electronics

MDPI

Article

Memory-Efficient Fixed-Length Representation of Synchronous Event Frames for Very-Low-Power Chip Integration

Ionut Schiopu * and Radu Ciprian Bilcu

Tampere Handset Camera Innovation Lab, Huawei Technologies Oy (Finland) Co., Ltd., 33720 Tampere, Finland; radu.ciprian.bilcu@huawei.com

* Correspondence: ionut.schiopu@huawei.com

Abstract: The new event cameras are now widely used in many computer vision applications. Their high raw data bitrate levels require a more efficient fixed-length representation for low-bandwidth transmission from the event sensor to the processing chip. A novel low-complexity lossless compression framework is proposed for encoding the synchronous event frames (EFs) by introducing a novel memory-efficient fixed-length representation suitable for hardware implementation in the very-low-power (VLP) event-processing chip. A first contribution proposes an improved representation of the ternary frames using pixel-group frame partitioning and symbol remapping. Another contribution proposes a novel low-complexity memory-efficient fixed-length representation using multi-level lookup tables (LUTs). Complex experimental analysis is performed using a set of group-size configurations. For very-large group-size configurations, an improved representation is proposed using a mask-LUT structure. The experimental evaluation on a public dataset demonstrates that the proposed fixed-length coding framework provides at least two times the compression ratio relative to the raw EF representation and a close performance compared with variable-length video coding standards and variable-length state-of-the-art image codecs for lossless compression of ternary EFs generated at frequencies bellow one KHz. To our knowledge, the paper is the first to introduce a low-complexity memory-efficient fixed-length representation for lossless compression of synchronous EFs, suitable for integration into a VLP event-processing chip.

Keywords: fixed-length coding; synchronous event camera frames; very-low-power chips; low-complexity coding; lookup-table-based event frame representation

Citation: Schiopu, I.; Bilcu, R.C. Memory-Efficient Fixed-Length Representation of Synchronous Event Frames for Very-Low-Power Chip Integration. *Electronics* **2023**, *12*, 2302. https://doi.org/10.3390/electronics12102302

Academic Editor: Kiat Seng Yeo

Received: 31 March 2023
Revised: 11 May 2023
Accepted: 17 May 2023
Published: 19 May 2023

1. Introduction

A new type of sensor called an event camera was recently developed based on the new biomimetic technologies proposed in the neuromorphic engineering domain [1]. The event camera is bio-inspired by the human brain, as each pixel operates separately to mimic the biological neural system and performs simple tasks with small energy consumption. More exactly, in contrast to the conventional camera where all are operating simultaneously, the event camera sensor introduces a novel design where each pixel detects and reports independently only the changes (increased or deceased) of the incoming light intensity above a threshold or remains silent otherwise.

The event camera proposes a new *paradigm shift for capturing visual data*, where only the dynamic information of the scene is captured using a sequence (stream) of events that are triggered asynchronously, without capturing the unnecessary static information represented by the background or skyline. This gives the event camera some very important properties of low energy consumption, low latency, high dynamic range (HDR), and high temporal resolution, as the asynchronous events can be triggered individually at the smallest timestamp distance of one microsecond (µs). Two types of sensors are now made available on the market: the dynamic vision sensor (DVS) [2], which captures sequences of asynchronous events; and the dynamic and active-pixel vision sensor (DAVIS) [3], which

adds a second camera to the DVS sensor, an active pixel sensor (APS) for capturing greyscale or color (RGB) frames.

In the computer vision domain, event cameras are now widely used as the solutions designed to process the RGB and event modalities already reached state-of-the-art performance for many applications such as deblurring [4], feature detection and tracking [5,6], optic flow estimation [7], 3D estimation [8], super-resolution [9], interpolation [10], visual odometry [11], and many others. For a comprehensive literature review of event-based applications in computer vision, we recommend the work presented in [12]. Since many of these upper-level applications are designed to process the event modality as an image rather than an asynchronous events sequence, the event data acquired by the DVS sensor is pre-processed, usually using an event-accumulation process to form a sequence of synchronous Event Frames (EFs), i.e., the EF contains a set of synchronous events having assigned the timestamp associated with the EF.

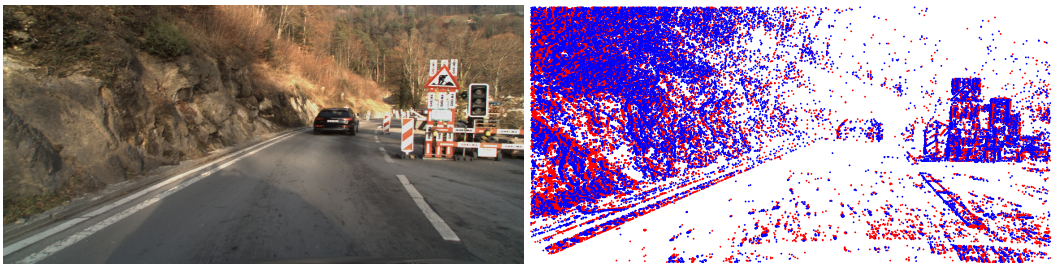
In general, an EF is represented using the ternary alphabet, $\mathcal{A}_3^{raw} = \{0, 1, 2\}$, where for each pixel position (x, y), one of the following three possible symbols is assigned: "0" to signal the position of an event with negative polarity (report a light intensity decrease), "1" to signal the position of a non-event, and "2" to signal the position of an event with positive polarity (report a light intensity increase). Please note that sometimes the representation of the first two symbols is swapped. Although $\mathcal{A}_3^{raw}$ is used to store the EF on the disk, one can note that in some cases, a ternary alphabet containing different symbols might be used for a better visual representation. E.g., $\mathcal{A}_3^{p} = \{-1, 0, +1\}$ is preferred for signaling the negative and positive event polarity; $\mathcal{A}_3^{int} = \{0, 127, 255\}$ is preferred for having an 8-bit intensity-like representation and guarantees compatibility with traditional image and video codecs. In the DVS sensor representation, each event is stored using a package of 8 Bytes (B). While in the raw EF representation, each pixel contains a ternary symbol that is stored using 2 bits, i.e., an EF having a $W \times H$ pixel resolution sensor is stored using a total of $2 \times H \times W$ bits.

The event data captured by the event sensor contains different distortions (e.g., noise, latency), which must be corrected by the sensor or the event-processing chip, i.e., on low-power chips with limited memory before it can be consumed by upper-level applications. Since the raw EF representation reaches high bitrate levels, a more efficient fixed-length representation is required for low-bandwidth transmission of the event data from the event sensor to the event-processing chip. One can note that such a method must satisfy several constraints. The codec must operate in a group of pixels as other algorithms are employed to correct the event data using a limited chip memory, i.e., the codec must provide a fixed-length representation to provide random access to any group of pixels by allocating an equal number of bits to encode each group. Note that in data compression, large coding gains are achieved by allocating a variable-length representation to each group of pixels, as the coding gain depends on the amount of data stored inside each group. The codec must have a low complexity so it can be hardware written, i.e., only simple computations are accepted. The codec must provide an efficient memory representation to justify the costs of hardware writing the lossless codec. Hence, in this paper, we propose a novel low-complexity lossless compression framework of synchronous EFs based on fixed-length coding, which is suitable for integration into very-low-power (VLP) processing chips.

In the literature, the event data compression problem remains understudied. Only a few coding solutions are proposed to either encode asynchronous event sequences [13,14] or synchronous EF sequences [15,16]. In [13], the authors propose to encode lossless asynchronous event sequences by exploiting the spatial and temporal characteristics of the event location information. The research was further extended in [14]. In [15], the Time Aggregation-based Lossless Video Encoding for Neuromorphic vision sensor data (TALVEN) algorithm is proposed, where an event-accumulation process is employed to generate EFs by concatenating the positive and negative polarity EF counts and form an EF sequence then compresses by the High-Efficiency Video Coding (HEVC) [17] standard. Finally, in [16], a lossy coding solution is proposed for the DAVIS sensor. One can note that

these solutions are much too complex for hardware implementation in the event-processing chip with limited memory. These are performance–oriented compression solutions that can be integrated only in a system-on-a-chip (SoC) where enough computation power is available. Here, we propose a low-complexity–oriented lossless compression framework of EFs suitable for integration into low-power event signal processing (ESP) chips or VLP event-processing chips, i.e., inside the sensor, where the event data are constrained to have fixed-length representation.

In our prior work, we employ an event-accumulation process where each asynchronous event sequence is split into spatiotemporal neighborhoods of time interval Δ, where each neighborhood generates an EF as follows: at each pixel position (x, y), the polarity sum of the events triggered in the time interval Δ is computed so that (x, y) is set as the sum's sign, see Figure 1. In our prior work [18], we proposed a context-based lossless image codec for encoding a stack of up to eight EFs using two data structures: an event map image (EMI) that stores the event spatial information; and the concatenated polarity vector (CPV) that stores the polarity information; a lossless codec suitable for integration into SoC. In our prior work [19], we proposed a low-complexity lossless coding framework by adapting the run-length encoding scheme and Elias coding for EF coding, suitable for integration into low-cost ESP chips. While in our prior work [20], we proposed a low-complexity lossless coding method for encoding the event data represented as asynchronous event sequences that employ only low-complexity coding techniques so that it is suitable for integration into low-cost ESP chips. Moreover, these solutions provide a variable-length representation, a strategy used by most state-of-the-art coding methods. In contrast, the goal of this work is to introduce a novel low-complexity lossless coding framework for fixed-length representation of EFs using multi-level look-up tables (LUTs) that is suitable for integration into VLP chips.



(**a**) (**b**)

Figure 1. A camera view example of two modalities (RGB and Event) captured by sequence "*interlaken*_00_*c*" in the DSEC training dataset [21] using a pair of a standard RGB camera and a Prophesee Gen3.1 (DVS) event camera. (**a**) Crop of first-captured RGB frame. (**b**) The EF was generated using the asynchronous event sequence (captured before, during, and after the frame exposure time) by employing a sum-accumulation process over the first spatiotemporal neighborhood in the sequence of $\Delta = 5.555$ ms (180 fps). Blue, red, and white background marks the negative polarity-sum positions, the positive polarity-sum positions, and the zero-sum (or non-event) positions, respectively.

In summary, the novel contributions proposed in this paper are listed below.

(1) A novel lossless compression framework for encoding synchronous EFs, suitable for hardware implementation in VLP event-processing chips; see Section 3.
(2) A novel ternary frame representationmusing group partitioning and symbol remapping; see Section 3.1.
(3) A novel memory-efficient low-complexity coding scheme for lossless compression of sparse images using a novel fixed-length representation based on multi-level LUTs; see Section 3.2.2.
(4) An improved fixed-length representation using a mask-LUT structure for encoding EFs using very-large groups of pixels; see Section 3.2.3.

(5) A codec that provides random access (RA) to any group of pixels using a fixed-length representation; see Section 3.3.

The remainder of this paper is organized as follows. Section 2 presents a discussion on state-of-the-art methods developed for the new event modality. Section 3 describes the proposed fixed-length coding framework. Section 4 presents the experimental evaluation of the proposed framework. Section 5 concludes this work.

2. State-of-the-Art Methods

Let us consider an event sensor having a resolution of $W \times H$ pixels. When the sensor detects a change in the incoming light intensity that is above a set threshold, an asynchronous event, denoted $e_i = (x_i, y_i, p_i, t_i)$, is triggered by encapsulating and transmitting 8 B of event data containing the following information: **(i)** the event spatial information, denoted (x_i, y_i), where $x_i \in [1, H]$, and $y_i \in [1, W]$, corresponding to the pixel position *where* the change was detected; **(ii)** the event polarity information, denoted $p_i \in \{-1, 1\}$, which signals the *type* of change using symbol " -1" for a decrease and symbol "1" for an increase in the incoming light intensity; and **(iii)** the event time information, t_i, i.e., the timestamp *when* the change was detected. Let us denote the asynchronous event sequence capture by a DVS sensor [2] as $\mathcal{S}_\mathcal{T} = \{e_i\}_{i=1,2,\ldots,N_e}$, where N_e events were triggered over the time period $\mathcal{T}$ µs.

One can note that the processing steps in event-based computer vision solutions are significantly different than in the case of conventional color cameras. In the event-signal processing framework, an important decision is represented by the selection of the event representation. $\mathcal{S}_\mathcal{T}$ is first extracted from the raw DVS data and then further transformed into the format interpretable by the following processing steps, which may differ from one application to another.

The section is organized as follows. Section 2.1 analysis the event representations proposed in the computer vision domain. Section 2.2 outlines the proposed state-of-the-art methods for lossless event data compression and analysis how to modify the traditional data compression codecs for lossless event data compression.

2.1. Event Data Representations in Computer Vision

The simplest event representation that gained a lot of popularity in recent years is spike processing, such as Spike Neural Networks (SNNs), which directly can process the asynchronous event sequences [22]. Another representation used in Computer Vision is the "image-like" representation, denoted here EF, as most of the upper applications prefer to consume the event data as an image rather than an asynchronous event sequence. One can note that the event data can be stored as a sequence of generated EFs in a lossless manner (no information is lost) only by setting Δ with the smallest time-window size, i.e., $\Delta = 1$ µs. Then the raw EF representation will require an impressive amount of space to store one second of captured event data, i.e., $\frac{H \times W}{4}$ MB (around 76.8 GB). Note that the raw EF representation (2-bit per pixel) is more efficient than the raw sensor representation (8 B per event) only if more than $p_\tau = 3.125\%$ of the positions in the generated EF sequence signal event positions, i.e., a capturing event density that the DVS sensor cannot reach using current technology. Therefore, Δ is usually set using values in the ms time range so that the generated EFs have a much higher matrix filling percentage than p_τ, which results in losing some part of the raw captured information.

