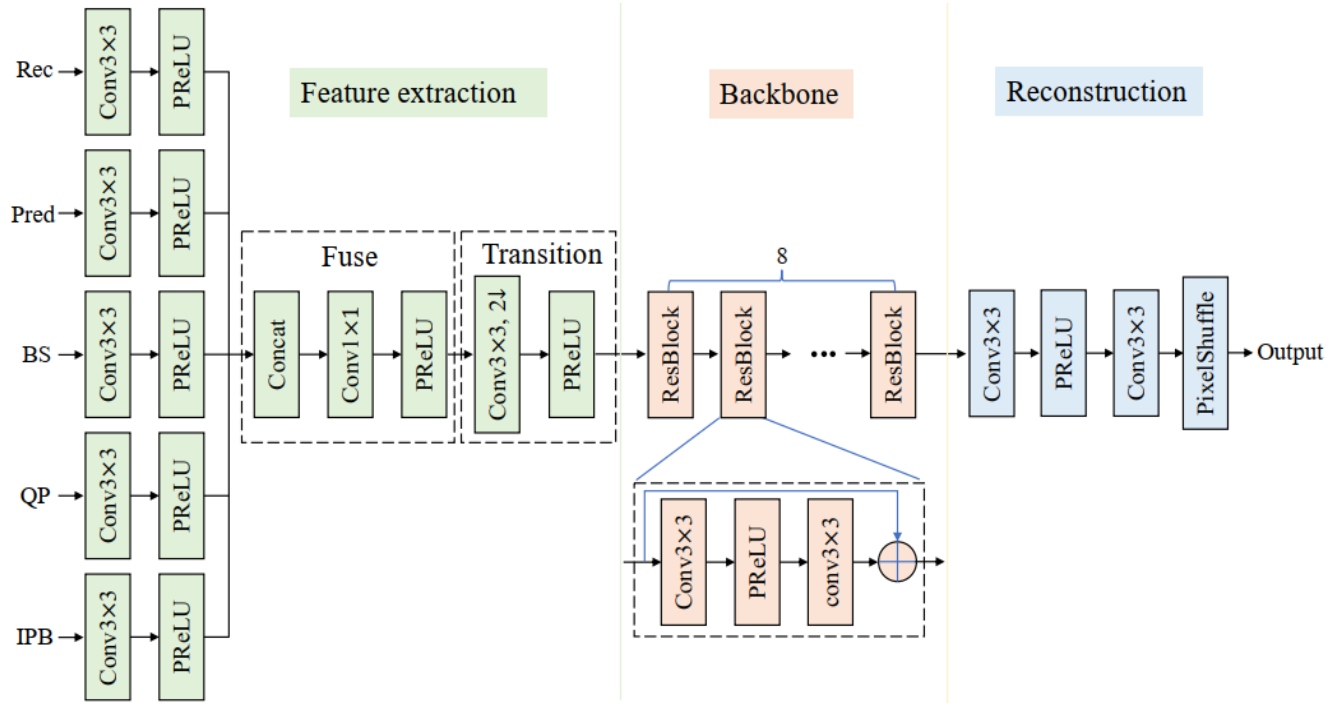
December 28, 2024 @ VCIP 2024

Homepage: https://zhuoyuanli1997.github.io/

Background

Neural network-based in-loop filtering (NNLF) *methods achieve remarkable coding gains beyond the capability of advanced video coding standards, which* ***becomes a powerful coding tool candidate for future video coding standards****.*

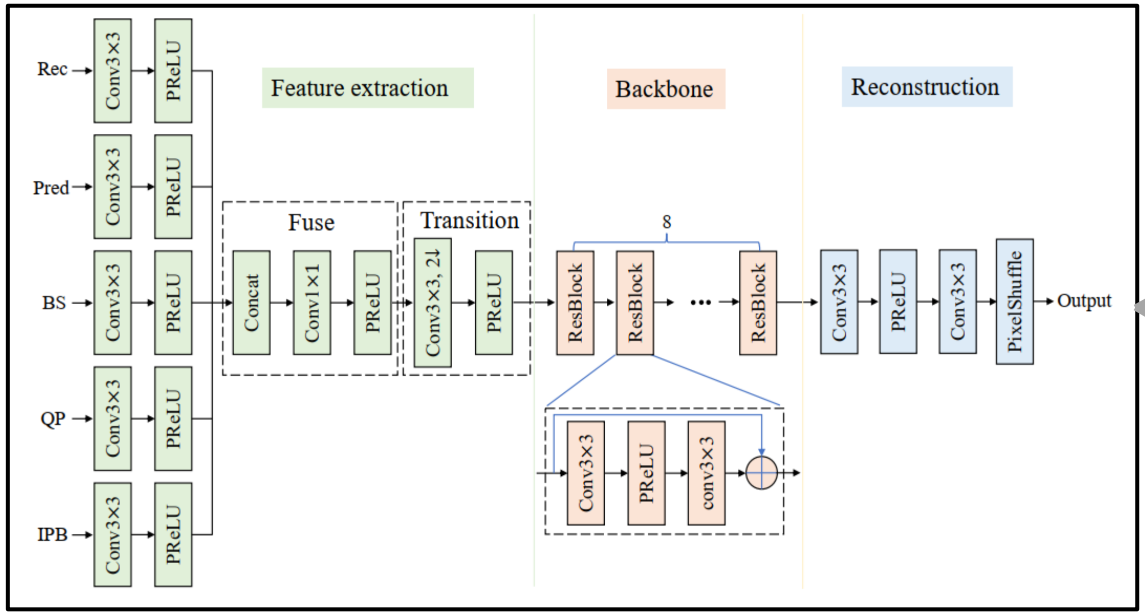
Backbone of NNLF in VVC
NN-based Exploration Extension (NNVC)
(e.g. JVET-AF0014 , JVET-AF0041 , JVET-AF0043)



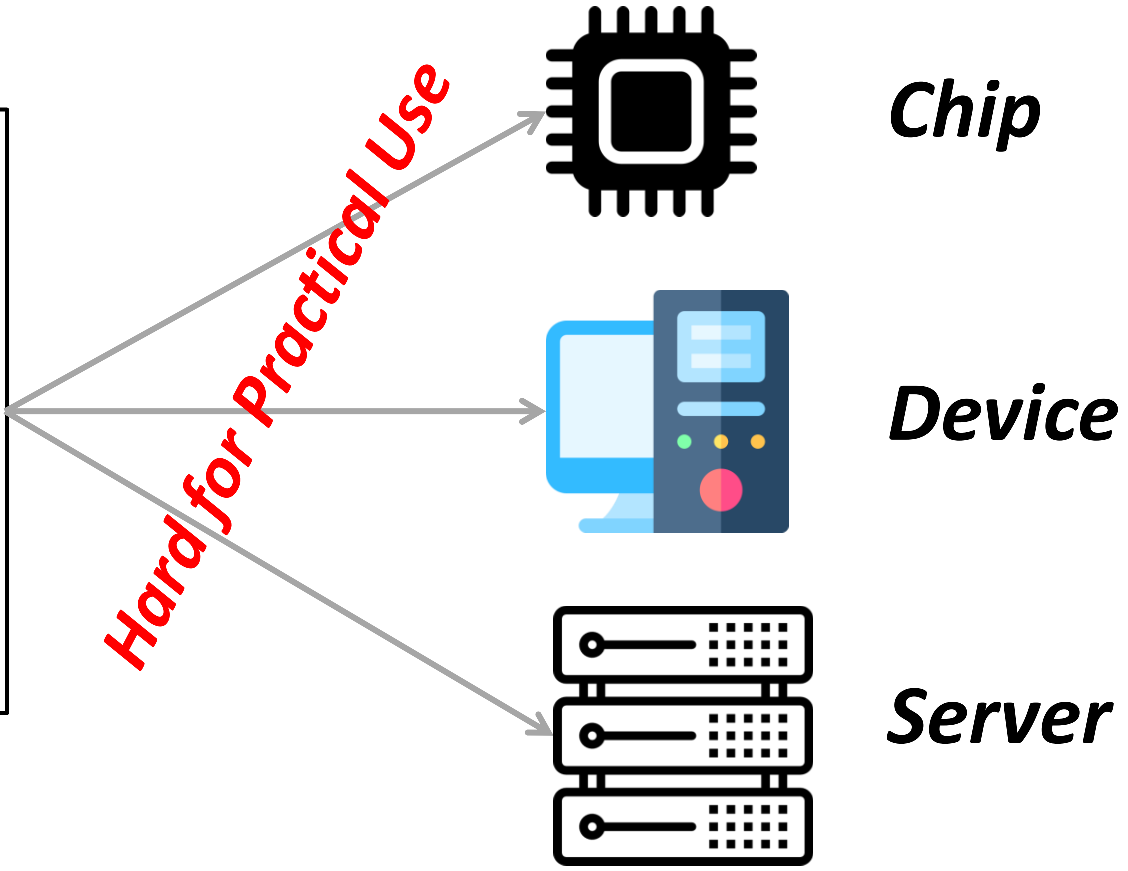
Performance: Lop-17.0 kMACs/pixel with -4.61%～-4.78% in AI , Hop-477.0 kMACs/pixel with -7.79%～-7.91% in AI

Concept

*The utilization of **deep neural networks (DNN)** brings **high computational complexity and raises high demand of dedicated hardware**, which is **challenging to apply into general use**.*



