Adaptive-length Coding of Image Data for Low-cost Approximate Storage

Qianqian Fan, David J. Lilja, and Sachin S. Sapatnekar

Abstract—In the past few years, ever-increasing amounts of image data have been generated by users globally, and these images are routinely stored in cold storage systems in compressed formats. This paper investigates the use of approximate storage that leverages the use of cheaper, lower reliability memories that can have higher error rates. Since traditional JPEG-based schemes based on variable-length coding are extremely sensitive to error, the direct use of approximate storage results in severe quality degradation. We propose an error-resilient adaptive-length coding (ALC) scheme that divides all symbols into two classes, based on their frequency of occurrence, where each class has a fixed-length codeword. This provides a balance between the reliability of fixed-length coding schemes, which have a high storage overhead, and the storage-efficiency of Huffman coding schemes, which show high levels of error on low-reliability storage platforms. Further, we use data partitioning to determine which bits are stored in approximate or reliable storage to lower the overall cost of storage. We show that ALC can be used with general non-volatile storage, and can substantially reduce the total cost compared to traditional JPEG-based storage.

Index Terms—Adaptive-length Coding, Error-resilience, Approximate Storage.

1 Introduction

The quantity of image data on cloud-based storage has skyrocketed in recent years. For example, in 2014 annual trends report, 1.8 billion images were uploaded to Facebook, Instagram, Snapchat, and WhatsApp every day [1], a number that has surely increased since. Furthermore, images are currently stored in most sites at multiple resolutions to support different devices and contexts, further increasing storage overheads [2]. These uploads require a significant amount of storage, which can be expensive. Therefore, cost-effective image storage is an active field of research. In [3], they reduced the metadata of photo to obtain lower cost storage with higher throughput. Progressive JPEG with customized encoding parameters for dynamic resizing was used to reduce bandwidth and storage overheads in [2]. Recent works applied cheaper, lower-reliability image and video storage at dramatically lower cost [4] [5] [6] [7]. This concept has been explored for raw images [7] and client devices presuming the availability of a high-fidelity copy in the cloud [4]. However, these prior works do not address the issue of reducing the cost of the long-term storage in the cloud, which is the subject of this paper.

Individual images are typically saved in compressed form in long-term storage rather than in a raw format where every pixel is stored. Apple is adopting the High Efficiency Image File (HEIF) format as its new photo format [8], but this format is still only a small part of the market. Dropbox uses Lepton to re-compress JPEG files, which replaces the lowest layer of the baseline JPEG compression with a parallelized arithmetic code (variable-length) to improve speed [9]. The PTC encoding algorithm is used in [4] for the purpose of approximate storage, which is a seldom-used image format and has low compression efficiency. To reach our target, the JPEG encoding algorithm is a good candidate because of its popularity [10] and high compression ratio.

The JPEG standard uses a discrete cosine transform (DCT) and saves the coefficients of the dominant spatial frequencies using variable-length coding (VLC), a Huffman coding scheme that further improves compression efficiency [11]. Modern image storage providers, such as Google Photos, typically re-compress JPEG files to reduce the image size. While this loses some information, it does so without any perceptual difference [9] [12]. Similarly, when approximate storage is used, the new storage scheme should keep each encoded image with no visually perceptible loss in quality. However, an error in compressed image data can change key attributes of the VLC-encoded image, and even with variable error correction mechanism, can still result in obvious distortions in the image [5]. Even a single error could completely corrupt part of the image (Fig. 1a). The direct use of error-prone low-reliability storage for saving VLC-encoded JPEG images is thus risky as it can cause unacceptable levels of error. Therefore, despite their low cost, error-prone storage is not considered viable for storing compressed VLC-encoded images.

(a) (b) (c)

Fig. 1: (a) VLC-based storage where a single error corrupts the the remaining blocks of image (b) VLC-based storage with variable error correction mechanism [5] under a 1% error rate, showing obvious degradation. (c) Our ALC-based scheme with a same percentage of data stored in approximate storage as (b) under a 1% error rate, showing no visible quality loss.

In this work, a novel adaptive-length coding (ALC) scheme is proposed to replace the error-sensitive Huffman coding in JPEG compression. Our ALC scheme is shown to be inherently resilient to errors. ALC divides all the symbols

into two classes: more frequently occurring symbols, which are encoded to a shorter fixed length, and more infrequent symbols that are mapped to a longer fixed length code. Since the codeword in each class has the same length, ALC is an encoding scheme where every word uses one of two allowable lengths. This differs from Huffman coding as used in VLC where each symbol could be mapped to a variable length encoding [11]. By limiting the number of code lengths to two, we obtain the best of both worlds, balancing the error-resilience of fixed-length encoding with the reduced storage needs of VLC. According to the statistics of an image, the algorithm adaptively and efficiently adds additional bits to compensate for the quality loss. We demonstrate that ALC maintains excellent compression efficiency while improving error resilience inherently. An example result is shown in Fig. 1c.

The increased error resilience of ALC allows most of the data bits of an image to be stored in low-cost approximate storage, but some critical parts of the image may still require reliable storage. We develop a data partitioning scheme that segments the ALC-encoded data into reliable and approximate storage, minimizing the total cost of storage with the quality degradation constrained. From source-channel separation theorem [13] aspect, we can optimize ALC and ECC of storage separately and can simply combine them in a cascaded manner. In this work, we mainly focus on the compressed image data instead of other applications with high error resilience [14]. Our scheme can be implemented for JPEG files recompression as [9], which only replace Huffman coding with error-resilient ALC compared to conventional JPEG scheme. Furthermore, ALC scheme is compatible with more sophisticated JPEG techniques, e.g. JPEGmini, which use image-specific quantization matrix to further improve compression efficiency [15].

2 Related work

The networking community has proposed a set of techniques for image transmission over lossy networks to improve the error resilience of image coding.

From the point of view of source coding, to overcome the resynchronization limitation of VLC, the error resilient entropy coding (EREC) method was proposed in [16]. In this code, blocks of various lengths are reorganized into fixed-length blocks to prevent error propagation beyond block boundaries. The approaches in [17] [18] [19] adopt the same idea to improve the error resilience of JPEG by preventing the errors propagating across the block boundaries. However, this cannot prevent quality degradation within blocks at the higher error rates seen in low-cost storage. In [20], the embedding and side-match vector quantization (VQ) is used to conceal the corrupted block, but their discussion shows that it is challenging for this scheme to achieve acceptable quality degradation under large error rates. The Hybrid Variable Length Code (HVLC) [21] uses multiple VLC coding structures to reduce the error propagation distance within one codeword, but the boundaries of the codeword are based on VLC, and if errors are introduced in these boundary code bits, the image data can be corrupted.

Channel coding also can be applied intelligently to improve the error resilience. An error-resilient unequal protection can robustly cope with packet loss in transmission [5] [22]. The Wyner-Ziv Error-Resilient scheme emphasizes protecting the Region of Interest area in the frame [23]. However, the goal of all these methods localize the error within blocks or larger segments and to prevent it from impacting the entire bit-stream. As we stated earlier, the impact of this block-based error on perceptual quality is still large for purposes of image storage, as shown in Fig. 1b.

Instead of using VLC, SoftCast [24] compresses data by discarding zero and near-zero DCT components. The codewords preserve the numerical properties of the original pixels, so the error resilience of image or video coding can be improved compared to the VLC-based scheme. Fixed-length coding (FLC) [25] ensures that errors do not propagate beyond the corrupted codeword, but at a much lower compression efficiency than VLC. Variable-to-Fixed-length codes, e.g. the Tunstall coding algorithm, map symbols to a fixed number of bits and can also avoid resynchronization issues in VLC. However, the compression efficiency of Tunstall coding is severely affected by rarely-used symbols compared with Huffman coding [26]. No existing scheme can overcome the error sensitivity of VLC with comparable compression efficiency.

3 Preliminaries and motivation

3.1 Cost-reliability trade-offs for approximate storage

There exists a clear trade-off between the cost of a storage device and its reliability. For instance, inexpensive HDDs have lower reliability than more expensive enterprise-class devices. As a result, the less expensive devices are typically used in a RAID configuration to improve the system reliability. There is a similar trade-off for SSD devices and for other types of storage technologies. Furthermore, we can make other trade-offs besides reliability and price cost. For example, there is a trade-off between the cost of adding ECC to a memory system and the resulting reliability. This trade-off has been used in prior work to build an approximate storage system [4] [5]. There also exists a trade-off in NAND flash technology between the cell storage efficiency (bits/cell) and reliability [27]. For the emerging STT-MRAM technology, the reliability is related to the energy required to change the device's state. A lower energy circuit will provide lower reliability [6] [7].

The goal of this work is to develop a methodology for storing a compressed image in approximate storage. We propose a general methodology and presenting the notion of the trade-off under various cost scenarios. As a result, we assume only two types of storage - reliable and approximate - that have different cost ratios. The specific implementation of these two types of storage is beyond the scope of this work.