Different pre-processing strategies were proposed in the literature to quantize the event data and further process it in such a way that an upper-level application can achieve state-of-the-art performance. In [23], the event polarity sum frames are computed by summing the polarity of events within a given time window for real-time visual-inertial odometry. In [24], the count-based event-accumulation representation is proposed for steering prediction for self-driving cars, where the events within a temporal window are counted for each polarity type to generate two EFs. In [25], the distance surface representation is proposed for pixel motion estimation, where the generated image contains

the spatial distance to the active pixel. In [26], the surface of active events representation is proposed for computing the visual flow, where the EF contains only the latest event triggered at each pixel position. In [27,28], the graph-based representation is proposed for object classification, where the events are transformed within a temporal window into a set of connected nodes. Although this representation is quite compact and efficient, the generated graphs can be computationally expensive. In [29], the voxel grid representation is proposed. In [30], the event spike tensor representation is proposed as a voxel grid variation. In [31], the time-ordered recent events representation is proposed for a wide range of applications (denoising, reconstruction, classification, pose estimation), where a first-in-first-out buffer is used to retain the most recent events at each location, while the others are ignored.

2.2. Event Data Representations in Lossless Compression

In the lossless compression domain, two event data representations, asynchronous event sequences, and event-accumulation EFs, were studied. In [13], the authors propose a solution for encoding the raw asynchronous event sequences by removing the redundancy of the spatial and temporal information using three strategies: adaptive macro-cube partitioning structure, the address-prior mode, and time-prior mode. An extension was proposed in [14] by introducing an event sequence octree-based cube partition and a flexible inter-cube prediction method based on motion estimation and motion compensation. In [32], the authors present a complex performance analysis of different lossless data compression algorithms employed to compete against the Spike Coding approach proposed in [14]. The study shows that several state-of-the-art traditional data compression codecs, such as the Lempel–Ziv–Markov chain algorithm (LZMA) [33] developed by Igor Pavlov and the dictionary-based codec Zeta Library (ZLIB) [34], provide a good coding performance. One can note that the comparison with [13] and [14] is not possible as their codec/source code is not made publicly available, and the PKU-DVS dataset [35] is partially made available only for academic research purpose.

The study in [32] was extended in [15] by introducing the TALVEN algorithm for encoding the count-accumulated EF sequences so that they can be further encoded by employing the HEVC codec [17]. For encoding event-accumulation EFs, in our prior work [18,19], we proposed to compare the coding performance with that of prior and current video coding standards, HEVC [17] and VVC [36], and of well-known state-of-the-art lossless image codecs, such as Context Adaptive Lossless Image Codec (CALIC) [37], and Free Lossless Image Format (FLIF) codec [38]. Experiments show that an improved performance is achieved when encoding an intensity-like event representation obtained using a symbol remapping technique where a set of five (consecutive) EFs is merged into a combined EF that contains (8-bit) intensity-like values.

In conclusion, the current coding solutions of event data, either based on event codecs [13–15] or traditional data/image/video codecs [17,33,34,36–38], provide competitive variable-length representations designed to optimize the codec's performance. In contrast, in this work, we propose a fixed-length representation designed to achieve low complexity and fast RA to any group of pixels, suitable for integration into VLP chips.

3. Proposed Framework

Figure 2 presents a comparison between the requirements imposed for different compression strategies. In a left-to-right order, the first representation is the raw image representation, where no compression strategy is employed as all pixels are represented uncompressed in memory using a fixed-length representation of the maximum number of bits required to store each symbol in the alphabet, and which provides random access to any pixel. The first compression strategy is the fixed-length representation, where on top of the fixed-length and random-access requirements, the codec must provide a more efficient representation of the image in the memory, i.e., to store the image using a reduced number of bits compared with raw representation. Moreover, such compression methods

usually propose to encode together a group of pixels (e.g., a $w \times h$ pixel-block) and must have a very low computational complexity so that it can be directly hardware written in the sensors. The second compression strategy is the variable-length representation with random access, where the coding performance is further improved by removing the fixed-length representation of each group of pixels and where a header is introduced to store additional information needed to provide random access to each group of pixels. The last compression strategy is storage-optimized, i.e., it is designed to provide the highest coding performance possible without imposing any constraints on algorithmic complexity, runtime, or memory requirements.

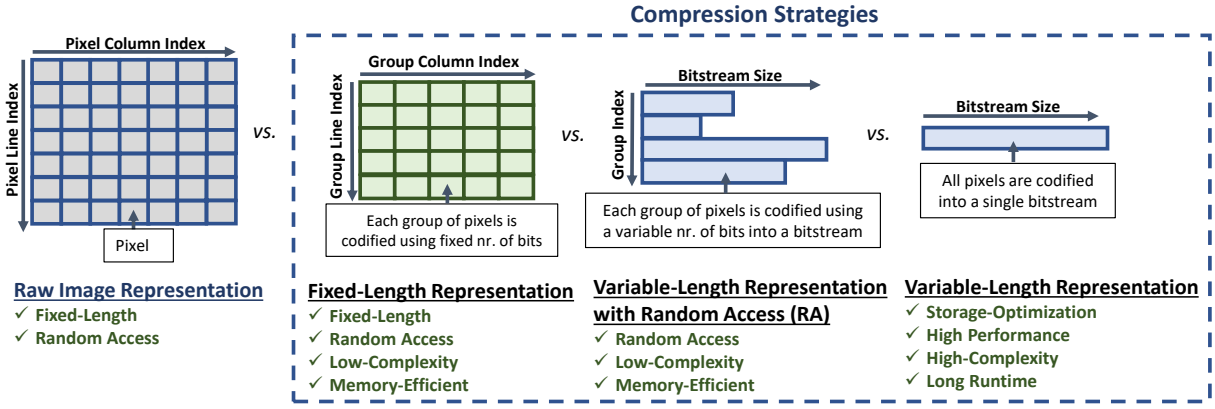


Figure 2. Comparison between the requirements of different compression strategies.

One can note that the fixed-length representation requirement is a very hard constraint and does not allow the solution to achieve the best performance. The fixed-length and variable-length representations offer the possibility to apply any pre-processing algorithm to a group of pixels while still storing in memory the entire package of data. In general, variable-length representations, also known as performance-oriented codecs, are optimized to achieve the best coding performance without imposing any constraints on complexity, runtime, or having access to groups of pixels.

Figure 3 presents what type of compression strategies can be used for each type of chip. E.g., the VLP chips may incorporate a lossless compression method as long as its complexity is very low so that it is possible to be written on hardware, and each group of pixels can be encoded separately so that they can be packetized together. The ESP chip may incorporate a lossless compression method to reduce the amount of data to be transmitted. Such a chip contains a limited memory of a few MB, and the codec must be simple enough to be hardware written. While in general, the SoC chip may incorporate any compression method as long as the SoC specifications allow the coding method to be run on the SoC. Finally, the data are transmitted from the SoC chip to be stored on the non-volatile memory of a mobile phone (e.g., memory card) or a computer (e.g., hard disk drive).

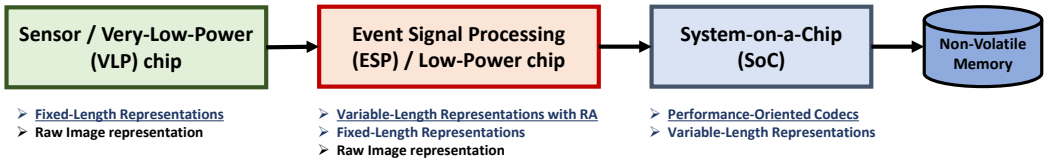


Figure 3. The compression strategy required for each type of chip.

In this work, we propose a fixed-length representation framework suitable for integration into VLP chips, which offers the possibility to store more efficiently in memory the input image compared with the raw image representation. Figure 4 presents the proposed fixed-length representation framework for encoding the input raw image representation. The proposed strategy consists in first dividing the raw image into groups of pixels of

$w \times h$ size, where each group is represented using a reduced set of (N_t) remapped symbols. The figure's central part shows that the raw image can be represented as remapped volume having group-partition resolution and remapped-group-size depth. While the right part shows the proposed method is employed to store the remapped volume using two data structures: a matrix for storing using a fixed-length representation of an index and additional information and a set of one or more LUTs for storing the unique symbols combination used to remap the original raw data.

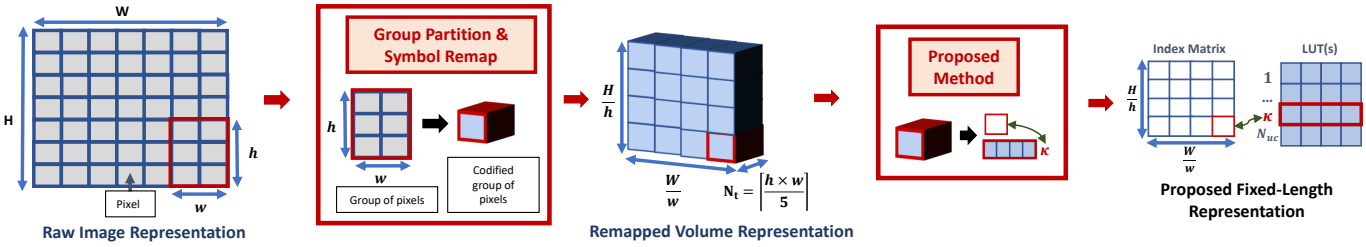


Figure 4. The proposed fixed-length LUT-based representation framework.

The section is organized as follows. Section 3.1 presents the group partitioning and symbol remapping processes. The remapped volume is used to generate the proposed fixed-length representation consisting of an index matrix and a set of one or more LUTs. Section 3.2 presents the proposed LUT-based Representations. Section 3.3 presents the final algorithmic details.

3.1. Group Partition and Symbol Remap

Let us denote **I** the input image of size $W \times H$ which stores using 2 bits a ternary symbol, $s \in \{0, 1, 2\}$, at each pixel position $(i_r, j_r), \forall i_r \in [1, H], j_r \in [1, W]$. In this work, we propose to partition **I** into $N_g = \frac{W}{h} \times \frac{H}{h}$ groups of pixels, where each group $G_{i_g j_g}, \forall i_g \in [1, \frac{H}{h}], j_g \in [1, \frac{W}{w}]$, of size $w \times h$ contains $N = wh$ ternary symbols. Firstly, each group of pixels $G_{i_g j_g}$ is vectorized as follows:

$$V^v_{i_g j_g} = [s_0\, s_1\, \ldots\, s_{N-1}]. \tag{1}$$

Next, $V^v_{i_g j_g}$ is then zero-padded up to the closest multiple-of-five length such that the zero-padded vector can be split into $N_t = \lceil \frac{N}{5} \rceil$ subgroups of five ternary symbols, see Figure 5. Let us denote the zero-padded group vector as $V^z_{i_g j_g}$, where

$$V^z_{i_g j_g} = [s_0\, s_1\, \ldots\, s_{5N_t-1}] = [[s_0\, s_1\, \ldots\, s_4]\, \ldots [s_{5k}\, s_{5k+1}\, \ldots\, s_{5k+4}]\, \ldots [s_{5N_t-5}\, s_{5N_t-4}\, \ldots\, s_{5N_t-1}]]. \tag{2}$$

Each subgroup k of five ternary symbols, $[s_{5k}\, s_{5k+1}\, \ldots\, s_{5k+4}]$, is then remapped into a symbol t in the alphabet $\{0, 1, \ldots, 3^5 - 1\}$ (represented using 8 bits), by employing the following symbol remapping formula:

$$t_k = \sum_{r=0}^{4} 3^{4-r} s_{5k+r}, k = 0, 1, \ldots, N_t. \tag{3}$$

Figure 5 shows that $V^z_{i_g j_g}$ is remapped as follows:

$$V_{i_g j_g} = [t_0\, t_1\, \ldots\, t_{N_t-1}]. \tag{4}$$

Furthermore, $V_{i_g j_g}$ can be further rearranged as a remapped unite volume of size $1 \times 1 \times N_t$.

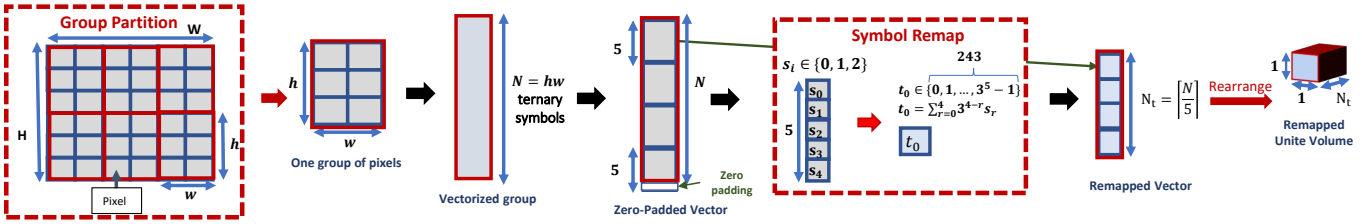


Figure 5. Proposed group partitioning and symbol remapping processes.

In conclusion, the proposed fixed-length representation first stores the input image $\mathbf{I}$ as a remapped volume $V(i_g, j_g, t_k), \forall i_g \in [1, \frac{H}{h}], j_g \in [1, \frac{W}{w}], k \in [0, N_t]$, of size $\frac{W}{w} \times \frac{H}{h} \times N_t$, as depicted in the middle part of Figure 4. One can note that the input image $\mathbf{I}$, having a raw image representation of $2WH$ bits, stored in memory using the proposed remapped volume representation, $\mathbf{V}$, using $8N_gN_t = 8\frac{WH}{wh}\lceil\frac{wh}{5}\rceil$, i.e., the input memory size is reduced with a percentage of

$$p_{mr} = 100 - \frac{400}{wh}\left\lceil\frac{wh}{5}\right\rceil (\%). \tag{5}$$

A maximum memory reduction of $p_{mr} = 20\%$ (or a compression ratio of 1.25, see Section 4) is obtained when the pixel-group size, N, is selected as a multiple-of-five number.

3.2. Proposed LUT-Based Representations

The proposed fixed-length representation is designed to encode each group $G_{i_gj_g}$ using an index value (represented on a specific number of bits) stored in the index matrix $\mathbf{M}$. The main idea is to find all unique combinations of N_t symbols found in any vector $V_{i_gj_g}$ and store them in memory using a novel LUT-based structure that continues to provide a fixed-length representation. Therefore, the remapped volume representation, $\mathbf{V}$, is represented using a fixed number of bits by the index matrix $\mathbf{M}$ and the LUT-based structure, see the right part of Figure 4, where $M(i_g, j_g)$ contains all the necessary information to extract $V_{i_gj_g}$ from the LUT-based structure. Several fixed-length representation solutions based on different levels of LUT indexing are proposed. A variable-length representation solution for storing the LUT-based structure is also studied.

