applied sciences

MDPI

Article

Rate–Distortion-Optimized Arithmetic Coding for Parallel JPEG

Jakub Stankowski, Tomasz Grajek, Adrian Dziembowski *, Adam Grzelka and Damian Karwowski

Institute of Multimedia Telecommunications, Poznan University of Technology, 61-131 Poznań, Poland; jakub.stankowski@put.poznan.pl (J.S.); tomasz.grajek@put.poznan.pl (T.G.); adam.grzelka@put.poznan.pl (A.G.); damian.karwowski@put.poznan.pl (D.K.)

* Correspondence: adrian.dziembowski@put.poznan.pl

Abstract

Despite the emergence of sophisticated image coding standards, the original JPEG format remains the most popular and widely used standard for digital photography due to its minimal resource requirements and wide software/hardware support. This paper explores the enhancement of JPEG compression efficiency by optimizing the normative, but rarely used, arithmetic coding defined in the JPEG standard. The authors propose a refined encoding strategy that integrates Rate–Distortion-Optimized Quantization (RDOQ) into the arithmetic coding of JPEG. Moreover, an optimized arithmetic codec core (including precise bit-count estimation) is introduced. To address the increased computational complexity inherent in arithmetic entropy coding, a parallel processing framework that leverages restart markers to exploit multicore architectures has also been developed and implemented. Experimental results demonstrate that the presented approach provides significant bitrate reductions of approximately 20–30% compared to standard Huffman-based encoders while maintaining strict bitstream conformance. These findings prove that JPEG still leaves substantial space for compression efficiency improvement.

Keywords: JPEG; image compression; arithmetic coding; entropy coding; parallel processing; restart markers; rate–distortion optimization; RDOQ

1. Introduction

Over three decades have passed since the development of the JPEG standard [1], yet it remains the dominant format for digital image representation across various products and workflows [2–7]. While more modern compression techniques such as JPEG 2000 [8] , JPEG-XR [9], JPEG-XL [10], HEVC Intra (HEIF) [11], and AV1 Intra (AVIF) [12] offer superior rate–distortion performance, JPEG's enduring popularity is sustained by its simplicity and the extensive ecosystem of hardware and software that supports it.

As discussed in the literature, JPEG can be viewed as being "on steroids" when common optimization techniques are applied to its existing framework [13]. Because the rate control and entropy coding decisions are non-normative, there remains significant potential to enhance compression efficiency long after the standard's inception [14,15]. Previous research has successfully utilized encoder-only techniques, such as Rate–Distortion-Optimized Quantization (RDOQ) [16], to refine transform coefficients before entropy encoding. Such advancements prove that JPEG's performance can be tuned to reach levels close to modern standards, such as JPEG XR, which has only a fraction of

Academic Editor: Christos Bouras

Received: 25 October 2026
Revised: 22 November 2026
Accepted: 25 November 2026
Published: 26 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

the computational complexity [13].

While much of the existing research focuses on quantization matrices or Huffman table optimization [17–19], this work concentrates on the rarely used arithmetic coding mode of JPEG. Although Huffman coding is the de facto standard for JPEG, the specification also includes a normative arithmetic coding option that provides improved compression efficiency (Figure 1).

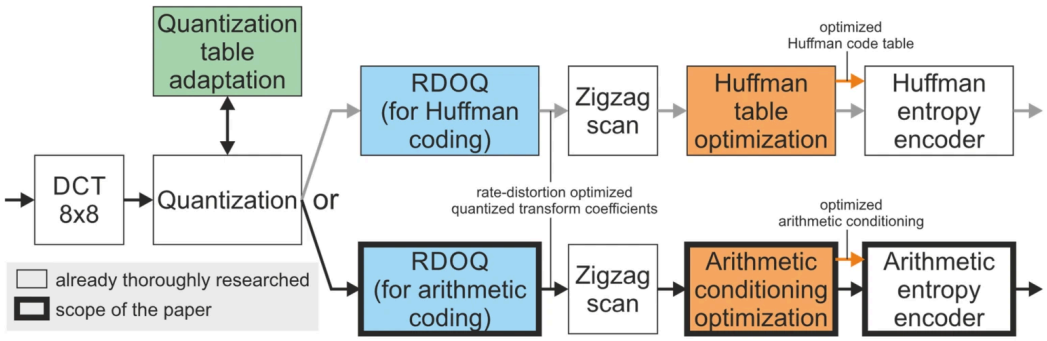


Figure 1. JPEG encoding pipeline for Huffman and arithmetic coding paths with possible encoder optimization (in color).

The core contributions of this work are: (1) a refined encoding strategy that integrates Rate–Distortion-Optimized Quantization (RDOQ) into arithmetic coding to maximize coding efficiency, (2) the introduction and evaluation of arithmetic conditioning optimization (ACO), and (3) the implementation of a parallelized framework for both the encoder and decoder to overcome the inherent computational complexity of arithmetic entropy coding.

The main motivation behind the research presented in this paper is the renewed interest in the use of arithmetic encoding in JPEG. Arithmetic encoding in JPEG was previously covered by several patents [1], which blocked its adoption due to potential licensing fees. After the expiration of these patents, several software implementations have added support for arithmetic encoding. However, these implementations did not take into account additional mechanisms—which are the subject of this work—that determine the final efficiency of arithmetic coding.

2. Arithmetic Coding in JPEG

The arithmetic coding within JPEG consists of several key components: binary arithmetic coding core, probability modeling of data symbols, transform coefficient block coding, and optional arithmetic conditioning.