3.2 Fundamentals of JPEG

JPEG is a commonly used technique that employs the discrete cosine transform (DCT) to perform lossy compression on digital images [28]. The DCT divides an image into a set of 8×8 pixel frames, converting the sub-image in each frame from the spatial domain into an 8×8 matrix of coefficients in the frequency domain, corresponding to various spatial frequencies. The zero-frequency coefficient of the DCT is referred to as its DC term, which is the average of the pixel values. It provides a baseline value for the encoding. The remaining elements for the AC terms provide information

about the successively higher frequency components, which represent color changes across the block. Since the human eye is insensitive to high-frequency spatial variations, a quantization step is used to divide each coefficient by the non-uniform entry in the quantization matrix (higher resolution for DC and low-frequency components), and an integer result is stored. As larger divisors are used at higher frequencies, and DCT coefficients tend to be lower at high frequencies, many coefficients go to zero. Only nonzero DCT coefficients are stored, and thus the volume of compressed data is greatly reduced from the raw image. The original image can be reconstructed from this data with little or no discernible loss in quality. The entries of the quantization matrix, for a quality factor Q, range from 1 to 100, where a higher number corresponds to higher quality ($Q = 90$ is widely used). The quantization matrix for any value of Q can be derived using the procedure in [28] that is based on standard quantization matrix for $Q = 50$.

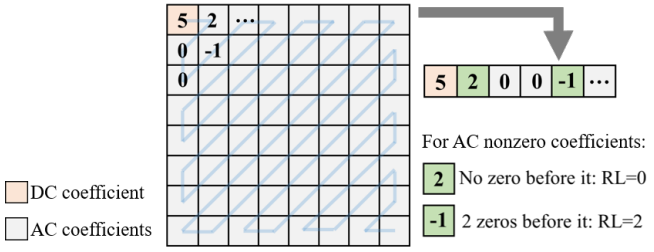


Fig. 2: The zigzag pattern that orders the DCT coefficients for storage.

The top left entry in the 8×8 DCT coefficient matrix is the DC coefficient. The DC coefficients store the basic information for the image, and employ differential coding, whereby the difference of the DC coefficients between successive blocks is saved. This enables more compact storage, but allows an error to affect multiple blocks. The remaining entries are the AC coefficients. In practice, since the nonzero AC coefficients tend to cluster near the top left entry of the matrix, the quantized coefficients are stored in a zigzag sequence, as shown in Fig. 2. Then DC and AC coefficients are encoded separately based on different Huffman tables specified in the JPEG standard [29].

TABLE 1: VLC-encoded AC coefficients in Fig. 2.

Nonzero coefficient	RL	1's complement representation	CAT	Codeword
2	0	10	2	01 10
-1	2	0	1	11100 0

In VLC, the value of a coefficient is stored using one's complement representation[1], where the sign is encoded by ensuring that the MSB for a positive [negative] number is 1 [0]. For example, consider a differentially coded DC coefficient of 5, represented in one's complement notation as 101, in Fig. 2. Then the category (CAT) of this coefficient will be encoded based on the Huffman table for the DC coefficient, which represents the number of bits required to store this value. Therefore, for this example, CAT = 3, which is translated to 100 after Huffman coding. The encoded DC coefficient then concatenates these to obtain the codeword 100 101. In VLC, the maximum value of CAT for DC coefficients is 11 bits.

Table 1 encodes the AC coefficient values in Fig. 2. The sequence of coefficients is translated to symbols that encode the run-length (RL) and the category (CAT) of the coefficient. Here, RL corresponds to the number of zeros preceding the coefficient, e.g., for the entry of -1, RL = 2 as it is preceded by two zeros. Each (RL, CAT) symbol is represented by a variable-length codeword based on Huffman tables, assigning shorter codes to more common symbols, e.g., in Table 1, (RL = 0, CAT = 2) is represented by the symbol 01. In VLC, RL is limited to 4 bits and CAT to 10 bits for AC coefficients; in case of larger run-lengths, symbols with (RL = 15, CAT = 0) can be concatenated.

For the block in Fig. 2, the precise bit string that is stored is a concatenation of the codewords for the DC coefficient and all AC coefficients in Table 1, i.e.,

$$1001010110111000\ldots1010$$

where the ellipsis represents the nonzero coefficients that are not shown in the figure. The last symbol, 1010, corresponds to end of block (EOB), a unique symbol that indicates the last nonzero coefficient of a block. The EOB symbol acts as a separator between consecutive blocks.

As shown in Fig. 3e, the storage overhead is dominated by the AC coefficients. Further, among these AC coefficients, the distribution of the symbols (RL, CAT) is quite unbalanced. For example, in the images, Girlface, Peppers and the two additional images Tiger, Bird, from the Imagenet dataset [30], the first 15 symbols can be seen to cover over 95% of the coefficients, with (0,1), (0,2) and (1,1) corresponding to over half of all 161 symbols, as shown in Fig. 3. This motivates the use of Huffman coding in VLC, but we will soon see how VLC is sensitive to errors.

3.3 Error resilience limitations of JPEG storage schemes

The error tolerance for traditional VLC-encoded JPEG, where the storage scheme uses a Huffman-based coding algorithm, is quite limited, and even a single error can bring a dramatic degradation, as illustrated in Fig. 1(a). This error occurs due to a misalignment caused by an erroneous symbol, which maps to a keyword with a different (RL, CAT) value and disrupts symbol boundaries. This creates noisy data in the blocks that follow. Moreover, some EOB symbols are left undetected, and the image is interpreted to have fewer blocks, with all-zero coefficients in the last few blocks, causing them to be black.

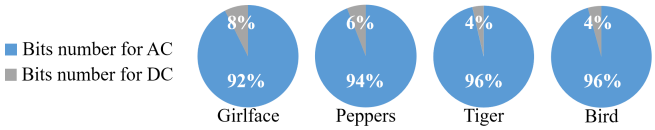
One way to avoid such scenarios is to annotate each block with the number of bits that it contains [17]. This improves the error resilience by reducing the possibility of error propagation across blocks, but under a larger number of errors, as shown in Fig. 1(b).

Another alternative is to use fixed-length coding (FLC) [25]. Instead of a variable-length codeword, FLC may use the maximum number of bits for RL (4 bits) and CAT (10 bits) if coding AC coefficients. For a constant number of CAT bits, it is essential to also explicitly store a sign bit since the sign can no longer be inferred from the leading bit of the

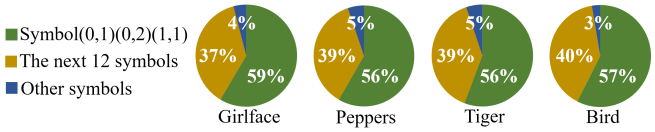
1. two's complement representation can also be applied, but it will not introduce any meaningful changes in the results.

(a) Girlface (b) Peppers

(c) Tiger (d) Bird



(e) Number of bits required to store the DC and AC coefficients using VLC



(f) Distribution of the (RL, CAT) symbols for the AC coefficients

Fig. 3: A set of sample images and the distribution of the symbols representing their AC and DC coefficients.

data. In other words, to represent signed binary AC coefficients, the sign-magnitude representation is needed for FLC instead of only using the one's complement representation in VLC. This implies that the length of each codeword in FLC can be 4+10+1 = 15 bits. This fixed codeword length allows the boundary between codewords in this scheme to be determined easily, so that errors no longer propagate to subsequent codewords, but this potential improvement in error resilience is accompanied by a severe degradation in compression efficiency. Furthermore, the error resilience may not actually be improved, since the large increase in the number of storage bits raises the likelihood of errors that cannot be corrected for.

4 Adaptive-Length Coding

In this section, we propose the adaptive-length coding (ALC) scheme to provide a balance between the reliability of fixed-length coding schemes and the storage-efficiency of Huffman coding schemes. The ALC method is described in three steps: symbol classification for the AC coefficients (Section 4.1), adaptation for additional bits (Section 4.2), and EOB identification (Section 4.3).

It is worth noting that the number of AC coefficients significantly exceeds the number of DC coefficients, as shown in Fig. 3e. Meanwhile, as we will see in Section 8.4, the DC coefficients contain the important characteristics of an image and are not very tolerant to errors. Therefore, we develop the ALC scheme to compactly store the AC coefficients while the DC coefficients are still encoded using VLC.

4.1 Step 1: Symbol classification

Our ALC scheme places all symbols for the AC coefficients into one of two classes. Class I corresponds to a shorter fixed codeword length for the most frequent symbols, (0,1), (0,2), and (1,1). Class II consists of all other symbols, which are encoded using a longer fixed-length.

The structure of the codewords is summarized in Fig. 4. The MSB indicates whether the symbol belongs to Class I or II. In case of Class I, all of the remaining bits are used to store the coefficient. For Class II, if RL = 0, the second-MSB is set to 0, the third bit is the sign bit, and four bits are used to save the coefficient magnitude. If RL $\neq$ 0, then the second-MSB is 1, followed by three bits for RL, the sign bit, and 1 bit for the coefficient magnitude. Similar to FLC, the sign-magnitude representation is used for the AC coefficients in ALC. Next, we justify the choice of the number of bits used to store RL and the coefficient.

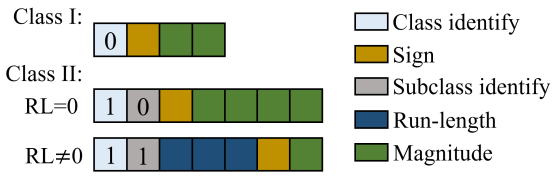


Fig. 4: Bit assignment for Class I and Class II symbols in the ALC scheme.

4.1.1 *Precise representation for Class I*

Table 2 shows our scheme for uniquely representing Class I codewords using a 4-bit fixed-length code. For Class I, the MSB is set to 0. The next bit indicates the sign, and is 0 for a positive value and 1 for a negative value. The last two bits denote the magnitude of the coefficient. Since the only symbols in Class I are (RL, CAT) = {(0,1), (0,2), (1,1)}, there are four cases without considering the sign bit, as shown in Table 2. Thus, the magnitude and RL of Class I can be precisely represented using a 2-bit fixed length field. For comparison, the corresponding VLC codes are shown, which have similar output length to the ALC codes.