Sections 3.2.1, 3.2.2 and 3.2.3 present the proposed fixed-length representation solutions based on a single-level LUT, double-level LUTs, and multi-level LUT structures, respectively. Section 3.2.4 presents the variable-length representation solution for compressing the LUT-based structure.

3.2.1. Single-Level LUT Solution

The proposed single-level LUT solution, called 1L-LUT, introduces a simple fixed-length representation by storing the N_{uc} unique combinations of N_t symbols using a single LUT table structure, which requires

$$\mathcal{L}_{LUT}^{1L} = 8N_tN_{uc} \text{ bits.} \tag{6}$$

In such case, $M(i_g, j_g)$ stores an index κ using $n_\kappa = \lceil log_2 N_{uc} \rceil$ bits, where $V_{i_gj_g}$ is extracted from LUT as $V_{i_gj_g} = LUT(\kappa)$. The index matrix $\mathbf{M}$ is represented using

$$\mathcal{L}_{M}^{1L} = N_gn_\kappa \text{ bits.} \tag{7}$$

The number of bits needed to represent $\mathbf{I}$ using the 1L-LUT solution is computed as follows:

$$\mathcal{L}^{1L} = \mathcal{L}_{M}^{1L} + \mathcal{L}_{LUT}^{1L} = N_gn_\kappa + 8N_tN_{uc} \text{ bits.} \tag{8}$$

Figure 6 depicts the proposed 1L-LUT solution. One can note that the value $\kappa - 1$ (instead of κ) is written in $M(i_g, j_g)$ using n_κ bits as $\kappa \in \{1, 2, \dots, N_{uc}\}$. For example,

if $N_{uc} = 4$, then only two bits are needed to represent the index $\kappa = 4$ instead of three ($4_{(10)} = 100_{(2)}$). $\mathcal{L}^{1L}$ depends on the group size, N, as follows:

(i) If N is too small, then $\mathbf{M}$ contains too much information, and N_{uc} is too large as too many unique combinations are stored by LUT;
(ii) If N is too large, then $\mathbf{M}$ contains too less information, and each line in LUT contains too much information as N_t is too large.

An optimal N^* provides the smallest memory representation for the 1L-LUT solution, however, its coding performance remains limited.

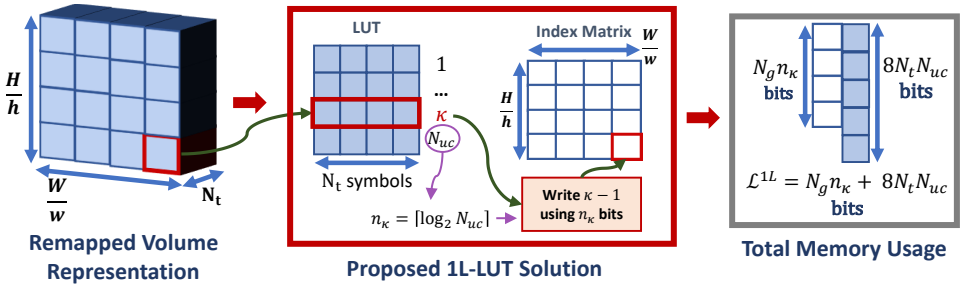


Figure 6. Proposed fixed-length single-level LUT (1L-LUT) solution. Green arrows show that the encoding process employs the following steps: *(e1)* each remapped group is searched in LUT to find index κ; *(e2)* κ is stored using n_κ bits in the index matrix. The decoding process simply follows these steps in the reverse order: *(d1)* read n_κ bits from the index matrix to decode κ; *(d2)* decode the remapped group as the κ-th line in LUT.

3.2.2. Double-Level LUT Solution

The proposed double-level LUT solution, called 2L-LUT, introduces a fixed-length representation where the N_{uc} unique combinations of N_t symbols are further divided into a set of N_t LUTs, $\{LUT_\ell\}_{\ell=1:N_t}$, to achieve an improve compression performance and faster search of the unique combinations. The main idea is that $V_{i_g j_g}$ is first classified into $N_{uc}+1$ classes by counting the number of non-zero values as ℓ. 2L-LUT propose to created N_t LUTs, where the ℓ-th LUT, denoted LUT_ℓ, stores N_{uc}^ℓ unique combinations of N_t symbols found in any vector $V_{i_g j_g}$. One can note that for $\ell = 0$, no LUT is needed to be stored in memory as a single deterministic case can be found: all values are zero. In such case, $M(i_g, j_g)$ stores the LUT index, ℓ, using n_ℓ bits, see Equation (9), and the index in the LUT, κ_ℓ, using n_κ^M bits, see Equation (10), i.e., using the maximum number of bits needed to represent the largest number of unique combinations in the set of LUTs. Hence, $M(i_g, j_g)$ stores ℓ using the first n_ℓ bits and κ_ℓ using the following n_κ^M bits. $V_{i_g j_g}$ is extracted from the ℓ-th LUT as $V_{i_g j_g} = LUT_\ell(\kappa_\ell)$. Therefore, $\mathbf{M}$ is stored by 2L-LUT using $\mathcal{L}_M^{2L}$ bits, see Equation (11).

$$n_\ell = \lceil log_2 N_t \rceil \tag{9}$$

$$n_\kappa^M = \lceil \log_2(\max\{N_{uc}^\ell\}_{\ell=1:N_t}) \rceil \tag{10}$$

$$\mathcal{L}_M^{2L} = N_g(n_\ell + n_\kappa^M) \tag{11}$$

The set of N_t LUTs, $\{LUT_\ell\}_{\ell=1:N_t}$, contains $N_{uc}^\ell = \sum_{\ell=1:N_t} N_{uc}^\ell$ entries. 2L-LUT makes use of the information that LUT_ℓ (the ℓ-th LUT) contains ℓ non-zero symbols by introducing a mask of N_t binary symbols, b_ℓ, where the ℓ-th mask symbol is set by checking if the corresponding symbol t_ℓ is or not symbol "0" in the alphabet, see Equation (12).

$$b_\ell = \begin{cases} 0 & \text{if } t_\ell = 0 \\ 1 & \text{if } t_\ell \neq 0 \end{cases}, \ \ell = 0, 1, \ldots, N_t - 1 \tag{12}$$

The ℓ non-zero symbols are stored using 8ℓ bits. Hence, 2L-LUT stores LUT_ℓ in memory using $\mathcal{L}^{2L}_{LUT_\ell}$ bits, see Equation (13), and $\mathbf{I}$ using $\mathcal{L}^{2L}$ bits, see Equation (14).

$$\mathcal{L}^{2L}_{LUT_\ell} = (N_t + 8\ell)N^\ell_{uc} \tag{13}$$

$$\mathcal{L}^{2L} = N_g(n_\ell + n^M_\kappa) + \sum_{\ell=1}^{N_t}(N_t + 8\ell)N^\ell_{uc} \tag{14}$$

Figure 7 depicts the proposed 2L-LUT solution. Note that n^M_κ can be computed only after populating the entire set of LUTs with unique combinations of N_t symbols. Therefore, the following bit-cleaner procedure, BitCleaner(n^M_κ), is employed to guarantee a single pass encoding of $\mathbf{I}$.

BitCleaner(n^M_κ) procedure:

(i) Compute $n_w = \lceil \log_2 N_g \rceil$ as $N^\ell_{uc} \leq N_g$, i.e., in the extreme case each group contains a unique combination of N_t symbols.

(e) Employ 2L-LUT and write each value $\kappa_\ell - 1$ using n_w bits instead of n^M_κ bits.

(b1) Compute n^M_κ using Equation (10) and $\{N^\ell_{uc}\}_{\ell=1:N_t}$.

(b2) Remove the unnecessary $n_w - n^M_\kappa$ bits, $n_\ell + n^M_\kappa + 1 : n_\ell + n_w$, from the representation of each index value $M(i_g, j_g)$.

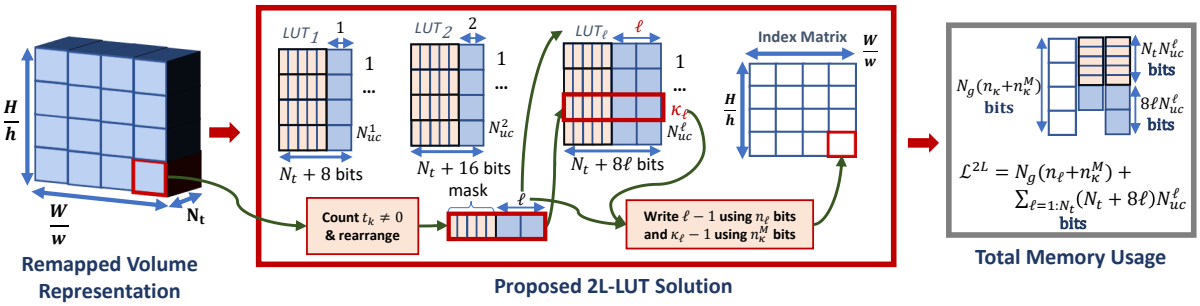


Figure 7. Proposed fixed-length double-level LUT (2L-LUT) solution. Green arrows show that the encoding process employs the following steps: *(i)* each remapped group is processed to find ℓ, the number of non-zero symbols; *(ii)* the rearranged vector is searched in LUT_ℓ to find κ_ℓ; *(iii)* ℓ and κ_ℓ are stored in the index matrix using n_ℓ and n^M_κ bits, respectively. The decoding process simply follows these steps in the reverse order: decode the two indices, read the rearranged vector at κ_ℓ-th line in LUT_ℓ, and generate the remapped group using the mask and ℓ non-zero symbols.

In the encoding algorithm, one can note that step *(i)* is employed in the initialization stage, step *(e)* in the encoding stage; and steps (b1)–(b2) in the final processing stage of the final bitstream, $\mathcal{B}_\mathcal{M}$, generated for encoding the index matrix.

A similar procedure was developed to post-process the bitstream and remove the unnecessary bits used in the signaling of ℓ for the large group size case, where not all LUTs are used in the proposed LUT-based representation, i.e., only the first few LUTs are used as, for example, LUT_{N_t} has the lowest probability to be used included in the LUT set. Therefore, the number of bits needed to represent ℓ is the number of bits needed to represent ℓ^*, the index of the last LUT in the set for which $N^{\ell^*}_{uc} \neq 0$. The bit-cleaner procedure, BitCleaner(n^*_ℓ), is employed.

BitCleaner(n^*_ℓ) procedure:

(i) Compute $n_\ell = \lceil \log_2 \frac{HW}{hw} \rceil$.

(e) Employ 2L-LUT and write each value $\ell - 1$ using n_ℓ bits.

(b1) Compute $n^*_\ell = \lceil \log_2(N^{\ell^*}_{uc}) \rceil$.

(b2) Remove the extra $n_\ell - n^*_\ell$ bits, $n^*_\ell + 1 : n_\ell$, from the representation of each $M(i_g, j_g)$.

Similarly, step (i) is employed in the initialization stage, (e) in the encoding stage; and steps (b1)–(b2) in the final processing stage of $\mathcal{B}_{\mathcal{M}}$.

3.2.3. Multi-Level LUT Solution

When the selected group size is very large, for example, for $N_t \geq 150$, the memory allocated to store the LUT-based structure of 2L-LUT contains a lot of redundancy. One can note that in such a case, the number of unique combinations in the set of LUTs is reduced, and the unique combinations are more populated with the symbol "0", the most frequent symbol in the alphabet. In this work, we propose to modify 2L-LUT for very large block sizes further and introduce the multi-level LUT solution called ML-LUT. ML-LUT introduces an extra LUT compared with 2L-LUT, called LUT mask and denoted LUT_M, to store a mask associated with each LUT in $\{LUT_\ell\}_{\ell=1:N_t}$, i.e., line ℓ in LUT_M is associated with LUT_ℓ. Therefore, the position k in line ℓ, $LUT_M(\ell,k)$, is set by checking if all the unique combinations stored by LUT_ℓ contain a symbol "0" on the k-th position, i.e., $LUT_\ell(q,k) = 0, \forall q = 1,2,\ldots,N_{uc}^\ell$, see Equation (15). Let us denote π as the number of lines in LUT_M, and $\mathcal{L}_{Mask}$ as the number of bits required to store LUT_M, computed using Equation (16).

$$LUT_M(\ell,k) = \begin{cases} 1 & \text{if } LUT_\ell(q,k) = 0, \forall q = 1,2,\ldots,N_{uc}^\ell \\ 0 & \text{otherwise} \end{cases}, \ \ell,k = 1,2,\ldots,N_t \tag{15}$$

$$\mathcal{L}_{Mask}^{ML} = N_t\pi \tag{16}$$

Figure 8 depicts the proposed ML-LUT solution. One can note that, by using LUT_M, only for the positions $LUT_M(\ell,k) = 0$, we can employ Equation (12) to generate the mask. Let us denote m_κ^ℓ the number of positions for which Equation (12) is employed. Since m_κ^ℓ depends on the structure of the found unique combination, a variable-length representation would be required to store in memory the LUT-based structure of ML-LUT efficiently. However, to guarantee a fixed-length representation, we propose to store each updated mask using a fixed-length representation of $m^\ell = min(\ell N_{uc}^\ell, N_t)$ bits. Hence, Equation (13) is updated as Equation (17), and Equation (14) is updated as Equation (18).

$$\mathcal{L}_{LUT_\ell}^{ML} = (m_\ell + 8\ell)N_{uc}^\ell \tag{17}$$

$$\mathcal{L}^{ML} = \mathcal{L}_M^{2L} + \mathcal{L}_{Mask}^{ML} + \sum_{\ell=1}^{N_t} \mathcal{L}_{LUT_\ell}^{ML} = N_g(n_\ell + n_\kappa^M) + N_t\pi + \sum_{\ell=1}^{N_t}(m^\ell + 8\ell)N_{uc}^\ell \tag{18}$$