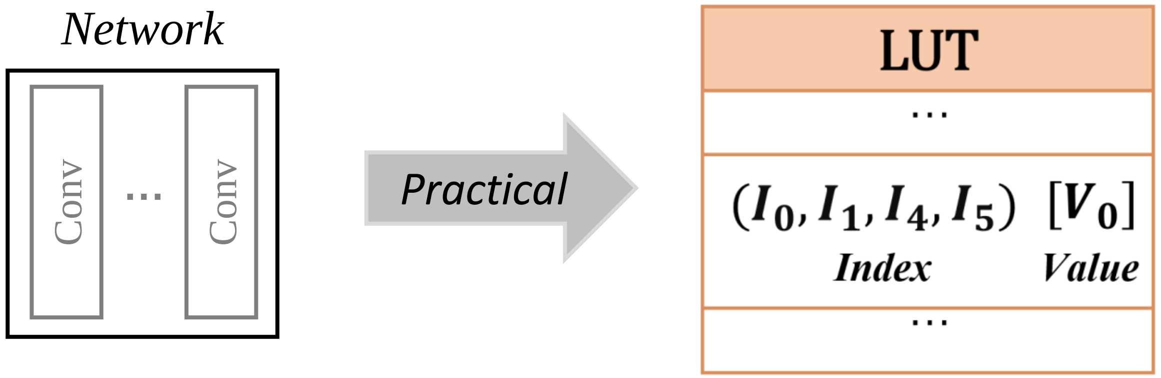
Deep neural networks (DNN)



***Resource Limitation:** Time complexity, Computational complexity, Energy cost*

Concept

The basic idea of the proposed scheme *is to adopt the* ***look-up operation (direct addressing)*** *of LUT to* ***replace the inference process of DNN in coding process****, which is also friendly for embedded systems to accelerate computation with far fewer floating-point operations.*

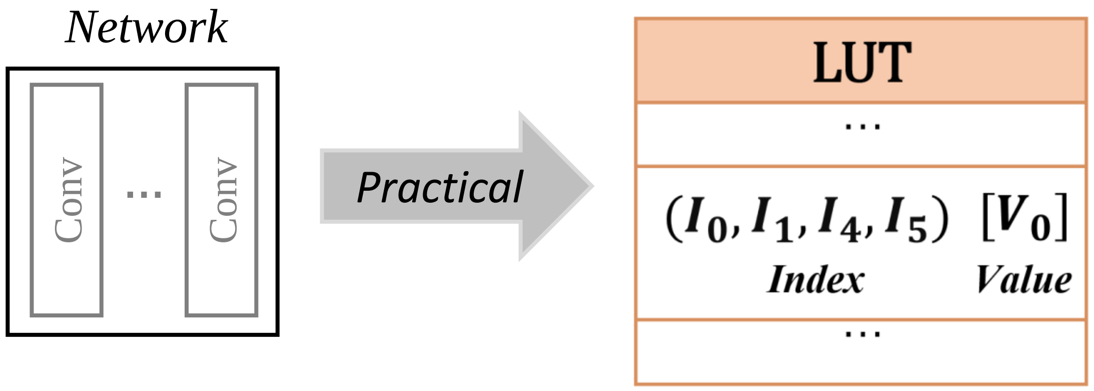


To achieve this goal, we establish a LUT-based in-loop filtering framework ***(termed LUT-ILF),*** *and introduce a series of LUT-related modules to strengthen its efficiency.*

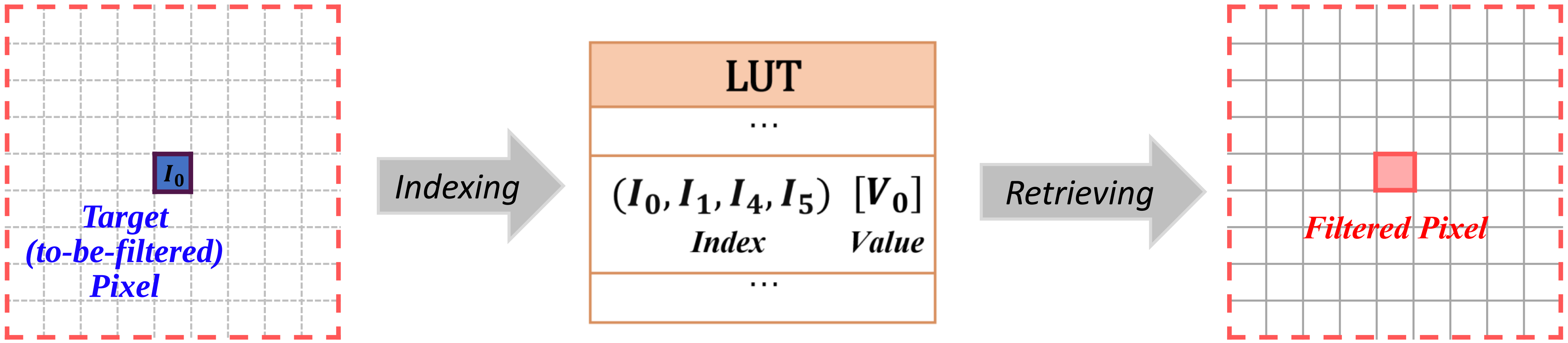
Our Basic Solution

We propose an efficient and practical in-loop filtering scheme by adopting the Look-up Table (LUT). We train the DNN of in-loop filtering within a fixed filtering reference range, and cache the output values of the DNN into a LUT via traversing all possible inputs.

Caching Network into LUT



Retrieving the Pixel in LUT



Our Basic Solution

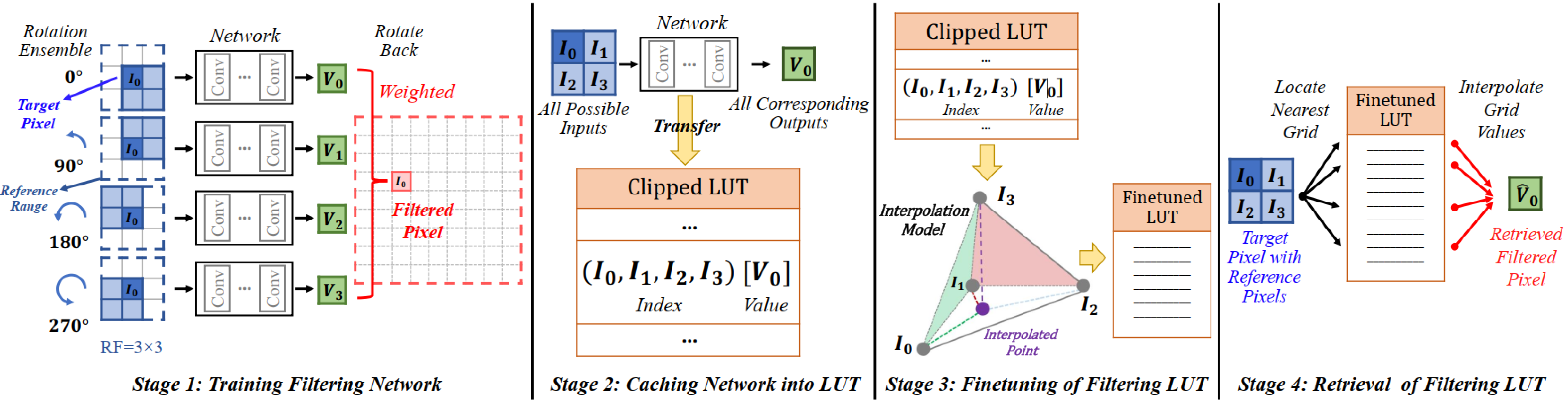


Illustration of the basic framework of look-up table-based in-loop filtering framework (LUT-ILF).