TABLE 2: Encoding Class I in ALC, and a comparison with VLC.

(RL, CAT)	Magnitude	ALC Codeword		VLC Codeword		Output length	
		Positive	Negative	Positive	Negative	ALC	VLC
(0, 1)	1	0 0 00	0 1 00	00 1	00 0	4	3
(0, 2)	2	0 0 01	0 1 01	01 10	01 01	4	4
	3	0 0 10	0 1 10	01 11	01 00	4	4
(1, 1)	1	0 0 11	0 1 11	1100 1	1100 0	4	5

4.1.2 *Approximate representation for Class II*

For elements in Class II, Fig. 5 shows the distribution of the DCT coefficient magnitudes without considering the symbols in Class I. This distribution determines the number of bits required for Class II. Four representative images are analyzed in Fig. 3a–3d, separating the scenarios where RL = 0 and RL $\neq$ 0. All results show the same trend: when RL $\neq$ 0, CAT is small for almost all possible values, and is larger only when RL = 0. Thus, finding symbols with large values of both RL and CAT is improbable. Based on this observation, the use of the fixed bit length for Class II

symbols varies with RL. When RL $= 0$, all bits are used to store the coefficient, and when RL $\neq 0$, the first few bits represent RL while the rest correspond to the coefficient, as illustrated in Fig. 4.

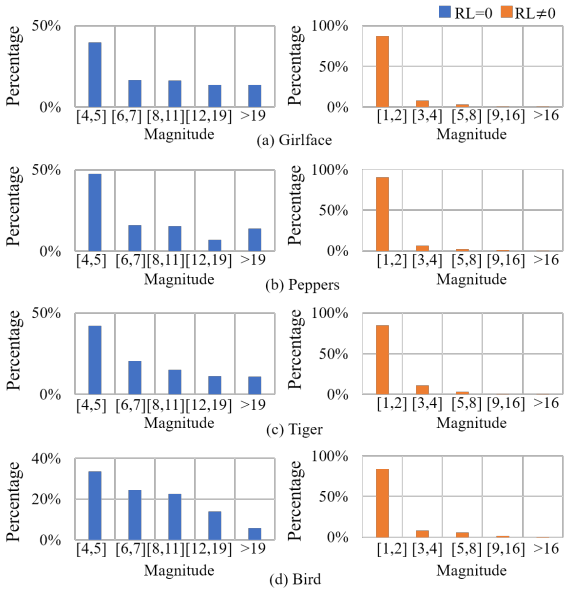


Fig. 5: A histogram of coefficient magnitudes for Class II symbols for a representative set of images, for RL $= 0$ and RL $\neq 0$. The values in the x-axis for each class group several magnitude values when one more bit is added in the Class II magnitude field.

4.1.2.1 Selecting the number of bits for RL $\neq 0$: Table 3 examines the symbols that cannot be represented by three RL bits, i.e., with RL $= 1, \cdots, 8$. Only a small fraction of symbols have RL > 8 and these tend to have small magnitudes. Thus, for this case, 3 bits are sufficient to represent RL. Any symbols with RL > 8, and all coefficients beyond this symbol, are discarded with little quality loss.

TABLE 3: Distribution of symbol magnitudes for RL > 8

		Girlface	Peppers	Tiger	Bird
Fraction of symbols with RL > 8		1.67%	5.18%	2.83%	0.42%
Distribution of the coefficient magnitude in RL > 8 symbols	Magnitude = 0	7.53%	13.29%	7.31%	3.28%
	Magnitude = 1	90.75%	86.30%	92.26%	96.72%
	Magnitude = 2	1.54%	0.41%	0.24%	0.00%
	Magnitude > 2	0.18%	0.00%	0.19%	0.00%

The observation from Table 3 can be extended to a larger set of images. Fig. 6 shows the distributions generated with 500 images in the dataset [30]. Figure. 6a shows that the fraction of symbols that require RL>8 is quite small. Thus, three bits for RL is sufficient for most situations. In Figure. 6b, only a few parts of the symbols with RL>8 have magnitude larger than 2, which means the information can be discarded with little impact on the final results.

4.1.2.2 Selecting the number of bits for the coefficient magnitude: According to the Huffman table for the AC coefficients, the magnitudes can be represented precisely using 10 bits. However, from Fig. 5, numbers that use all 10 bits are seldom encountered. Therefore, we reduce the number of magnitude bits while still covering the vast majority of scenarios. The figure shows that the distribution

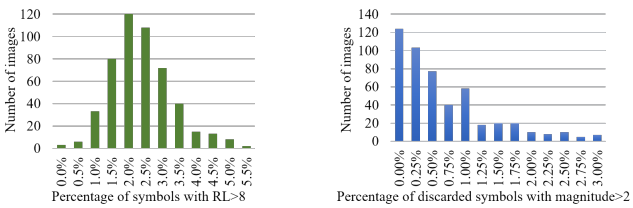


Fig. 6: The distribution of images based on the different percentage of symbols that cannot be covered by Class II representation for RL.(a) Discard the symbols with RL>8.(b) The discarded symbols with the magnitude larger than 2.

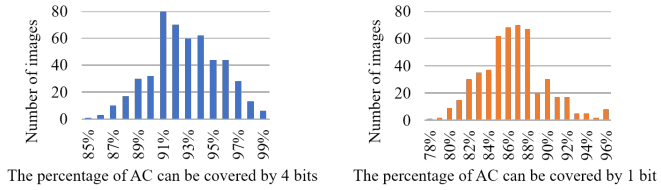


Fig. 7: The distribution of images based on the different percentage of AC coefficients can be covered by Class II representation for magnitude.

of magnitudes for RL $= 0$ has a wider spread than for RL $\neq 0$, but even here, the number of coefficients whose values exceed 19 is small. Therefore, our ALC encoding represents the magnitude using 4 bits. Since any value of RL > 8 is treated as an EOB, it is not necessary to store coefficient values of 0 to store long run lengths, and Class II has no magnitudes of 1 and 2 when RL $= 0$. Therefore four bits can be used to store magnitudes ranging from 4 to 19. If the actual coefficient magnitude exceeds 19, a value of 19 is stored. Similarly, for RL $\neq 0$, the vast majority of coefficient magnitudes are ≤ 2, so that ALC uses 1 bit to store coefficients[2] of 1 or 2. Coefficient magnitudes of 3 or larger are capped at 2. This is a large reduction over FLC, which requires 10 bits to represent the coefficient magnitude.

The distribution of these four images are representative and the number of bits covering most of the scenarios works for a larger set of images. Fig. 7 shows the distributions generated with 500 images in the dataset [30]. When RL=0, most images have over 90% of the AC coefficients that can be covered by 4 bits for magnitude; when RL≠0, most images have over 85% of the AC coefficients that can be covered by 1 bit. This parameter selection is verified on a larger dataset of images in Section 8.

4.2 Step 2: Adaptation per image

The basic ALC scheme described above combines the spirit of the Huffman coding scheme in VLC with the error-resilience of FLC. However, if used directly, only using the few bits in Section 4.1.2 to represent the magnitude of Class II is not enough and it can lead to a significant degradation of the image quality.

2. When RL=1, the magnitude representation can be extended to encode 2 or 3 since (RL, CAT)=(1,1) is already encoded in Class I.

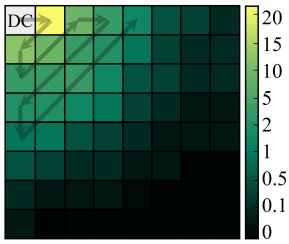


Fig. 8: The average AC coefficient magnitudes, over all frames, of all the blocks in Girlface. A lighter color represents a higher magnitude.

To better understand the reason, we study the average magnitudes of all the blocks in a representative image, Girlface, (other images show similar overall trends) for each AC frequency band. Fig. 8 presents a color-coded map of these magnitudes for each element in the 8×8 coefficient matrix. The figure indicates that larger-magnitude coefficients are located near the beginning of the zigzag storage sequence. In particular, the first few lower-frequency elements of the sequence have larger coefficients with more important information than higher-frequency components. For instance, we can see that there is an important AC frequency band whose average value is about 20, but this results in an overflow in our basic ALC scheme, which only allows a maximum four bits to store a coefficient.

We alter the basic scheme to improve quality by selectively adding extra bits to the codewords of Class II located in lower-frequency positions, which allows storing these high magnitude coefficients more precisely. In order to develop a computationally simple methodology, the number of additional bits and the number of codewords with these additional bits are the same for all blocks in one image. Therefore, adding one more bit can result in a large change in output length. To ensure high compression efficiency, rather than choosing a worst-case value over all images, we employ image-specific adaptation, where the number of additional bits is chosen based on the characteristics of a specific image.

The number of additional bits required to precisely represent the first T AC codewords is determined by all block samples from an image, where T is an optimization parameter, in the zigzag pattern based on this data. We choose only the first T AC codewords because the AC codewords near the end of the block normally have lower magnitudes. Then we find the maximum and median values of the required additional bits among all block samples. If we use the Structural Similarity Index (SSIM) [31] as the quality metric (defined in Eq. (3) in Section 7.3), in the absence of error, we find that using the maximum value of the required number of additional bits provides the best quality, but at highest overhead (because this value may be rare). Another alternative is to use the median value, which has relatively low overhead, but this still results in large quality degradation, as illustrated in Fig. 9.