2.1. Binary Arithmetic Coding Core

The arithmetic coding core within the JPEG codec is an example of an integer-based calculation binary arithmetic codec (BAC) that directly realizes the concept of the QM-codec [20]. The integer BAC stores its internal state in two integer registers: A and C. The A register expresses the size of the coding interval, whereas register C expresses the lower end of the interval. This approach avoids the requirements of infinite-precision arithmetic and allows for incremental emission of the encoded bitstream.

The JPEG arithmetic codec core works with 113 predefined least probable symbol (LPS) probability estimates (*Qe*), which are provided by the standard. The internal state of the A and C registers is updated based on the value of the currently encoded/decoded binary symbol (the so-called bin) and the corresponding *Qe* value.

After each coded bin, the arithmetic codec core checks whether a renormalization of

registers is needed (to avoid overflow). If so, the encoder rearranges their states and writes out part of the encoded data stream. On the decoder side, the renormalization results in reading new bytes from the input stream of data.

The BAC in JPEG defines two types of coded bins (as shown in Figure 2): first, with probability modeling, and second, with identical fixed-probability values (50%) for symbols 0 and 1. In the first mode, the BAC requires a probability model that contains two pieces of information: the LPS probability estimate (*Qe*) and the value of the most probable symbol (MPS). During the encoding/decoding process, the model is updated for every binary symbol (bin) according to the value of the encoded/decoded bin.

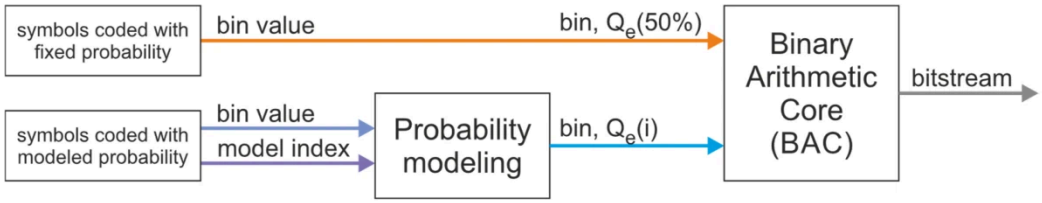


Figure 2. Illustration of two encoding paths of binary symbols used in JPEG's BAC.

2.2. Probability Modeling of Data Symbols

The probability model in JPEG contains the LPS probability estimate index *Qe*(*i*) and the value of the most probable symbol (MPS). The probability modeling in JPEG BAC is standardized as a state machine. Each state of this machine is associated with a specific probability value of the LPS. The model can be in one of 226 states—113 possible LPS probability indices and two possible MPS values (0 or 1). For model updates, the standard defines a table describing transitions between machine states according to the current state and the value of the coded bin.

At the beginning of encoding/decoding, all models are set to the same initial state—*MPS* = 0 and probability index $i = 0$, corresponding to probability $p(LPS) = 50\%$.

The JPEG entropy encoder requires hundreds of independent probability models (state machines) to encode/decode an image. Coding of DC and AC coefficients requires the use of 49 and 245 probability estimators, respectively. Moreover, in a typical case, separate encoders are used for different component types (luma, chroma), leading to 588 models for the shared chroma case and 882 for the case where each chroma has its own set of models. This is a substantial number, especially since they are used for a block of 8 × 8 DCT (Discrete Cosine Transform) coefficients only. The excessive number of models leads to suboptimal efficiency of statistical data modeling due to the context dilution problem, but it is definitely forgivable in the case of such a mature technique. For comparison, HEVC requires 154 models [21,22] for all its syntax elements.

2.3. Transform Coefficient Block Coding

Arithmetic coding in JPEG defines separate paths for DC and AC coefficients and uses different sets of probability models (state machines) for each coefficient type.

The DC coefficients are encoded differentially (similarly to the Huffman-based path), so ΔDC is calculated as the difference between the current and previous value. The ΔDC values are then categorized into five categories based on previously encoded ΔDC. Categories include: big negative, small negative, close to zero, small positive, and big positive. Each category has its own set of four probability models (S) used to encode bins signaling the following conditions: $\Delta DC \neq 0$, $sign(\Delta DC)$, $\Delta DC > 1$, and $\Delta DC < -1$. If the magnitude of ΔDC is bigger than 1, the encoder uses the re$sign(\Delta DC)$aining models (X and M) to encode $abs(\triangle DC) - 1$.

When AC coefficients are considered, the encoding is more complex. For each possible AC coefficient index k, a separate group of three models (S-models) is used to encode bins signaling the following: end of block (EOB), $AC_k \neq 0$, and $abs(AC_k) > 1$. In the case of AC coefficients, the sign bit is encoded in a constant (50%) probability mode. If the magnitude of AC_k is bigger than 1, the encoder uses a similar scheme to ΔDC encoding, however, with a noticeable difference. There are two sets of X- and M-models. The first is used for lower AC coefficient indices ($k \leq K_x$) and the second for higher indices ($k > K_x$), where the default $K_x = 5$.

2.4. Arithmetic Conditioning

Similar to Huffman tables transmitted in "define-Huffman-tables marker" (DHT), JPEG allows the transmission of so-called arithmetic conditioning as "define-arithmetic-coding-conditioning marker" (DAC). The arithmetic conditioning for DC allows the selection of two bound values, L (lower) and U (upper), used to assign ΔDC value to a specific category (see 2.3). The default values are $L = 0$ and $U = 1$, and allowed values are restricted by the formula $0 \leq L \leq U \leq 15$, giving a total of 136 possible combinations.