Stage 1: Training Filtering Network

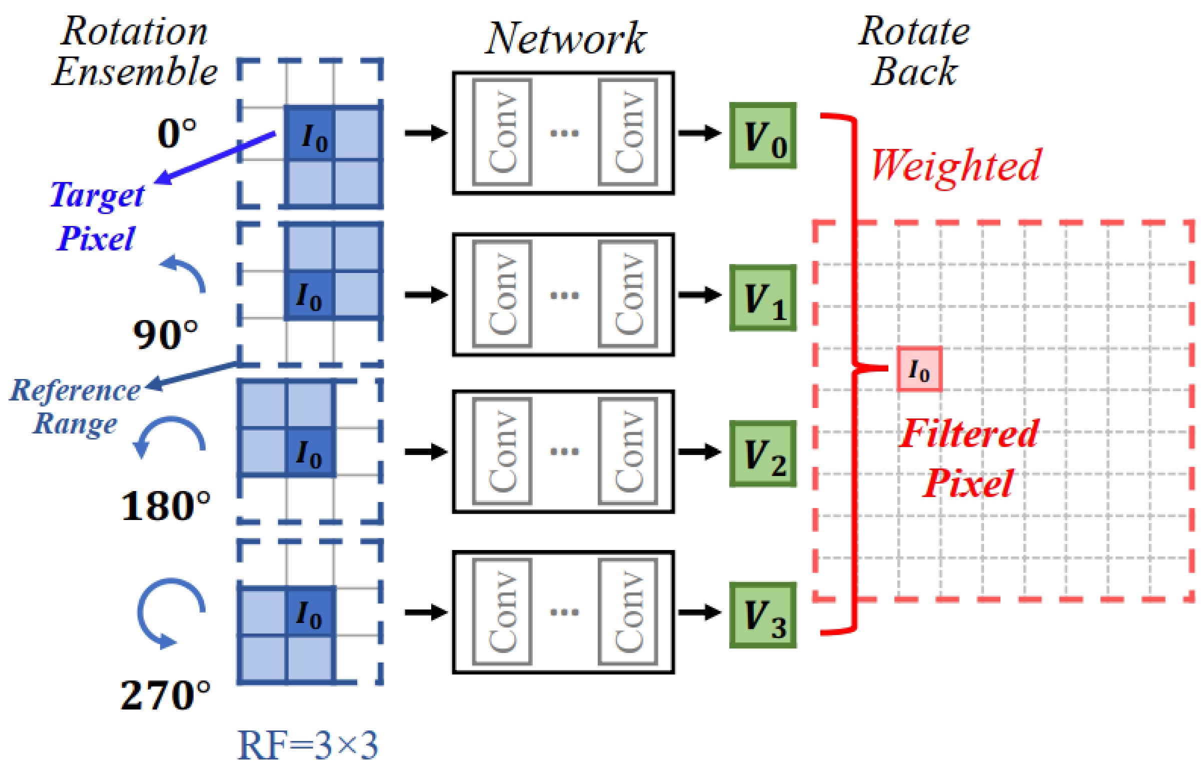
Stage 2: Caching Network into LUT

Stage 3: Finetuning of Filtering LUT

Stage 4: Retrieval of Filtering LUT

Our Basic Solution

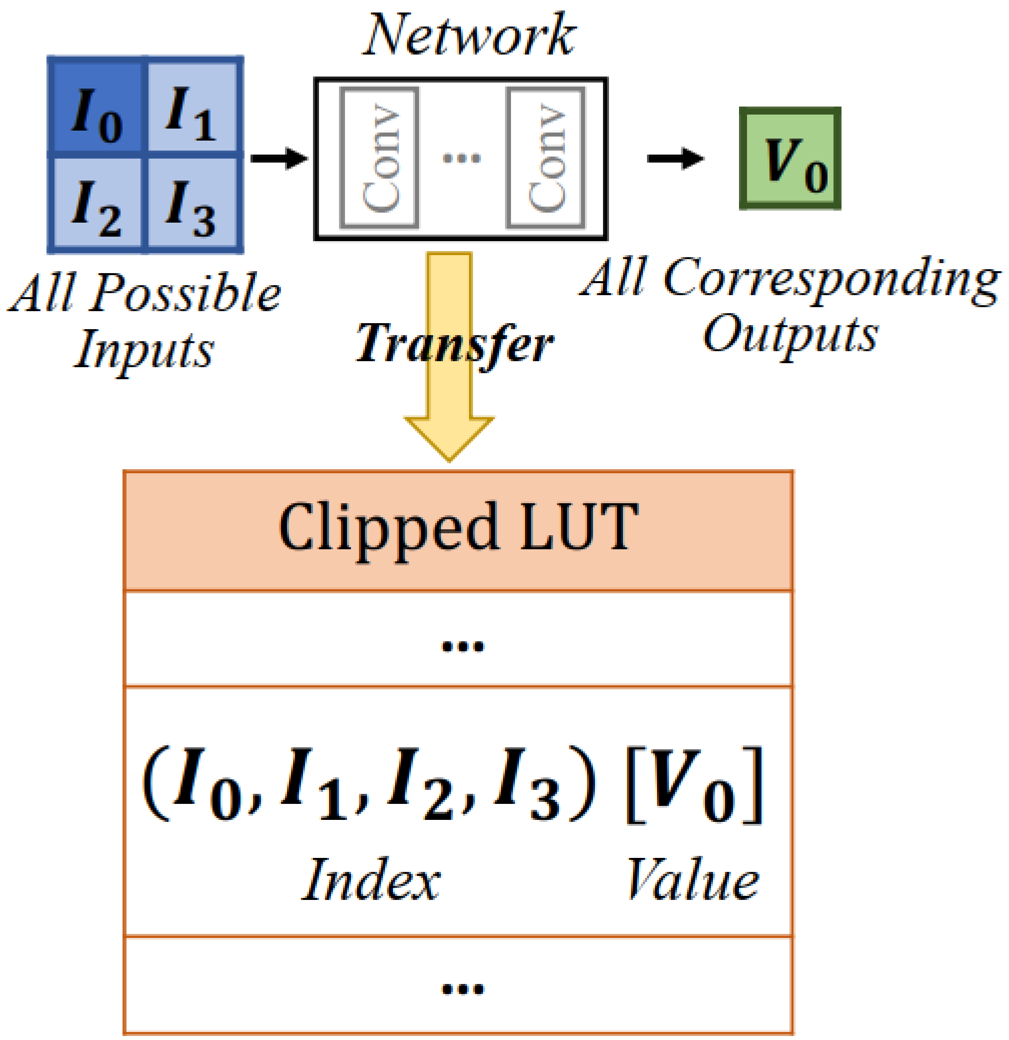
Stage 1: Training Filtering Network



Due to the size of LUT grows exponentially as the dimension of indexing entries (i.e., target pixel with reference pixels) increases, the lightweight filtering network is trained with the constraint of a small reference range (receptive field, RF) in an end-to-end manner.

Taking the 2×2 reference range (4D LUT) as an example, the target pixel (to-be-filtered / reconstructed pixel) with three surrounding reference pixels (solid line) serves as the input to the network. To enlarge the size of RF, the rotation ensemble trick is used to cover the 3×3 reference range (dotted line). The final output value is averaged by all outputs of the 4 rotations ($V_0 \sim V_3$).

Our Basic Solution

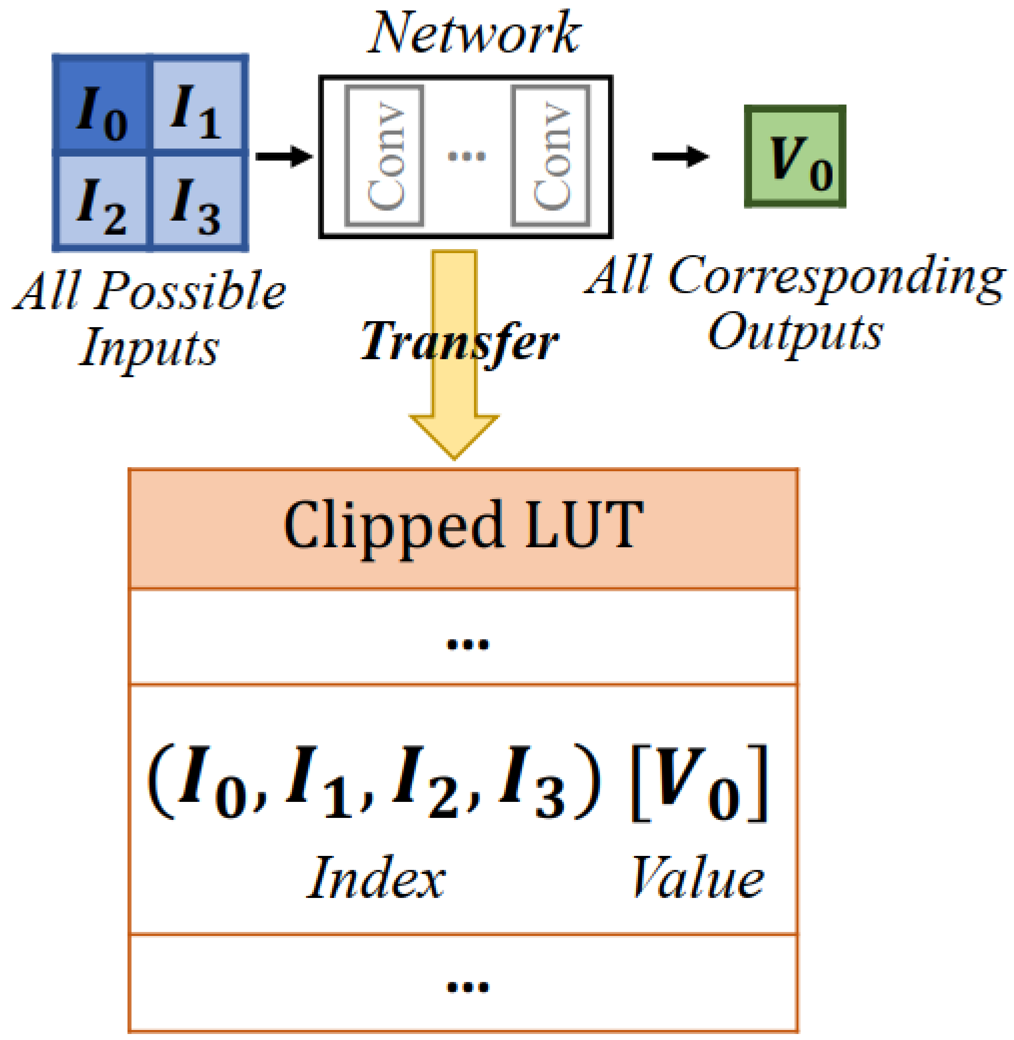


Stage 2 : Caching Network into LUT

With the network being trained, the 4D LUT is transferred and cached from the output values of the network via traversing all possible inputs (target pixel with reference pixels, [0~255] [0~255] [0~255] [0~255] for int8 case of input), as shown in Fig.