We find that using $median\_value + \alpha \times (max\_value - median\_value)$ provides an effective trade-off between the goal of reducing the overhead while delivering adequate quality, where α can be set to any value from 0 to 1. Fig. 9 shows the results for $\alpha = 0.25$ and $\alpha = 0.5$. In our experiments, $\alpha = 0.25$ is used.

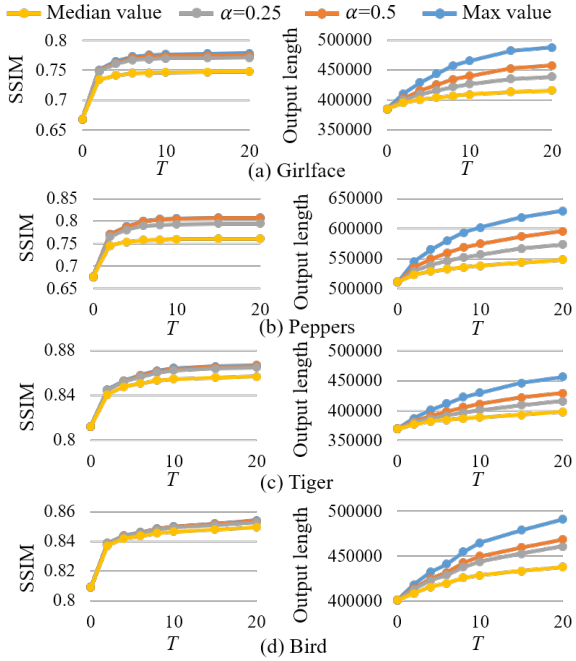


Fig. 9: The impact of using different methods to decide the number of bits to be added to the first T AC codewords.

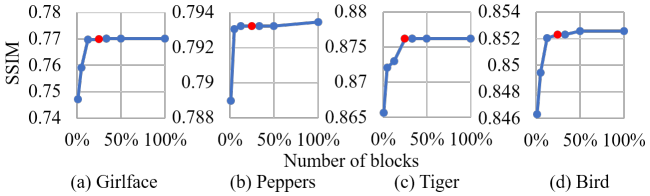


Fig. 10: The impact of using different number of blocks to decide the number of bits to be added to the first T AC codewords.

Moreover, as shown in Fig. 9, the effectiveness in improving the quality goes down steeply after the first few codewords. Thus, in our experiments, we only test the value of T from 0 to 10 to satisfy different quality requirements.

Considering the high correlation of content among adjacent blocks, we uniformly select a subset of the blocks from an image and determine the number of additional bits required to precisely represent the first T AC codewords. From Fig. 10, it can be seen that once a subset of the blocks are sampled, a representative value of T can be chosen. The red dot shows the SSIM value when 25% of the blocks are chosen, and empirically this is seen to be a safe threshold. In principle, this also depends on the order in which the blocks are sampled, but our experiments find the 25% threshold to be safe.

4.3 Step 3: EOB identification

In the ALC scheme, since the codeword lengths are fixed, we do not use the EOB symbol, described in Section 3.2. Instead, for each block, we record the number of codewords in the block for block alignment. Fig. 11 shows the number of blocks with up to k codewords, for various values of k along the x-axis. This indicates that if we limit the number

of codewords in each block to 32, over 95% of all blocks can be covered. Therefore, we choose to use 5 bits to record the number of codewords in a block. If the number of codewords is larger than 32, the higher coefficients are discarded, and we empirically observe that this results in minimal quality degradation.

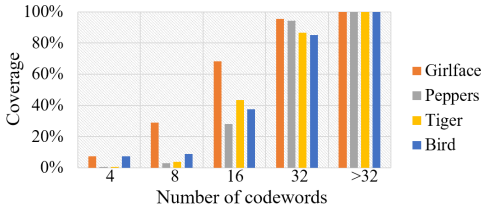


Fig. 11: The cumulative distribution function of the number of codewords in each block.

Note that recording the number of codewords can prevent errors propagating to other blocks. For example, a bit flip in the RL field could shift the coefficients to an incorrect frequency band. However, since we store the number of codewords per block, and each codeword has a known length, such an error can be limited to the block and will not corrupt subsequent blocks.

5 Partitioning Data Between Reliable and Approximate Storage

The image data is partitioned into two parts, one into reliable storage and the other into the approximate storage. An error in the more important data, such as the DC coefficients, could cause a critical failure. Consequently, this data is placed in the reliable storage using the conventional VLC coding. Critical data for the AC coefficients, such as the number of codewords in each block, also are placed in reliable storage. However, the remaining AC coefficient data, which requires a large number of bits, can be placed in the less expensive approximate storage. The decision to place specific data into the reliable or approximate storage areas is based on the quality requirement and the cost of the approximate storage relative to the reliable storage.

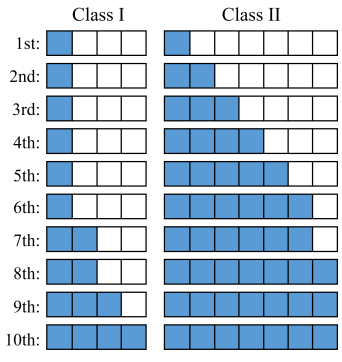
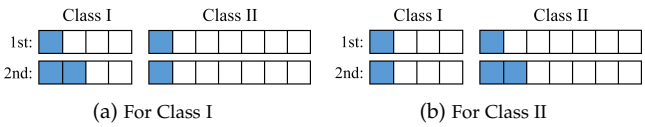


Fig. 12: The sequence of data partitioning patterns based on the importance order of bits in Class I and Class II, where the blue and white bits are stored in reliable and approximate storage, respectively.

Fig. 12 shows ten different patterns that partition data bits between reliable and approximate storage. As we go from the first to the tenth pattern, the quality of the image will improve, at the cost of increased storage costs. In our approach, the final data partitioning pattern between reliable and approximate storage is determined by sequentially trying the patterns in Fig. 12 until the quality requirement is met. Once the pattern has been determined, it will be applied to all codewords in Class I and Class II. To restore an image, we have to access both reliable and approximate storage to get the bits of each codeword according to the data partitioning pattern.



(a) For Class I (b) For Class II

Fig. 13: The two cases for geneating the second pattern: one more bit should be added in reliable storage, either for Class I or for Class II, compared with the first pattern.

Typically, the MSBs of a codeword are more likely to be placed in reliable storage, and the ten patterns successively place larger numbers of MSBs in Class I and Class II in reliable storage. The precise sequence is obtained based on the importance order of bits in Class I and Class II, which have different impacts on output quality, as tested on the sample images, Girlface, Peppers, Tiger and Bird. For example, if we want to generate the second pattern, one more bit should be added in reliable storage compared with the first pattern. As shown in Fig. 13, there are two cases: one is placing the first two MSBs in Class I and MSB in Class II in reliable storage for all codewords; the other is placing the MSB in Class I and the first two MSBs in Class II in reliable storage. Under a given error rate (1% error rate is used in this work), the case with higher quality will be chosen as the second pattern. The sequence of data partitioning patterns is derived following this process.

For data partitioning patterns, we assume the length of codewords in Class II are fixed to simplify the complexity. As explained in Section 4.2, only the first few codewords have variable length due to additional bits. The lengths of the codewords are adaptively changed for different images. Thus, the data partitioning patterns should be varied for different codewords and images. The data partitioning process will be less complicated if we assume the lengths of all codewords are fixed. Ignoring these additional bits in the data partitioning pattern will lead to always placing them in approximate storage. However, the errors that occur in these additional bits have a small impact on the final results, since these additional bits are used to compensate for the least significant bits of the magnitude. Therefore, it is likely to put these additional bits in approximate storage and our simplification for the length of codewords is acceptable.

6 The Impact of Dividing the Symbols Into More Classes

If we divide the symbols into three classes and have three fixed lengths as shown in Figure. 14a: Class I corresponds to the shortest fixed codeword length for the most frequent symbols, (0,1), (0,2), and (1,1); Class II corresponds to the

middle fixed codeword length for the following most frequent symbol (0,3) and a part of symbol (0,4); Class III consists of all other symbols, which are encoded using a longer fixed-length. We optimize the number of bits when using these three classes in the same manner as the previous sections. For Class I, the encoding process is the same as Table II. For class II, the three bits for the magnitude can use 000 111 to represent the values from 4 to 11. For class III, when RL=0, the four bits can be used to store magnitudes ranging from 12 to 27. If the actual coefficient magnitude exceeds 27, the value 27 is stored. When RL≠0, the process is the same as the case with two classes.

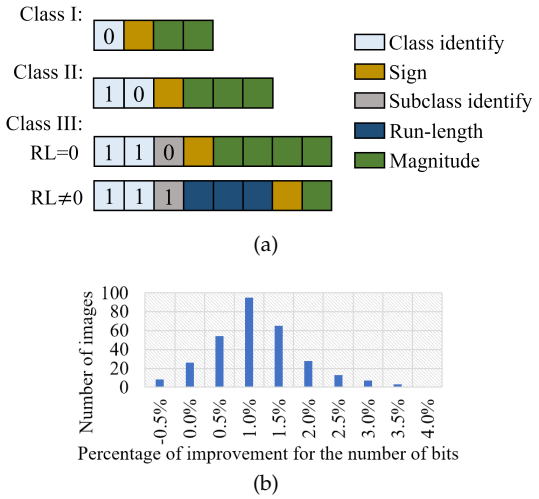


Fig. 14: (a) Bit assignment for three-class scheme (b)The distribution of images with different improvement in the total number of bits.

