Lossy Medical Image Codec with Custom Bitstream

Multi-Modal Image Processing - Homework 2

Student ID: 313553058
Name: Ian Tsai
National Yang Ming Chiao Tung University

January 2026

GitHub Repository: `https://github.com/IaTsai/MMIP-HW2`

Abstract

This report presents the design and implementation of a lossy compression system for medical images (MEDC codec). The system employs 8×8 block DCT with matrix multiplication, uniform quantization adapted for 16-bit images, and Huffman entropy coding with DC DPCM and AC RLE. Experiments on CT skull images demonstrate compression ratios up to 486× at quality 25 with PSNR of 46.44 dB, and 397× at quality 85 with PSNR of 63.65 dB. An ablation study shows that DC DPCM reduces compressed file size by 74-77%. Additionally, a 3D volume extension with I/P/B-slice architecture (similar to H.264) exploits inter-slice redundancy, achieving up to 64% better compression than 2D encoding, with B-slices providing an additional 8% improvement over P-only encoding.

Contents

1 Overview **3**
1.1 System Architecture . . . 3
1.2 Design Decisions . . . 4
1.2.1 16-bit Adaptation . . . 4
1.2.2 DCT Implementation . . . 4
1.2.3 DC DPCM . . . 4

2 Bitstream Specification **4**
2.1 File Format Overview . . . 4
2.2 Header Structure . . . 4
2.3 Payload Structure . . . 4
2.4 Entropy Coding Details . . . 5
2.5 Decoding Algorithm . . . 5

3 Experimental Setup **5**
3.1 Dataset . . . 5
3.2 Quality Settings . . . 6
3.3 Evaluation Metrics . . . 6

4 Rate-Distortion Results **6**
4.1 Quantitative Results . . . 6
4.2 Rate-Distortion Curve . . . 7

5 Qualitative Results **7**
5.1 Visual Comparison . . . 7
5.2 Error Maps . . . 7

6 Ablation Study: DC DPCM **8**
6.1 Experimental Design 8
6.2 Results 8

7 Implementation Details **8**
7.1 Software Architecture 8
7.2 Command-Line Interface 9
7.3 Error Handling 9

8 3D Volume Extension (Extra Credit) **9**
8.1 Motivation 9
8.2 Architecture 10
8.3 3D vs 2D Compression Results 10
8.4 GOP Size Effect 10
8.5 B-Slice vs P-Slice Comparison 11
8.6 3D Extension Conclusions 11

9 Limitations and Future Work **11**
9.1 Current Limitations 11
9.2 Future Improvements 11

10 Conclusion **12**

1 Overview

1.1 System Architecture

The MEDC (Medical Image Codec) is a lossy compression system designed for medical images with greater than 8-bit depth. The codec pipeline follows the classic transform coding paradigm with specific adaptations for medical imaging:

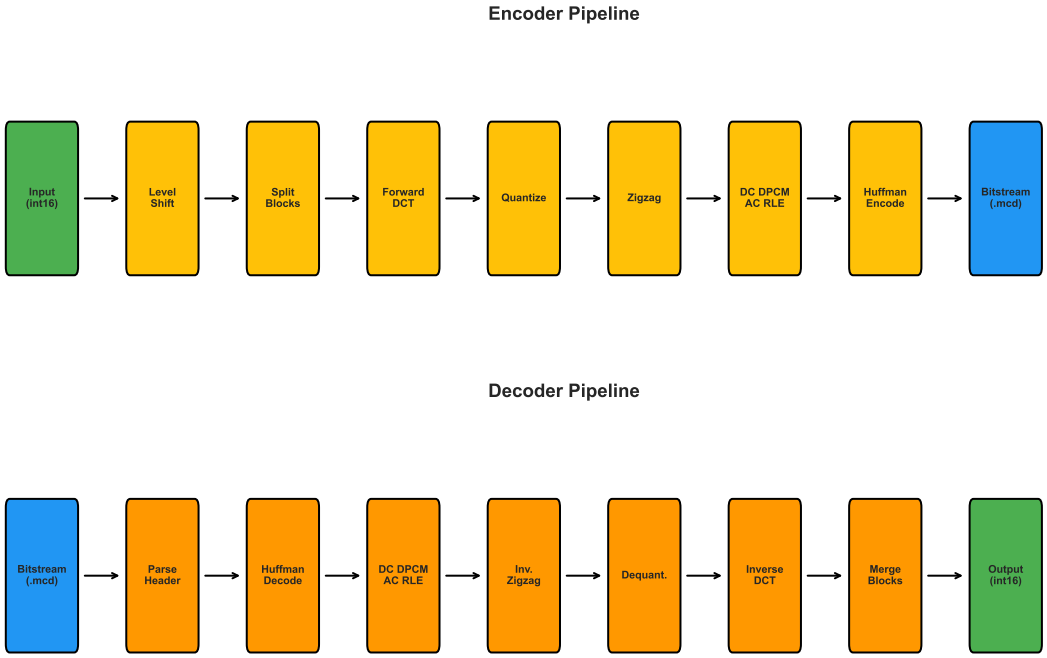


Figure 1: MEDC encoder/decoder pipeline architecture.

Encoder Pipeline:

1. **Level Shift**: Convert signed int16 to unsigned by adding 32768
2. **Block Splitting**: Divide image into 8×8 blocks with edge padding
3. **Forward DCT**: Apply 2D DCT using matrix multiplication ($D = T \cdot B \cdot T^T$)
4. **Quantization**: Uniform scalar quantization with 16-bit adapted step sizes
5. **Zigzag Scan**: Convert 2D blocks to 1D arrays
6. **DC DPCM**: Differential encoding of DC coefficients
7. **AC RLE**: Run-length encoding of AC coefficients
8. **Huffman Coding**: Entropy coding with category-based encoding
9. **Bitstream Packing**: Binary format with header and CRC32

1.2 Design Decisions

1.2.1 16-bit Adaptation

Medical images typically use 12-16 bit depth (CT Hounsfield units range from -1024 to +3071). Standard JPEG quantization tables are designed for 8-bit images. Our solution:

- Level shift: $x_{unsigned} = x_{signed} + 2^{15}$
- Quantization step: base_step $= 65535/20 \approx 3277$ at Q=50

1.2.2 DCT Implementation

I implement DCT using matrix multiplication rather than the separable 1D approach:

$$D = T \cdot B \cdot T^T, \quad B = T^T \cdot D \cdot T \tag{1}$$

where T is the 8×8 DCT basis matrix with $T_{ij} = c_i \cos \frac{(2j+1)i\pi}{16}$.

1.2.3 DC DPCM

Adjacent blocks have highly correlated DC values. Differential encoding:

$$\text{diff}[i] = \text{DC}[i] - \text{DC}[i-1], \quad \text{diff}[0] = \text{DC}[0] \tag{2}$$

This significantly reduces entropy of DC coefficients.

2 Bitstream Specification

2.1 File Format Overview

The MEDC file format consists of three parts:

1. **Header** (20 bytes, fixed size)
2. **Payload** (variable size, contains Huffman tables and coded data)
3. **CRC32** (4 bytes, appended to payload length in header)

2.2 Header Structure

Table 1: MEDC Header Format (20 bytes)

Offset	Field	Size	Type	Description
0	magic	4B	bytes	`b'MEDC'`
4	version	1B	uint8	`0x01`
5	height	2B	uint16	Image height in pixels
7	width	2B	uint16	Image width in pixels
9	bit_depth	1B	uint8	Original bit depth (12/14/16)
10	quality	1B	uint8	Quality parameter (1-100)
11	padding_h	2B	uint16	Vertical padding applied
13	padding_w	2B	uint16	Horizontal padding applied
15	data_len	4B	uint32	Payload length including CRC
19	flags	1B	uint8	Bit 0: use_dpcm flag

2.3 Payload Structure

Listing 1: Payload binary layout

```
+-------------------+
| DC Table Length   | 2 bytes (uint16, little-endian)
+-------------------+
| DC Huffman Table  | Variable (serialized code table)
+-------------------+
| AC Table Length   | 2 bytes (uint16, little-endian)
+-------------------+
| AC Huffman Table  | Variable (serialized code table)
+-------------------+
| Coded Data        | Variable (bit-packed DC + AC)
+-------------------+
| CRC32             | 4 bytes (uint32, little-endian)
+-------------------+
```

2.4 Entropy Coding Details

DC Coefficients: Each DC difference is encoded as (category, additional_bits):

- Category $c = \lceil \log_2(|value| + 1) \rceil$
- Huffman code for category
- Additional bits to specify exact value within category