Note that the storage of LUT with a large input/output range will bring heavy storage cost, for example, the full size of 4D LUT is calculated as $256^4 \times 1 \times 8$ bit = 4096 MB (4 GB), 256^4 *bins for possible input value (0~255), 1 for 8-bit output value.*

Our Basic Solution

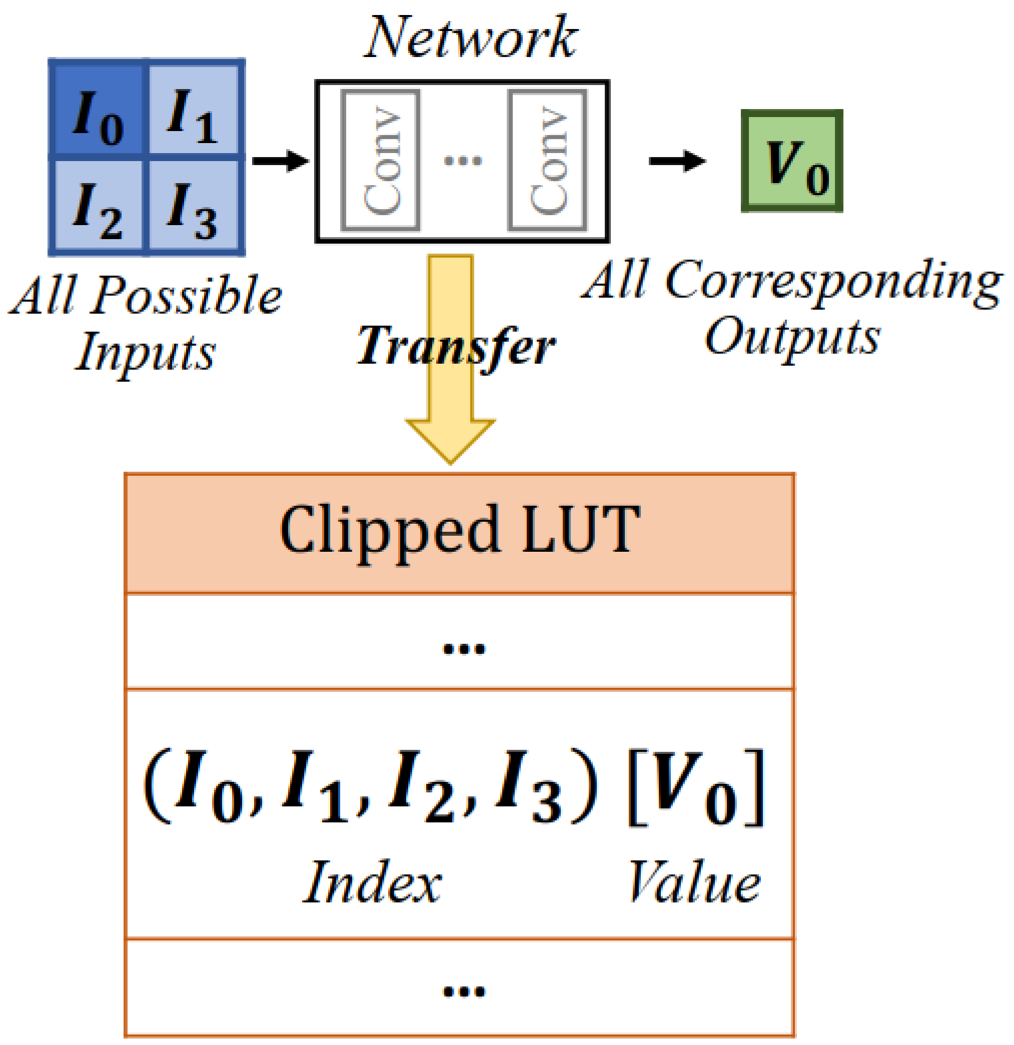


Stage 2 : Caching Network into LUT

To avoid the heavy storage cost, the indexes of full LUT are *uniformly sampled and stored in the small LUT (named Clipped LUT), which only caches the output value of the most significant bits (MSB) of the input pixel value.*

In our design, the 8-bit input pixel value is uniformly sampled to 4 MSBs, and the 4 MSBs serve as the initial (nearest) index for the indexing of input pixel. The input/output range of indexing is degraded to [0,16,…,240,255][0,16,…, 240,255] [0,16,…, 240,255] [0,16,…, 240, 255], and *the size of Clipped LUT is calculated as* $17^4 \times 1 \times 8$ bit = 81.56 KB.

Our Basic Solution

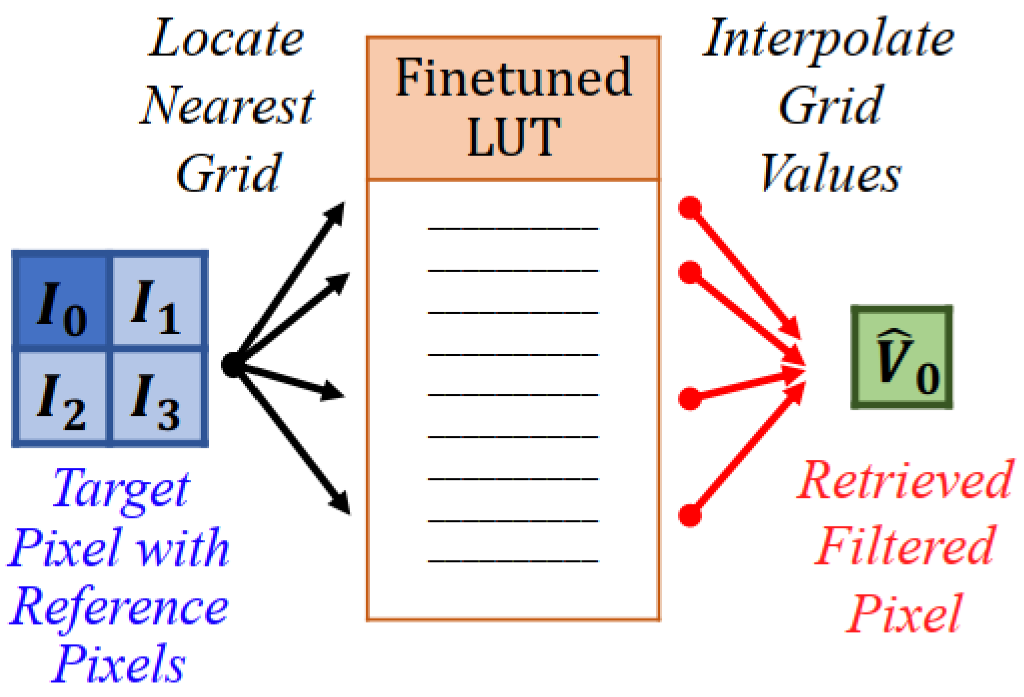


Stage 3 : Finetuning of Filtering LUT

To compensate for the degradation of LUT Clipping, the finetuning of Clipped LUT is performed to adapt to the uniform sampling and the interpolation model, facilitating the interpolation of the final *retrieved filtered pixel* value of non-sampled indexes of LUT from the nearest sampled indexes of LUT.

In finetuning, *the values of Clipped LUT are activated as the trainable parameters* and finetuned by the same setting of filtering network training.

Our Basic Solution



Stage 4 : Retrieval of Filtering LUT

In the retrieval process of finetuned filtering LUT, *with the indexing of the MSB of input pixels* (I_0, I_1, I_2, I_3) *in 4D Clipped LUT, the obtained output values of the nearest index and least significant bits (LSB) of the input pixels are used to interpolate the final retrieved filtered pixel by linear interpolation model.*

For the interpolation method of Clipped LUT, we follow the *4-Simplex interpolation model.*

Bottleneck of Basic LUT-ILF

1. First, the filtering reference range (RF, only 3×3) is limited with the constraint of LUT size, which is verified as an important factor in traditional filtering tools (such as ALF with 7×7 reference range).

2. Second, the selection of reference pixels is very relevant to the filtering (such as the ALF with a diamond shape).

To address these limitations, the reference, progressive, weighted indexing mechanism is introduced to enhance the above issues in our framework.