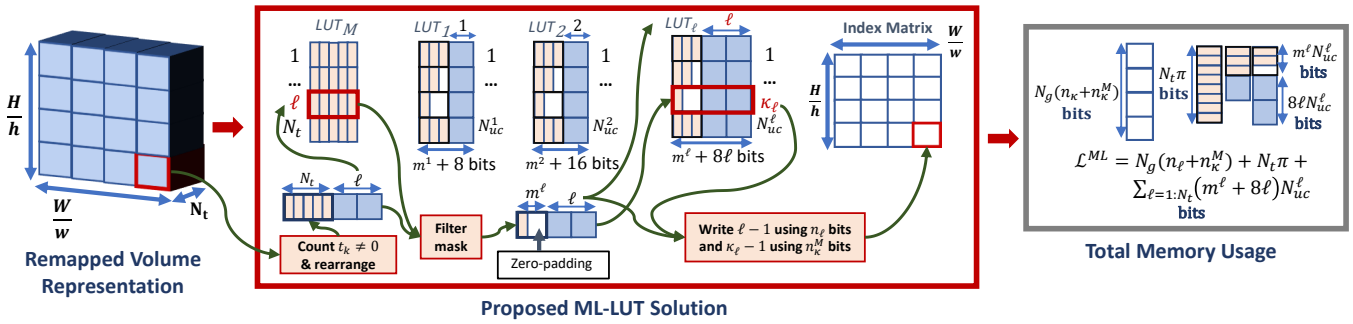


Figure 8. Proposed fixed-length multi-level LUT (ML-LUT) solution. Green arrows show that the encoding process employs the following steps: *(i)* each remapped group is processed to find ℓ, the number of non-zero symbols, and generated the rearranged vector (mask of N_t bits and ℓ non-zero symbols of 8 bits each); *(ii)* read from LUT_M the ℓ-th line and filter the rearranged vector to generate the filtered vector (filtered mask of m^ℓ bits and ℓ non-zero symbols); *(iii)* the filtered vector is searched in LUT_ℓ to find κ_ℓ; *(iv)* ℓ and κ_ℓ are stored in the index matrix using n_ℓ and n_κ^M bits, respectively. The decoding process simply follows these steps in the reverse order: decode the two indices, read the filtered vector at κ_ℓ-th line in LUT_ℓ, generate the rearranged vector using the ℓ-th line in LUT_M, and finally generate the remapped group using the mask and ℓ non-zero symbols.

3.2.4. Variable-Length Solution

In this work, we also studied the possibility of relaxing the constraint of fixed-length representation for encoding the LUT-based structure and introducing a variable-length multi-level LUT solution called MLv-LUT, see Figure 9. MLv-LUT simply modifies the fixed-length representation version, ML-LUT, by introducing two variable-length strategies for storing each LUT_ℓ without where the zero-padding procedure is removed from the generation of the filtered mask and the last non-zero symbol t_{N_t-1}.

Firstly, instead of storing each filtered mask using a fixed number of bits, m_κ, we propose to store each filtered mask using the exact number of bits, m_κ^ℓ, and avoid the zero-padding process. The cost of storing the filtered mask of LUT_ℓ is computed by (19).

$$\mathcal{L}_{LUT_\ell}^{MLv} = \sum_{\kappa=1}^{N_{uc}^\ell} \left(m_\kappa^\ell + 8\ell N_{uc}^\ell \right) \tag{19}$$

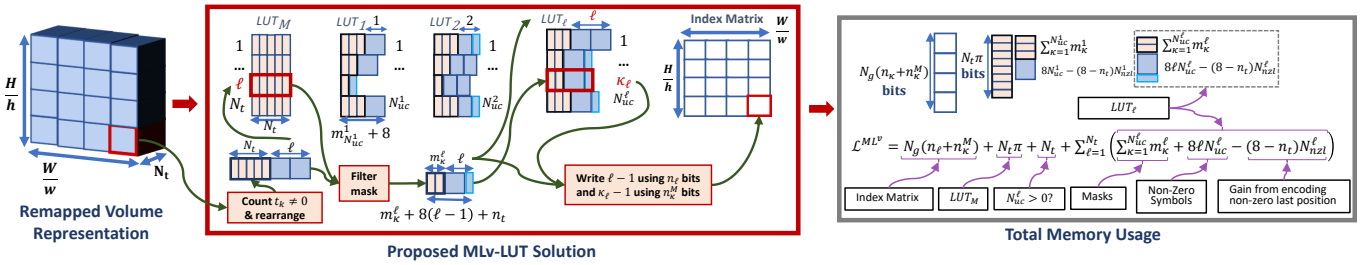


Figure 9. Proposed variable-length multi-level LUT (MLv-LUT) solution. Green arrows show that the same encoding process is employed as ML-LUT. Only a simple modification is introduced: instead of a fixed-length representation, the filtered vector has a variable-length representation where the filtered mask is stored using m_κ^ℓ bits, the first $\ell - 1$ non-zero symbols are represented on 8 bits each, and the ℓ-th non-zero symbol is represented either using n_t bits if it placed on the last position in the remapped vector or using 8 bits otherwise. The decoding process follows similar steps as ML-LUT.

Secondly, the last non-zero symbol t_{N_t-1} can be efficiently represented using less then 8 bits by removing the zero-padding process of $V_{i_g j_g}^v$. In such case, the last group of ternary symbols in $V_{i_g j_g}^v$ is $[s_{5N_t-5} \dots s_N]$ and has a length of r_l symbols, where r_l is computed by Equation (20). We can now update Equation (3) as Equation (21) for remapping t_{N_t-1}. If $r_l < 5$, then $t_{N_t-1} \in \{0, 1, \dots, 3^{r_l} - 1\}$ and, instead of 8 bits, only n_t bits are needed to store t_{N_t-1}, where n_t is computed using Equation (22). Let us denote N_{nzl}^ℓ, the number of unique combinations in LUT_ℓ which contain a non-zero symbol on the last position. The gain obtained by employing such a strategy is computed by (23). Due to the variable-length representation of the LUT-based structure, an additional cost of $\mathcal{L}_{vl}^{MLv}$ bits, computed by (24), is required to signal which LUTs are included in the LUT set, i.e., for which $N_{uc}^\ell > 0$.

$$r_l = N - 5(N_t - 1) \tag{20}$$

$$t_{N_t-1} = \sum_{r=0}^{r_l} 3^{r_l - r} s_{5k+r} \tag{21}$$

$$n_t = \lceil \log_2 3^{r_l} \rceil \tag{22}$$

$$\mathcal{L}_{gain}^{MLv} = (8 - n_t) N_{nzl}^\ell \tag{23}$$

$$\mathcal{L}_{vl}^{MLv} = N_t \tag{24}$$

Finally, MLv-LUT stores in memory $\mathbf{I}$ using $\mathcal{L}^{MLv}$ bits, where $\mathcal{L}^{MLv}$ is computed by (25) using (11), (16), (19), (23) and (24).

$$\begin{aligned}\mathcal{L}^{MLv} &= \mathcal{L}_M^{2L} + \mathcal{L}_{Mask}^{ML} + \mathcal{L}_{vl}^{MLv} + \sum_{\ell=1}^{N_t}\left(\mathcal{L}_{LUT_\ell}^{MLv} - \mathcal{L}_{gain}^{MLv}\right) \\ &= N_g(n_\ell + n_\kappa^M) + N_t\pi + N_t + \sum_{\ell=1}^{N_t}\left(\sum_{\kappa=1}^{N_{uc}^\ell} m_\kappa^\ell + 8\ell N_{uc}^\ell - (8-n_t)N_{nzl}^\ell\right)\end{aligned} \tag{25}$$

3.3. Algorithmic Details

The *encoder implementation* of the proposed 2L-LUT/ML-LUT fixed-length representation is presented in Algorithm 1. The threshold $\mathcal{T}$ is used to select the appropriate solution based on the group size information. One can note that the index matrix is initially encoded as bitstream $\mathcal{B}_\mathcal{M}$ using steps 5–22 in Algorithm 1. The proposed LUT-based representation is encoded as bitstream $\mathcal{B}_{\mathcal{LUT}}$ using steps 23–39 in Algorithm 1. Bitstream $\mathcal{B}_\mathcal{I}$ is finally updated using steps 40–41 in Algorithm 1 by removing the unnecessary bits based on the information provided by the proposed LUT-based representation. The output bitstream, $\mathcal{B}$, concatenates $\mathcal{B}_{\mathcal{LUT}}$ and $\mathcal{B}_\mathcal{M}$ in this order.

Algorithm 1: Encode using ML-LUT

Data: **I** of size $W \times H$ containing symbols in alphabet $\mathcal{A}_3 = \{0\,(\text{non-event}), 1\,(\text{positive}), 2\,(\text{negative})\}$; $w \times h$, group size; $\mathcal{T}$, group size threshold;
Result: $\mathcal{B}$, output bitstream;

1 **if** **I** *is full of non-event symbols* **then** // Encode an empty **I** using 8 bits.
2 **Return** $\mathcal{B} \leftarrow$ [Write 8 bits of 0]; // Bits needed to represent N_{uc}.
3 **Compute:** $N_t = \lceil \frac{hw}{5} \rceil$, $n_w = \lceil \log_2 \frac{HW}{hw} \rceil$, $n_\ell = \lceil \log_2 N_t \rceil$;
4 **Initialize:** $\{N_{uc}^\ell\}_{\ell=1:N_t} \leftarrow 0$; $\mathcal{B}_\mathcal{M} \leftarrow []$; $\mathcal{B}_{\mathcal{LUT}} \leftarrow []$;
5 **for** $j_g = 1, 2, \ldots, \frac{W}{w}$ **do** // Encode **I** into $\mathcal{B}_\mathcal{M}$.
6 **for** $i_g = 1, 2, \ldots, \frac{H}{h}$ **do**
7 $G_{i_g j_g} = vec(I(h(i_g - 1) + 1 : hi_g, w(j_g - 1) : wj_g))$; // Current group of pixels.
8 $V_{i_g j_g} \leftarrow$ **Remap** $G_{i_g j_g}$ using (1)–(4); // Symbol remapping, see Section 3.1.
9 $\ell \leftarrow$ **Count** $t_k \neq 0$, $k = 1, 2, \ldots, N_t$; // Find ℓ, the LUT index.
10 **if** $\ell = 0$ **then**
11 $\mathcal{B}_I \leftarrow$ [Write $n_\ell + n_w$ bits of 0]; // All-zero groups are encoded as $n_\ell + n_w$ bits of 0.
12 **else**
13 **if** $N_{uc}^\ell = 0$ **then** // Add 1^{st} line to LUT_ℓ.
14 $N_{uc}^\ell \leftarrow 1$; $LUT_\ell(N_{uc}^\ell) \leftarrow V_{i_g j_g}$; $eIndex = 1$;
15 **else**
16 **if** $V_{i_g j_g}$ *is already in* LUT_ℓ *at position* κ_ℓ **then** // Find κ_ℓ in LUT_ℓ.
17 $eIndex = \kappa_\ell$
18 **else** // Add a new line to LUT_ℓ.
19 $N_{uc}^\ell \leftarrow N_{uc}^\ell + 1$; $LUT_\ell(N_{uc}^\ell) \leftarrow V_{i_g j_g}$; $eIndex = N_{uc}^\ell$;
20 $\mathcal{B}_\mathcal{M} \leftarrow$ [Write $\ell - 1$ on n_ℓ bits; Write $eIndex$ on n_w bits]; // Write $M(i_g, j_g)$ in $\mathcal{B}_\mathcal{M}$.
21 **end**
22 **end**
23 **for** $\ell = 1, 2, \ldots, N_t$ **do** // Encode the LUT-based representation into $\mathcal{B}_{\mathcal{LUT}}$.
24 **Compute** $n_\kappa(\ell) = \lceil \log_2 N_{uc}^\ell \rceil$;
25 **if** $N_t < \mathcal{T}$ **then**
26 **Employ** 2L-LUT, see Section 3.2.2; // Use the 2L-LUT solution.
27 $\mathcal{B}_{\mathcal{LUT}} \leftarrow [[b_0, b_1, \ldots, b_{N_t - 1}]$ using (12)$]$; // Mask information.
28 $\mathcal{B}_{\mathcal{LUT}} \leftarrow$ [Write ℓ non-zero symbols on 8ℓ bits]; // Non-zero symbol information.
29 **else**
30 **Employ** ML-LUT, see Section 3.2.3; // Use the ML-LUT solution.
31 **if** $N_{uc}^\ell > 0$ **then**
32 $\mathcal{B}_{\mathcal{LUT}} \leftarrow [1;\ LUT_M(\ell)$ on N_t bits; N_{uc}^ℓ on n_w bits]; // LUT_M information.
33 $\mathcal{B}_{\mathcal{LUT}} \leftarrow [[b_0, b_1, \ldots, b_{N_t - 1}]$ using (12)$]$; // Mask information.
34 $\mathcal{B}_{\mathcal{LUT}} \leftarrow$ [Write ℓ non-zero symbols on 8ℓ bits]; // Non-zero symbol information.
35 **else**
36 $\mathcal{B}_{\mathcal{LUT}} \leftarrow [0]$; // Signal $N_{uc}^\ell = 0$.
37 **end**

Algorithm 1: *Cont.*

38 **Compute:** $N_{uc} = \sum_{\ell=1:N_t} N_{uc}^{\ell}$; $n_\kappa = \lceil log_2 N_{uc} \rceil$;
39 $\mathcal{B}_{LUT} \leftarrow$ [Write n_κ on 8 bits; Write N_{uc} on n_κ bits; $\mathcal{B}_{LUT}$]; // All info to decode $\mathcal{B}_{\mathcal{LUT}}$.
40 $\mathcal{B}_{\mathcal{LUT}} \leftarrow$*BitCleaner*($n_\ell^*, \mathcal{B}_{\mathcal{LUT}}$).(b1)–(b2); // Remove unnecessary bits in the representation of ℓ.
41 $\mathcal{B}_{\mathcal{LUT}} \leftarrow$*BitCleaner*($n_\kappa^M, \mathcal{B}_{\mathcal{LUT}}$).(b1)–(b2); // Remove unnecessary bits to avoid a 2^{nd} image pass.
42 **Return** $\mathcal{B} \leftarrow [\mathcal{B}_{\mathcal{LUT}}; \mathcal{B}_{\mathcal{M}}]$; // $\mathcal{B}$ concatenates $\mathcal{B}_{\mathcal{LUT}}$ and $\mathcal{B}_{\mathcal{M}}$.