However, if we compare the three-class scheme with the two-class scheme in the previous section, the improvement is quite limited. First, one more bit is required for class identification. In our methodology, the bit for class identification of a codeword is always placed in reliable storage to make sure the size of the codeword is always known accurately. Therefore, the amount of reliable storage needed to store the class identification bits increases as more classes are used. Second, adding the additional class for symbol (0,3) and a part of symbol (0,4) comes at the expense of adding one more bit for the other symbols in Class III. Fig. 14b shows that the distribution of images with different improvements in the total number of bits. The x-axis represents the improvement of the number of bits compared with using only the two-class scheme. It shows that the benefit of adding the additional class to reduce the total number of bits is very limited.

Considering that we obtain good improvement with only two fixed lengths, and that adding more classes will not produce a large benefit, especially for low-cost ratio, we see that there is no benefit in using three or more lengths.

7 Evaluation Methodology

To evaluate the performance of ALC-based storage compared to traditional JPEG-based storage, the following error injection and error correction models for storage are developed and the criteria for the quality of the result and the storage cost are also described.

7.1 Error model

The data array organization is based on the structure in NAND Flash memories and is a reasonable representation for a storage system built from any technology. A logical page is the smallest addressable unit for reading and writing, and each page is typically made up of a main area for data and a spare area for ECC [32].

We apply an error model based on [33] where the errors are randomly distributed in one page, which follows the random bit-error characteristics of NAND Flash memory. If the error rate of a storage bit-cell is p, then the probability that n cells fail in a storage array with size M follows the binomial distribution:

$$P\left(N=n\right)=\binom{M}{n}p^{n}\left(1-p\right)^{M-n} \tag{1}$$

7.2 Protection of error correction codes

Bose-Chaudhuri-Hocquenghem (BCH) codes are commonly used in storage [33] because they are efficient in correcting single-bit errors. If the BCH error correction capability is t bits, and the number of failed cells is n, which follows the distribution in Eq. (1), then the failure probability of error correction can be defined as:

$$P\left(n>t\right)=\sum_{n=t+1}^{M}\binom{M}{n}p^{n}\left(1-p\right)^{M-n} \tag{2}$$

The BCH scheme can construct a code with length 2^m-1, which includes data bits and parity bits, over the Galois Field $GF(2^m)$. The number of parity bits required for BCH can be computed as mt with error correction capability t [34]. In our case, the errors can occur in both data bits and parity bits with the same error rate, p.

7.3 Evaluation criteria

The quality of the image retrieved from the storage system, and the total cost in bits of the storage system, are used to compare the different coding schemes. In our evaluation, SSIM is used to measure the quality of the decompressed images. The SSIM is an index measuring the structural similarity between two images. It is a well-known objective image quality metric [35] [36]and is defined as

$$SSIM\left(x,y\right)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \tag{3}$$

where μ_x is the average of x, μ_y is the average of y, σ_{xy} is the covariance of x and y, σ_x^2 is the variance of x, σ_y^2 is the variance of y, c_1 and c_2 are two variables to stabilize the division with a weak denominator. It is valued between -1 and 1. When two images are nearly identical, their SSIM is close to 1 [31]. The baseline image for comparison is a conventional JPEG-compressed image with a quality factor of $Q=90$, which we denote as $\text{SSIM}_{Q_{90}}$. If the quality of the image using our scheme is given by $\text{SSIM}_{\text{image}}$, then the percentage degradation is:

$$\text{Quality Degradation}=\frac{\text{SSIM}_{Q_{90}}-\text{SSIM}_{\text{image}}}{\text{SSIM}_{Q_{90}}}\times 100\% \tag{4}$$

Our evaluation places a user-specified limit on the maximum acceptable percentage quality degradation. Image

data is stored in both inexpensive approximate storage and more expensive reliable storage. Therefore, C_{total}, the total cost of storing data, is the sum of the total cost associated with reliable and approximate storage. The cost of each of these components can be defined as the product of the cost per bit and the number of bits used in this type of storage. In other words, if we store N_r (N_a) data bits and E_r (E_a) ECC bits in reliable (approximate) storage, then the total cost is given by:

$$(N_r + E_r) \times C_r^{bit} + (N_a + E_a) \times C_a^{bit} \quad (5)$$

where C_r^{bit} and C_a^{bit} are, respectively, the cost per bit for reliable and approximate storage. Since our objective is to minimize Eq. (5), it is the relative cost between these components that is important, and therefore this minimization is equivalent to minimizing the function:

$$C_{total} = (N_r + E_r) + (N_a + E_a) \times r \quad (6)$$

where the cost ratio r can be defined as

$$r = \frac{C_a^{bit}}{C_r^{bit}} \quad (7)$$

We now consider the cost of various types of image storage schemes:

Conventional VLC-based JPEG image storage method: This uses VLC to store Huffman-coded compressed JPEG images. The high sensitivity to error (demonstrated in Fig. 1) implies that all data must be stored in reliable storage, i.e., $N_a = E_a = 0$, and E_r is determined by the number of BCH protection bits that are used.

Our basic approximate storage scheme: We use ALC encoding to enhance the error-resilience of data stored in approximate memory. This implies that N_r is relatively small and most of the stored image data corresponds to N_a. Specifically, the bits stored in reliable storage correspond to:

- The number of additional bits and the number of codewords with these additional bits, which are the same for all blocks.
- The DC coefficients, stored using VLC.
- Five bits per block that represent the number of symbols in the block, as described in Section 4.3.
- Critical data for the AC coefficients, as described in Section 5.

For a given image with cost $C_{total,\text{image}}$, we define the percentage cost improvement relative to the $C_{total,Q_{90}}$, the cost of conventional VLC-based JPEG image storage with $Q = 90$ as:

$$\text{Cost Improvement} = \frac{C_{total,Q_{90}} - C_{total,\text{image}}}{C_{total,Q_{90}}} \times 100\% \quad (8)$$

8 Experimental results

We evaluate the performance of the proposed ALC algorithm using the basic structure of a generic NAND flash memory [32]. The main area of a page contains 2KB data and it will be further split into four subpages. Each subpage contains 4096 bits of data. ECC is applied to each subpage. The organization of a page is depicted in Fig. 15. The four subpages are continuous in a page and their corresponding ECC parity bits are located in spare area at the end.

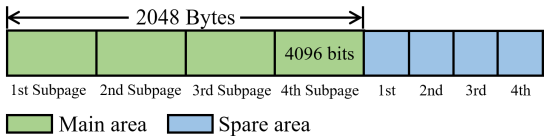


Fig. 15: Organization of a page with the large page divided into small subpages to better implement ECC.

The error rate, p, of reliable storage is 10^{-6} and the common storage industry reliability requirement corrects bit errors to reach a failure probability of 10^{-15} [37]. To satisfy this reliability requirement, eq. (2) indicates that we need an error correction capability of $t = 4$. In our experiments, we evaluate approximate storage at error rates of $p = 0.1\%$, 0.5%, 1% and 1.5%. These error rates are much higher than normal storage as our scheme investigates the use of low-cost, low-reliability storage in order to reduce the overall cost. If the approximate storage tries to reach 10^{-15} failure probability by using ECC, the number of required parity bits becomes prohibitive. For example, when $e = 1\%$, there is a 10^{-15} probability that the number of bit flips will be greater than 123 based on Eq. (2). In other words, we should set the error correction capability parameter to $t = 123$ to ensure the reliability of approximate storage. Based on Section 7.2, for each 4096 bits of data ($GF(2^{13})$ and $m = 13$), $123 \times 13 = 1599$ parity bits are required, corresponding to an overhead of 39% for ECC (i.e., 71% efficiency). In contrast, for the same value of e, in our method, reliable storage requires 52 parity bits, with an overhead of 1% (i.e., 99% efficiency). At a cost ratio of $r = 0.9$, our unreliable storage requires no ECC bits for this error rate, and thus has an overhead of 0% (i.e., 100% efficiency); when $r = 0.3$, the overhead is 12% (i.e., 89.5% efficiency).

Therefore, only weak ECC protection is applied to approximate storage. If the number of errors is beyond the ECC error correction capability, all the errors in this subpage cannot be corrected. For traditional SSD, once this situation happens, the operating system will be notified that these data are corrupted. However, in our approximate storage scheme, we allow errors to happen and the operating system does not need to be notified [4]. Therefore, for approximate storage, various values of t for BCH map to different failure probabilities of each subpage, as shown in Fig. 16, which is generated using Eq. (1) and Eq. (2). The weak ECC protection results in a higher failure probability than high-reliability storage. For example, when $e = 0.5\%$, a choice of $t = 20$ can correct 40% of the errors in the subpages successfully. For different cost ratios, r, defined in Eq. (7), the capability t is varied to find the minimum total cost, as described in Section 7.2.

Our simulations inject errors into 500 image samples from the Imagenet dataset [30]. The resolutions of these image samples range from 1600×520 to 144×144. Since we used the images Girlface, Peppers, Tiger and Bird to set the parameters of the ALC scheme, for a fair evaluation, these images are excluded from the set of 500 images that are used in our evaluation.

We compare our JPEG-based ALC scheme only with the JPEG-based VLC scheme in the evaluation. Our objective is to place the data into approximate storage, while maintaining an acceptable quality and compression efficiency. Thus,

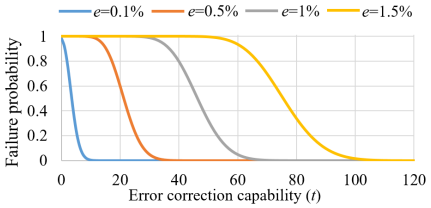


Fig. 16: Failure probability of error correction for different error rates, p, as a function of the error correction capability, t.