Module 1: Reference Indexing

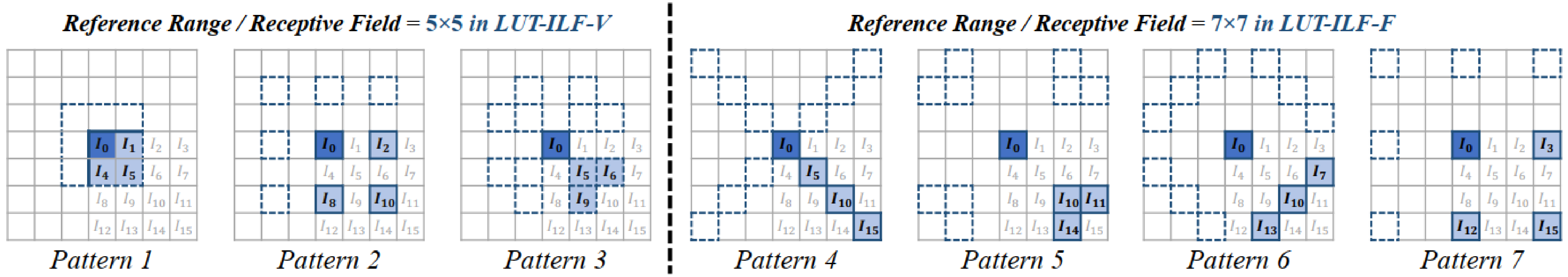
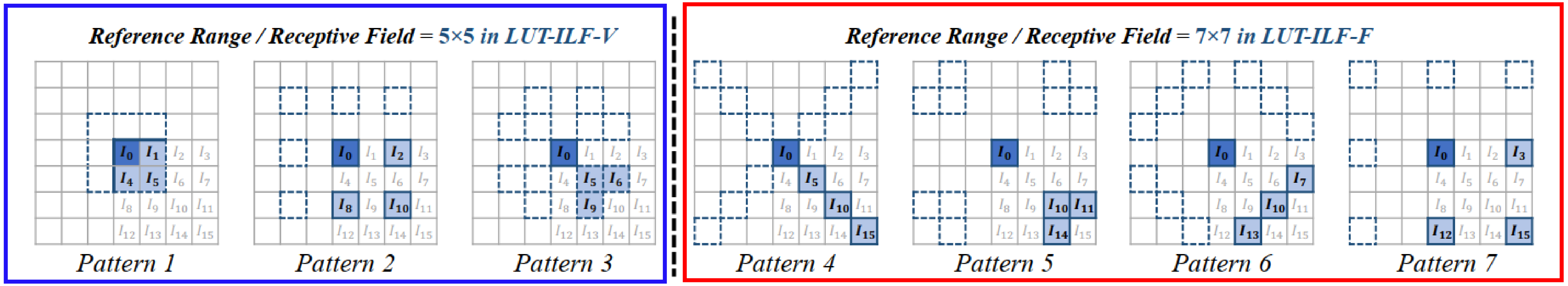

Reference Range / Receptive Field = 5×5 in LUT-ILF-V
Reference Range / Receptive Field = 7×7 in LUT-ILF-F
Pattern 1
Pattern 2
Pattern 3
Pattern 4
Pattern 5
Pattern 6
Pattern 7

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

To avoid the exponential growth in the size of LUT with the dimension, the complementary reference indexing is used to increase the reference range of target pixel by parallelizing more complementary indexing patterns to address more reference pixels and capture the rich local structures. As shown in Fig, it can cover a wide reference range.

Module 1: Reference Indexing



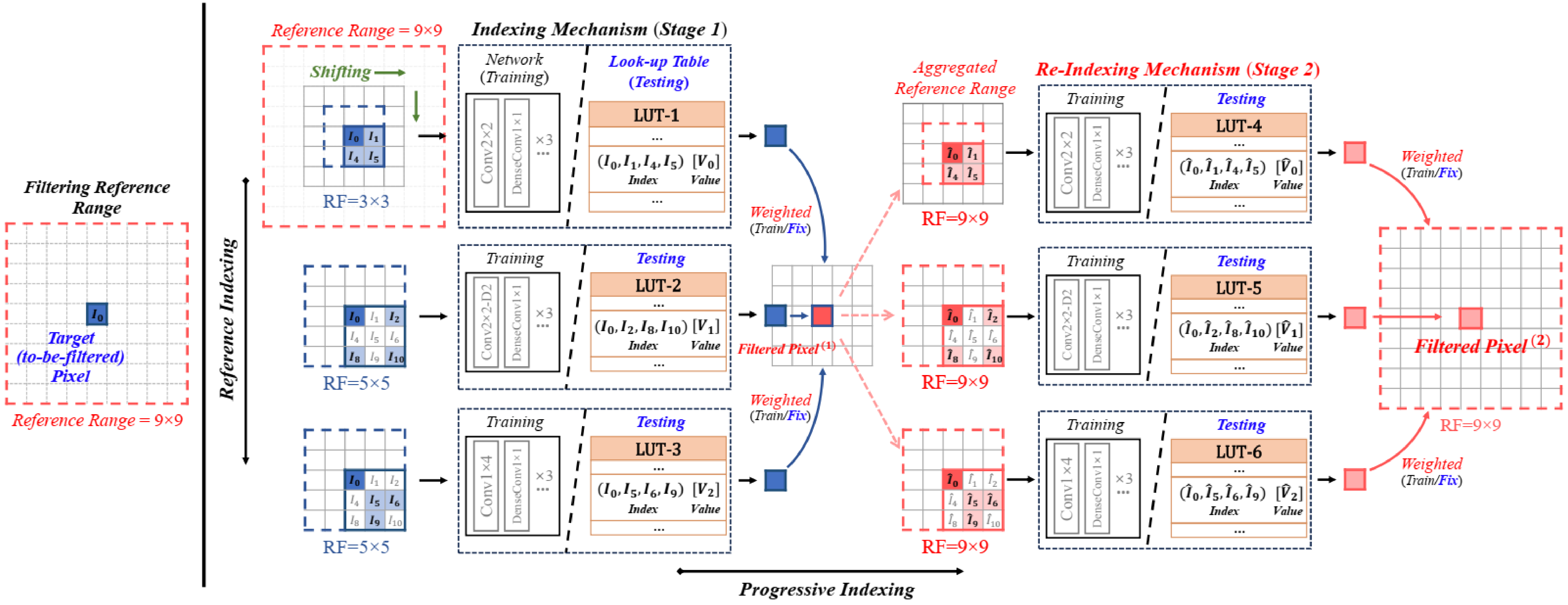
Reference Range / Receptive Field = 5×5 in LUT-ILF-V
Reference Range / Receptive Field = 7×7 in LUT-ILF-F
Pattern 1
Pattern 2
Pattern 3
Pattern 4
Pattern 5
Pattern 6
Pattern 7

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

In LUT-ILF-V, besides the standard indexing pattern of LUT-ILF (Pattern 1, RF=3×3), complementary Pattern 2 and Pattern 3 are used to cover the 5×5 reference range. For the patterns of LUT-ILF-F, it can cover the 7×7 reference range.

In this way, the total size of cached LUTs grows linearly (3 times a 4D Clipped LUT, $3\times17^4\times1\times8$ bit = 244.69 KB), instead of exponentially (the full size of a 25D LUT with an equivalent 5×5 reference range is 256^{25-4} times a 4D LUT), in a single stage of reference indexing mechanism.

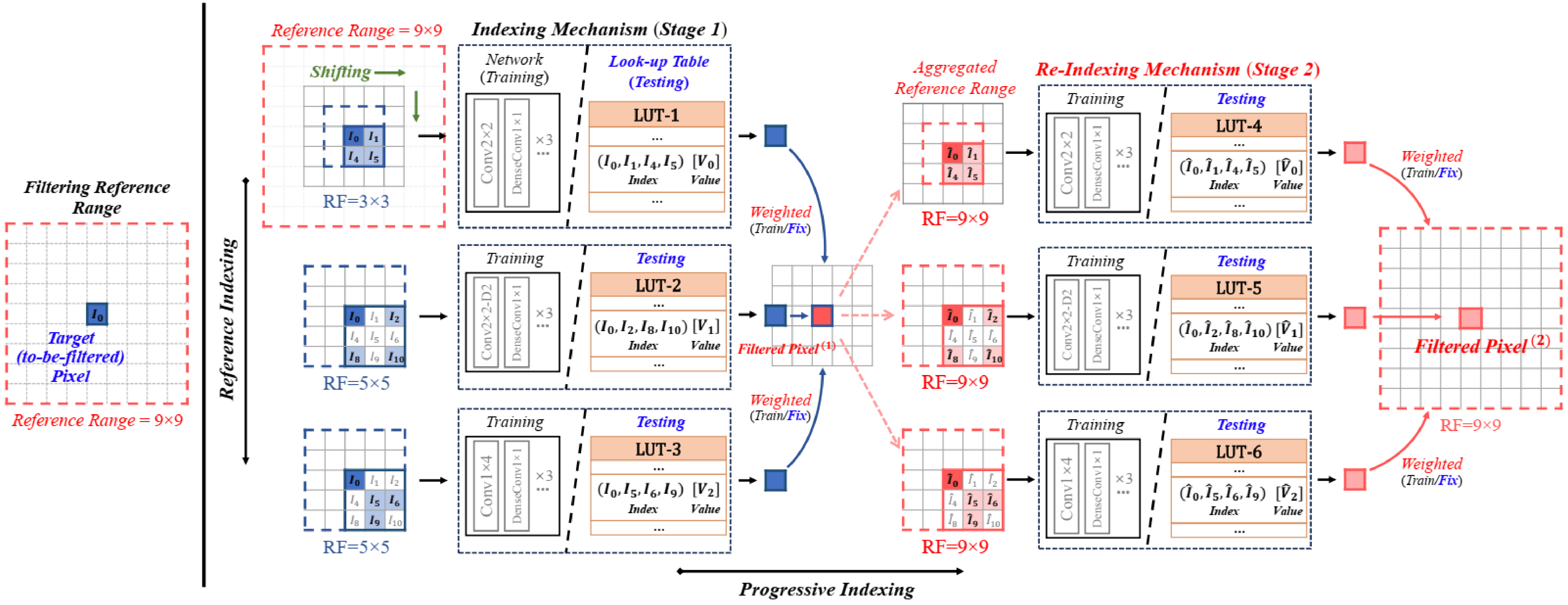
Module 2 & 3: Progressive Indexing and Learnable Weighting



