Block-based Learned Image Compression without Blocking Artifacts

Supplementary Material

This supplementary material provides detailed mathematical derivations, extended experimental results, and illustrative examples to support the claims made in the main paper. Section 1 revisits the arithmetic of standard and transposed convolutions to establish the matrix-based foundation of our method. Section 2 details the specific assumptions and alignment rules used to ensure consistent downsampling and upsampling. Section 3 derives the recursive formulas for calculating the minimum required overlap. Section 4 presents the specific overlap calculation results for various LIC models. Finally, Section 5 and Section 6 provide additional experimental data on resource efficiency and ablation studies, confirming the effectiveness of the proposed method.

1. Standard Convolution Arithmetic

In this section, we revisit the matrix formulation of standard and transposed convolutions. The goal is to explicitly demonstrate how transposed convolution acts as the inverse operation of standard convolution in terms of spatial transformation, and to justify the "zero-insertion + stride-1 convolution" interpretation used in our derivations.

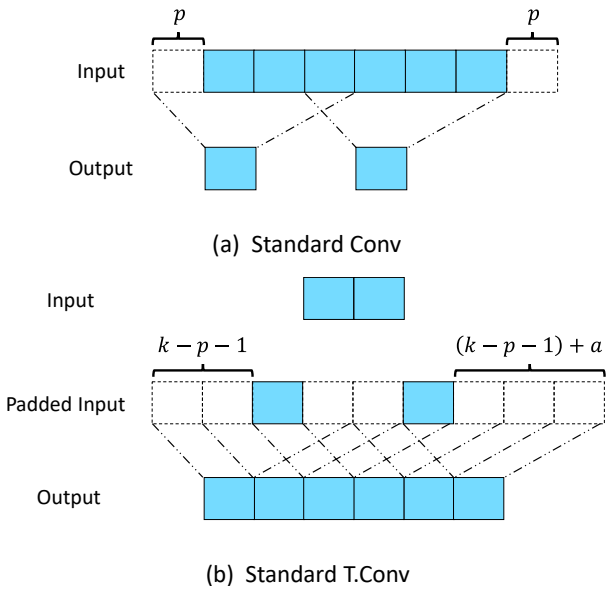

Figure 8. Toy 1D example of the standard convolution and its associated transposed convolution used in this section. (a) Standard convolution with $i = 6, k = 4, s = 3, p = 1$. (b) The transposed convolution that shares the same (k, s, p).

In the main paper, Sec. 2.3 summarizes the arithmetic of standard convolutions and transposed convolutions using 1D signals. For an input of size i, padding p, kernel size k, and stride s, the output size o of a standard convolution is given by Eq. (1). For the toy example in Fig. 8(a), where $(i, k, s, p) = (6, 4, 3, 1)$, substituting these values into Eq. (1) yields $o = 2$. Because $s > 1$, the last input element is never covered by the kernel, so it does not contribute to any output position even though it lies inside the padded support.

As stated in Eq. (2), a standard convolution can be written as a matrix–vector product $\mathbf{y} = \mathbf{W}\mathbf{x}$, where $\mathbf{x}$ is the (unpadded) input, $\mathbf{W}$ is the weight matrix induced by the kernel, and $\mathbf{y}$ is the output. For the example in Fig. 8(a), we set

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]^T, \quad \mathbf{y} = [y_1, y_2]^T$$

and the kernel weight vector as $\mathbf{k} = [w_0, w_1, w_2, w_3]^T$. The corresponding weight matrix $\mathbf{W}$ is

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & w_0 & w_1 & w_2 & w_3 \end{bmatrix}}_{\mathbf{W}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \tag{20}$$

Each row of $\mathbf{W}$ implements one valid kernel placement over the input, shifting by $s = 3$ positions between consecutive rows while simply omitting entries corresponding to padded zeros. For example, in the first row the coefficient w_0 never appears, because at the first valid position it is always multiplied by the left padding.

The main paper defines the associated transposed convolution through the transpose of this weight matrix, as in Eq. (3):

$$\mathbf{y}' = \mathbf{W}^T \mathbf{x}',$$

where $\mathbf{x}'$ and $\mathbf{y}'$ denote the input and output of the transposed convolution, respectively. Using $\mathbf{W}$ from Eq. (20) and

$$\mathbf{x}' = [x'_1, x'_2]^T, \quad \mathbf{y}' = [y'_1, y'_2, y'_3, y'_4, y'_5, y'_6]^T,$$

we obtain the explicit matrix form

$$\begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ y'_5 \\ y'_6 \end{bmatrix} = \underbrace{\begin{bmatrix} w_1 & 0 \\ w_2 & 0 \\ w_3 & w_0 \\ 0 & w_1 \\ 0 & w_2 \\ 0 & w_3 \end{bmatrix}}_{\mathbf{W}^T} \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} \tag{21}$$

This operation upsamples the length-2 input to a length-6 output while reversing the spatial transformation in Fig. 8(a), as discussed in the main text.

