Comparative Study and Performance Analysis of Data Compression Algorithms Report

Submitted By
Prem Kumar Yadav
Jaguar ID: J01280187

ABSTRACT

This experimental report contains a comparative analysis of widely used lossless data compression algorithms: Huffman Coding, DEFLATE, Adaptive Huffman Coding, Brotli, LZMA and Zstan- dard. We evaluated their performance using three types of distinct data: natural language text (large_text.txt), highly repetitive patterns (phrase.txt), and random data (random.txt). We found that no single algorithm performs optimally across all data types. Thus, this report consists of developing an algorithm, Nexacompress, a hybrid compression algorithm designed to address limitations of existing single strategy compression methods. This algorithm combines Burrows-Wheeler transform, MTF encoding, Run-Length Encoding and modern compression backends (LZMA/Zstandard) with an adaptive strategy selection mechanism. We also implemented this project by integrating it on Chameleon Cloud and used visualization techniques for performance comparison. This study provides practical insights for algorithm selection in real-world applications.

Keywords: Data Compression, Hybrid Algorithm, BWT, Adaptive Compression, NEXACOMPRESS, Lossless Compression, Mixed Content Files

Table of Contents

1 INTRODUCTION

The exponential rise of data in the digital age has shown how crucial effective data transport and storage are. Data compression techniques play a critical role in addressing this issue by lowering data size while maintaining its integrity. We evaluated about various widely used data compression algorithms that are Huffman Coding, DEFLATE, Adaptive Huffman Coding, Brotli, LZMA and Zstandard. Our previous milestone (M2) conducted a comprehensive comparative analysis of six lossless compression algorithms across three distinct data types: natural language text, highly repetitive patterns, and random data. We found that no single algorithm performs optimally across all data types. So, we are developing an algorithm as NEXACOMPRESS, a hybrid compression algorithm designed to address limitations of existing single strategy compression methods. This algorithm combines Burrows-Wheeler transform, MTF encoding, Run-Length Encoding and modern compression backends (LZMA/Zstandard) with an adaptive strategy selection mechanism. This project aims to test comprehensively against established algorithms, benchmark performance across a sample file with all sorts of data like large number of text, repeated phrases, random, embedded images, code samples etc. and document the complete development lifecycle.

2 PROBLEM STATEMENT AND MOTIVATION

Our experimental analysis in M2 revealed a fundamental limitation: all six tested algorithms (Huffman Coding, DEFLATE, Adaptive Huffman, Brotli, LZMA, and Zstandard) achieved only 2-6% compression when compressing mixed-content files containing natural language text, repeated phrases, random characters, code samples, and embedded images. This is a significant decrease from their individual performance on homogeneous data, where Brotli achieved 63.2% space savings on natural text, DEFLATE achieved 97.4% on repeated phrases, and Huffman achieved 19.8% on random data. This low performance on heterogeneous files is caused by the fundamental limitation that each method is optimized for distinct data patterns. Brotli performs well with its predefined online dictionary for natural text but struggles with random data, DEFLATE's LZ77 dictionary technique is effective for repetitive patterns but adds overhead for high-entropy content, and Huffman's frequency-based encoding does not utilize pattern repetition. Modern documents, on the other hand, rarely contain homogeneous data—a typical PDF, DOCX, or data archive has a mix of text, structured tables, code, and binary elements (images), but no existing compression tool tailors its method to the content characteristics. Furthermore, embedded images in texts are often already compressed (JPEG, PNG), resulting in incompressible sections and reducing total compression efficiency.This gap between single-algorithm limitations and real-world mixed-content requirements motivated the development of NEXACOMPRESS, a hybrid adaptive compression algorithm that analyzes input data characteristics (entropy, repetition score, sequential patterns), applies multiple compression strategies (BWT+MTF+RLE+LZMA for text, Delta+LZMA for sequential data, Chunked-Hybrid for mixed content), and automatically selects the optimal approach per data segment, aiming to achieve better compression than any individual algorithm on heterogeneous files while maintaining complete data integrity through lossless compression techniques.