The pseudo-code provided by Algorithm 1 is used to encode the input image and store the data on the disk, while (14) and (18) compute the total memory needed to store in memory all the necessary information.

The *decoder implementation* of the proposed 2L-LUT/ML-LUT fixed-length representation is presented in Algorithm 2. Similarly, the threshold $\mathcal{T}$ is used to select the appropriate solution based on the group size information. The proposed LUT-based representation is first decoded from the first part of $\mathcal{B}$ using steps 7–24 in Algorithm 2. The index matrix, **M**, is decoded from the remaining part of $\mathcal{B}$ using steps 26–29 in Algorithm 2, i.e., ℓ and κ_ℓ are decode on steps 28 and 29 in Algorithm 2, respectively. Note that the inverse remap process consists in first representing each symbol t_k in $V_{i_g j_g}$ in base 3 to generate five ternary symbols and then selecting $G_{i_g j_g}$ with the first N_t ternary symbols. Finally, **I** is generated using the proposed LUT-based representation and **M** using steps 30–36 in Algorithm 2.

Algorithm 1 and Algorithm 2 are used to introduce the encoder and decoder of the proposed fixed-length ML-LUT method, respectively. In the case of the proposed variable-length MLv-LUT method, the two algorithms may be further modified using the algorithmic description presented in Section 3.2.4, i.e., by modifying steps 33–34 in Algorithm 1 and steps 21–22 in Algorithm 2.

The size of $\mathcal{B}$, i.e., the compressed file size, is usually smaller than the total memory usage size computed by $\mathcal{L}^{2L}$ or $\mathcal{L}^{ML}$. In this work, $\mathcal{L}^{1L}$, $\mathcal{L}^{2L}$, and $\mathcal{L}^{ML}$ are computed to report the total memory usage of each proposed fixed-length method, while $\mathcal{L}^{MLv}$ is computed to report the optimal storage size of the proposed framework.

Algorithm 2: Decode using ML-LUT

Data: $\mathcal{B}$, input bitstream; $W \times H$, size of **I**; $w \times h$, group size; $\mathcal{T}$, group size threshold;
Result: **I** containing symbols in alphabet $\mathcal{A}_3 = \{0$ (non-event), 1 (positive event), 2 (negative event)$\}$;
1 **Initialize:** $\mathbf{I} \leftarrow \mathbf{0}_{H,W}$; // Fill the output with non-event symbols.
2 **Decode:** $n_\kappa \leftarrow$ *Bin2Dec*(first 8 bits in $\mathcal{B}$); // $\mathcal{B}(1:8)$ stores the binary representation of n_κ.
3 **if** $n_\kappa = 0$ **then** // No other information is encoded.
4 Return **I**; // **I** is full of non-event symbols.
5 **Decode:** $N_{uc} \leftarrow$ *Bin2Dec*(next n_κ bits in $\mathcal{B}$); // Decode N_{uc}.
6 **Compute:** $N_t = \lceil \frac{hw}{5} \rceil$, $n_w = \lceil \log_2 \frac{HW}{hw} \rceil$, $n_\ell = \lceil \log_2 N_t \rceil$;
7 **if** $N_t < \mathcal{T}$ **then** // 2L-LUT Solution.
8 **Decode:** $Mask_{LUTs}^{2L} \leftarrow$ Reshape(next $N_{uc}N_t$ bits in $\mathcal{B}$ as $N_{uc} \times N_t$); // Decode all mask info.
9 **Compute:** $\{N_{uc}^{\ell}\}_{\ell=1:N_t}$ by counting the non-zero positions in each line of $Mask_{LUTs}^{2L}$;
10 **Split** $Mask_{LUTs}^{2L}$ into $\{Mask_{LUT_\ell}^{2L}\}_{\ell=1:N_t}$ using $\{N_{uc}^{\ell}\}_{\ell=1:N_t}$; // Decode each LUT_ℓ mask info.
11 **for** $\ell = 1 : N_t$ **do** // Process each LUT_ℓ
12 **Decode** $N_{uc}^{\ell}\ell$ symbols $\leftarrow$ Bin2Dec(next $8N_{uc}^{\ell}\ell$ bits in $\mathcal{B}$ as $N_{uc}^{\ell}\ell$ symbols); // 8 bits/symbol.
13 **Set** LUT_ℓ using $Mask_{LUT_\ell}^{2L}$ and the set of $N_{uc}^{\ell}\ell$ non-zero symbols;
14 **Compute** $n_\kappa(\ell) = \lceil \log_2 N_{uc}^{\ell} \rceil$;
15 **end**
16 **else** // ML-LUT Solution.
17 $d^\ell \leftarrow$ *Bin2Dec*(next bit in $\mathcal{B}$); // Decode the decision regarding $N_{uc}^{\ell} > 0$.
18 **if** $d^\ell = 1$ **then** // An LUT having at least on line, i.e., $N_{uc}^{\ell} > 0$
19 **Decode** $LUT_M(LUT_\ell) \leftarrow$ Reshape(next N_t bits in $\mathcal{B}$); // Decode one line in LUT_M.
20 **Decode** $N_{uc}^{\ell} \leftarrow$ Bin2Dec(next n_w bits in $\mathcal{B}$); // Decode N_{uc}^{ℓ}.
21 **Decode** $Mask_{LUT_\ell}^{ML} \leftarrow$ Reshape(next $N_{uc}^{\ell}\ell$ bits in $\mathcal{B}$ as $N_{uc}^{\ell} \times \ell$); // Decode $Mask_{LUT_\ell}$.
22 **Decode** $N_{uc}^{\ell}\ell$ symbols $\leftarrow$ Bin2Dec(next $8N_{uc}^{\ell}\ell$ bits in $\mathcal{B}$ as $N_{uc}^{\ell}\ell$ symbols); // 8 bits/symbol.
23 **Set** LUT_ℓ using $Mask_{LUT_\ell}^{ML}$ and the set of $N_{uc}^{\ell}\ell$ non-zero symbols;
24 **Compute** $n_\kappa(\ell) = \lceil \log_2 N_{uc}^{\ell} \rceil$;

Algorithm 2: *Cont.*

Compute: n_{κ}^{M} using (10) and $n_{\ell}^{*} = \lceil \log_2(N_{uc}^{\ell^*}) \rceil$; // Find the representation of ℓ and κ_{ℓ}.
for $j_g = 1, 2, \ldots, \frac{W}{w}$ **do** // Decode each $M(i_g, j_g)$.
  for $i_g = 1, 2, \ldots, \frac{H}{h}$ **do**
    Decode $\ell \leftarrow$ Bin2Dec(next n_{ℓ}^{*} bits in $\mathcal{B}$); // Decode ℓ, the LUT index.
    Decode $\kappa_{\ell} \leftarrow$ Bin2Dec(next n_{κ}^{M} bits in $\mathcal{B}$); // Decode κ_{ℓ}, line in LUT_{ℓ} containing $V_{i_g j_g}$.
    if $\ell > 0$ **then** // No action is needed if $\ell = 0$ (all-zero group), see step 1 above.
      if $N_t < \mathcal{T}$ **then** // 2L-LUT Solution.
        Set $V_{i_g j_h} \leftarrow LUT_{\ell}(\kappa_{\ell})$; // Use group mask and ℓ non-zero symbols to generate $V_{i_g j_g}$.
      else // ML-LUT Solution.
        Set $V_{i_g j_h} \leftarrow$ GenerateGroup($LUT_{\ell}(\kappa_{\ell}), LUT_M(\ell)$); // Use $LUT_M(\ell)$ and filtered mask to generate group mask. Use group mask and ℓ symbols to generate $V_{i_g j_g}$.
      $G_{i_g j_h} \leftarrow$ Remap $V_{i_g j_g}$ to generate $G_{i_g j_g}$ using the inverse process of (1)–(4);
      $I(h(i_g - 1) + 1 : hi_g, w(j_g - 1) : wj_g) \leftarrow$ Reshape($G_{i_g j_g}, h \times w$); // Reshape group.
  end
end
Return I;

4. Experimental Evaluation

4.1. Experimental Setup

In our work, the experimental evaluation is carried out on the large-scale outdoor stereo event camera datasets [21], called DSEC, containing a set of 82 asynchronous event sequences captured for network training using the Prophesee Gen3.1 event sensor placed on top of a moving car, having a $W \times H = 640 \times 480$ resolution.

In this work, the total memory usage of the three proposed fixed-length representations, 1L-LUT, 2L-LUT, and ML-LUT, and the compressed file size of the proposed variable-length representation, MLv-LUT, are reported. All proposed methods are implemented in the MATLAB programming language. One can note that both 2L-LUT and ML-LUT results are reported as ML-LUT, where the corresponding method is selected based on the group size, N. An EF sequence is generated using the sum-accumulation process for each asynchronous event sequence and each time-window size Δ. Four frame rates are studied for generating the EFs: (i) $\Delta = 5555$ µs (180 fps); (ii) $\Delta = 1000$ µs (10^3 fps); (iii) $\Delta = 100$ µs (10^4 fps); and (iv) $\Delta = 1$ µs (10^6 fps), i.e., all acquired asynchronous events are collected by EFs as all events having the same timestamp are collected by one EF. The proposed methods are designed to store-on-disk/represent-in-memory each EF separately. The performance of the proposed framework is studied using a large variety of group sizes. The group configuration provides RA to blocks of pixels having the $w \times h$ block size set as follows: (g_1) 8×4; (g_2) 16×4; (g_3) 8×8; (g_4) 16×8, (g_5) 64×4, (g_6) 16×16, (g_7) 32×32, and (g_8) 64×32. More exactly, ML-LUT selects the 2L-LUT method for (g_1)–(g_6) and the ML-LUT method for (g_7) and (g_8).

The performance of the proposed lossless coding framework is compared with the following state-of-the-art methods, which encode lossless intensity-like event representation that remaps a sequence of five EFs (see [18]):

(1) The HEVC standard [17] using the *FFmpeg* implementation [39] which was employed using the following command:

ffmpeg -f rawvideo -vcodec rawvideo -s W x H -r 30 -pix_fmt gray -i in.yuv
-c:v libx265 -x265-params lossless=1 out.mp4 (26)

(2) The Versatile Video Codec (VVC) standard [36] using the VVC Test Model [40] implementation which was employed using the following command:

EncoderAPP -c encoder_intra_vtm.cfg -c lossless.cfg –FrameRate=30 -wdt W -hgt H
–InputChromaFormat=400 –FramesToBeEncoded=1 –InputBitDepth=8
–OutputBitDepth=8 –BDPCM=1 -i in.yuv -b out.vvc -o out.yuv (27)

(3) The Context-based, Adaptive, Lossless Image Codec (CALIC) lossless image codec [37];

(4) The Free Lossless Image Format (FLIF) lossless image codec [38]; and with our performance-oriented prior work Lossless Compression of Event Camera Frames (LCECF) [18], where eight EFs are represented as a pair of an event map image (EMI) and a concatenated polarity vector.

The *Raw data* size is computed using 2 bits per pixel, i.e., each pixel symbol is stored using 2 bits; therefore, the size of one EF is $2HW$ bits. Moreover, a similar procedure as in [18,19] is employed: the EF sequences generated for only the first 20 s, 2 s, and 20 ms are extracted from each asynchronous event sequence for $\Delta = 1000$ µs, $\Delta = 100$ µs, and $\Delta = 1$ µs, respectively, i.e., a maxim of 20,000 EFs are generated for each sequence.

The coding performance of all codecs is measured using the compression ratio (CR) metric defined as follows:

$$CR = \frac{\text{Raw data } size}{\text{Compressed file } size}. \tag{28}$$

For ML-LUT, Equation (28) is computed using the size of output bitstream, $\mathcal{B}$, generated by Algorithm 1. See Section 4.2 for the lossless compression results over DSEC.

The memory usage performance of the proposed fixed-length codecs is measured using the compression ratio—memory usage (CR_{MU}) metric defined as follows:

$$CR_{MU} = \frac{\text{Raw data } size}{\text{Total memory usage } size}. \tag{29}$$

More exactly, Equation (29) is computed using (8), (14), (18) and (25), for the proposed fixed-length representation solutions, 1L-LUT, 2L-LUT, ML-LUT, and MLv-LUT, respectively. See Section 4.3 for the memory usage results over DSEC for the proposed fixed-length representations. See Section 4.4 for the average runtime results per EF over DSEC.

4.2. Lossless Compression Results

Figure 10 shows the lossless compression results over DSEC for four different values of the time window of the spatiotemporal neighborhood. One can note that on one hand, when Δ is very small ($\Delta = 1$ µs), see Figure 10a, a larger block size provides improved performance as the EF is sparsely populated with event frames as a small amount of information is stored in each EF, while a small block size provides a limited improvident as a few bits are still necessary to signal empty group (i.e., full of non-event symbols). While on the other hand, when Δ is much larger ($\Delta = 5555$ µs), see Figure 10d, the largest block size might not provide the best performance as the EF size is limited to $W \times H$, and the cost of encoding ℓ might become too high as too many LUTs are generated with only a few unique combinations. Figure 10 shows that, in general, the 32×32 group size provides the best results for any of the selected time-window sizes. Therefore, the ML-LUT 32×32 solution was selected as the anchor fixed-length representation, and the DSEC sequences are sorted in ascending order of the ML-LUT 32×32 result.

Table 1 shows the average compression results over DSEC, where the bold font marks the best results. One can note that ML-LUT 32×32 is able to provide a CR performance of at least 3.15 at $\Delta = 5555$ µs and up to 310.13 at $\Delta = 1$ µs. If the chip memory requirements are stricter, then a small group size can be used, however, the CR performance decreases by up to 95.15% at $\Delta = 1$ µs. One can note that the proposed method is able to provide a minimum CR performance of at least 2, i.e., the ML-LUT can offer the sensor to store in memory at least two compressed EF and apply other pre-processing methods (using a block-processing approach) compared with a single EF using the raw representation. The results also show that a larger group size might not provide improved performance as a large group size and the possible number of unique combinations increases too much.