In the case of arithmetic conditioning for AC encoding, the K_x value can be transmitted. K_x sets the boundary between low-frequency ($k \leq K_x$) and high-frequency ($k > K_x$) AC coefficients and influences the selection of probability models. K_x can be in the range $1 \leq K_x \leq 63$.

3. Rate–Distortion-Optimized Quantization (RDOQ)

The Rate–Distortion-Optimized Quantization (RDOQ) [23] is an encoder-only, non-normative technique that enhances compression efficiency by analyzing quantized transform coefficients before final entropy encoding. Such an optimization can be applied to any compression technique.

The RDOQ algorithm performs rate–distortion optimization (RDopt) within a single DCT transform block. It analyzes a block of quantized transform coefficients and tries to adjust the values of AC coefficients in order to reduce the rate–distortion cost (*RDcost*) of block encoding. For each non-zero coefficient AC_k, up to four options are evaluated: $c_1 = AC_k$, $c_2 = AC_k - 1$, $c_3 = AC_k + 1$, and $c_4 = 0$ (Figure 3). For each c_x option, the *RDcost* for the whole block is calculated. *RDcost* is based on the number of bits (R) used to encode the considered transform block and the distortion (D) of the reconstructed image block (typically calculated as SSD, Sum of Squared Differences, which is a widely used metric in RDOQ and correlates well with perceived visual quality). The R and D components are combined using a Lagrange multiplier (λ) [23]:

$$RDcost = D + \lambda \cdot R, \quad (1)$$

where the selection of the Lagrange multiplier (λ) strictly follows the empirical methodology established in our previous work [16]. To adapt the λ parameter to the characteristics of each encoded image, its calculation is performed dynamically at the picture level. The λ multiplier is derived by pre-calculating distortion and number of bits for Q and $Q - 1$ values. Next the slope of the rate–distortion curve is derived by:

$$\lambda = -\frac{D_Q - D_{Q-1}}{R_Q - R_{Q-1}}, \quad (2)$$

where D_x and R_x represent the distortion and the number of bits needed to encode the entire image for a given quality factor setting, respectively [16].

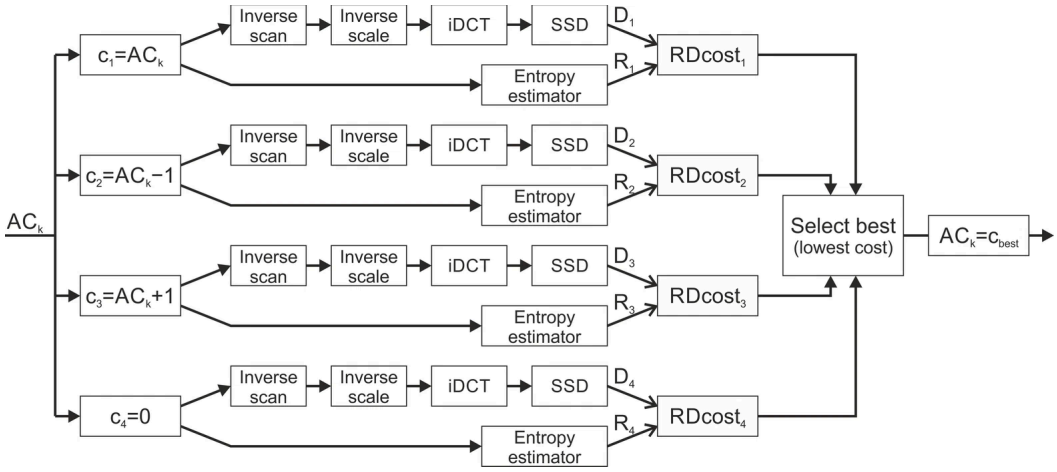


Figure 3. Simplified diagram of the exemplary RDOQ algorithm.

Finally, the variant with the lowest *RDcost* is selected (Figure 3). Typically, this procedure is repeated iteratively from high- to low-frequency coefficients. A detailed description of the RDOQ adoption in a JPEG encoder using Huffman entropy coding may be found in [13,16].

4. JPEG Arithmetic Encoder Challenges

This paper aims to incorporate RDOQ into JPEG with arithmetic coding. The application of RDopt in the JPEG encoder requires knowledge about the number of bits used to encode the selected transform block and the distortion of the reconstructed image block. Unfortunately, the typical use of an arithmetic encoder makes the precise calculation of the number of encoded bits practically impossible. The arithmetic core writes out bytes only if needed to avoid overflow in integer registers. For example, bytes emitted during a single DCT block encoding can also contain data from the previous block of the image, or, even worse, no data can be written out. This leads to an obvious conclusion that for the precise calculation of bits spent on a single DCT block, a dedicated algorithm has to be developed. In this paper, it is called an arithmetic bit-count estimator and is described in the following section.

4.1. Arithmetic Bit-Count Estimator

The integer binary arithmetic core emits bytes in batches during the renormalization phase. Therefore, it is impossible to get clear information about the number of bits used to encode a single bin of data. Furthermore, arithmetic encoding can use a fractional number of bits to represent a selected bin.

The only reasonable solution to this problem is to calculate an approximate number of bits based on specific arithmetic coding properties. Fortunately, the approximated number of bits can be calculated based on symbol probability.