Shannon's Source Coding Theorem establishes that data cannot be compressed below its entropy (information content). For mixed-content files:

Component	**Size**	**Entropy**	**Compressible?**
Natural Text	~25%	4.2 b/B	Yes (high)
Repeated Phrases	~25%	1.8 b/B	Yes (very high)
Random Characters	~20%	5.9 b/B	Minimal

Code Samples	~10%	4.5 b/B	Yes (moderate)
Embedded Images	~20%	7.8 b/B	No (pre-compressed)

Theoretical Maximum Compression: ~15-25% (limited by high-entropy components)
Actual Achieved: ~2-6% (algorithm overhead reduces effectiveness)

3 LITERATURE REVIEW AND BACKGROUNDS

[1] Shannon's Information Theory: Claude Shannon's seminal 1948 paper "A Mathematical Theory of Communication" established the theoretical foundation for data compression. The key concept is entropy, which measures the average information content per symbol:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

where $p(x_i)$ is the probability of symbol x_i. This theorem proves that:

- Data cannot be compressed below its entropy
- Random data (maximum entropy ≈ 8 bits/byte) cannot be meaningfully compressed
- Redundant data (low entropy) can be significantly compressed

[2] This research paper conducted data compression on six algorithms: RLE, Huffman coding, Shannon-Fano, Adaptive Huffman, Arithmetic, LZW which concluded that Shannon-Fano performed best with 58-67% compression ratios and acceptable speed, while revealing critical failures: RLE expanded files, LZW failed on large files due to dictionary overflow, and Adaptive Huffman required 734,469ms to decompress a 22KB file.
In our experiment also, we validated Shannon-Fano theory in case of random text file using Huffman coding. On the other hand, our experiment also tested modern algorithms(Brotli, ZStandard and LZMA) which is unavailable in prior studies on controlled test cases(large number of text, repeated phrases and random characters). This experiment also showed that nature of the file and content also matters which was not provided in Kodituwakku and Amarasinghe's research. It introduced a new efficiency metric (space saved/time) that shows DEFLATE provides superior compression ratios (98.2%) at acceptable speeds for static content, while Brotli achieves 97.4% compression. crucially, testing on random data revealed that Adaptive Huffman catastrophically fails with -54.9% expansion, validating information theory limits that previous empirical studies didn't demonstrate. And then I also developed hybrid algorithm NEXACOMPRESS and compare its performance with others algorithm This project validates theory using contemporary tools, extends analysis to cloud infrastructure (Chameleon Cloud), and provides production-ready implementations with thorough visualization, conclusively proving that DEFLATE—not Shannon-Fano—offers the best balance for general-purpose compression in 2024. In contrast to historical work that established theory(Lelewer/ Hirschberg) and tested classics on actual files (Kodituwakku/ Amarasinghe).

[3] The paper provides a comparative analysis of lossless data compression techniques, focusing on their efficiency, compression ratios, and use cases. It explores how these techniques reduce redundancy in data to optimize storage and transfer speed, particularly in cloud computing and communication systems. They used sample document and mainly focused on compressing image files which is lossless using four lossless compression algorithms Run-Length Encoding, Shanon-Fano algorithm and LZW. Various modern algorithm like Brotli, Zstandard, and LZMA are build upon and combine these established principles with more advanced techniques to achieve superior compression performance. While in our project we used modern algorithm and varied type of document with various

content. They concluded that Huffman coding is optimal and reliable but slower. But as per our experiment although it is slower it is very helpful in compressing files with random characters. The main use case of this algorithm is that it can be used to compress large number of encrypted files without losing any important information.