Figure 11 shows the lossless compression results over DSEC for comparison with state-of-the-art methods, using different values of Δ. One can note that when Δ is very small ($\Delta = 1$ µs), see Figure 11a, the performance of the proposed ML-LUT 32×32 method

is worse than the state-of-the-art methods as the fixed-length representation constraint is must be satisfied by still using a few bits to signal empty groups. This small extra cost per group adds up to a high extra cost compared with state-of-the-art methods; however, ML-LUT 32×32 provides an average CR compared with the raw data representation. While on the other hand, when Δ is much larger, see Figure 11d, the proposed fixed-length representation ML-LUT 32×32 provides competitive results compared with the state-of-the-art variable-length representations, especially to the video coding standards. One can note that ML-LUT must work under powerful constraints: must provide a fixed-length representation, i.e., for each group of pixels, we must allocate an equal number of bits, although maybe some groups of pixels can be encoded using a much less bitrate; must have a low-complexity so it is suitable for integration into VLP event-processing chips, i.e., it can be written on hardware (e.g., the use of basic coding concepts such as arithmetic coding or adaptive Markov modeling are too complex); and must provide an efficient memory representation, i.e., the coding gains must be large enough to justify the introduction of the proposed coding method in the sensor.

Table 2 shows the average compression results over DSEC for comparison with state-of-the-art, where the bold font marks the best results. One can note that the proposed ML-LUT 32×32 solution, although offering a fixed-length representation, is able to provide a close CR performance compared with variable-length video coding standards and variable-length state-of-the-art image codecs for $\Delta = 1000$ µs and $\Delta = 5555$ µs, i.e., at frequencies below 1 KHz. When Δ is too small, the fixed-length representation feature of the proposed framework affects the CR performance.

Table 1. Lossless compression results over DSEC for ML-LUT using different group sizes.

Method		**ML-LUT 8 × 4**	**ML-LUT 16 × 4**	**ML-LUT 8 × 8**	**ML-LUT 16 × 8**	**ML-LUT 64 × 4**	**ML-LUT 16 × 16**	**ML-LUT 32 × 32**	**ML-LUT 64 × 32**
$\Delta = 1$ µs	CR	15.04	29.64	29.67	58.32	113.33	111.17	310.13	**312.98**
	Improv.	−95.15%	−90.44%	−90.43%	−81.20%	−63.46%	−64.15%	–	0.92%
$\Delta = 100$ µs	CR	5.97	9.92	9.89	13.92	15.83	16.07	27.18	**27.67**
	Improv.	−78.03%	−63.52%	−63.61%	−48.80%	−41.76%	−40.88%	–	1.79%
$\Delta = 1000$ µs	CR	3.30	4.45	4.46	5.22	5.47	5.58	**6.84**	6.43
	Improv.	−51.76%	−34.96%	−34.84%	−23.59%	−20.04%	−18.37%	–	−5.95%
$\Delta = 5555$ µs	CR	2.17	2.62	2.60	2.88	3.03	3.03	**3.15**	3.00
	Improv.	−31.12%	−16.69%	−17.39%	−8.36%	−3.81%	−3.62%	–	−4.51%

Table 2. Lossless compression results over DSEC for comparison with state-of-the-art.

Method		**HEVC [17]**	**VVC [36]**	**CALIC [37]**	**FLIF [38]**	**LCECF [18]**	**ML-LUT 32 × 32**
Representation		Variable-Length	Variable-Length	Variable-Length	Variable-Length	Variable-Length	Fixed-Length
Chip Integration		SoC	SoC	SoC	SoC	SoC	VLP/ESP
$\Delta = 1$ µs	CR	1715.40	1507.25	2309.98	2775.60	**5533.72**	310.13
	Improv.	453.12%	386.01%	644.84%	794.98%	1684.32%	–
$\Delta = 100$ µs	CR	46.21	50.17	45.12	67.52	**80.10**	27.18
	Improv.	70.01%	84.58%	66.00%	148.42%	194.70%	–
$\Delta = 1000$ µs	CR	7.26	7.84	7.21	10.25	**12.87**	6.84
	Improv.	6.14%	14.62%	5.41%	49.85%	88.16%	–
$\Delta = 5555$ µs	CR	3.24	3.49	3.51	4.40	**5.15**	3.15
	Improv.	2.86%	10.79%	11.43%	39.68%	63.49%	–

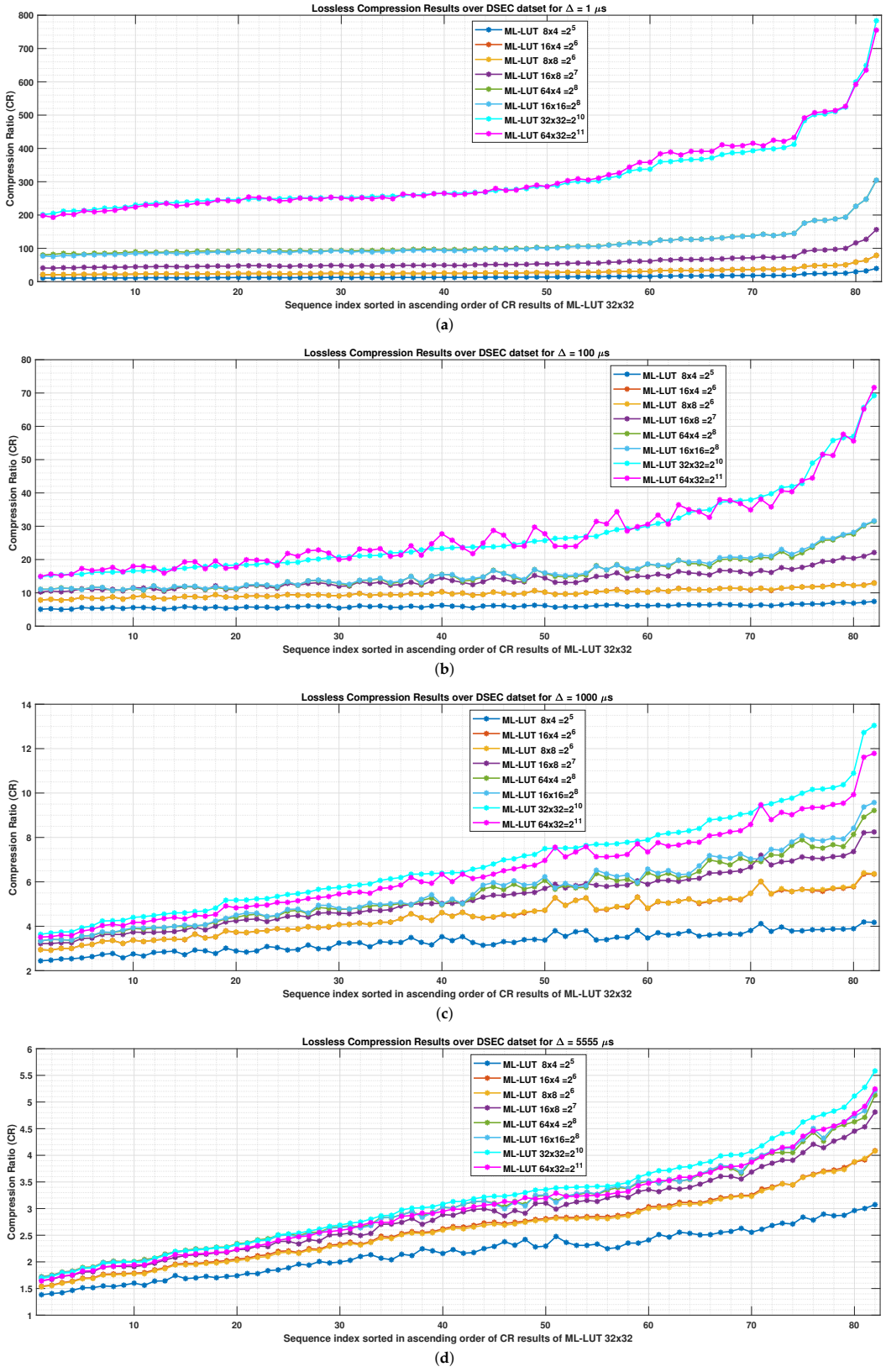


Figure 10. Lossless compression results over DSEC, measured using Compression Ratio (CR), for ML-LUT using different group sizes when encoding EF sequences generated using a time-window of (**a**) $\Delta = 1$ µs; (**b**) $\Delta = 100$ µs; (**c**) $\Delta = 1000$ µs; and (**d**) $\Delta = 5555$ µs. The DSEC sequence is ordered in ascending order of the CR results of ML-LUT for a group size of $w \times h = 32 \times 32$.

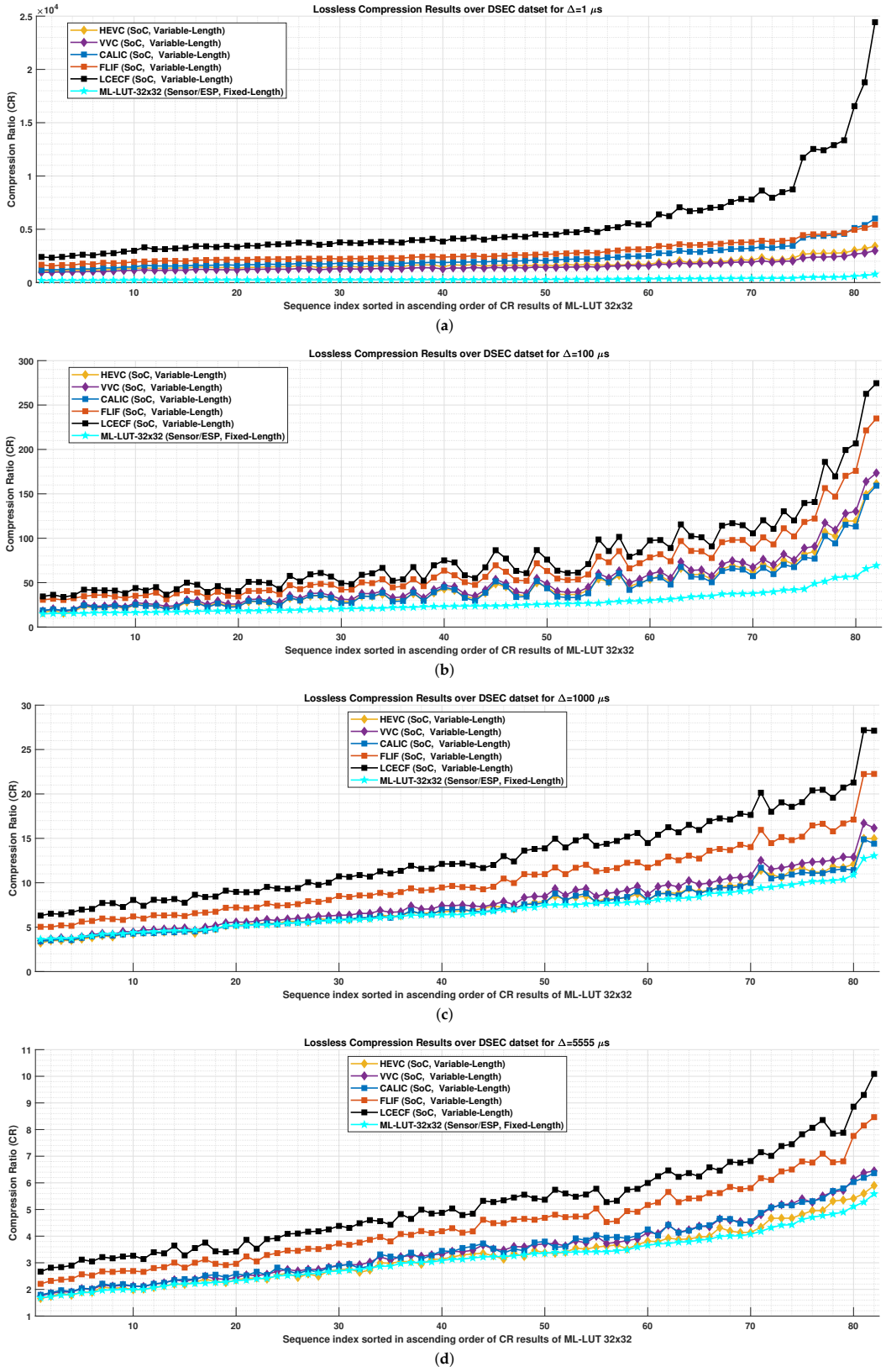


Figure 11. Lossless compression results over DSEC, measured using Compression Ratio (CR), for state-of-the-art methods and ML-LUT 32×32, when encoding EF sequences generated using a time-window of (**a**) $\Delta = 1$ µs; (**b**) $\Delta = 100$ µs; (**c**) $\Delta = 1000$ µs; and (**d**) $\Delta = 5555$ µs. The DSEC sequence is ordered in ascending order of the CR results of ML-LUT for a group size of $w \times h = 32 \times 32$.

4.3. Memory Usage Results

Figure 12 shows the memory usage results over DSEC for 1L-LUT using different group sizes. One can note that for each Δ, a different group size provides the best performance. A large group size is affecting the overall performance as too much information is stored by the LUT-based structure. However, the proposed 1L-LUT solution has the lowest complexity, and the provided performance might be enough for some types of applications.

Figure 13 shows the memory usage results for ML-LUT and MLv-LUT using different group sizes. Note that the 32×32 group size usually provides the best performance. For $\Delta = 1$ µs, a larger group size provides a much better performance than a smaller group size. Moreover, variable-length representation solutions provide only a small improvement compared with the corresponding fixed-length representation solutions.

Table 3 shows the average compression results for ML-LUT and MLv-LUT, where the bold font marks the best results. Note that the memory usage results are similar to the lossless compression results; see Table 1. The variable-length representation solutions always provide improved performance compared with the corresponding fixed-length representation solutions.