JPEG uses 113 predefined LPS probability estimates ($Qe(i)$) and does not directly provide probabilities corresponding to each of the 113 probability indices. However, the standard also defines the Qe value used for encoding the sign of AC coefficients with a fixed 50% probability—$Qe(50\%)$. For every probability index i, the LPS probability $p_{LPS}(i)$ can be calculated in the following way:

$$p_{LPS}(i) = \frac{Qe(i)}{2 \cdot Qe(50\%)} \quad (3)$$

Knowing the value of $p_{LPS}(i)$, one can calculate the number of bits used to represent

a single bin. Since this number depends on the bin value and the current MPS value, two values have to be calculated:

$$b(i, Bin) = \begin{cases} -log_2(1 - p_{LPS}(i)) & if \ \ Bin = MPS \\ -log_2(p_{LPS}(i)) & if \ \ Bin \neq MPS \end{cases} \quad (4)$$

The calculated numbers of bits are not integers. This leads to inaccuracies due to floating-point number accumulation. Therefore, we decided to represent $b(i,Bin)$ as fixed-point numbers and use integer arithmetic. We chose 15-bit precision of the fractional part and calculated the fixed-point values $B(i,Bin)$ as follows:

$$B(i, Bin) = \ round\left(b(i, Bin) \ \cdot 2^{15}\right) \quad (5)$$

The use of the RDopt mechanism requires encoding the DCT block multiple times. This leads to a new class of problems, which was not present when using Huffman codes [13,16]. The Huffman estimator is stateless—it just calculates the number of bits that are used to represent a symbol. However, the arithmetic encoder (and estimator) has its unique state (which determines the efficiency of encoding subsequent data symbols). The state refers to the exact values of arithmetic core registers and the current stage of the probability models (which is constantly updated). Since the design of the arithmetic core in JPEG makes model updating dependent on the A register, the estimator also needs to keep track of it. As a result, the arithmetic estimator has to be designed with the possibility to back up and restore its state. Since the calculation of the number of bits is only an approximation, the estimator's precision is crucial and needs to be evaluated (see Section 6.1).

4.2. Arithmetic Conditioning Optimization (ACO)

JPEG encoders using a Huffman-based entropy codec can improve the compression ratio by determining optimal Huffman code tables and sending them in the bitstream to the decoder.

In the case of JPEG with arithmetic coding chosen, the arithmetic conditioning optimization (ACO) can be seen as a functional equivalent of Huffman table optimization, especially since the size of the DAC marker is significantly smaller than the size of the entire Huffman tables. DAC contains only one byte for AC or DC conditioning, so the entire marker uses 8 bytes for the common chroma case (or 10 bytes if each chroma has its own conditioning). The ACO requires a global decision for the entire picture and is performed before the final entropy encoding of image data.

5. Implementation

In order to evaluate JPEG arithmetic encoding performance in combination with RDOQ and ACO, a production-quality JPEG encoder and decoder were implemented based on the Huffman-only framework described in [24]. This baseline was substantially extended to incorporate the three core contributions of this work: (1) the design of a novel fixed-point arithmetic bit-count estimator that enables efficient RDOQ within the arithmetic pipeline, (2) the integration of arithmetic conditioning optimization (ACO), and (3) the adaptation of the multi-threaded architecture to properly manage the stateful nature of arithmetic entropy coding. The effort to create a very fast, highly optimized JPEG encoder was needed to properly evaluate both compression efficiency and computational complexity tradeoffs.

5.1. Optimized JPEG Encoder/Decoder

The developed encoder contains a Rate–Distortion-Optimized Quantization (RDOQ) implementation for both the Huffman and arithmetic coding paths. Moreover, the implementation includes both Huffman table optimization (HTO) [13] and arithmetic conditioning optimization (ACO). The implementation of the Huffman coding path was intended to provide a proper reference for subsequent studies of the efficiency and complexity of JPEG with arithmetic coding.

For comparison against a real-world case, most performance-critical functions (DCT, iDCT, quantization, inverse scaling, zigzag scan, inverse scan, etc.) were implemented using AVX512 instructions [25] available on modern x86-64 processors.

Additionally, highly optimized encoders, decoders, and estimators for both Huffman and arithmetic coding were implemented. The optimizations to the Huffman codec are out of the scope of this paper, so only the arithmetic part will be described in the following paragraphs.

As mentioned, the probability modeling in JPEG is based on a state machine with two internal variables: probability estimate index (i) and most probable symbol (MPS). Since there are 113 probability indices, the index (i) can be stored using 7 bits (out of 8 in a byte). The last bit can be used to store MPS. The state is then represented as $State = (i<<1) + MPS$.

The transition between states during model update is described in the JPEG standard by a complex table (D.3 in [1]) describing the next index based on encoded value (MPS or LPS) and information on whether MPS should be altered. Using this approach directly leads to many branches in the critical update path.

In this paper, a different approach is proposed. The state transition is based on a lookup table (LUT). The model state combines probability index and MPS, so there are 226 possible states. There are also two possible values of encoded bins (0 and 1). Finally, the LUT contains 452 one-byte entries. An additional table is used by the arithmetic estimator. It stores a fixed-point representation of the numbers of bits used to encode a bin —$B(i,Bin)$. This table also contains 226 entries, each 4 bytes wide. Obviously, there is a third, unavoidable table containing 113 entries representing $Qe(i)$ values, also using 4 bytes per entry.

All mentioned tables occupy a reasonable 1808 bytes of memory and will fit within the 1st-level cache of any modern CPU. This table-based approach allowed us to avoid unnecessary branches or complex calculations in critical code paths.