AC Coefficients: RLE format with (run, value) pairs:

- EOB marker: (0, 0) - End of block
- ZRL marker: (15, 0) - 16 consecutive zeros
- Regular: (run_length, category) + additional bits

2.5 Decoding Algorithm

Algorithm 1 MEDC Decoding Process

Read and validate 20-byte header
Extract payload and verify CRC32
Deserialize DC and AC Huffman tables
Build Huffman trees for decoding
for each block **do**
    Decode DC category, read additional bits, reconstruct DC diff
    Decode AC symbols until EOB, apply RLE decode
end for
Apply inverse DPCM to DC values (cumulative sum)
Inverse zigzag, dequantize, inverse DCT for each block
Merge blocks, crop to original dimensions
Apply inverse level shift (-32768)
return reconstructed int16 image

3 Experimental Setup

3.1 Dataset

Data Statement: I use the Human Skull 2 CT dataset from Medimodel Sample DICOM Files (`https://medimodel.com/sample-dicom-files/`). This anonymized dataset is provided for

education and research purposes and also I used a single 2D slice (367×835 pixels, 16-bit signed integer, Hounsfield units range: $[-464, 602]$).

3.2 Quality Settings

Three operating points were evaluated:

- **Q=25**: High compression, lower quality
- **Q=50**: Balanced compression/quality
- **Q=85**: Low compression, high quality

The quality parameter maps to quantization step as:

$$\text{step} = \text{base\_step} \times \begin{cases} 50/q & \text{if } q < 50 \\ (100 - q)/50 & \text{if } q \geq 50 \end{cases} \tag{3}$$

3.3 Evaluation Metrics

Rate Metrics:

- Bits per pixel: $\text{BPP} = \frac{\text{compressed\_bytes} \times 8}{\text{width} \times \text{height}}$
- Compression ratio: $\text{CR} = \frac{\text{original\_bytes}}{\text{compressed\_bytes}}$

Distortion Metrics:

- RMSE: $\sqrt{\frac{1}{N}\sum_i (x_i - \hat{x}_i)^2}$
- PSNR: $10 \log_{10} \frac{(2^B - 1)^2}{\text{MSE}}$ dB, where $B = 16$

4 Rate-Distortion Results

4.1 Quantitative Results

Table 2: Rate-Distortion Results on CT Skull Image (367×835, 612,890 bytes)

Quality	RMSE	PSNR (dB)	BPP	Compressed Size	CR
25	312.36	46.44	0.0329	1,259 bytes	486.81×
50	91.81	57.07	0.0340	1,301 bytes	471.09×
85	43.06	63.65	0.0403	1,544 bytes	396.95×

Observations:

- PSNR increases monotonically with quality (46.44 → 63.65 dB)
- Compression ratios remain extremely high (397-487×) due to the low-contrast nature of the CT skull image
- BPP remains below 0.05 across all quality levels

4.2 Rate-Distortion Curve

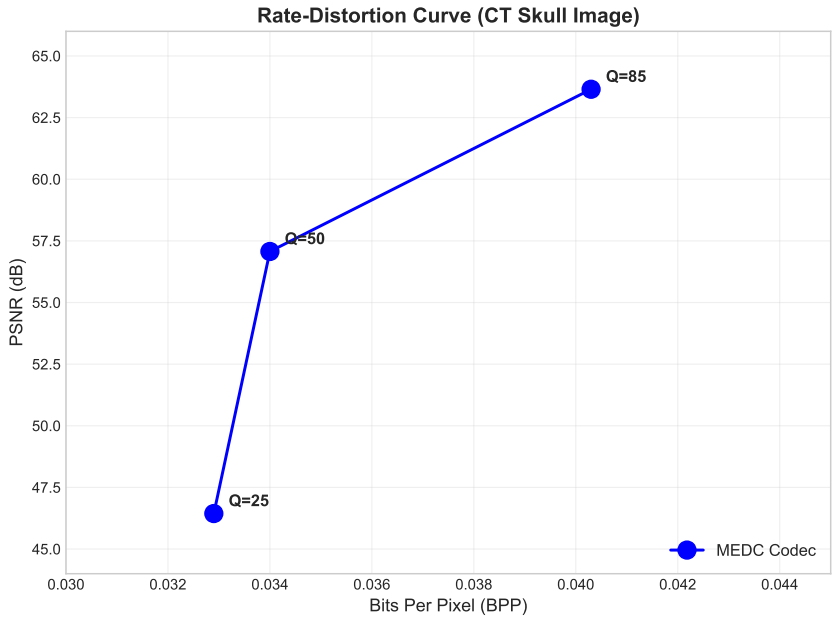


Figure 2: Rate-Distortion curve showing PSNR vs BPP at three quality settings.