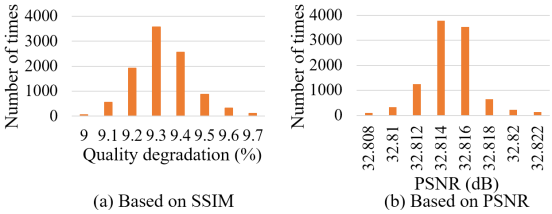


Fig. 18: The distributions of 10^4 Monte Carlo samples on one image with different error locations caused by approximate storage.

our objective is fundamentally different from prior methods described in Section 1, such as [7], which stores raw images (instead of JPEG), or [4], which uses the relatively rarely-used PTC compression format. Moreover, the impact of the block-based errors is large when techniques such as [5] [16]–[23], described in Section 2, are used. Therefore, only traditional JPEG and JPEG with a reduced Q [2] are compared in this section.

8.1 Quality degradation of ALC algorithm

8.1.1 Inherent quality degradation

As illustrated in Section 4, ALC-based storage results in inherent quality degradation due to approximations in representation and the impact of some discarded coefficients. Fig. 17a shows the distribution of quality degradation for the above set of 500 images, considering only ALC-inherent information loss. We also show the quality in terms of peak signal-to-noise ratio (PSNR) as the comparison for the SSIM degradation results, as shown in Fig. 17b. In the absence of errors, we use $median\_value+\alpha\times(max\_value-median\_value)$ with $\alpha = 0.25$ to find the additional bits and apply it to the first $T = 10$ AC codewords to achieve the minimum degradation for each image. This is the largest value in our experiments, as described in Section 4.2. The figures indicate that the image quality histograms for these images are concentrated around the small values of the degradation metric or large value for PSNR, so that the ALC algorithm can effectively maintain most of the information for each image.

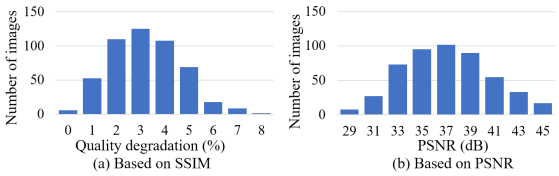


Fig. 17: The distributions of ALC inherent quality degradation based on SSIM and based on PSNR caused only by the approximate representation for 500 different images.

8.1.2 Quality degradation for various error locations

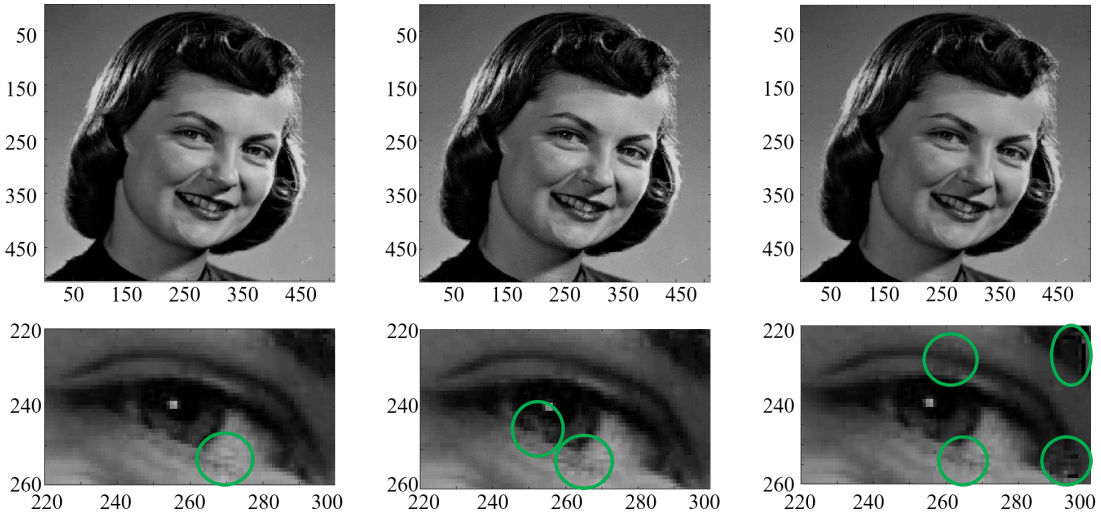
Additionally, for the same number of errors caused by approximate storage and a fixed data partition between reliable and approximate storage, we perform 10^4 Monte Carlo samples on one randomly selected image (other images show similar results). The difference between the samples is in the error locations, which are randomly distributed. In our simulation, we use the expected value of the number of errors with 1% error rate, so that the number of errors in each sample is fixed. Each sample uses the original image and the errors will be randomly distributed again. The quality degradation for each sample will be computed and we get the histograms of these degradation values, as shown in Fig. 18a. The quality degradation of these samples are seen to vary within a small range, as shown in Fig. 18a. Therefore, unlike traditional VLC-encoded JPEG, the ALC algorithm is less sensitive to the location of the errors that occur in less important data. In other words, the quality degradation of an image in ALC can be controlled.

8.1.3 Visual output of ALC scheme

Fig. 19 shows the results of using the ALC scheme to store the Girlface image under a 1% error rate. The three images correspond to different data partitions between reliable and approximate storage, resulting in different levels of quality degradation, as computed using Eq. (4), of approximately 5%, 10% and 20%. The values of PSNR also decrease for the larger degradation and the quality performance in terms of PSNR shows the same trend compared with SSIM. Therefore, only SSIM degradation is shown in the following parts. It can be seen from the figures that the errors in the former two cases are barely discernible. Even for the latter case, due to the built-in error-resilience of ALC, the errors affect the quality only in a localized manner instead of affecting the image globally, as in the VLC cases shown in Fig. 1. As the degradation increases, more obvious errors appear in the image, as shown in the zoomed-in details. For our other test images, Peppers, Tiger and Bird, the quality degradation results (not shown here) are similar, leading to the conclusion that a quality degradation of at most 10% is acceptable.

8.1.4 Compatibility with various quantization table JPEG techniques

More sophisticated JPEG techniques have been proposed, that use image-specific quantization tables [15] or that directly reduce the quality factor Q of the quantization table [2] to achieve further compression efficiency. Our ALC-based scheme is orthogonal to these techniques. Fig. 20 shows the quality degradation of ALC and JPEG schemes for various quality factor Q of the quantization table. These results show the average value of all 500 images from the dataset. For each image, the number of bits for the ALC scheme is constrained to be nearly the same but no larger than JPEG-based scheme. Our goal in this section is to quantify the quality degradation caused by quantization, and therefore, the impact of errors in approximate storage is not considered. For large quality factor Q, the ALC scheme



(a) 4.44% degradation
PSNR=37.1455

(b) 9.63% degradation
PSNR=35.7241

(c) 19.81% degradation
PSNR=34.5008

Fig. 19: Visual output of the ALC scheme using 1% error injection under different quality degradations in the original form (upper) and zoomed in on the details (lower). Due to space constraints, the upper row of figures is shrunk from the original size of the image. However, the errors can be observed at original size and more obvious errors occur in the figures with larger quality degradation.

has greater quality degradation than JPEG due to approximations in representation of the values. However, thanks to the additional bits adaptively added to compensate for quality loss, the degradation of JPEG can be followed by the ALC scheme closely. As the quality factor Q decreases, the limited length of codewords for the ALC scheme can cover most situations and the degradation of the ALC scheme is dominated by the quantization. Therefore, the number of bits for ALC-encoded images can be further reduced by decreasing the quality factor, as is done with JPEG.

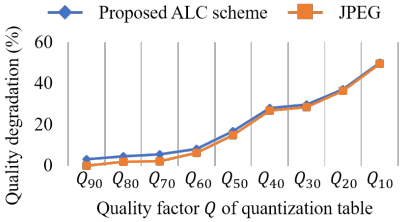


Fig. 20: Quality degradation for ALC scheme and JPEG with the same number of bits.

8.2 Impact of the error rate on ALC-based storage

For the case of a mainstream non-volatile storage technology, at different error rates, p, in approximate storage, Fig. 21 shows the number of bits is placed in reliable and approximate storage using the ALC algorithm and JPEG with a same quality degradation. The results are generated based on a representative image out of the 500 evaluated testcases in Imagenet. Other images show similar trends. Due to the sensitivity of VLC to errors, all of the data for JPEG must be saved in reliable storage. Thus, the number of bits for JPEG is independent of the error rates in approximate storage and remains constant for all cases.

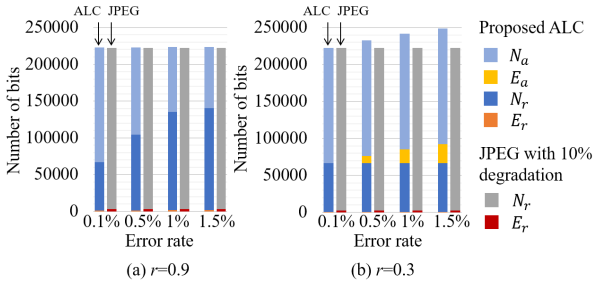


Fig. 21: Number of bits in reliable and approximate storage for the ALC scheme, as compared with VLC-encoded JPEG using reliable storage.