To connect this formulation to the "zero-insertion + stride-1 convolution" view used in Eqs. (3) and (4), we separate the transposed convolution into two steps: (i) inserting zeros between input samples, and (ii) applying a standard convolution with stride 1 and a suitably padded input. For the same example, we can rewrite Eq. (21) as

$$\begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ y'_5 \\ y'_6 \end{bmatrix} = \underbrace{\begin{bmatrix} w_1 & w_0 & 0 & 0 \\ w_2 & w_1 & w_0 & 0 \\ w_3 & w_2 & w_1 & w_0 \\ 0 & w_3 & w_2 & w_1 \\ 0 & 0 & w_3 & w_2 \\ 0 & 0 & 0 & w_3 \end{bmatrix}}_{\text{stride-1 convolution with flipped kernel}} \underbrace{\begin{bmatrix} x'_1 \\ 0 \\ 0 \\ x'_2 \end{bmatrix}}_{\text{zero-inserted input}} \tag{22}$$

Here, the right-most vector corresponds to inserting $(s-1) = 2$ zeros between the original inputs. The middle matrix is arranged so that, as required for a standard convolution with stride 1, each row is obtained by shifting the kernel coefficients by exactly one position relative to the previous row. Under this constraint, matching the mapping in Eq. (22) forces the kernel coefficients to appear in reversed order, i.e., as the flipped kernel $[w_3, w_2, w_1, w_0]$. Thus, Eq. (22) explicitly shows how a transposed convolution can be implemented as a standard convolution with stride 1 on an upsampled input. Looking at the first row of the convolution matrix in Eq. (22), we observe that the coefficient w_0, which was omitted in the forward convolution matrix in Eq. (20), now appears at the second column. In general, the number of kernel coefficients that are omitted in the first row of the matrix W^T corresponding to the transposed convolution is $k-p-1$. This quantity is exactly the symmetric padding p' introduced in Eq. (4):

$$p' = (k-p-1).$$

Intuitively, to allow the kernel to reach the spatial boundaries of the original input and restore its full size, the transposed convolution applies additional padding around the zero-inserted input.

When $s > 1$, the stride can also cause a misalignment at the right boundary: some padded input samples in the forward convolution are never covered by the kernel, even after the last valid kernel placement. The number of such unused samples is given by the remainder term in Eq. (4),

$$a = (i + 2p - k) \bmod s.$$

From the perspective of the transposed convolution, this mismatch is compensated by adding an extra asymmetric padding of size a to the right of the zero-inserted input, as illustrated in Fig. 8(b). With this additional padding, the stride-1 convolution in Eq. (22) produces an output whose length exactly matches the input size i of the associated standard convolution.

In summary, Fig. 8 and Eqs. (20) to (22) provide a concrete matrix-level example of the general procedure described in the main paper: the transposed convolution in Eq. (3) can be implemented by (i) inserting $(s-1)$ zeros between input samples, (ii) symmetrically padding the result by $p' = (k-p-1)$ on both sides and adding an extra right padding of size $a = (i + 2p - k) \bmod s$ as in Eq. (4), and (iii) applying a standard convolution with stride 1 using the flipped kernel.

2. Alignment Rules and Padding Assumptions

This section provides a detailed explanation of Assumptions 1–2 and Eq. (5)–Eq. (12) in the main paper. We use 1-D examples to illustrate how input alignment and padding are strictly defined to ensure that the downsampling and upsampling operations are perfectly matched in spatial dimensions.

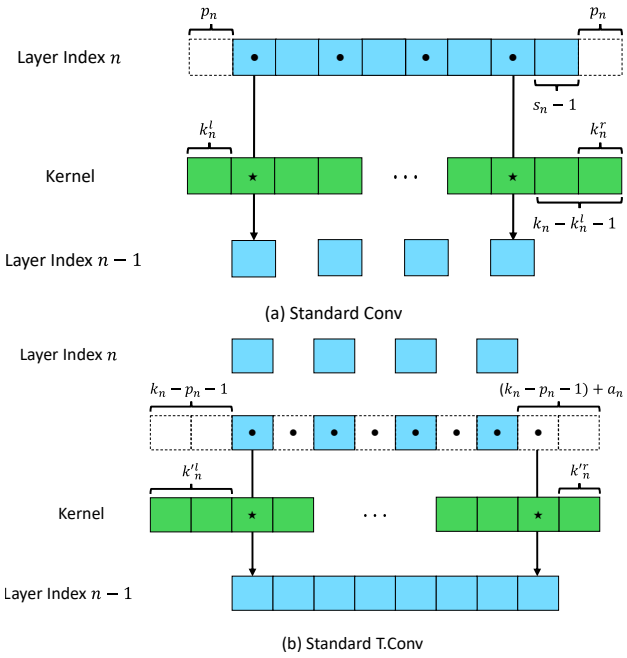

Figure 9. (a) A diagram showing the calculation process of a standard convolution with $X_n = 8$, $k_n = 4$, $s_n = 2$, and $p_n = 1$. (b) A diagram showing the calculation process of the corresponding standard transposed convolution ($a_n = 0$). In both (a) and (b), the dotted parts in X_n indicate where the actual computation occurs, and the star in the kernel indicates its center.

Assumption 1 and Eq. (5) state that the input size X_n (and similarly B_n) is an integer multiple of the stride s_n, and that the convolution downsamples the signal by a factor of exactly s_n, i.e., $X_{n-1} = X_n/s_n$ and $B_{n-1} = B_n/s_n$. Under this assumption, the convolution in Fig. 9(a) with

$X_n = 8$, $k_n = 4$, $s_n = 2$, and $p_n = 1$ is a representative example.

Equation (7) defines the center index c_n of a kernel of size k_n when the kernel indices are numbered from left to right as $[0, \ldots, k_n - 1]$. For the example in Fig. 9(a), we have $k_n = 4$, and Eq. (7) yields $c_n = 1$. The star in the kernel in Fig. 9 visualizes this center index.

Assumption 2 specifies the alignment rule for all convolutions in this work: the first convolution operation starts with the kernel center aligned to the first element of the unpadded input, and subsequent operations shift the kernel by s_n elements at a time from left to right. In Fig. 9(a), this means that the star (kernel center) is aligned with the leftmost non-padded input value at the first operation.