4.4. Runtime Results

Figure 14 shows the encoding runtime results. Table 4 shows the average compression results for ML-LUT, where a larger group size always provides a smaller runtime. The hardware implementation of the proposed method can provide a much smaller runtime as the proposed framework is designed to employ only low-complexity coding techniques and a reduced number of operations, compared with state-of-the-art methods which are developed to compute more complex structures and employ complex coding techniques. To be able to hardware write such complex methods, several other optimization techniques must be applied while accepting a drop in the coding performance and paying a high price for developing such a specialized chip. Therefore, here, these complex methods are evaluated as unsuitable for integration into VLP chips.

Table 3. Memory usage results over DSEC for ML-LUT and MLv-LUT using different group sizes.

Method		**ML-LUT 8 × 4**	**ML-LUT 16 × 4**	**ML-LUT 8 × 8**	**ML-LUT 16 × 8**	**ML-LUT 64 × 4**	**ML-LUT 16 × 16**	**ML-LUT 32 × 32**	**MLv-LUT 32 × 32**	**ML-LUT 64 × 32**	**MLv-LUT 64 × 32**
$\Delta =$ 1 µs	CR_{MU}	15.04	29.66	29.68	58.38	113.58	111.42	353.70	353.82	411.43	**411.49**
	Improv.	−95.75%	−91.61%	−91.61%	−83.49%	−67.89%	−68.50%	–	0.03%	16.32%	16.34%
$\Delta =$ 100 µs	CR_{MU}	5.97	9.91	9.89	13.91	15.83	16.07	26.38	27.77	**28.35**	28.58
	Improv.	−77.37%	−62.43%	−62.51%	−47.27%	−39.99%	−39.08%	–	5.27%	7.47%	8.34%
$\Delta =$ 1000 µs	CR_{MU}	3.27	4.44	4.44	5.20	5.46	5.57	6.56	**6.91**	6.44	6.50
	Improv.	−50.15%	−32.32%	−32.32%	−20.73%	−16.77%	−15.09%	–	5.34%	−1.83%	−0.91%
$\Delta =$ 5555 µs	CR_{MU}	2.12	2.61	2.59	2.86	3.01	3.02	2.92	**3.17**	2.96	3.02
	Improv.	−27.40%	−10.62%	−11.30%	−2.05%	3.08%	3.42%	–	8.56%	1.37%	3.42%

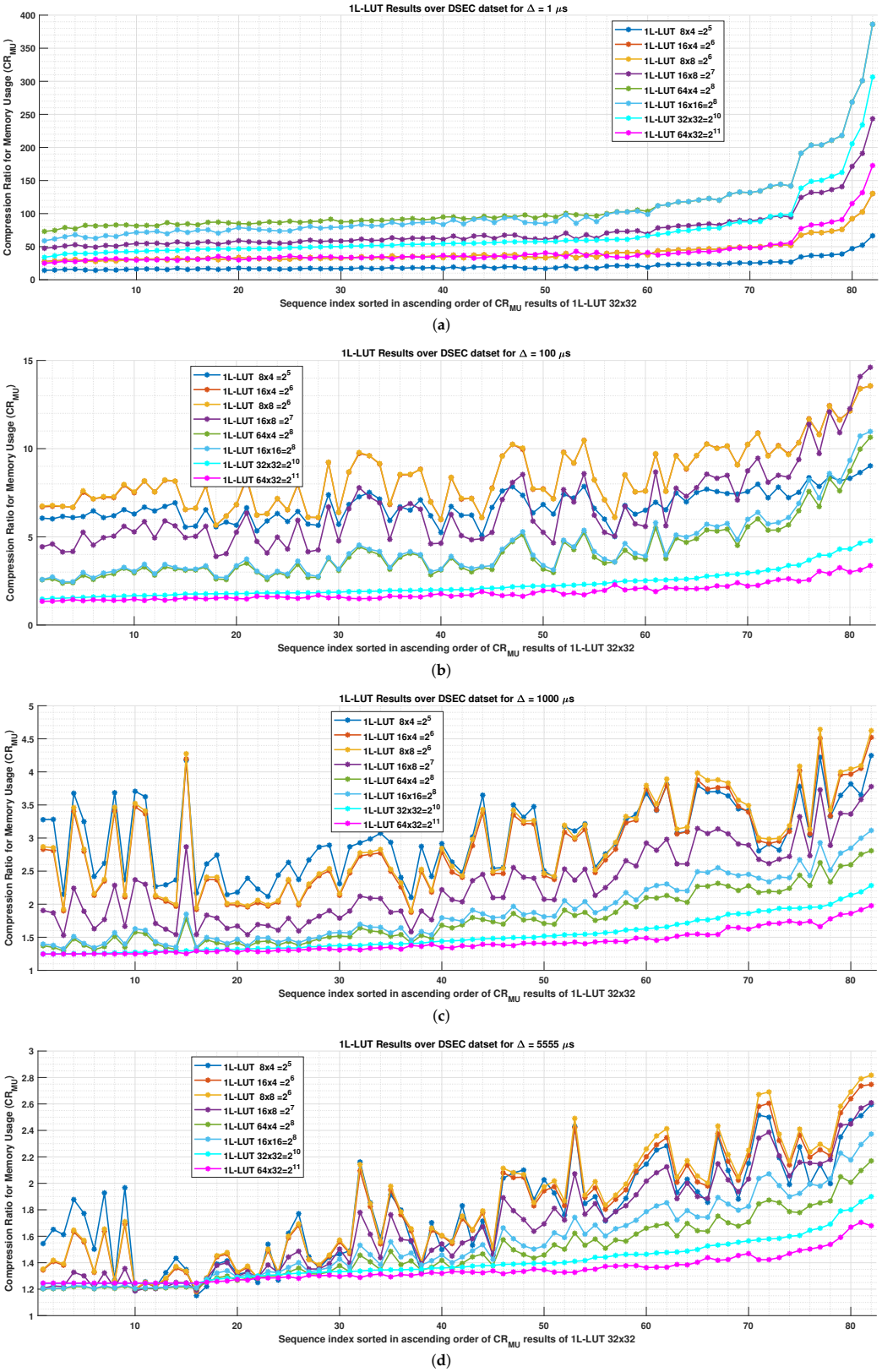
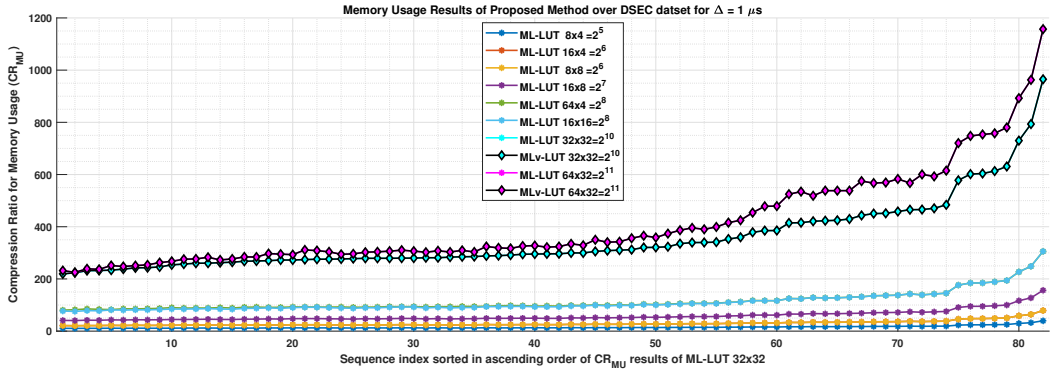
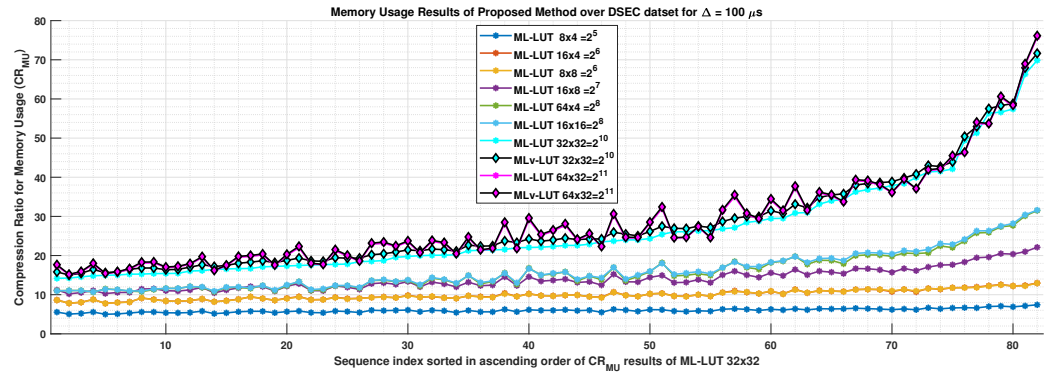


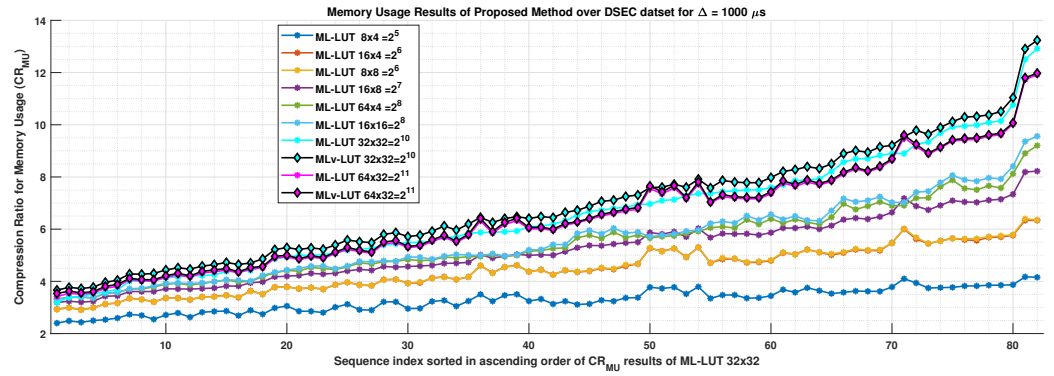
Figure 12. Memory usage results over DSEC, measured using CR_{MU}, for 1L-LUT using different group sizes, when encoding EF sequences generated using a time-window of (**a**) $\Delta = 1$ µs; (**b**) $\Delta = 100$ µs; (**c**) $\Delta = 1000$ µs; and (**d**) $\Delta = 5555$ µs. The DSEC sequence is ordered in ascending order of the CR_{MU} results of 1L-LUT 32×32.



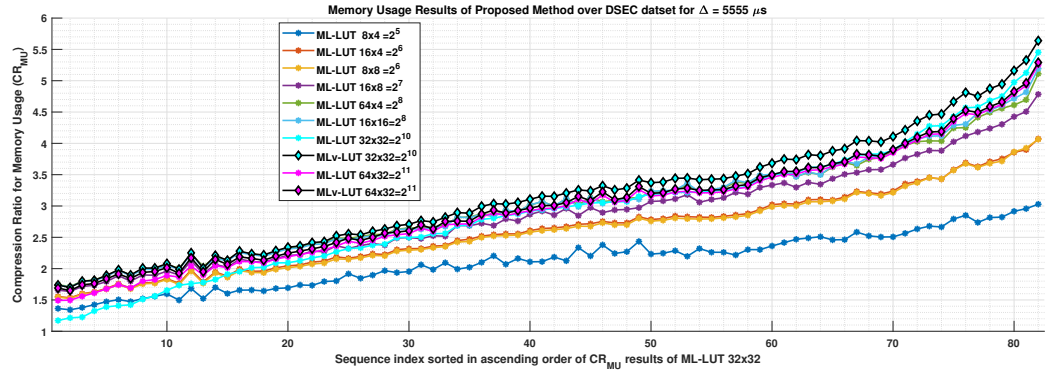
(**a**)



(**b**)



(**c**)



(**d**)

Figure 13. Memory usage results over DSEC, measured using CR_{MU}, for ML-LUT and MLv-LUT using different group sizes, when encoding EF sequences generated using a time-window of (**a**) $\Delta = 1$ µs; (**b**) $\Delta = 100$ µs; (**c**) $\Delta = 1000$ µs; and (**d**) $\Delta = 5555$ µs. The DSEC sequence is ordered in ascending order of the CR_{MU} results of ML-LUT 32×32.