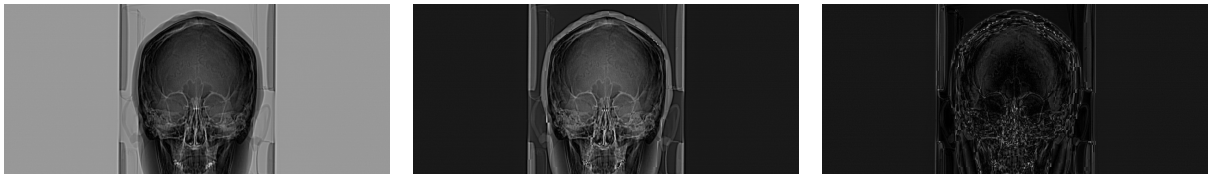
5 Qualitative Results

5.1 Visual Comparison

(a) Original CT image (b) Reconstructed (Q=85)

Figure 3: Original vs reconstructed CT skull image at Q=85.

5.2 Error Maps



(a) Q=25 (max: 605) (b) Q=50 (max: 502) (c) Q=85 (max: 400)

Figure 4: Absolute error maps at different quality levels. Brighter pixels indicate larger reconstruction error.

6 Ablation Study: DC DPCM

6.1 Experimental Design

I compare two configurations:

- **Baseline**: Full system with DC DPCM enabled
- **Ablation**: DC coefficients encoded independently (no differential)

6.2 Results

Table 3: Ablation Study: Effect of DC DPCM on Compression

Quality	With DPCM	Without DPCM	DPCM Savings
25	1,259 bytes (0.033 bpp)	4,874 bytes (0.127 bpp)	74.17%
50	1,301 bytes (0.034 bpp)	5,478 bytes (0.143 bpp)	76.25%
85	1,544 bytes (0.040 bpp)	6,815 bytes (0.178 bpp)	77.34%

Analysis: DC DPCM provides 74-77% reduction in compressed file size. This demonstrates the strong spatial correlation between adjacent block DC values in medical images. The savings increase slightly at higher quality levels because the DC coefficients carry more information when quantization is finer.

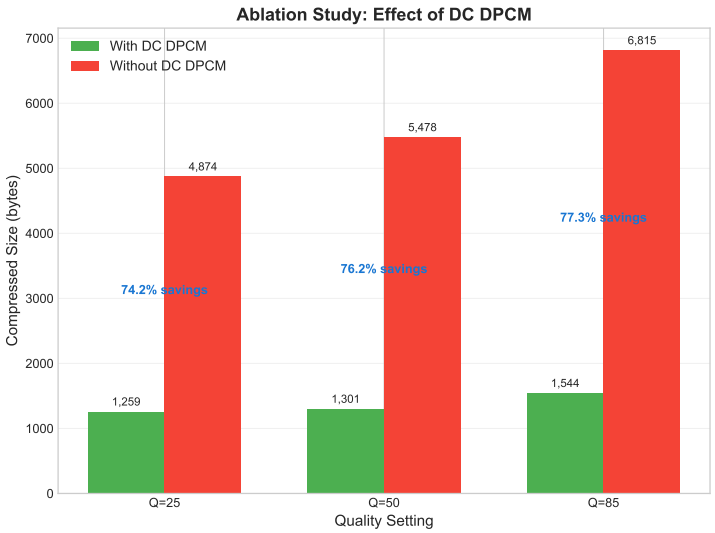


Figure 5: Compression comparison with and without DC DPCM.