Next, consider the number of kernel elements that extend beyond the left boundary of the input at this first operation. Let k_n^l denote this quantity. Because Assumption 2 forces the first input element to coincide with the kernel center, the number of kernel taps that overhang the left boundary is exactly the number of taps to the left of the center. Therefore, k_n^l is equal to the center index c_n, as expressed in Eq. (8). In the example of Fig. 9(a), this gives $k_n^l = c_n = 1$.

Similarly, let k_n^r denote the number of kernel elements that extend beyond the right boundary of the input at the last convolution operation. The total number of elements to the right of the kernel center is $(k_n - 1 - k_n^l)$. However, due to the stride-s_n sampling and the integer-multiple condition in Assumption 1, the $(s_n - 1)$ elements immediately after the center remain within the input region at the last operation. Consequently, at most $(k_n - 1 - k_n^l) - (s_n - 1)$ elements can overhang the right boundary. When the stride is relatively large so that $(s_n - 1) > (k_n - 1 - k_n^l)$, the kernel never exceeds the right boundary, so the overhang should be clamped to zero. This reasoning leads to the expression for k_n^r summarized in Eq. (9). The configuration at the last convolution position in Fig. 9(a) visually confirms this behavior.

Because the first convolution operation causes an overhang of k_n^l elements on the left, we must add exactly k_n^l zeros to the left of the input to ensure that all valid operations are computed. To preserve the spatial alignment and keep the output centered, the same amount of padding is applied to the right. Therefore, the symmetric padding amount p_n required for a standard convolution that satisfies Assumptions 1 and 2 is given by $p_n = k_n^l$, as stated in Eq. (11). In the example of Fig. 9(a), this yields $p_n = 1$.

For the corresponding transposed convolution, shown in Fig. 9(b), we rely on the standard interpretation that a transposed convolution is equivalent to a convolution with stride 1 whose kernel is spatially flipped. Under this view, the roles of the left and right overhangs are swapped. Hence, if the original convolution uses overhangs k_n^l and k_n^r, the flipped kernel has overhangs k'^l_n and k'^r_n that satisfy Eq. (10), i.e., k'^r_n equals the original left overhang k_n^l, and k'^l_n is determined by the remaining taps to the left. In Fig. 9(b), this leads to $k'^r_n = 1$ and $k'^l_n = 2$.

Unlike the standard convolution, the transposed convolution first performs zero interpolation and zero padding on its input, and then applies a stride-1 convolution. Therefore, the $(s_n - 1)$ term that appears in the derivation of k_n^r in Eq. (9) is no longer relevant for the transposed operation. Instead, the transposed convolution requires an additional asymmetric output padding term a_n so that its spatial transform exactly inverts the corresponding convolution. This output padding is defined in Eq. (12) in terms of the convolution parameters X_n, p_n, k_n, and s_n. In the example of Fig. 9(b), we obtain $a_n = 0$, meaning that no extra output padding is needed.

Finally, combining Assumption 1 with the above alignment and padding rules ensures that the paired convolution and transposed convolution change the spatial size and the corresponding support variables by an exact integer factor s_n, as summarized in Eq. (5) and Eq. (6). Under these conditions, the downsampling and upsampling operations are perfectly matched in terms of spatial alignment.

3. Derivation of Recursive Overlap Formulas and Examples

In this section, we derive the exact recursive formulas for the minimum overlap required by convolution and transposed convolution operations. These derivations mathematically support the overlap propagation rules summarized in the main paper.

3.1. Convolution

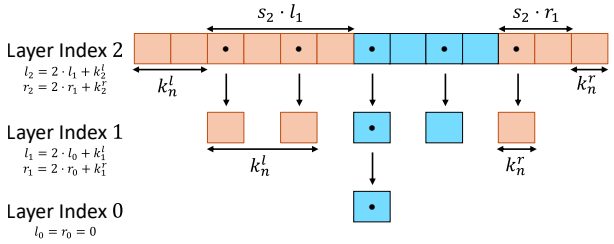

Figure 10. This depicts a convolution in the proposed method with $k_1 = k_2 = 5$ and $s_1 = s_2 = 2$. The dotted areas indicate the regions where the actual computation is performed.

A simple 1-D example is shown in Fig. 10. At the coarsest layer $n = 0$ in this example, the central patch exactly coincides with the region of interest, so no extra overlap is required and we can set

$$l_0 = 0, \qquad r_0 = 0. \tag{23}$$

The next layer $n = 1$ applies a standard convolution with stride $s_1 = 2$ and kernel size k_1 that satisfies Assumptions 1–2. By Assumption 2, each unit shift in the feature

map at layer $n = 1$ corresponds to a shift of s_1 samples in the input at layer $n = 0$. Consequently, the overlap needed at layer 0 to support a given overlap at layer 1 is expanded by a factor of s_1. In other words, the left/right overlaps l_0 and r_0 are mapped to $s_1 \cdot l_0$ and $s_1 \cdot r_0$ at layer 1.

In addition to this scaled overlap, we must also account for the kernel overhangs at layer 1. As shown in Sec. 1 , the standard convolution with kernel size k_1 and stride s_1 requires an overhang of k_1^l elements to the left at the first convolution position, and k_1^r elements to the right at the last convolution position (see Eqs. (8) and (9)). Therefore, the total overlap required at layer 1 is obtained by adding these kernel-dependent terms to the scaled overlaps:

$$l_1 = s_1 \cdot l_0 + k_1^l, \qquad r_1 = s_1 \cdot r_0 + k_1^r. \tag{24}$$

In the 1-D example in Fig. 10, the kernel parameters are chosen such that $k_1^l = 2$ and $k_1^r = 1$. Substituting $l_0 = r_0 = 0$ then yields $l_1 = 2$ and $r_1 = 1$, in agreement with the diagram.