We use a stacked bar chart to show the different fraction of the number of bits in these two schemes with various colors, where $E_r + N_r$ correspond to the total number of bits in reliable storage, and $E_a + N_a$ correspond to the bits in approximate storage. For each subfigure, the left stacked bar represents our ALC scheme and the right one represents the JPEG scheme. With a degradation specification of 10%, evaluated according to Eq. (4), the total number of bits for our ALC-based scheme is nearly the same as JPEG at $r = 0.9$ and not more than 11% compared with JPEG in the worst case at $r = 0.3$. Moreover, the increase for the total number of bits is caused by E_a for the low-cost ratio $r = 0.3$ scenario.

When cost ratio r is large, for example $r = 0.9$, it is difficult to obtain a cost benefit for the weak ECC in the approximate storage compared to directly placing the data in reliable storage. Therefore, the limited or even no ECC is applied when r is large. For different error rates, the various data partitioning patterns are used to satisfy the quality requirement instead of changing the capability of the weak

ECC. As the error rate decreases along the x-axis, the total number of bits stays nearly the same and the percentage of bits that can be placed in reliable storage decreases from 68.0% at $e = 1.5\%$ to 29.9% at $e = 0.1\%$, as shown in Fig. 21.

When the cost ratio r is small, for example $r = 0.3$, the weak ECC protection allows fewer data to placed in reliable storage and the large cost difference between approximate storage and reliable storage can compensate for the cost of the parity bits for the weak ECC. As shown in Fig. 21b, for different error rates, the number of bits placed in reliable storage remains the same, using only the first data partitioning pattern defined in Section 5. As the error rate decreases, the number of parity bits required for the weak ECC decreases, so the number of bits placed in the approximate storage decreases.

8.3 Impact of the cost ratio r on ALC-based storage

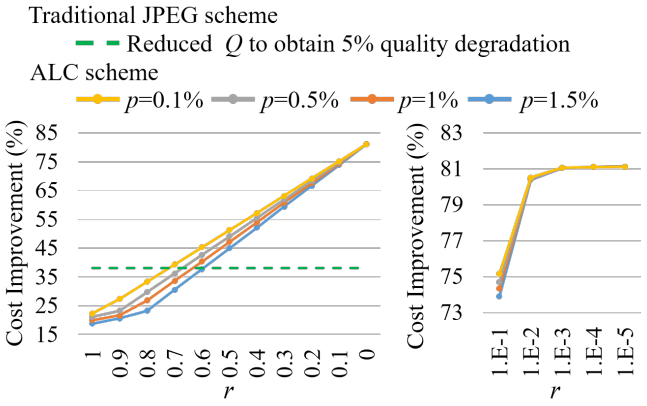
For any non-volatile memory technology, the relationship between the error rate, p, and the cost ratio, r, are strongly context-dependent. Even for the same type of storage technology, this relationship may vary from manufacturer to manufacturer, or from year to year. Meanwhile, the target of our paper is providing a methodology to allow the image to be compressed and placed in approximate storage. To preserve the generality of the approach and to provide a clear view of the landscape, we do not use the specific cost ratio or price, so that anyone can use any ratio when implementing our methodology. Therefore, rather than using a fixed function that maps p to r, we evaluate the proposed ALC-based storage scheme for general non-volatile solid-state storage by conducting a sweep of the value of the cost ratio, r, for different error rates, p.

Using the original JPEG scheme as the baseline, Fig. 22 shows the cost improvement for our approach compared to the VLC-based JPEG approach that uses a reduced Q value, which has a similar concept in [2]. In both cases, the quality degradation is constrained to be under 5% (Fig. 22a) and 10% (Fig. 22b). The results show the average of all 500 images from the Imagenet dataset. We sweep the value of r from 1 to 0 and also zoom in on the range from 10^{-1} to 10^{-5} to show the details for these smaller possible values. The dashed green line represents the conventional JPEG scheme with the adaptively reduced Q factor. The remaining lines show the ALC scheme with different error rates, p.

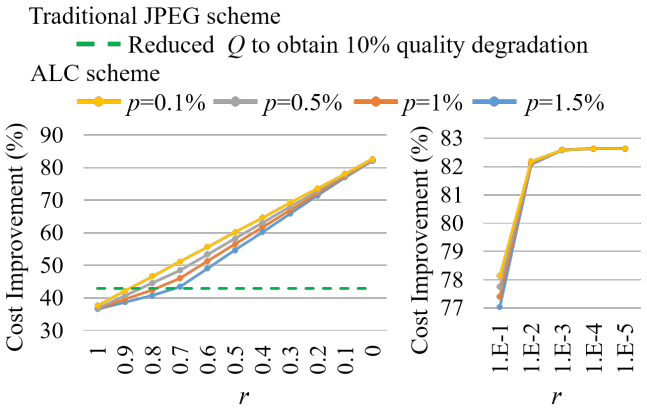
The JPEG version with a reduced Q value lowers the cost compared to the baseline due to its reduced storage requirements. This improvement is independent of r and p since all data are placed in reliable storage. When the degradation is constrained to be less than 5% and 10%, the results of reducing the Q value in the JPEG encoding changes the texture of the image, as shown in Fig. 23, but the image stills appears acceptable. The other test images, Girlface and Peppers, show similar results and are not included here due to space limitations.

For our ALC-based storage scheme, when r is large, there is no advantage compared to using JPEG with a smaller Q value since the data volume is similar or even larger than the reduced-Q JPEG case. However, as the cost ratio, r, gets smaller, the benefit of using approximate storage increases due to its lower cost, as shown in Fig. 22a and Fig. 22b.

Section 8.2 showed that, when r is close to 1, there is no advantage to partitioning the data between reliable and approximate memory since they cost about the same. When r is close to 0, the weak ECC protection in the approximate storage allows the first pattern of data partitioning for any error rate, which means that the number of bits placed in reliable storage is fixed. When the cost of the approximate storage is very low relative to the reliable storage, the total cost of the storage system is dominated by the number of bits in reliable storage. Therefore, the cost improvement converges as r approaches 0 so that all of the curves for the different error rates nearly coincide with each other in Fig. 22.



(a) 5% degradation



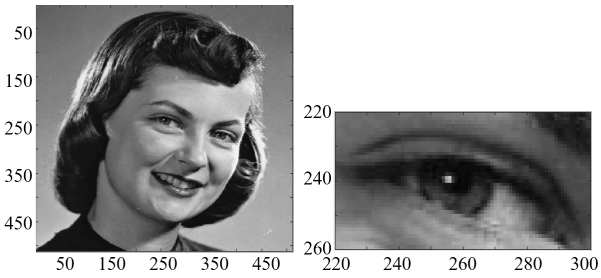
(b) 10% degradation

Fig. 22: Cost improvement for various values of r and p for the ALC scheme (left). The right graph expands the range from 10^{-1} to 10^{-5}.

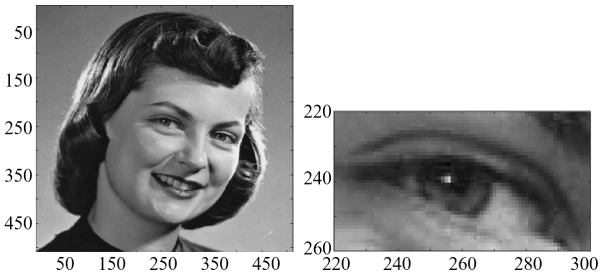
8.4 Limit to ALC-based storage

According to the analysis in Section 8.3, when the cost ratio is small, the entire cost of the storage system is dominated by the number of bits stored in reliable storage, which contains all of the DC coefficients, the block information, and critical data for the AC coefficients. For each image, the block information is fixed. When the cost ratio is small, the first pattern of data partitioning is always used, which fixes the critical data for the AC coefficients. To further reduce the cost, the only thing we can do is decrease the number of bits used to store the DC coefficients in the reliable storage.

In our case, all DC coefficients are still stored in reliable storage encoded using VLC, but the total number of bits



(a) $Q = 85$, 4.61% degradation



(b) $Q = 75$, 9.61% degradation

Fig. 23: Visual output of the JPEG encoded images with various Q factor reductions showing different quality degradations (Eq. (4)) in the original form (upper) and zoomed in on the details (lower). There is a slight degradation from (a) to (b), but both are acceptable.

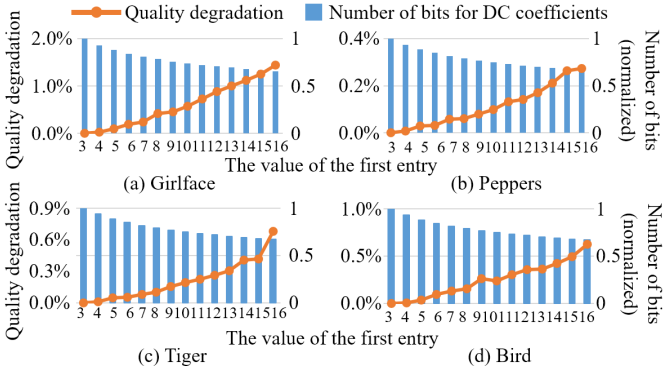


Fig. 24: Impact of the value of the first entry of the quantization matrix on the image quality and the number of bits for the DC coefficients.

allocated to the DC coefficients is decreased by reducing the resolution of the values. This requires a custom quantization matrix in which only the first entry (1, 1) is changed to a larger value compared to the default matrix. If we increase only the first entry while keeping the other entries the same as the $Q = 90$ matrix, the quality will degrade and the number of bits used to store the DC coefficients will decrease, as shown in Fig. 24. The first entry in the default quantization matrix with $Q = 90$ is 3, so the simulation starts from 3. This point has no quality degradation and requires the largest number of bits to store the DC coefficients. This figure shows that, when the value of the first entry increases to 16, the quality degrades less than 2% for all four images, while the number of bits decreases at least 34%. Fig. 25 shows that, when the first entry is 16, the visual output maintains an acceptable quality with only limited texture loss.