5.2. Parallel JPEG Encoder/Decoder with Arithmetic Coding

Lastly, both the encoder and decoder were implemented in a way that allowed the use of parallel (multi-threaded) processing and exploited the possibilities of high-core-count CPUs. The JPEG standard was created when no one was expecting that multicore processors would become affordable for end-user devices, and it does not contain explicit parallelization tools. However, its authors provided restart markers. Restart markers were meant to be used to improve robustness against transmission errors by splitting the entire image into independent segments containing an integer number of MCUs (minimum coded units). The use of restart markers was contrary to the creators' intentions—to parallelize encoding and decoding. In the proposed approach, each segment between restart markers is encoded or decoded using a dedicated thread [24].

The entire implementation effort allows for accurate comparison of coding efficiency/performance tradeoffs for high-performance JPEG implementations. This element increases the usefulness of the presented solution.

6. Experimental Evaluation and Results

The experiment was conducted based on images from the Netflix/MPEG test sequence dataset [26]. It contains 16 diverse sequences (*BasketballDrive*, *blue_sky*, *BQTerrace*, *Cactus*, *Kimono1*, *ParkScene*, *pedestrian_area*, *riverbed*, *rush_hour*, *station2*, *sunflower*, *tennis*, *toys_and_calendar*, *tractor*, *vintage*, *walking_couple*) in FullHD format with 4:2:0 chroma subsampling, yielding a total of ~3200 images.

The experiments were performed for three operating points related to the strength of the quantization used, respectively, for $Q_L \in \{20, 25, 30, 35\}$, $Q_M \in \{50, 55, 60, 65\}$, and $Q_H \in \{80, 85, 90, 95\}$. In JPEG, a Q value means "quality" and a higher Q is associated with weaker quantization resulting in a higher bitrate. The tests were performed on a 12-core (24 threads) processor (AMD Ryzen 7900X). The software was compiled using MSVC 19.44 with default optimizations enabled in the "Release" mode. Each encoding was performed independently, and only one experiment was run at a time. The time was measured using the processor's TSC (Time Stamp Counter), read by using the RDTSCP instruction [27].

6.1. Accuracy of Arithmetic Bit-Count Estimator

The first step of evaluation focuses on measuring the arithmetic estimator accuracy. Two experiments were performed. The first experiment measures the performance of the arithmetic estimator core (corresponding to the arithmetic encoder core) by encoding a pseudo-random stream of bins. The second experiment measures the accuracy of the entire arithmetic estimator by encoding a pseudo-random set of transform coefficient blocks (with coefficient distribution resembling the coefficients representing a block of a natural image, including different Q settings). Results were gathered for different numbers of bins (2^{10}–2^{19}) and blocks (2^{6}–2^{16}). For each input length, 4096 independent random samples were gathered. The results are presented as relative estimation error (ΔE) calculated based on the number of encoded bits and the number of estimated bits:

$$\Delta E = 100\,\% \cdot \frac{NumBitsEnc - NumBitsEst}{NumBitsEnc} \tag{6}$$

As shown in Table 1, the arithmetic estimator achieves sufficient accuracy (ΔE below 1% for the random bin test and ΔE ≈ 3.81% for the random block test). The estimator systematically underestimates the number of bits required to represent a transform coefficient block by ~3.81%. Since the pseudo-random DCT block generator was designed based on all of the test images across all evaluated Q levels, the 3.81% bias was representative. Given Equations 1 and 2, in which R is subject to the same multiplicative error, these errors cancel out when computing the *RDcost*, and the cost is therefore determined correctly. Using the same estimator for λ and *RDcost* preserves the relative ordering of *RDcost*, ensuring that global RDOQ decisions remain unaffected. Figure 4 shows that the distribution of estimation error resembles a normal distribution.

Table 1. Estimation error in arithmetic bit estimator module.

Random Bin Evaluation		Random Block Evaluation	
Number of Bins	**ΔE [%] Average (Std. Deviation)**	**Number of DCT Blocks**	**ΔE [%] Average (Std. Deviation)**
1024	1.10 (±0.642)	64	3.86 (±0.154)
2048	1.00 (±0.435)	128	3.88 (±0.112)
4096	0.95 (±0.293)	256	3.86 (±0.079)
8192	0.92 (±0.213)	512	3.83 (±0.057)
16,384	0.90 (±0.147)	1024	3.81 (±0.041)
32,768	0.90 (±0.105)	2048	3.80 (±0.029)
65,536	0.89 (±0.074)	4096	3.80 (±0.020)
131,072	0.89 (±0.052)	8192	3.79 (±0.014)
262,144	0.89 (±0.037)	16,384	3.79 (±0.010)
524,288	0.89 (±0.026)	32,768	3.79 (±0.008)

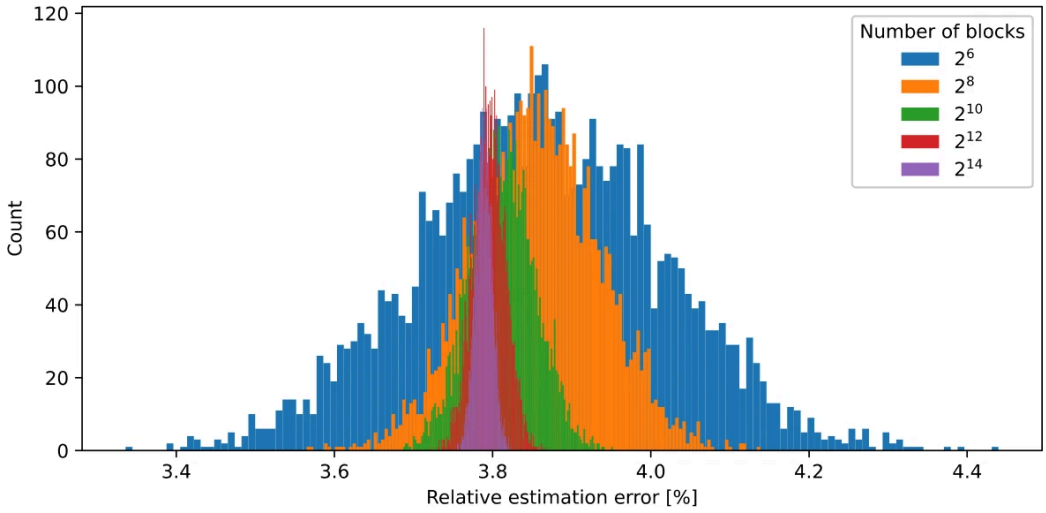


Figure 4. Distribution of relative estimation error (ΔE).