The same reasoning applies when propagating the overlap from layer $n-1$ to layer n for arbitrary $n \in \mathbb{N}_L$. Because a stride-s_n convolution at layer n downsamples the signal by a factor of s_n, the overlaps (l_{n-1}, r_{n-1}) at layer $n-1$ expand to $(s_n \cdot l_{n-1}, s_n \cdot r_{n-1})$ when expressed in the coordinates of layer n. The convolution at layer n then adds an additional kernel-dependent overhang of k_n^l and k_n^r on the left and right, respectively. This yields the general recursion

$$l_n = s_n \cdot l_{n-1} + k_n^l, \qquad r_n = s_n \cdot r_{n-1} + k_n^r, \qquad n \in \mathbb{N}_L. \tag{25}$$

These relations coincide with the overlap propagation rule for convolution summarized in the main paper.

3.2. Transposed Convolution

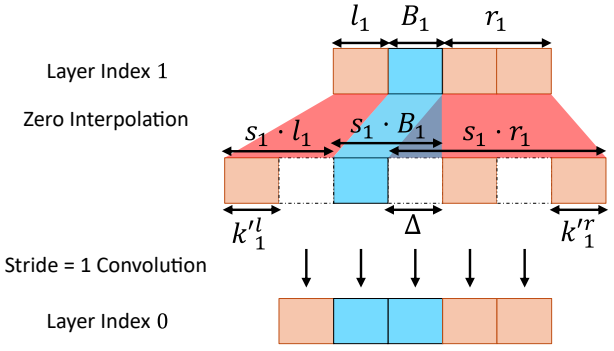

Figure 11. This depicts a transposed convolution in the proposed method with $k_1 = 3$ and $s_1 = 2$.

We next derive the corresponding recursion for transposed convolution in our proposed method. As in Sec. 1, we adopt the standard interpretation of a transposed convolution as a stride-1 convolution with a spatially flipped kernel applied to a zero-inserted and padded input.

To build intuition, we first consider the 1-D example illustrated in Fig. 11. Suppose that $k_n = 3$, $s_n = 2$, and $l_1 = 1, r_1 = 2$ at layer $n = 1$, with kernel overhangs $k'^l_1 = 1$ and $k'^r_1 = 1$ as implied by Eq. (28). Zero insertion expands the overlaps at layer $n = 1$ so that the left and right overlaps become $s_1 l_1 = 2$ and $s_1 r_1 = 4$, respectively. From the perspective of layer $n = 0$, part of these expanded overlaps is consumed by the flipped kernel to reconstruct the central block B_0, and only the remaining portion should be counted as overlap. After subtracting the contributions corresponding to the left and right kernel overhangs, k'^l_1 and k'^r_1, we must also subtract the $\Delta = (s_1 - 1)$ zero positions that are already covered by the central block from the right overlap r_1. Consequently, the resulting overlaps at layer 0 are

$$l_0 = 2 \cdot 1 - 1 = 1, \tag{26}$$

$$r_0 = 2 \cdot 2 - 1 - 1 = 2, \tag{27}$$

which matches the diagram in Fig. 11. This concrete case illustrates how the overlap at a lower layer is obtained by starting from the expanded overlap at the higher layer and subtracting the portions already used to reconstruct the central block.

We now generalize this construction. Consider again two consecutive layers, n and $n-1$, and let (l_n, B_n, r_n) denote the left overlap, central block, and right overlap at layer n. After inserting $(s_n - 1)$ zeros between neighboring samples at layer n, the three segments l_n, B_n, and r_n are all expanded by a factor of s_n, so that their lengths become $s_n \cdot l_n$, $s_n \cdot B_n$, and $s_n \cdot r_n$, respectively. We denote by k'^l_n and k'^r_n the left and right kernel overhangs of the associated transposed convolution. As discussed in Eq. (28), these are related to the overhangs k_n^l and k_n^r of the forward convolution by

$$k'^r_n = k_n^l, \qquad k'^l_n = k_n - 1 - k'^r_n. \tag{28}$$

We now examine how much of the expanded overlap remains outside the central block when viewed from the perspective of layer $n-1$. On the left-hand side, the expanded overlap has length $s_n \cdot l_n$, but its first k'^l_n samples are consumed by the flipped kernel to reconstruct the left portion of l_{n-1}. Therefore, the required left overlap at layer $n-1$ is

$$l_{n-1} = s_n \cdot l_n - k'^l_n. \tag{29}$$

On the right-hand side, two corrections are subtracted from the overlap length $s_n \cdot r_n$. First, the right kernel overhang k'^r_n is used for reconstruction, and second, the $\Delta = (s_n - 1)$ newly inserted zeros are considered part of the central block at layer $n-1$. After excluding these contributions, the remaining portion constitutes the right overlap

at layer $n-1$:

$$r_{n-1} = s_n \cdot r_n - {k'}_n^r - (s_n - 1). \tag{30}$$

Since the above argument does not depend on the specific numerical values of l_n and r_n, it can be applied recursively to all layers in the transposed convolution operation. Combining Eq. (29) and Eq. (30) thus yields the general backward recursion

$$\begin{aligned} l_{n-1} &= s_n \cdot l_n - {k'}_n^l, \\ r_{n-1} &= s_n \cdot r_n - {k'}_n^r - (s_n - 1), \end{aligned} \quad n \in \mathbb{N}_L, \tag{31}$$

which corresponds to the transposed convolution overlap propagation rule stated in the main paper.