Filtering Reference Range
I_0
Target (to-be-filtered) Pixel
Reference Range = 9×9
Reference Indexing
Reference Range = 9×9
Shifting
I_0 I_1 I_4 I_5
RF=3×3
RF=5×5
RF=5×5
Indexing Mechanism (Stage 1)
Network (Training)
Look-up Table (Testing)
Conv2×2
DenseConv1×1
×3
LUT-1
(I_0, I_1, I_4, I_5) $[V_0]$
Index Value
Training
Testing
Conv2×2-D2
LUT-2
(I_0, I_2, I_8, I_{10}) $[V_1]$
Conv1×4
LUT-3
(I_0, I_5, I_6, I_9) $[V_2]$
Weighted (Train/Fix)
Filtered Pixel$^{(1)}$
Aggregated Reference Range
RF=9×9
Re-Indexing Mechanism (Stage 2)
LUT-4
$(\hat{I}_0, \hat{I}_1, \hat{I}_4, \hat{I}_5)$ $[\hat{V}_0]$
LUT-5
$(\hat{I}_0, \hat{I}_2, \hat{I}_8, \hat{I}_{10})$ $[\hat{V}_1]$
LUT-6
$(\hat{I}_0, \hat{I}_5, \hat{I}_6, \hat{I}_9)$ $[\hat{V}_2]$
Filtered Pixel$^{(2)}$
RF=9×9
Progressive Indexing

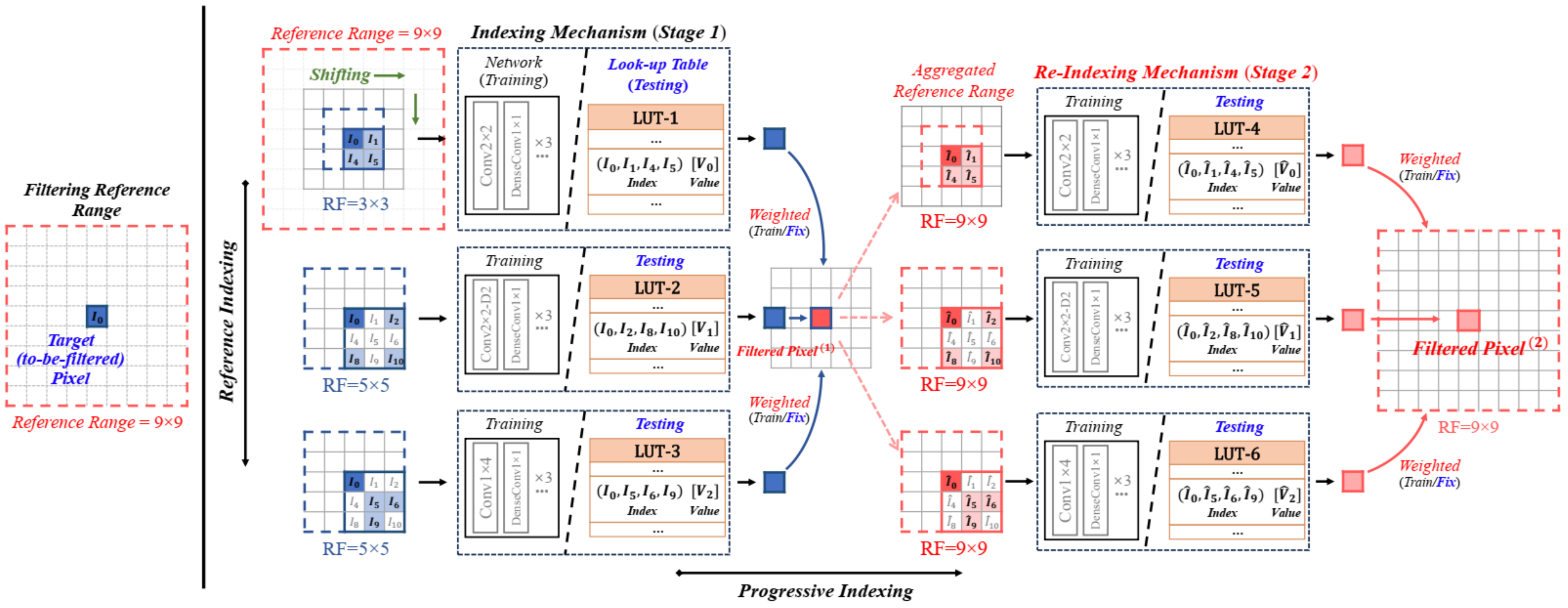
As shown in Fig, the re-indexing mechanism is used to link the cascaded framework between multiple 4D LUTs. With the filtering of target pixel by multiple indexing patterns (5×5 reference range) in stage 1 of progressive indexing, the filtered pixel of stage 1 contains the local information of 5×5 reference range implicitly.

Module 2 & 3: Progressive Indexing and Learnable Weighting


Filtering Reference Range
Target (to-be-filtered) Pixel
Reference Range = 9×9
Reference Indexing
Reference Range = 9×9
Shifting
RF=3×3
RF=5×5
RF=5×5
Indexing Mechanism (Stage 1)
Network (Training)
Look-up Table (Testing)
LUT-1
(I_0, I_1, I_4, I_5) $[V_0]$
Index Value
Training
Testing
LUT-2
(I_0, I_2, I_8, I_{10}) $[V_1]$
LUT-3
(I_0, I_5, I_6, I_9) $[V_2]$
Conv2×2
Conv2×2-D2
Conv1×4
DenseConv1×1
×3
Weighted (Train/Fix)
Filtered Pixel$^{(1)}$
Aggregated Reference Range
RF=9×9
Re-Indexing Mechanism (Stage 2)
LUT-4
$(\hat{I}_0, \hat{I}_1, \hat{I}_4, \hat{I}_5)$ $[\hat{V}_0]$
LUT-5
$(\hat{I}_0, \hat{I}_2, \hat{I}_8, \hat{I}_{10})$ $[\hat{V}_1]$
LUT-6
$(\hat{I}_0, \hat{I}_5, \hat{I}_6, \hat{I}_9)$ $[\hat{V}_2]$
Filtered Pixel$^{(2)}$
Progressive Indexing

By shifting the filtering window in the 9×9 reference range, the local information of 9×9 reference range can be aggregated into a 5×5 aggregated reference range. In stage 2 of progressive indexing, the reindexing mechanism can be used to filter the target pixel in the aggregated reference pixels to achieve the larger reference range implicitly.

Module 2 & 3: Progressive Indexing and Learnable Weighting



Above these ways, with the utilization of reference and progressive indexing, the total size of cached LUTs is linear to its indexing capacity (6 times a Clipped 4D LUT, $6 \times 17^4 \times 1 \times 8$ bit = 489.38 KB), instead of exponentially (the full size of an 81D LUT with an equivalent 9×9 reference range is 256^{81-4} times a 4D LUT), in the whole process of very fast setting of LUT-ILF (LUT-ILF-V). For the fast setting (LUT-ILF-F), the total size of cached LUTs is 2.3× LUT-ILF-V's size, instead of exponentially (the full size of a 169D LUT with an equivalent 13×13 reference range is 256^{169-4} times a 4D LUT).

Performance

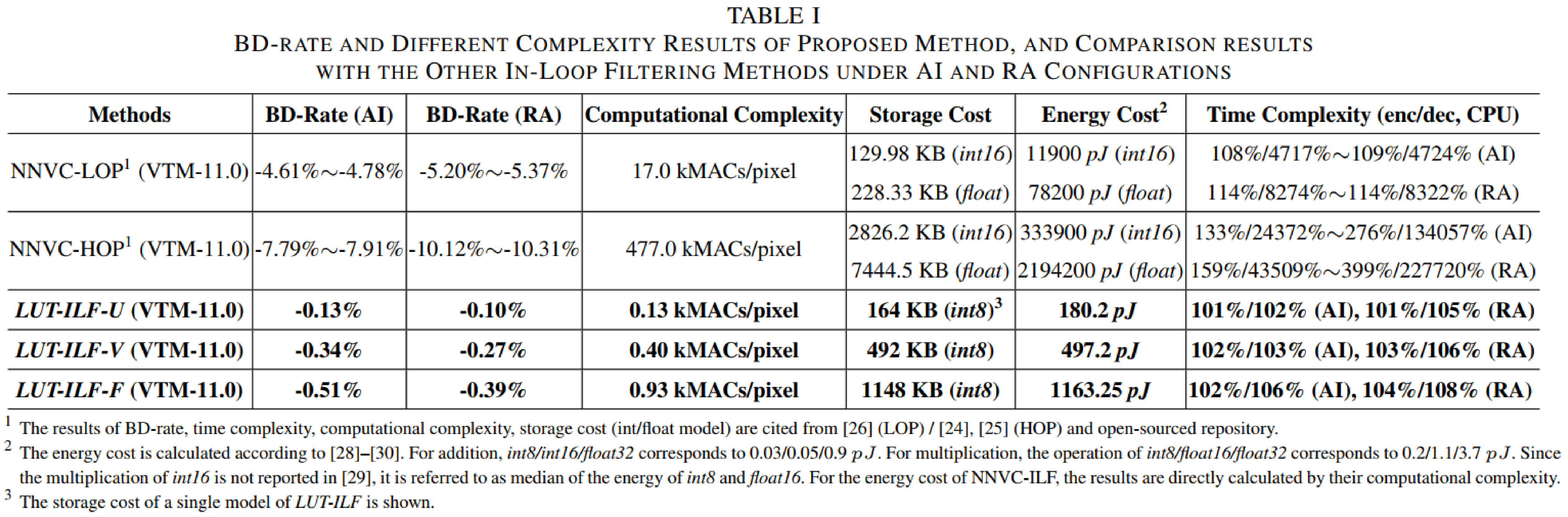


TABLE I
BD-RATE AND DIFFERENT COMPLEXITY RESULTS OF PROPOSED METHOD, AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING METHODS UNDER AI AND RA CONFIGURATIONS