6.2. Throughput of Entropy Codec

The second step of evaluation was to perform a synthetic measurement for the throughput of the entropy stage, for both encoding and decoding. These studies also covered the mechanism of estimating the number of bits produced by the entropy encoder. As shown in Table 2, the arithmetic coding is 7 ÷ 9 times slower than Huffman-based coding, slowing down the entire encoding/decoding process. One can also notice that our approach to fast estimation of the number of bits allows for ~20% throughput improvement when compared to full entropy encoding.

Table 2. Entropy coding and arithmetic core throughput.

DCT Coefficient Block Coding Throughput [×10^6 Blocks/s]			
Entropy Codec	**Encoder**	**Decoder**	**Estimator**
Huffman	3.628	4.269	5.804
Arithmetic	0.527	0.502	0.636
Arithmetic Core Throughput [×10^6 Bins/s]			
Mode	**Encoder**	**Decoder**	**Estimator**
Modeled	196.9	178.4	268.6
Fixed (50%)	244.0	237.9	411.0

6.3. Efficiency of Entropy Coding Method

An important part of the research was to assess the efficiency of a given entropy compression method, including the relative execution time of the method. The performance comparison between different encoding variants was measured as the Bjøntegaard delta [28] and presented as the ΔRate for the luma component. The reported values were obtained by calculating the BD rate independently for each test image and computing the arithmetic mean across the entire dataset. Negative values mean coding efficiency improvement (percentage of bitrate reduction for the same quality).

Table 3 presents a comparison of coding performance and processing time between arithmetic coding and Huffman coding in plain JPEG. Switching to arithmetic coding allows for saving ~18% of bits, with the cost of ~3-times-longer encoding and decoding times. This bit saving is very large considering that it is achieved within a lossless coding part of the algorithm.

Table 3. The comparison of coding performance and processing time for arithmetic coding vs. Huffman coding in plain JPEG.

Scenario	**ΔRate [%]**	**Encoding Slowdown [x]**	**Decoding Slowdown [x]**
Q_L	−25.24	2.14	1.84
Q_M	−15.35	2.76	2.41
Q_H	−12.55	3.91	3.80

6.4. Influence of Encoder Control Mechanism

The more interesting aspect is the influence of encoder optimizations (RDOQ and/or HTO/ACO) on compression efficiency and processing times. Table 4 contains results for the Huffman-based path (for comparison) and the arithmetic-based one. Depending on the scenario, the optimized quantization (RDOQ) allows for reducing the image size by ~8% at the cost of 10 ÷ 20× encoder slowdown—this is comparable to results achieved for the Huffman-based case.

Table 4. Summary of encoder compression efficiency improvements.

Scenario	Huffman Entropy Coding						Arithmetic Entropy Coding					
	RDOQ		HTO		RDOQ + HTO		RDOQ		ACO		RDOQ + ACO	
	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]
Q_L	−8.54	9.40	−15.01	1.19	−23.02	9.52	−7.27	9.40	−0.12	36.97	−7.510	41.23
Q_M	−8.64	12.30	−5.94	1.24	−14.18	12.50	−7.05	11.92	−0.10	43.07	−7.153	49.59
Q_H	−10.37	21.27	−1.85	1.34	−11.69	21.63	−8.52	20.30	−0.16	49.63	−8.639	65.54

Surprisingly, the results gathered for ACO are slightly disappointing. ACO can reduce the data size by an insignificant 0.1% at the cost of an excessive complexity increase. While ACO completes the theoretical optimization framework for arithmetic coding, it is not recommended for practical deployment due to its unfavorable cost-to-benefit ratio. Therefore, ACO was excluded from further experiments, as it provides no significant coding gains while its parameter estimation is excessively time-consuming.

The comparison with all reasonable performance optimizations included is summarized in Table 5. When compared to "plain" Huffman-based JPEG, the proposed combination of arithmetic coding with RDOQ reduces the size of the encoded image by ~25%. Even when compared to improved encoders (as in [13,16]), the gain is still noticeable, at ~10%.

Table 5. Coding performance improvements for the JPEG encoder with full optimization.

Scenario	Arithmetic + RDOQ vs. plain Huffman		Arithmetic + RDOQ vs. Huffman + RDOQ + HTO		Arithmetic + RDOQ
	ΔRate [%]	Slowdown Enc [x]	ΔRate [%]	Slowdown Enc [x]	Encoding Time [ms]
Q_L	−31.08	20.31	−10.85	2.11	73.46
Q_M	−21.49	33.22	−8.61	2.63	163.35
Q_H	−20.00	79.70	−9.46	3.66	675.11

It is important to note that while relative slowdowns appear high, the absolute encoding time for a single FullHD image on a modern CPU remains in the order of fractions of a second, which is highly viable for non-real-time storage and archiving applications.