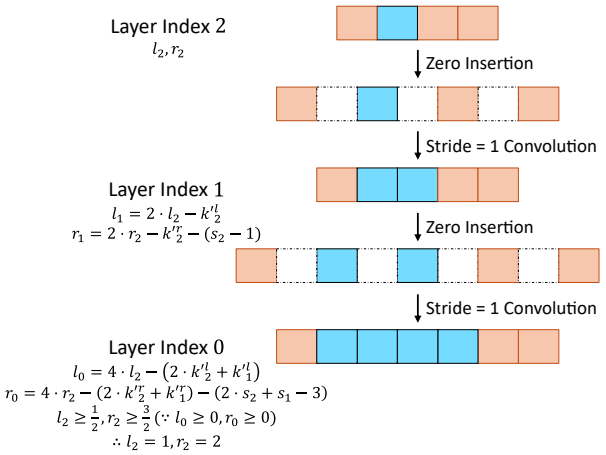

Figure 12. This depicts a transposed convolution in the proposed method with $k_1 = k_2 = 3$ and $s_1 = s_2 = 2$.

Referring to Fig. 12, we treat l_2 and r_2 as unknown variables and use Eq. (31) iteratively down to $n = 0$ to express l_0 and r_0 in terms of l_2 and r_2, respectively. Given the constraint that l_0 and r_0 must be the smallest non-negative integers, we can determine the minimum integer values for l_2 and r_2. Once l_2 and r_2 are found, we can then apply Eq. (31) again to calculate l_n and r_n for all values of n.

4. Overlap Specifications for LIC Models

This section presents the detailed calculation process and results for the overlap of various LIC models, which were omitted from the main text for brevity. We provide these details to demonstrate that our proposed method is generally applicable across different architectures and to serve as a reference for reproducibility. In a multipath network, for convenience, the convolutional networks in the primary path are represented by their layer indices. To simplify this representation, networks with a kernel size and stride of one ($k_n = s_n = 1$) are excluded from the indexing.

Table 6. Overlap propagation for Cheng with CKBD. The results for CKBD are identical to those in the CKBD table of ELIC. PS denotes PixelShuffle.

g_a					g_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
-	-	-	-	-	25	Conv	3	1	(11, 11)
-	-	-	-	-	24	Conv	3	1	(10, 10)
-	-	-	-	-	23	Conv	3	1	(9, 9)
-	-	-	-	-	22	Conv	3	1	(8, 8)
-	-	-	-	-	21	Conv	3	1	(7, 7)
-	-	-	-	-	20	Conv	3	1	(6, 6)
19	Conv	3	2	(117, 102)	19	PS ($u = 2$)	-	-	(5, 5)
18	Conv	3	1	(58, 51)	18	Conv	3	1	(10, 10)
17	Conv	3	1	(57, 50)	17	Conv	3	1	(9, 9)
16	Conv	3	1	(56, 49)	16	Conv	3	1	(8, 8)
15	Conv	3	2	(55, 48)	15	Conv	3	1	(7, 7)
14	Conv	3	1	(27, 24)	14	PS ($u = 2$)	-	-	(6, 6)
13	Conv	3	1	(26, 23)	13	Conv	3	1	(12, 12)
12	Conv	3	1	(25, 22)	12	Conv	3	1	(11, 11)
11	Conv	3	1	(24, 21)	11	Conv	3	1	(10, 10)
10	Conv	3	1	(23, 20)	10	Conv	3	1	(9, 9)
9	Conv	3	1	(22, 19)	9	Conv	3	1	(8, 8)
8	Conv	3	2	(21, 18)	8	Conv	3	1	(7, 7)
7	Conv	3	1	(10, 9)	7	Conv	3	1	(6, 6)
6	Conv	3	1	(9, 8)	6	PS ($u = 2$)	-	-	(5, 5)
5	Conv	3	1	(8, 7)	5	Conv	3	1	(10, 10)
4	Conv	3	2	(7, 6)	4	Conv	3	1	(9, 9)
3	Conv	3	1	(3, 3)	3	Conv	3	1	(8, 8)
2	Conv	3	1	(2, 2)	2	Conv	3	1	(7, 7)
1	Conv	3	1	(1, 1)	1	PS ($u = 2$)	-	-	(6, 6)
0	-	-	-	(0, 0)	0	-	-	-	(12, 12)
h_a					h_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
-	-	-	-	-	7	Conv	3	1	(4, 4)
-	-	-	-	-	6	Conv	3	1	(3, 3)
5	Conv	3	1	(7, 4)	5	PS ($u = 2$)	-	-	(2, 2)
4	Conv	3	1	(6, 3)	4	Conv	3	1	(4, 4)
3	Conv	3	2	(5, 2)	3	Conv	3	1	(3, 3)
2	Conv	3	1	(2, 1)	2	PS ($u = 2$)	-	-	(2, 2)
1	Conv	3	2	(1, 0)	1	Conv	3	1	(4, 4)
0	-	-	-	(0, 0)	0	-	-	-	(3, 3)

Table 7. Overlap propagation in the Hyperprior with CKBD. The results for CKBD are identical to those in the CKBD table of ELIC.

g_a					g_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
4	Conv	5	2	(30, 15)	4	T.Conv	5	2	(2, 3)
3	Conv	5	2	(14, 7)	3	T.Conv	5	2	(2, 3)
2	Conv	5	2	(6, 3)	2	T.Conv	5	2	(2, 3)
1	Conv	5	2	(2, 1)	1	T.Conv	5	2	(2, 3)
0	-	-	-	(0, 0)	0	-	-	-	(2, 3)
h_a					h_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
3	Conv	3	1	(7, 4)	3	T.Conv	5	2	(2, 3)
2	Conv	5	2	(6, 3)	2	T.Conv	5	2	(2, 3)
1	Conv	5	2	(2, 1)	1	Conv	3	1	(2, 3)
0	-	-	-	(0, 0)	0	-	-	-	(1, 2)