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

[1] The results of BD-rate, time complexity, computational complexity, storage cost (int/float model) are cited from [26] (LOP) / [24], [25] (HOP) and open-sourced repository.
[2] The energy cost is calculated according to [28]–[30]. For addition, *int8/int16/float32* corresponds to 0.03/0.05/0.9 *pJ*. For multiplication, the operation of *int8/float16/float32* corresponds to 0.2/1.1/3.7 *pJ*. Since the multiplication of *int16* is not reported in [29], it is referred to as median of the energy of *int8* and *float16*. For the energy cost of NNVC-ILF, the results are directly calculated by their computational complexity.
[3] The storage cost of a single model of *LUT-ILF* is shown.

In our experiment, the VVC reference software VTM-11.0 is used as the baseline. The codec adopts the configuration of all intra (AI) and random access (RA) according to the VVC Common Test Condition (CTC). For each test sequence, quantization parameter (QP) values are set to 22, 27, 32, 37, 42, and Bjontegaard Delta-rate (BD-rate) is used as an objective metric to evaluate coding performance. For the complexity metrics, time complexity, computational complexity (kMAC/pixel), theoretical energy cost (pJ), and storage cost (KB) are evaluated.

Performance

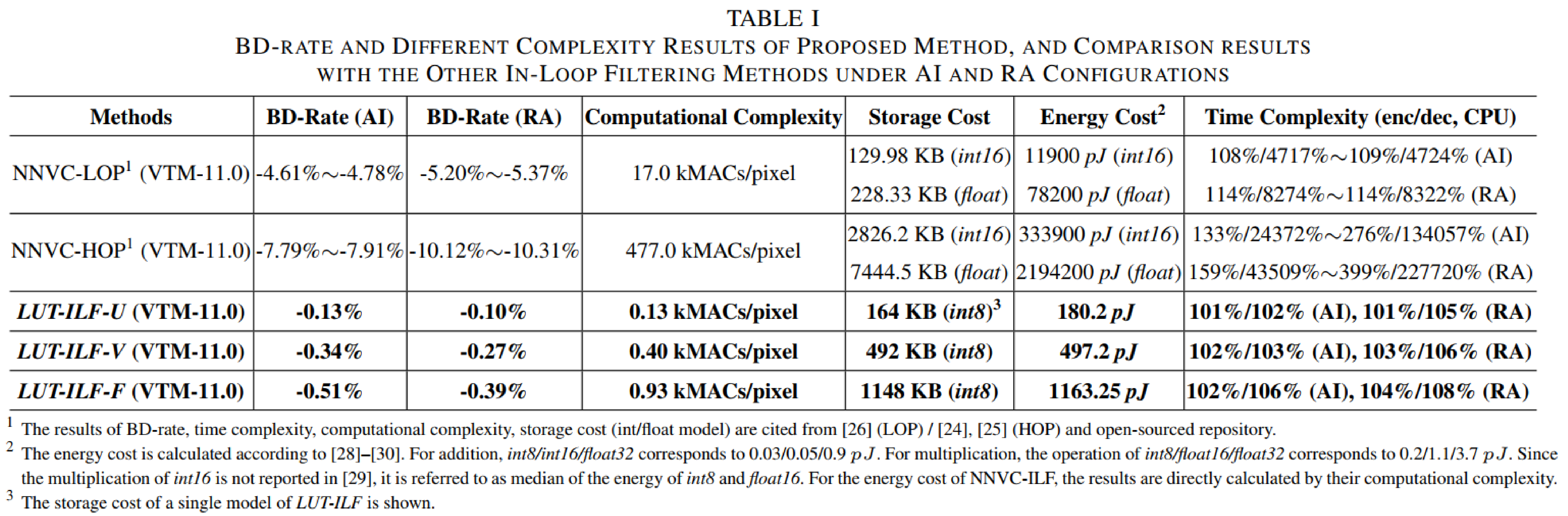


TABLE I
BD-RATE AND DIFFERENT COMPLEXITY RESULTS OF PROPOSED METHOD, AND COMPARISON RESULTS WITH THE OTHER IN-LOOP FILTERING METHODS UNDER AI AND RA CONFIGURATIONS

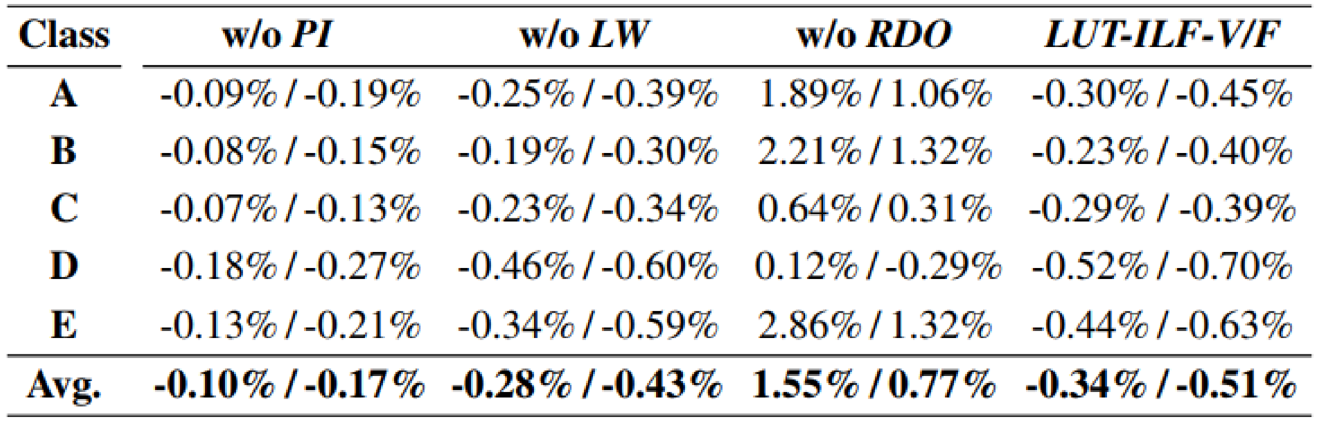
Methods	BD-Rate (AI)	BD-Rate (RA)	Computational Complexity	Storage Cost	Energy Cost[2]	Time Complexity (enc/dec, CPU)
NNVC-LOP[1] (VTM-11.0)	-4.61%∼-4.78%	-5.20%∼-5.37%	17.0 kMACs/pixel	129.98 KB (*int16*)	11900 *pJ* (*int16*)	108%/4717%∼109%/4724% (AI)
				228.33 KB (*float*)	78200 *pJ* (*float*)	114%/8274%∼114%/8322%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

[1] The results of BD-rate, time complexity, computational complexity, storage cost (int/float model) are cited from [26] (LOP) / [24], [25] (HOP) and open-sourced repository.
[2] The energy cost is calculated according to [28]–[30]. For addition, *int8/int16/float32* corresponds to 0.03/0.05/0.9 pJ. For multiplication, the operation of *int8/float16/float32* corresponds to 0.2/1.1/3.7 pJ. Since the multiplication of *int16* is not reported in [29], it is referred to as median of the energy of *int8* and *float16*. For the energy cost of NNVC-ILF, the results are directly calculated by their computational complexity.
[3] The storage cost of a single model of *LUT-ILF* is shown.

From Table, we can find that the different modes (ultrafast, very fast, fast) of *our proposed LUT-ILF provide a series of new trade-off points between the performance and efficiency for practical applications*. For the quantitative comparisons of performance and complexity, the computational complexity and decoding time complexity of LUT-ILF are *130×~3600× and 46×~2200× lower than that of popular NN-based ILF methods*, and LUT-ILF also shows good performance potential.