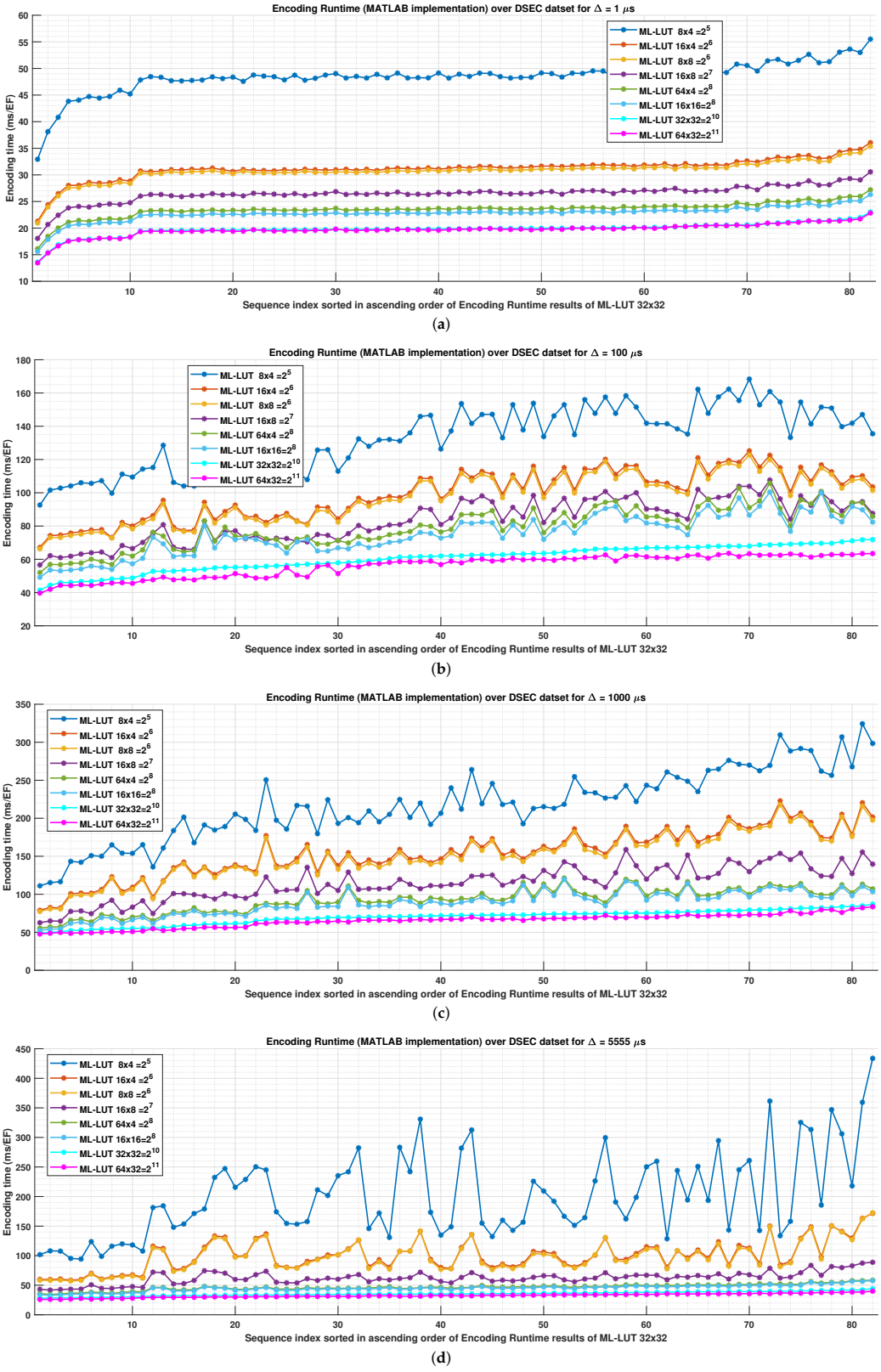


Figure 14. Encoding runtime results (using MATLAB code implementation) over DSEC for ML-LUT using different group sizes when encoding EF sequences generated using a time-window of (**a**) $\Delta = 1$ µs; (**b**) $\Delta = 100$ µs; (**c**) $\Delta = 1000$ µs; and (**d**) $\Delta = 5555$ µs. The DSEC sequence is ordered in ascending order of the encoding runtime results of ML-LUT 32×32.

Table 4. Encoding runtime results over DSEC for ML-LUT using different group sizes.

Method		ML−LUT 8 × 4	ML−LUT 16 × 4	ML−LUT 8 × 8	ML−LUT 16 × 8	ML−LUT 64 × 4	ML−LUT 16 × 16	ML−LUT 32 × 32	ML−LUT 64 × 32
$\Delta = 1$ µs	time (ms/EF)	48.44	31.19	30.70	26.48	23.52	22.76	19.81	**19.68**
	Improv.	144.52%	57.45%	54.97%	33.67%	18.73%	14.89%	–	−0.66%
$\Delta = 100$ µs	time (ms/EF)	132.87	99.05	97.24	83.54	79.14	75.21	60.57	**56.05**
	Improv.	119.37%	63.53%	60.54%	37.92%	30.66%	24.17%	–	−7.46%
$\Delta = 1000$ µs	time (ms/EF)	217.68	154.18	150.69	115.08	92.41	88.72	69.82	**65.18**
	Improv.	211.77%	120.82%	115.83%	64.82%	32.35%	27.07%	–	−6.65%
$\Delta = 5555$ µs	time (ms/EF)	200.99	100.12	98.08	62.45	46.67	45.09	35.51	**32.49**
	Improv.	466.01%	181.95%	176.20%	75.87%	31.43%	26.98%	–	−8.50%

5. Conclusions

The paper proposed a novel low-complexity lossless compression framework for encoding synchronous EFs by introducing a novel memory-efficient fixed-length representation suitable for hardware implementation in the VLP event-processing chip. The proposed framework first partitions the ternary EFs into groups and remaps the symbols. Several solutions are proposed using different levels of LUTs. 2L-LUT is the low-complexity lossless compression solution for encoding EFs, which provides fixed-length representation based on two-level LUTs. ML-LUT provides an improved fixed-length representation based on multi-level LUTs using very-large groups of pixels. The proposed framework provides RA to any group of pixels using a fixed-length representation.

The experimental evaluation demonstrates that the proposed fixed-length framework provides at least two times the compression ratio relative to the raw EF representation and a close performance compared with the traditional variable-length lossless compression methods, such as CALIC and FLIF, and with the variable-length video coding standards, HEVC and VVC, for lossless compression of the ternary EFs generated at frequencies below 1 KHz. To our knowledge, the paper is the first to explore low-complexity lossless compression solutions which provide fixed-length representation of synchronous EFs, suitable for integration into very-low-power processing chips.

Author Contributions: Conceptualization, I.S.; methodology, I.S.; software, I.S.; validation, I.S.; formal analysis, I.S.; investigation, I.S.; writing—original draft preparation, I.S.; writing—review and editing, I.S. and R.C.B.; visualization, I.S.; supervision, R.C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DVS	Dynamic Vision Sensor
APS	Active Pixel Sensor
DAVIS	Dynamic and Active-pixel VIsion Sensor
EF	Event Frame
LUT	LookUp Table
RA	Random Access

TALVEN	Time Aggregation-based Lossless Video Encoding for Neuromorphic sensor
ESP	Event Signal Processing
VLP	Very-Low-Power
EMI	Event Map Image
CPV	Concatenated Polarity Vector
HEVC	High-Efficiency Video Coding
SNN	Spike Neural Network
CALIC	Context Adaptive Lossless Image Codec
FLIF	Free Lossless Image Format

References

1. Monroe, D. Neuromorphic Computing Gets Ready for the (Really) Big Time. *Commun. ACM* **2014**, *57*, 13–15. [CrossRef]
2. Lichtsteiner, P.; Posch, C.; Delbruck, T. A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE J. Solid-State Circuits* **2008**, *43*, 566–576. [CrossRef]
3. Brandli, C.; Berner, R.; Yang, M.; Liu, S.C.; Delbruck, T. A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE J. Solid-State Circuits* **2014**, *49*, 2333–2341. [CrossRef]
4. Pan, L.; Scheerlinck, C.; Yu, X.; Hartley, R.; Liu, M.; Dai, Y. Bringing a Blurry Frame Alive at High Frame-Rate With an Event Camera. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 6813–6822. [CrossRef]
5. Gehrig, D.; Rebecq, H.; Gallego, G.; Scaramuzza, D. Asynchronous Photometric Feature Tracking using Events and Frames. *Int. J. Comput. Vis.* **2020**, *128*, 750–765. [CrossRef]
6. Iaboni, C.; Lobo, D.; Choi, J.W.; Abichandani, P. Event-Based Motion Capture System for Online Multi-Quadrotor Localization and Tracking. *Sensors* **2022**, *22*, 3240. [CrossRef]
7. Zhu, A.; Yuan, L.; Chaney, K.; Daniilidis, K. EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras. *arXiv* **2018**, arXiv:1802.06898. [CrossRef]
8. Brandli, C.; Mantel, T.; Hutter, M.; Höpflinger, M.; Berner, R.; Siegwart, R.; Delbruck, T. Adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor. *Front. Neurosci.* **2014**, *7*, 275. [CrossRef]
9. Li, S.; Feng, Y.; Li, Y.; Jiang, Y.; Zou, C.; Gao, Y. Event Stream Super-Resolution via Spatiotemporal Constraint Learning. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; pp. 4460–4469. [CrossRef]
10. Yu, Z.; Zhang, Y.; Liu, D.; Zou, D.; Chen, X.; Liu, Y.; Ren, J. Training Weakly Supervised Video Frame Interpolation with Events. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; pp. 14569–14578. [CrossRef]
11. Wang, Y.; Yang, J.; Peng, X.; Wu, P.; Gao, L.; Huang, K.; Chen, J.; Kneip, L. Visual Odometry with an Event Camera Using Continuous Ray Warping and Volumetric Contrast Maximization. *Sensors* **2022**, *22*, 5687. [CrossRef]
12. Gallego, G.; Delbrück, T.; Orchard, G.; Bartolozzi, C.; Taba, B.; Censi, A.; Leutenegger, S.; Davison, A.J.; Conradt, J.; Daniilidis, K.; et al. Event-Based Vision: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 154–180. [CrossRef]
13. Bi, Z.; Dong, S.; Tian, Y.; Huang, T. Spike Coding for Dynamic Vision Sensors. In Proceedings of the 2018 Data Compression Conference, Snowbird, UT, USA, 27–30 March 2018; pp. 117–126. [CrossRef]
14. Dong, S.; Bi, Z.; Tian, Y.; Huang, T. Spike Coding for Dynamic Vision Sensor in Intelligent Driving. *IEEE Internet Things J.* **2019**, *6*, 60–71. [CrossRef]
15. Khan, N.; Iqbal, K.; Martini, M.G. Time-Aggregation-Based Lossless Video Encoding for Neuromorphic Vision Sensor Data. *IEEE Internet Things J.* **2021**, *8*, 596–609. [CrossRef]
16. Banerjee, S.; Wang, Z.W.; Chopp, H.H.; Cossairt, O.; Katsaggelos, A.K. Lossy Event Compression Based On Image-Derived Quad Trees And Poisson Disk Sampling. In Proceedings of the 2021 IEEE International Conference on Image Processing (ICIP), Anchorage, AK, USA, 19–22 September 2021; pp. 2154–2158. [CrossRef]
17. Sullivan, G.J.; Ohm, J.R.; Han, W.J.; Wiegand, T. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Trans. Circuits Syst. Video Technol.* **2012**, *22*, 1649–1668. [CrossRef]
18. Schiopu, I.; Bilcu, R.C. Lossless Compression of Event Camera Frames. *IEEE Signal Process. Lett.* **2022**, *29*, 1779–1783. [CrossRef]
19. Schiopu, I.; Bilcu, R.C. Low-Complexity Lossless Coding for Memory-Efficient Representation of Event Camera Frames. *IEEE Sens. Lett.* **2022**, *6*, 1–4. [CrossRef]
20. Schiopu, I.; Bilcu, R.C. Low-Complexity Lossless Coding of Asynchronous Event Sequences for Low-Power Chip Integration. *Sensors* **2022**, *22*, 14. [CrossRef] [PubMed]
21. Gehrig, M.; Aarents, W.; Gehrig, D.; Scaramuzza, D. DSEC: A Stereo Event Camera Dataset for Driving Scenarios. *IEEE Robot. Autom. Lett.* **2021**, *6*, 4947–4954. [CrossRef]
22. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; Nam, G.J.; et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1537–1557. [CrossRef]

23. Henri Rebecq, T.H.; Scaramuzza, D. Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Nonlinear Optimization. In Proceedings of the British Machine Vision Conference (BMVC), London, UK, 4–7 September 2017; Kim, T.-K., Stefanos Zafeiriou, G.B.; Mikolajczyk, K., Eds.; BMVA Press: Durham, UK, 2017; pp. 16.1–16.12. [CrossRef]
24. Maqueda, A.I.; Loquercio, A.; Gallego, G.; Garcia, N.; Scaramuzza, D. Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018. [CrossRef]
25. Almatrafi, M.; Baldwin, R.; Aizawa, K.; Hirakawa, K. Distance Surface for Event-Based Optical Flow. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 1547–1556. [CrossRef]
26. Benosman, R.; Clercq, C.; Lagorce, X.; Ieng, S.H.; Bartolozzi, C. Event-Based Visual Flow. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 407–417. [CrossRef]
27. Bi, Y.; Chadha, A.; Abbas, A.; Bourtsoulatze, E.; Andreopoulos, Y. Graph-Based Object Classification for Neuromorphic Vision Sensing. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 491–501. [CrossRef]
28. Bi, Y.; Chadha, A.; Abbas, A.; Bourtsoulatze, E.; Andreopoulos, Y. Graph-Based Spatio-Temporal Feature Learning for Neuromorphic Vision Sensing. *IEEE Trans. Image Process.* **2020**, *29*, 9084–9098. [CrossRef]
29. Zhu, A.; Yuan, L.; Chaney, K.; Daniilidis, K. Unsupervised Event-Based Learning of Optical Flow, Depth, and Egomotion. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 989–997. [CrossRef]
30. Gehrig, D.; Loquercio, A.; Derpanis, K.; Scaramuzza, D. End-to-End Learning of Representations for Asynchronous Event-Based Data. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 5632–5642. [CrossRef]
31. Baldwin, R.; Liu, R.; Almatrafi, M.M.; Asari, V.K.; Hirakawa, K. Time-Ordered Recent Event (TORE) Volumes for Event Cameras. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 2519–2532. [CrossRef]
32. Khan, N.; Iqbal, K.; Martini, M.G. Lossless Compression of Data From Static and Mobile Dynamic Vision Sensors-Performance and Trade-Offs. *IEEE Access* **2020**, *8*, 103149–103163. [CrossRef]
33. Pavlov, I. LZMA SDK (Software Development Kit). Available online: https://www.7-zip.org/ (accessed on 19 July 2021).
34. Deutsch, P.; Gailly, J.L. *Zlib Compressed Data Format Specification*, version 3.3; 1996. Available online: https://www.ietf.org/rfc/rfc1950.txt.pdf (accessed on 19 July 2021).
35. National Engineering Laboratory for Video Technology, P.U. PKU-DVS Dataset. Available online: https://pkuml.org/resources/pku-dvs.html (accessed on 10 October 2021).
36. Bross, B.; Chen, J.; Ohm, J.R.; Sullivan, G.J.; Wang, Y.K. Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC). *Proc. IEEE* **2021**, *109*, 1463–1493. [CrossRef]
37. Wu, X.; Memon, N. Context-based, adaptive, lossless image coding. *IEEE Trans. Commun.* **1997**, *45*, 437–444. [CrossRef]
38. Sneyers, J.; Wuille, P. FLIF: Free lossless image format based on MANIAC compression. In Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 25–28 September 2016; pp. 66–70. [CrossRef]
39. FFmpeg. FFmpeg Homepage. Available online: http://ffmpeg.org (accessed on 1 February 2021).
40. HHI, F. VVC Test Model (VTM). Available online: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM (accessed on 1 July 2021).