Table 8. Overlap propagation in ELIC.

g_a					g_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
19	Conv	5	2	(132, 117)	19	Conv	3	1	(9, 10)
18	Conv	3	1	(65, 58)	18	Conv	3	1	(8, 9)
17	Conv	3	1	(64, 57)	17	Conv	3	1	(7, 8)
16	Conv	3	1	(63, 56)	16	T.Conv	5	2	(6, 7)
15	Conv	5	2	(62, 55)	15	Conv	3	1	(10, 11)
14	Conv	3	1	(30, 27)	14	Conv	3	1	(9, 10)
13	Conv	3	1	(29, 26)	13	Conv	3	1	(8, 9)
12	Conv	3	1	(28, 25)	12	T.Conv	5	2	(7, 8)
11	Conv	3	1	(27, 24)	11	Conv	3	1	(12, 13)
10	Conv	3	1	(26, 23)	10	Conv	3	1	(11, 12)
9	Conv	3	1	(25, 22)	9	Conv	3	1	(10, 11)
8	Conv	5	2	(24, 21)	8	Conv	3	1	(9, 10)
7	Conv	3	1	(11, 10)	7	Conv	3	1	(8, 9)
6	Conv	3	1	(10, 9)	6	Conv	3	1	(7, 8)
5	Conv	3	1	(9, 8)	5	T.Conv	5	2	(6, 7)
4	Conv	5	2	(8, 7)	4	Conv	3	1	(10, 11)
3	Conv	3	1	(3, 3)	3	Conv	3	1	(9, 10)
2	Conv	3	1	(2, 2)	2	Conv	3	1	(8, 9)
1	Conv	3	1	(1, 1)	1	T.Conv	5	2	(7, 8)
0	-	-	-	(0, 0)	0	-	-	-	(12, 13)
h_a					h_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
3	Conv	3	1	(7, 4)	3	T.Conv	5	2	(2, 3)
2	Conv	5	2	(6, 3)	2	T.Conv	5	2	(2, 3)
1	Conv	5	2	(2, 1)	1	Conv	3	1	(2, 3)
0	-	-	-	(0, 0)	0	-	-	-	(1, 2)
ChARM					**CKBD**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
3	Conv	5	1	(6, 6)	-	-	-	-	-
2	Conv	5	1	(4, 4)	-	-	-	-	-
1	Conv	5	1	(2, 2)	1	Conv	5	1	(2, 2)
0	-	-	-	(0, 0)	0	-	-	-	(0, 0)

Table 9. Overlap propagation in the Hyperprior without AR context.

g_a					g_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
4	Conv	5	2	(30, 15)	4	T.Conv	5	2	(2, 3)
3	Conv	5	2	(14, 7)	3	T.Conv	5	2	(2, 3)
2	Conv	5	2	(6, 3)	2	T.Conv	5	2	(2, 3)
1	Conv	5	2	(2, 1)	1	T.Conv	5	2	(2, 3)
0	-	-	-	(0, 0)	0	-	-	-	(2, 3)
h_a					h_s				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
3	Conv	3	1	(7, 4)	3	T.Conv	5	2	(2, 3)
2	Conv	5	2	(6, 3)	2	T.Conv	5	2	(2, 3)
1	Conv	5	2	(2, 1)	1	Conv	3	1	(2, 3)
0	-	-	-	(0, 0)	0	-	-	-	(1, 2)

Table 10. Overlap propagation in the JPEG-AI reference software with the proposed method. PS denotes PixelShuffle.

Encoder(Y)					**Encoder(UV)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
7	Conv	3	2	(29, 14)	-	-	-	-	-
6	Conv	3	1	(14, 7)	6	Conv	3	2	(21, 14)
5	Conv	3	2	(13, 6)	5	Conv	3	1	(10, 7)
4	Conv	3	1	(6, 3)	4	Conv	3	2	(9, 6)
3	Conv	3	2	(5, 2)	3	Conv	3	1	(4, 3)
2	Conv	3	1	(2, 1)	2	Conv	3	2	(3, 2)
1	Conv	3	2	(1, 0)	1	Conv	3	1	(1, 1)
0	-	-	-	(0, 0)	0	-	-	-	(0, 0)
Decoder(Y)					**Decoder(UV)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
8	Conv	3	1	(4, 4)	-	-	-	-	-
7	T.Conv	4	2	(3, 3)	-	-	-	-	-
6	Conv	3	1	(4, 4)	6	Conv	3	1	(4, 4)
5	T.Conv	4	2	(3, 3)	5	T.Conv	4	2	(3, 3)
4	Conv	3	1	(4, 4)	4	Conv	3	1	(4, 4)
3	Conv	3	1	(3, 3)	3	Conv	3	1	(3, 3)
2	Conv	3	1	(2, 2)	2	Conv	3	1	(2, 2)
1	PS ($u = 4$)	-	-	(1, 1)	1	PS ($u = 8$)	-	-	(1, 1)
0	-	-	-	(4, 4)	0	-	-	-	(8, 8)
Hyper Encoder					**Hyper Decoder**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
5	Conv	3	1	(7, 4)	-	-	-	-	-
4	Conv	3	1	(6, 3)	4	Conv	1	1	(2, 2)
3	Conv	3	2	(5, 2)	3	T.Conv	4	2	(2, 2)
2	Conv	3	1	(2, 1)	2	Conv	3	1	(2, 2)
1	Conv	3	2	(1, 0)	1	Conv	3	1	(1, 1)
0	-	-	-	(0, 0)	0	-	-	-	(0, 0)

Table 11. Overlap propagation in the baseline JPEG-AI reference software. PS denotes PixelShuffle.