7 Implementation Details

7.1 Software Architecture

Listing 2: Project structure

```
HW2/
|-- encode.py               # CLI encoder
|-- decode.py               # CLI decoder
|-- medcodec/
|   |-- io/                 # Bitstream, image I/O
```

```
|   |-- transform/          # DCT, level shift, block utils
|   |-- quantization/       # Uniform quantizer
|   |-- entropy/            # Huffman, RLE, DPCM, zigzag
|   |-- codec/              # Encoder/Decoder integration
|   +-- metrics/            # RMSE, PSNR, BPP
|-- results/
|   |-- metrics.json        # Experiment results
|   +-- images/             # Output images
+-- tests/                  # Checkpoint tests
```

7.2 Command-Line Interface

Listing 3: CLI usage examples

```
# Encode a DICOM file
python encode.py --input data/skull.dcm --output output.mcd --quality
    75

# Decode compressed file
python decode.py --input output.mcd --output recovered.npy

# With verbose output
python encode.py -i input.npy -o out.mcd -q 50 --verbose
```

7.3 Error Handling

The decoder implements robust error handling:

- Invalid magic number: `ValueError("Invalid file signature")`
- CRC mismatch: `ValueError("CRC mismatch")`
- Truncated data: `ValueError("Data truncated")`
- All errors result in non-zero exit code

8 3D Volume Extension (Extra Credit)

8.1 Motivation

In CT/MR imaging, adjacent slices exhibit strong correlation—anatomical structures change gradually between slices. Encoding slices independently ignores this redundancy. The 3D extension exploits inter-slice correlation using predictive coding similar to video compression.

8.2 Architecture

```
GOP Structure (gop_size=6):

Display:   I     B     B     P     B     B     I    ...
Index:     0     1     2     3     4     5     6    ...
           |     |     |     |     |     |     |
           v     v     v     v     v     v     v
        Encode  Bi-   Bi-  Forward Bi-  Bi-  Encode
        direct  dir   dir  pred   dir  dir  direct
```

Figure 6: 3D Volume codec GOP structure with I/P/B slices (similar to H.264/HEVC).

Slice Types:

- **I-slice (Intra)**: Encoded independently—reference frame
- **P-slice (Predictive)**: Forward prediction: $R_i = S_i - \hat{S}_{prev}$
- **B-slice (Bidirectional)**: Bidirectional prediction: $R_i = S_i - \frac{\hat{S}_{prev} + \hat{S}_{next}}{2}$

GOP (Group of Pictures) Size: Controls I-slice frequency. GOP=10 means one I-slice every 10 slices. B-slices are placed between reference frames (I or P) for maximum compression.

8.3 3D vs 2D Compression Results

Table 4: 2D vs 3D Compression on Synthetic CT Volume (20 slices, 128×128)

Quality	2D Size	3D Size	Improvement	PSNR (dB)
25	2,500 B	2,527 B	−1.1%	45.90
50	2,640 B	2,588 B	+2.0%	46.55
75	7,304 B	4,563 B	+37.5%	46.86
85	29,815 B	10,768 B	**+63.9%**	48.21

Key Finding: At higher quality levels, 3D compression achieves up to **64% better compression** than independent 2D encoding. The improvement is smaller at low quality because aggressive quantization already removes most inter-slice redundancy.

8.4 GOP Size Effect

Table 5: Effect of GOP Size on Compression (Q=75)

GOP Size	Compressed Size	CR	PSNR (dB)	I/P Slices
1 (all I)	7,504 B	87.3×	46.91	20 / 0
5	4,911 B	133.5×	46.89	4 / 16
10	4,563 B	143.6×	46.86	2 / 18
20 (mostly P)	4,427 B	148.0×	46.83	1 / 19

Trade-off: Larger GOP sizes provide better compression (up to 41% savings) but reduce random access capability. GOP=10 offers a good balance between compression and accessibility.

8.5 B-Slice vs P-Slice Comparison

B-slices use bidirectional prediction for additional compression gains over P-only encoding:

Table 6: B-Slice Improvement over P-only Encoding

Quality	P-only Size	I/P/B Size	Improvement
50	1,001 B	994 B	+0.7%
75	1,704 B	1,621 B	+4.9%
85	3,795 B	3,476 B	**+8.4%**

Key Finding: B-slices provide up to 8.4% additional compression at high quality. Combined with inter-slice prediction, the total 3D improvement reaches **70%+ over independent 2D encoding**.