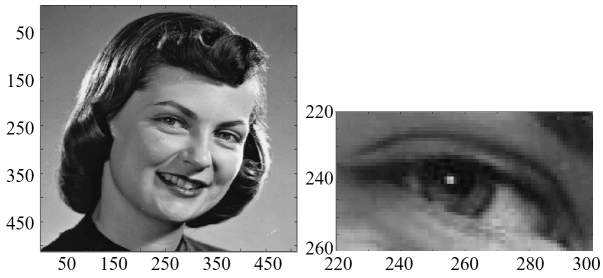


Fig. 25: The visual output for changing the first entry of the $Q = 90$ quantization matrix to 16.

We conclude, directly increasing the size of the first entry of the quantization table to decrease the total number of bits used for the DC coefficients can further improve the cost of the ALC encoding.

9 Conclusion

This work has introduced a new adaptive-length coding (ALC) algorithm that can be used to reduce the cost of a long-term "cold storage" system for image data by taking advantage of low cost memory devices that can have extremely high bit error rates. The ALC scheme partitions the bits used to store the encoded image into two classes – the most significant data that needs to be highly reliable, and the less important data that can tolerate some errors. Due to the built-in error-resilience of the ALC scheme, the errors that occur in the less important data, which is placed in low-cost approximate storage, affects the quality of the retrieved image in only a localized manner so that any changes in the images due to errors are barely discernible. Furthermore, when the difference in cost between the reliable storage and the unreliable (approximate) storage is large, the fraction of each encoded block that should be placed in the approximate storage depends on the error rate of the memory and the quality degradation limit desired by the user. Finally, the cost improvement of the ALC encoding increases as the cost ratio reduces and, when larger quality degradation can be tolerated, ALC can achieve even greater cost benefits.

The target of our paper is providing a methodology to allow the image to be stored in a compressed format and placed in approximate storage. Thus, we develop a new error resilient coding scheme. In this work, we examine only the fundamental idea instead of addressing system-level issues. Our objective is to make it clear that there is a viable trade-off between storage cost and image quality. Details of the implementation are specific to the precise storage scheme used and involve many subtleties beyond what can be considered in a single paper. These system-level concerns are a topic for future work.

Acknowledgments

This work was supported in part by National Science Foundation grant no. CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] "Mary meeker's annual internet trends report," 2014.
[2] E. Yan, *et al.*, "Customizing progressive jpeg for efficient image storage," in *USENIX Workshop on Hot Topics in Storage and File Systems*, 2017.
[3] D. Beaver, *et al.*, "Finding a needle in haystack: Facebook's photo storage," in *USENIX Symposium on Operating Systems Design and Implementation*, vol. 10, pp. 1–8, 2010.
[4] Q. Guo, *et al.*, "High-density image storage using approximate memory cells," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 413–426, 2016.
[5] D. Jevdjic, *et al.*, "Approximate storage of compressed and encrypted videos," in *Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 361–373, 2017.
[6] A. Ranjan, *et al.*, "Approximate storage for energy efficient spintronic memories," in *Proceedings of the ACM Conference on Design Automation Conference*, p. 195, 2015.
[7] H. Zhao, *et al.*, "Approximate image storage with multi-level cell stt-mram main memory," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 268–275, 2017.
[8] "Apple worldwide developers conference: High efficiency image file format," 2017.
[9] D. R. Horn, *et al.*, "The design, implementation, and deployment of a system to transparently compress hundreds of petabytes of image files for a file-storage service," in *USENIX Symposium on Networked Systems Design and Implementation*, pp. 1–15, 2017.
[10] "W^{3}techs reports," 2018.
[11] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. New York, NY: Van Nostrand Reinhold, 1992.
[12] "Google photos: High quality," 2018.
[13] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.
[14] C.-H. Huang, *et al.*, "Acoco: Adaptive coding for approximate computing on faulty memories," *IEEE Transactions on Communications*, vol. 63, no. 12, pp. 4615–4628, 2015.
[15] "Jpegmini," 2018.
[16] D. W. Redmill and N. G. Kingsbury, "The EREC: An error-resilient technique for coding variable-length blocks of data," *IEEE Transactions on Image Processing*, vol. 5, no. 4, pp. 565–574, 1996.
[17] Y. Yoo and A. Ortega, "Constrained bit allocation for error resilient JPEG coding," in *Proceedings of the IEEE Signals, Systems & Computers*, vol. 2, pp. 985–989, 1997.
[18] Y. Fang, "Erec-based length coding of variable-length data blocks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 10, pp. 1358–1366, 2010.
[19] Y.-S. Lee, *et al.*, "Error-resilient image coding (eric) with smart-idct error concealment technique for wireless multimedia transmission," *IEEE transactions on circuits and systems for video technology*, vol. 13, no. 2, pp. 176–181, 2003.
[20] L.-W. Kang and J.-J. Leou, "An error resilient coding scheme for jpeg image transmission based on data embedding and side-match vector quantization," *Journal of Visual Communication and Image Representation*, vol. 17, no. 4, pp. 876–891, 2006.
[21] Y.-S. Lee, *et al.*, "Hvlc: Error correctable hybrid variable length code for image coding in wireless transmission," in *Proceedings of the IEEE Acoustics, Speech, and Signal Processing*, vol. 4, pp. 2103–2106, 2000.
[22] H. Cai, *et al.*, "Error-resilient unequal error protection of fine granularity scalable video bitstreams," *EURASIP Journal on Advances in Signal Processing*, vol. 2006, no. 1, p. 045412, 2006.
[23] Z. Xue, *et al.*, "Error-resilient scheme for wavelet video codec using automatic roi detection and wyner-ziv coding over packet erasure channel," *IEEE Transactions on Broadcasting*, 2010.
[24] S. Jakubczak and D. Katabi, "Softcast: One-size-fits-all wireless video," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 449–450, 2011.
[25] S. Roman, *Introduction to Coding and Information Theory*. New York, NY: Springer-Verlag, 1997.
[26] H. Hashempour, *et al.*, "Error-resilient test data compression using tunstall codes," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 316–323, 2004.
[27] G. Dong, *et al.*, "Estimating information-theoretical nand flash memory storage capacity and its implication to memory system design space exploration," *IEEE Transactions on VLSI Systems*, vol. 20, no. 9, pp. 1705–1714, 2012.
[28] K. Cabeen and P. Gent, "Image compression and the discrete cosine transform, Math 45 Project, College of the Redwoods, Eureka, CA," 1998.
[29] M. Ghanbari, *Standard Codecs: Image Compression to Advanced Video Coding*. London,UK: The Institution of Engineering and Technology, 3rd ed., 2011.
[30] "Imagenet large scale visual recognition challenge," 2016.
[31] Z. Wang, *et al.*, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
[32] R. Micheloni, *et al.*, *Inside Solid State Drives (SSDs)*. New York, NY: Springer Dordrecht, 2012.
[33] H. Choi, *et al.*, "Vlsi implementation of bch error correction for multilevel cell nand flash memory," *IEEE Transactions on VLSI Systems*, vol. 18, no. 5, pp. 843–847, 2010.
[34] R. Micheloni, *et al.*, *Error correction codes for non-volatile memories*. New York, NY: Springer Dordrecht, 2012.
[35] S. S. Channappayya, *et al.*, "Rate bounds on ssim index of quantized images," *IEEE Transactions on Image Processing*, vol. 17, no. 9, pp. 1624–1639, 2008.
[36] S. Wang, *et al.*, "Ssim-motivated rate-distortion optimization for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 4, pp. 516–529, 2012.
[37] Y. Cai, *et al.*, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 94–101, 2012.

Qianqian Fan received the B.S. degree in electronic engineering from Shandong University, China, in 2013, and the M.S. degree in electrical engineering from University of Minnesota in 2015. Since June 2015, she has been working toward the Ph.D. degree at the University of Minnesota. Her research focuses on the storing error-resilient data into approximate memory or approximate storage, communication in extreme-scale systems, and corresponding performance evaluation and optimization.

David J. Lilja received the B.S. degree in computer engineering from Iowa State University in Ames, IA, USA, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in Urbana, IL, USA. He is currently a Professor of Electrical and Computer Engineering at the University of Minnesota in Minneapolis, MN, USA, where he also serves as a member of the graduate faculties in Computer Science, Scientific Computation, and Data Science. Previously, he served ten years as the head of the ECE department at the University of Minnesota, and worked as a research assistant at the Center for Supercomputing Research and Development at the University of Illinois, and as a development engineer at Tandem Computers Incorporated in Cupertino, California. He was elected a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and a Fellow of the American Association for the Advancement of Science (AAAS).

Sachin S. Sapatnekar (S'86, M'93, F'03) received the B. Tech. degree from the Indian Institute of Technology, Bombay, the M.S. degree from Syracuse University, and the Ph.D. degree from the University of Illinois. He taught at Iowa State University from 1992 to 1997 and has been at the University of Minnesota since 1997, where he holds the Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair in the Department of Electrical and Computer Engineering. He has received seven conference Best Paper awards, a Best Poster Award, two ICCAD 10-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence award and the SI University Research Award. He is a Fellow of the ACM and the IEEE.