Encoder(Y)					**Encoder(UV)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
7	Conv	3	2	(32, 32)	-	-	-	-	-
6	Conv	3	1	(16, 16)	6	Conv	3	2	(16, 16)
5	Conv	3	2	(16, 16)	5	Conv	3	1	(8, 8)
4	Conv	3	1	(8, 8)	4	Conv	3	2	(8, 8)
3	Conv	3	2	(8, 8)	3	Conv	3	1	(4, 4)
2	Conv	3	1	(4, 4)	2	Conv	3	2	(4, 4)
1	Conv	3	2	(4, 4)	1	Conv	3	1	(2, 2)
0	-	-	-	(2, 2)	0	-	-	-	(2, 2)
Decoder(Y)					**Decoder(UV)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
8	Conv	3	1	(2, 2)	-	-	-	-	-
7	T.Conv	4	2	(2, 2)	-	-	-	-	-
6	Conv	3	1	(4, 4)	6	Conv	3	1	(2, 2)
5	T.Conv	4	2	(8, 8)	5	T.Conv	4	2	(2, 2)
4	Conv	3	1	(8, 8)	4	Conv	3	1	(4, 4)
3	Conv	3	1	(8, 8)	3	Conv	3	1	(4, 4)
2	Conv	3	1	(8, 8)	2	Conv	3	1	(4, 4)
1	PS ($u = 4$)	-	-	(8, 8)	1	PS ($u = 8$)	-	-	(4, 4)
0	-	-	-	(32, 32)	0	-	-	-	(32, 32)
Hyper Encoder					**Hyper Decoder**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
5	Conv	3	1	(2, 2)	-	-	-	-	-
4	Conv	3	1	(2, 2)	4	Conv	1	1	(1, 1)
3	Conv	3	2	(2, 2)	3	T.Conv	4	2	(1, 1)
2	Conv	3	1	(1, 1)	2	Conv	3	1	(2, 2)
1	Conv	3	2	(1, 1)	1	Conv	3	1	(2, 2)
0	-	-	-	(1, 0)	0	-	-	-	(2, 2)

5. Peak Memory & Peak MACs in 2K Resolution

This section extends the resource efficiency analysis in the main paper to 2K resolution scenarios. By evaluating peak memory and peak MACs, we confirm that the same resource-saving trends are observed, further supporting the effectiveness of the proposed block-based approach.

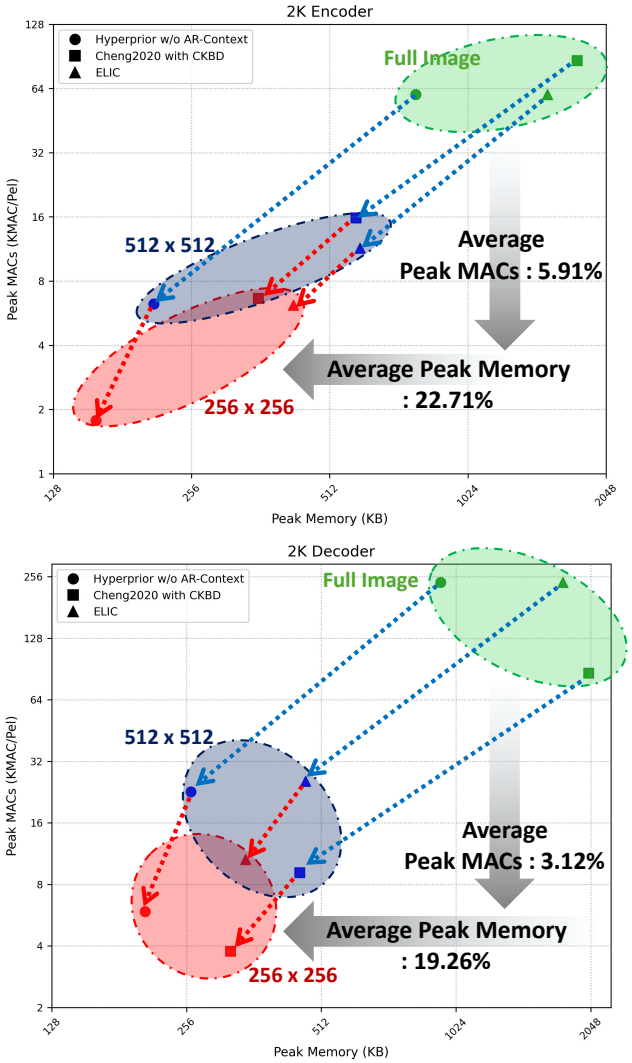

Figure 13. Comparison of Peak Memory and Peak Computation across various LIC models. Specifically, as the block size decreases, both Peak Memory and Peak MACs drop drastically. **Notably, as highlighted by the dotted arrows, this consistent pattern across all LIC models confirms that our block-based strategy is universally effective for reducing resource peaks.** The curve for Hyperprior with CKBD is omitted because it overlaps with the others.

Fig. 13 also exhibits a trend similar to that of Fig. 1. As shown in the 2K encoder and decoder results, resource efficiency improves remarkably as the input processing unit is subdivided from the Full Image (green region) to 512×512 (blue region) and 256×256 (red region). This implies that our proposed block-based method is a universally effective strategy for mitigating peak memory and peak computational costs.

6. Extended Experiments in JPEG-AI

This section serves as an ablation study for the experiments presented in Section 7. Here, we expand our analysis from Peak MACs to also include Total MACs and provide a comprehensive analysis of the UV encoder and decoder. We also visually demonstrate the quality of the reconstruction and verify the absence of artifacts.