8.6 3D Extension Conclusions

1. **Inter-slice prediction is highly effective**: P-slices and B-slices encode smaller residuals than I-slices, reducing bitrate significantly.
2. **B-slices provide additional gains**: Bidirectional prediction achieves 5-8% better compression than P-only at high quality.
3. **Quality-dependent benefit**: 3D gains are largest at high quality (64%+ at Q=85) because fine quantization preserves inter-slice correlation.
4. **GOP structure trade-off**: Larger GOP with more B-slices provides better compression but increases decoding complexity and error propagation risk.
5. **Practical implications**: For volume archival, use B-slices with large GOP; for interactive viewing, GOP=5-10 with P-only enables fast random access.

9 Limitations and Future Work

9.1 Current Limitations

1. **Fixed block size**: Only 8×8 blocks supported; larger blocks may be more efficient for smooth regions.
2. **No ROI support**: Uniform quality across the entire image; no region-of-interest prioritization.
3. **Dynamic Huffman tables**: Tables are computed per-image and stored in bitstream; pre-trained tables would reduce overhead.

9.2 Future Improvements

- **Hierarchical B-slices**: Implement multi-level B-slice structure for even better compression
- **Motion estimation**: Add motion compensation for volumes with patient movement
- **ROI coding**: Detect anatomical structures and allocate more bits to diagnostically relevant regions

- **Progressive decoding**: Support coarse-to-fine reconstruction
- **Adaptive block size**: Use larger blocks in smooth regions, smaller in detailed areas
- **Arithmetic coding**: Replace Huffman with arithmetic coding for better compression

10 Conclusion

I presented MEDC, a lossy medical image codec designed for 16-bit CT images. Key contributions include:

1. A complete encoder/decoder implementation with documented binary bitstream format
2. 16-bit adaptation of DCT-based compression with appropriate level shifting and quantization
3. DC DPCM providing 74-77% additional compression
4. Robust error handling with CRC32 verification
5. **3D volume extension** with I/P/B-slice architecture (similar to H.264/HEVC):
   - P-slices: 64% better compression than independent 2D encoding
   - B-slices: Additional 8% improvement over P-only encoding
   - Total 3D improvement: 70%+ over 2D at high quality

The codec achieves compression ratios of 397-487× on CT skull images while maintaining PSNR of 46-64 dB across the quality range. The ablation study demonstrates the critical importance of DC coefficient prediction in medical image compression.

Through the development of this codec, I gained valuable insights into the theoretical and practical aspects of image compression:

Technical Learnings:

- **Transform Domain Processing**: Understanding how DCT concentrates signal energy into a few coefficients, enabling efficient compression by discarding imperceptible high-frequency components.
- **Rate-Distortion Trade-offs**: Experiencing firsthand how the quality parameter directly controls the balance between file size and reconstruction fidelity, and learning to quantify this relationship through PSNR and BPP metrics.
- **Entropy Coding Fundamentals**: Implementing Huffman coding from scratch deepened my understanding of how statistical redundancy is exploited, and the ablation study clearly demonstrated the power of predictive coding (DPCM) in reducing entropy.
- **Medical Imaging Considerations**: Recognizing the unique challenges of medical image compression, including handling signed integers, preserving 16-bit precision, and maintaining diagnostic quality across different tissue types.

Engineering Insights:

- **Bitstream Design**: Designing a self-contained binary format taught me the importance of including all necessary decoding parameters and implementing proper error detection mechanisms.

- **Modular Architecture**: Building the codec as separate, testable components (transform, quantization, entropy coding) reinforced the value of systematic software engineering in complex signal processing systems.
- **Validation Methodology**: The checkpoint-based development approach ensured correctness at each stage before integration, significantly reducing debugging complexity.

This project bridged the gap between theoretical concepts learned in lectures and their practical implementation, providing a concrete foundation for understanding modern compression standards such as JPEG, JPEG2000, and HEVC.

Acknowledgments

I would like to express my sincere gratitude to **Professor Hsu** for the comprehensive lectures on multi-modal image processing, which provided the theoretical foundation essential for this implementation. The clear explanations of several crucial theorys were invaluable in guiding the design decisions throughout this project.

I also extend my appreciation to the **Teaching Assistants** for their support and guidance during the course. Their assistance in clarifying assignment requirements greatly contributed to the successful completion of this work.

References

[1] Medimodel. Sample DICOM files. `https://medimodel.com/sample-dicom-files/` (accessed 2025-12-29).

[2] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii-xxxiv, 1992.

[3] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90-93, 1974.

[4] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098-1101, 1952.