6.5. Encoder Performance in Multi-Threaded Implementation

All previously presented results were obtained for a single-threaded scenario. However, most modern devices are equipped with multicore processors, and parallelization is the obvious way to overcome the so-called "single-threaded performance wall". Therefore, the prospect of parallelization of both Huffman-based and arithmetic-based optimized JPEG encoders was measured. To enable parallel encoding, the image was divided into independent segments using restart markers. The size of each segment was set to one row of MCUs [24].

As shown in Table 6, the parallelization results are encouraging. The processor used during evaluation (AMD Ryzen 7900X) is equipped with 12 physical cores, while each core is capable of executing two concurrent threads (due to the use of Simultaneous Multi-Threading—the SMT approach). For a theoretical, straightforward, 12-core CPU, the theoretical maximum speedup is 12×. However, the presence of SMT and utilization-dependent clocking makes this estimation much more complex. According to test results performed on a similar architecture [29], the average SMT gain is about 13%; however, for some workflows, the gain reaches 72%. Since the arithmetic encoding contains a lot of data-dependent branches (causing severe branch misprediction penalties) we can assume that the usage of SMT can be very beneficial and estimate a ~21× speedup as a reasonable maximum for 24-thread execution. As shown in Table 6, the speedup achieved for Q_H (high number of non-zero transform coefficients) is very close to the estimated theoretical maximum.

Table 6. JPEG parallel-variant evaluation versus single-thread one.

Scenario	Huffman + RDOQ + HTO			Arithmetic + RDOQ		
	ΔRate [%]	Speedup Enc [x]	Speedup Dec [x]	ΔRate [%]	Speedup Enc [x]	Speedup Dec [x]
Q_L	0.29	9.06	6.96	1.51	13.73	10.13
Q_M	0.15	11.36	7.30	1.30	16.56	12.46
Q_H	0.06	16.62	8.80	1.04	20.14	17.61

Obviously, the introduction of restart markers slightly reduced the compression efficiency (~1.3% for the arithmetic encoder, mostly due to reinitialization of probability models at the beginning of each segment—see Table 6). On the other hand, the multicore speedup compensates for the slowdown caused by encoder optimizations (RDOQ and HTO).

7. Conclusions

This paper presents a highly optimized JPEG encoder that integrates the standard arithmetic coding mode with a non-normative Rate–Distortion-Optimized Quantization (RDOQ) technique. To enable efficient rate–distortion optimization within the arithmetic coding pipeline, a high-precision bit-count estimation method based on fixed-point lookup tables was developed. Experimental results indicate that the proposed estimator achieves a small relative error (with correctable constant bias), confirming its accuracy and suitability for practical implementations.

Experimental evaluation demonstrates that the proposed combination of arithmetic coding and RDOQ achieves a significant bitrate reduction of approximately 25% compared to conventional Huffman-based JPEG encoding. Furthermore, when compared against state-of-the-art Huffman encoders employing both RDOQ and optimized Huffman tables, the arithmetic coding-based approach provides a gain of approximately 10% in coding efficiency.

While arithmetic coding is undoubtedly more computationally demanding, leading to a 7 ÷ 9× reduction in entropy coding throughput, this approach is highly practical for scenarios such as server-side batch archiving, where maximizing storage savings outweighs the importance of real-time encoding speeds. Moreover, this overhead is effectively mitigated through parallelization. Specifically, restart markers are employed to partition the image and distribute the encoding workload across multiple CPU cores, resulting in speedups that compensate for the increased complexity introduced by the optimization techniques.

These results confirm that, when combined with sophisticated optimization strategies and modern parallel implementations, the compression efficiency of the legacy JPEG standard can be further improved.

Author Contributions: Conceptualization, J.S., D.K., and T.G.; methodology, J.S. and D.K.; software, J.S.; validation, A.G.; visualization: A.D. and J.S.; writing, J.S., T.G., D.K., A.D. and AG. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the Ministry of Science and Higher Education of the Republic of Poland.

Data Availability Statement: The Netflix/MPEG test sequences used in this study are publicly available and their specific names are listed in Section 6. The source code and evaluation scripts used to support the findings are available from the corresponding author upon reasonable request.