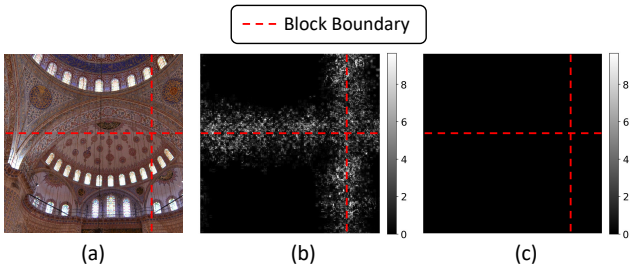

Figure 14. MSE comparison between the baseline and proposed methods for full-image reconstruction with a block size of 960 × 960. (a) full-image reconstruction. (b) MSE in baseline. (c) MSE in proposed. **Unlike the baseline, which displays artifacts along block boundaries, our method produces artifact-free results.**

As shown in Fig. 14, it is evident that the Baseline retains visible artifacts even in the pure JPEG-AI setting with a block size of 960×960, whereas our proposed method demonstrates an artifact-free reconstruction.

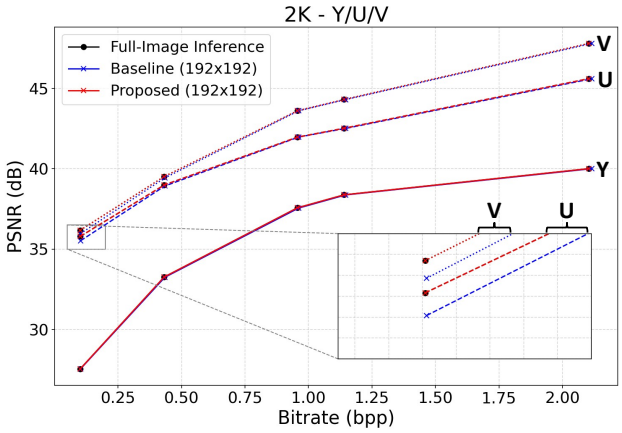

Figure 15. RD curves in JPEG-AI experiment. (2K, Y/U/V).

Consistent with the artifact-free results in Fig. 14, the RD curves of our proposed method in Fig. 15 perfectly overlap with those of full-image inference, further validating its bit-identical reconstruction performance.

Table 12. Comparison of Peak MAC, Total kMAC, and Decoding Time. Percentages in parentheses indicate the ratio of the Proposed method relative to the Baseline. **The results show that while patch-based processing introduces slight overheads in total operations, the proposed method effectively mitigates peak computational cost compared to the Baseline.**

Resolution	Blocksize	Method	Peak MAC / pel				Total kMAC / pel				Dec T. (sec)
			Enc. (Y)	Enc. (UV)	Dec. (Y)	Dec. (UV)	Enc. (Y)	Enc. (UV)	Dec. (Y)	Dec. (UV)	
2K	Full		11353.7	2838.4	5046.1	2838.4	812.9	220.4	367.0	132.2	0.93
	192×192	Baseline	218.3	54.6	97.0	54.6	1366.5	370.5	616.9	222.3	1.24
		Proposed	176.2 (80.7%)	52.9 (96.9%)	74.3 (76.6%)	54.6 (100.0%)	1140.3 (83.4%)	371.2 (100.2%)	529.4 (85.8%)	218.5 (98.3%)	1.45 (116.9%)
	448×448	Baseline	873.3	218.3	388.1	218.3	1002.8	271.9	452.7	163.1	0.99
		Proposed	786.9 (90.1%)	214.9 (98.4%)	341.2 (87.9%)	218.3 (100.0%)	960.0 (95.7%)	286.4 (105.3%)	438.5 (96.9%)	169.8 (104.1%)	1.06 (107.1%)
	960×960	Baseline	3493.4	873.4	1552.6	873.4	901.6	244.4	407.0	146.7	1.04
		Proposed	3318.3 (95.0%)	866.5 (99.2%)	1457.1 (93.8%)	873.4 (100.0%)	891.8 (98.9%)	255.4 (104.5%)	405.0 (99.5%)	152.1 (103.7%)	1.02 (98.1%)
4K	Full		9216.0	2304.0	4096.0	2304.0	775.4	210.2	350.0	126.1	1.43
	192×192	Baseline	87.6	21.9	38.9	21.9	1323.9	358.9	597.7	215.3	1.98
		Proposed	70.7 (80.7%)	21.2 (96.8%)	29.8 (76.6%)	21.9 (100.0%)	1071.1 (80.9%)	346.2 (96.5%)	496.5 (83.1%)	203.8 (94.7%)	2.46 (124.2%)
	448×448	Baseline	350.5	87.6	155.8	87.6	988.5	268.0	446.2	160.8	1.59
		Proposed	315.8 (90.1%)	86.3 (98.5%)	136.9 (87.9%)	87.6 (100.0%)	906.3 (91.7%)	268.9 (100.3%)	413.6 (92.7%)	159.5 (99.2%)	1.63 (102.5%)
	960×960	Baseline	1401.8	350.5	623.0	350.5	867.9	230.6	391.8	141.2	1.62
		Proposed	1331.6 (95.0%)	347.7 (99.2%)	584.7 (93.9%)	350.5 (100.0%)	846.5 (97.5%)	241.7 (104.8%)	384.3 (98.1%)	144.0 (102.0%)	1.75 (108.0%)

According to Tab. 12, patch-based methods generally incur higher total MACs than the full-image approach due to overlap overhead. Nevertheless, the proposed method consistently outperforms the baseline, particularly for the Y component, and this advantage becomes more pronounced as the block size decreases. Although the Proposed method requires slightly longer decoding time than the Baseline, this gap is likely due to implementation-level overhead rather than algorithmic inefficiency.