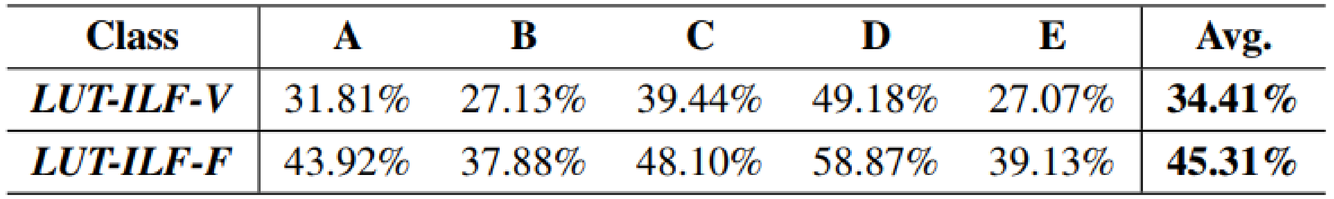
Performance

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



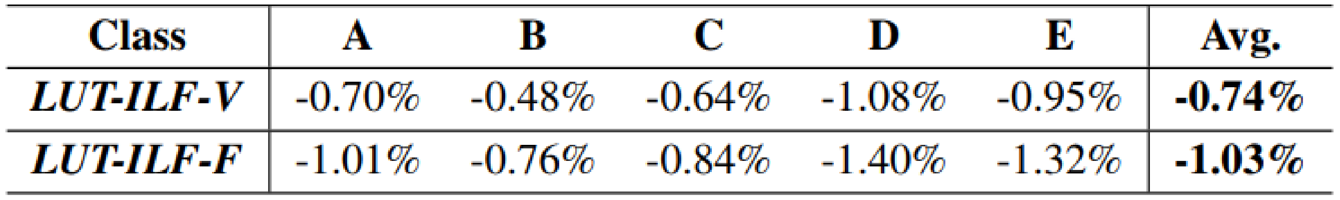
Class	w/o *PI*	w/o *LW*	w/o *RDO*	*LUT-ILF-V/F*
A	-0.09% / -0.19%	-0.25% / -0.39%	1.89% / 1.06%	-0.30% / -0.45%
B	-0.08% / -0.15%	-0.19% / -0.30%	2.21% / 1.32%	-0.23% / -0.40%
C	-0.07% / -0.13%	-0.23% / -0.34%	0.64% / 0.31%	-0.29% / -0.39%
D	-0.18% / -0.27%	-0.46% / -0.60%	0.12% / -0.29%	-0.52% / -0.70%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

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



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

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



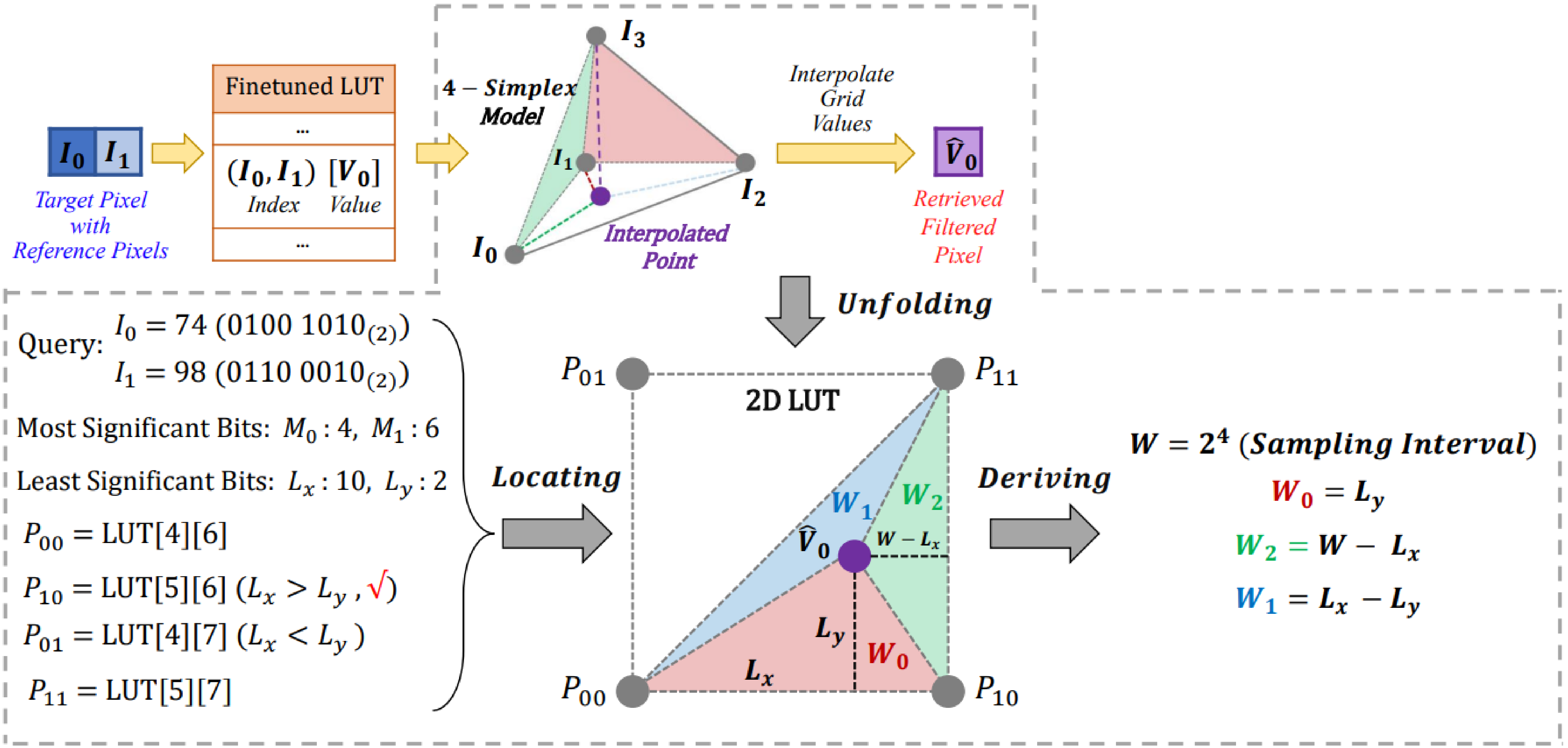
Class	A	B	C	D	E	Avg.
LUT-ILF-V	-0.70%	-0.48%	-0.64%	-1.08%	-0.95%	**-0.74%**
LUT-ILF-F	-1.01%	-0.76%	-0.84%	-1.40%	-1.32%	**-1.03%**

Ablation Study: To validate the contributions of core modules in our scheme, we conduct the ablation experiments on proposed *progressive indexing* (*PI*), *learnable weighting* (*LW*), and the CTU-level RDO (*RDO*), under AI configuration.

Usage Ratio: To verify the efficiency of *LUT-ILF*, we evaluate its usage ratio, which is calculated by, *Ratio* = N_{test}/N_{total}.

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

Supplementary


I_0 I_1
Target Pixel with Reference Pixels
Finetuned LUT
...
(I_0, I_1) $[V_0]$
Index Value
...
4 – Simplex Model
I_3
I_1
I_2
I_0
Interpolated Point
Interpolate Grid Values
$\hat{V}_0$
Retrieved Filtered Pixel
Unfolding
Query: $I_0 = 74\ (0100\ 1010_{(2)})$
$I_1 = 98\ (0110\ 0010_{(2)})$
Most Significant Bits: $M_0 : 4,\ M_1 : 6$
Least Significant Bits: $L_x : 10,\ L_y : 2$
$P_{00} = \mathrm{LUT}[4][6]$
$P_{10} = \mathrm{LUT}[5][6]\ (L_x > L_y, \checkmark)$
$P_{01} = \mathrm{LUT}[4][7]\ (L_x < L_y)$
$P_{11} = \mathrm{LUT}[5][7]$
Locating
2D LUT
P_{01} P_{11} P_{00} P_{10}
W_1 W_2 $\hat{V}_0$ $W - L_x$ L_y W_0 L_x
Deriving
$W = 2^4$ (Sampling Interval)
$W_0 = L_y$
$W_2 = W - L_x$
$W_1 = L_x - L_y$

Interpolation Process of Filtering LUT

TABLE IV
THE COMPUTATIONAL COMPLEXITY RESULTS AND SPECIFIC OPERATION NUM OF OUR PROPOSED BASIC FRAMEWORK (*LUT-ILF-U/V*) ON THE PIXEL (PER PIXEL) AND FRAME LEVEL (A 1920 × 1080 HD FRAME).

Detailed Calculation of Computational Complexity / Energy Cost of LUT-ILF-U/V/F			
Operation	**Level**	***Operation Num of LUT-ILF-U***	***Operation Num of LUT-ILF-V***
int8 Add	Pixel-wise	70	206
int8 Multiply		4	4
int32 Add		68	190
int32 Multiply		55	152
Total Add		138	396
Total Multiply		59	156
int8 Add	Frame-wise	145,152,000	427,161,600
int8 Multiply		8,294,400	8,294,400
int32 Add		141,004,800	393,984,000
int32 Multiply		114,048,000	315,187,200
Total Add	**Frame-wise**	**286,156,800**	**821,145,600**
Total Multiply		**122,342,400**	**323,481,600**
Worse-case Computational Complexity[1] (kMACs/pixel)	**Pixel-wise**	**0.13**	**0.40**
Energy Cost[2] (pJ)	**Frame-wise**	**180.2**	**497.2**

[1] The "worse-case" means that if the number of additions is more than the number of multiplications, the calculation of kMAC tends to be additions.
[2] The energy cost is calculated according to [7]–[9]. For addition, *int8/int16/float32* corresponds to 0.03/0.05/0.9 pJ. For multiplication, the operation of *int8/float16/float32* corresponds to 0.2/1.1/3.7 pJ.

Thanks for your listening!

Intelligent Visual Data Coding Laboratory (iVC)

University of Science and Technology of China (USTC)

December 28, 2024 @ VCIP 2024

Homepage: https://zhuoyuanli1997.github.io/