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

References

1. *ISO/IEC 10918-1*; Information Technology—Digital Compression and Coding of Continuous-Tone Still Images—Requirements and Guidelines. International Organization for Standardization: Geneva, Switzerland, 1992.
2. Yue, Y.; Zhang, M.; Lai, P.; Di, F. A JPEG Reversible Data Hiding Algorithm Based on Block Smoothness Estimation and Optimal Zero Coefficient Selection. *Appl. Sci.* **2025**, *15*, 10282. [Google Scholar] [CrossRef]
3. Borkowski, D.; Janczak-Borkowska, K. Reduction of JPEG Artifacts using BSDEs. In *Computer Science Research Notes, Proceedings of the 29th WSCG*; Václav Skala–UNION Agency: Plzen, Czech Republic, 2021; Volume 3101. [Google Scholar]
4. Jiao, S.; Jin, Z.; Chang, C.; Zou, W.; Li, X. Compression of phase-only holograms with JPEG standard and deep learning. *Appl. Sci.* **2018**, *8*, 1258. [Google Scholar] [CrossRef]
5. Hudson, G.; Léger, A.; Niss, B.; Sebestyén, I.; Vaaben, J. JPEG-1 standard 25 years: Past, present, and future reasons for a success. *J. Electron. Imaging* **2018**, *27*, 040901. [Google Scholar] [CrossRef]
6. W3Techs, Usage Statistics of JPEG for Websites. 2024. Available online: https://w3techs.com/technologies/details/im-jpeg (accessed on 2026-11-15).
7. Urbaniak, I.A. Using Compressed JPEG and JPEG2000 Medical Images in Deep Learning: A Review. *Appl. Sci.* **2024**, *14*, 10524. [Google Scholar] [CrossRef]
8. *ISO/IEC IS 15444*; Information Technology—JPEG 2000 Image Coding System. International Organization for Standardization: Geneva, Switzerland, 2019.
9. *ITU-T Rec. T.832*; Information Technology—JPEG XR Image Coding System. International Telecommunication Union: Geneva, Switzerland, 2020.
10. *ISO/IEC 18181:2022*; Information Technology—JPEG XL Image Coding System. International Organization for Standardization: Geneva, Switzerland, 2022.
11. *ISO/IEC 23008-12*; Information Technology—High Efficiency Coding and Media Delivery in Heterogeneous Environments—Part 12: Image File Format. International Organization for Standardization: Geneva, Switzerland, 2017.
12. Concolato, C.; Klemets, A. (Eds.) *AV1 Image File Format (AVIF). Alliance for Open Media*; 2021. Available online: https://aomediacodec.github.io/av1-avif/.
13. Richter, T. JPEG on STEROIDS: Common optimization techniques for JPEG image compression. In Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 25–28 September 2016; pp. 61–65. [Google Scholar]
14. Beach, A.; Owen, A. *Video Compression Handbook*, 2nd ed.; Peachpit Press: San Francisco, CA, USA, 2018. [Google Scholar]
15. Richardson, I. *Video Codec Design: Developing Image and Video Compression Systems*; Wiley: Chichester, UK, 2002. [Google Scholar]
16. Grajek, T.; Stankowski, J. Rate-Distortion Optimized Quantization in Motion JPEG. In Proceedings of the 30th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2022, Plzen, Czech Republic, 17–20 May 2022. [Google Scholar]
17. Luo, X.; Talebi, H.; Yang, F.; Elad, M.; Milanfar, P. The Rate-Distortion-Accuracy Tradeoff: JPEG Case Study. In Proceedings of the 2021 Data Compression Conference (DCC), Snowbird, UT, USA, 23–26 March 2021. [Google Scholar]
18. Safonov, I.V.; Kurilin, I.V.; Rychagov, M.N.; Tolstaya, E.V. Fast Control of JPEG Compression Rate. In *Document Image Processing for Scanning and Printing*; Signals and Communication Technology; Springer: Cham, Switzerland, 2019. [Google Scholar]
19. Wang, Q.; Liu, P.; Zhang, L.; Cheng, F.; Qiu, J.; Zhang, X. Rate–distortion optimal evolutionary algorithm for JPEG quantization with multiple rates. *Knowl.-Based Syst.* **2022**, *244*, 108500. [Google Scholar] [CrossRef]
20. Pennebaker, W.B.; Mitchell, J.L.; Langdon, G.G.; Arps, R.B. An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder. *IBM J. Res. Dev.* **1988**, *32*, 717–726. [Google Scholar] [CrossRef]
21. Sze, V.; Budagavi, M. High Throughput CABAC Entropy Coding in HEVC. *IEEE Trans. Circuits Syst. Video Technol.* **2012**, *22*, 1778–1791. [Google Scholar] [CrossRef]
22. Stankowski, J.; Karwowski, D.; Wegner, K.; Stankiewicz, O.; Klimaszewski, K.; Grajek, T. Analysis of CABAC Context Models Efficiency in the HEVC. In Proceedings of the 57th International Symposium ELMAR-2015, Zadar, Croatia, 28–30 September 2015; pp. 217–220. [Google Scholar]

23. Karczewicz, M.; Ye, Y.; Chong, I. *Rate Distortion Optimized Quantization*; Document ITU-T SG16 Q.6, VCEG-AH21; Video Coding Experts Group (VCEG): Antalya, Turkey, 2008. [Google Scholar]
24. Grajek, T.; Stankowski, T.; Klimaszewski, K.; Grzelka, A. Parallel JPEG with RDOQ (PJRD)—A straightforward, modular implementation of a parallel JPEG encoder with encoder control optimizations. *SoftwareX* **2026**, *35*, 102774. [Google Scholar] [CrossRef]
25. Demikhovsky, E. Intel® AVX-512 Architecture. Comprehensive vector extension for HPC and enterprise. In *LLVM Developers' Meeting*; LLVM Foundation: San Francisco, CA, USA, 2013. [Google Scholar]
26. Bampis, C.G.; Li, Z.; Moorthy, A.K.; Katsavounidis, I.; Aaron, A.; Bovik, A.C. *LIVE Netflix Video Quality of Experience Database*; The University of Texas at Austin: Austin, TX, USA, 2016. [Google Scholar]
27. Microsoft Developer Network Library. Acquiring High-Resolution Time Stamps. 2020. Available online: https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408 (accessed on 2026-11-15). [Google Scholar]
28. Bjøntegaard, G. *Improvements of the BD-PSNR Model*; ITU-T SG16 Q.6 Document, VCEG-AI11; Video Coding Experts Group (VCEG): Berlin, Germany, 2008. [Google Scholar]
29. Larabel, M. SMT Remains Very Advantageous for 5th Gen AMD EPYC Performance. *Processors* **2025**, 1–6. Available online: https://www.phoronix.com/review/amd-epyc-zen5-smt (accessed on 2026-11-15). [Google Scholar]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.