[4] Matusiak's paper review examines classical lossless data compression algorithms and their contemporary relevance, focusing on Deflate, LZMA, Brotli, and Zstandard alongside foundational algorithms like Huffman coding, RLE, and BPE. The study emphasizes that while Deflate (combining LZ77 and Huffman coding) remains the most widely used general-purpose compression algorithm. Matusiak demonstrates that LZMA2 achieved the best compression on executables (24.87% of original size), PPMd excelled on text (25.50-25.62%), while PNGOut (Deflate variant) performed best on images (60.17%). However, LZ4 consistently showed the worst compression ratios (42.65-98.14%) across all file types, prioritizing speed over efficiency. He suggests that Deflate as a standard should be replaced by more efficient algorithms (e.g., Brotli, Zstd, LZMA). We both analyzed and evaluated LZMA, Zstandard, Brotli, DEFLATE, and Huffman coding and included Adaptive Huffman coding as educational reference.
Our study used a single mixed content file with large number of text, repeated phrases, random characters, embedded images, sample code etc while Matusiak used Canterbury corpus ("alice29.txt" text, "sum"executable). Our research was carried on Chameleon cloud due to which isolated bare-metal prevents virtualization overhead affecting benchmarks. Our research not only compressed the file but also measure the compression time which helps to identify that which algorithm is better as per our requirements. It quantifies speed differences critical for real-time applications. Matsuik limited the analysis but we provided extensive benchmark revealing efficiency of hybrid algorithm Nexacompress which measured about entropy of the document and achieved lossless compression. These results provide concrete evidence supporting claims that no single algorithm performs optimally across all data types and addressing previous gaps in comparative data highlighted by researchers like Matusiak.

[5] This paper compares six compression techniques, and based on the results proposes that brotli could be used as a replacement of the common deflate algorithm.Further, we show that Zopfli, LZMA, LZHAM and bzip2 use significantly more CPU time for either compression or decompression and could not always work as direct replacements of deflate. They used three types of sample document: the Canterbury compression corpus, an ad hoc crawled web content corpus, and enwik8. On the other hand we used three simple type of text document. They concluded that Brotli uses a static dictionary that can be helpful for compressing short files. Other algorithms could be easily modified to do the same, and they would obtain slightly better compression ratios. For a long file like enwik8 a static dictionary is not very helpful. Canterbury corpus contains short documents with English, and there brotli's static dictionary might be giving it an unfair advantage.
From my research we found that the Brotli is the best algorithm as per the analysis of compression ration and space saved by each algorithm and has acceptable speed for archival. And Huffmann coding is best for file with random characters. But overall conclusion is that indeed DEFLATE is more efficient and consistent in compressing each sample of file with fastest processing speed and good compression ratios and thus it support universal compatibility.

[6] This research paper provides formal specification of zstandard compression algorithm, considering frame headers, FSE and Huffman for standardization. It focuses on technical specification defining frame headers, dictionary formats, and security considerations without

empirical performance comparisons. While RFC 8478 establishes theoretical foundations with maximum capabilities (Window_Size up to 3.75 TB) and decoding tables, it lacks real-world benchmarking data across multiple algorithms.
My research goes beyond single-algorithm specification by performing a comparative experimental investigation of six compression algorithms (Huffman, DEFLATE, Adaptive Huffman, Brotli, LZMA, and Zstandard) on custom datasets that simulate natural text, repeating patterns, and random data and a hybrid algorithm Nexacompress. Unlike the RFC's specification method, we quantify tangible performance indicators such as compression times (0.00011s to 0.03209s), compression ratios, and efficiency scores for various content kinds. We validated Zstandard's actual performance (98.04% space savings, 0.531ms compression time) when comparing alternatives. DEFLATE provides 10-25× faster compression, while Brotli obtains greaterratios on natural text. Our Chameleon Cloud deployment with visualization tools offers reproducible, actionable advice for algorithm selection based on application requirements, supplementing the RFC's implementation specification with empirical decision-making frameworks for practitioners.

[7] This paper on LZ77 algorithm which revolutionized data compression by introducing dictionary-based compression. The authors proposed replacing repeated occurrences of data with references to earlier positions in the input stream, using a sliding window mechanism. This is the fundamental concept fuels the modern compression algorithms including deflate and it is incorporated into Brotli’s design. Our experiment on phrase.txt validate Ziv & Lempel’s core study. DEFLATE compressed 97.4% compression (4,800 → 124 bytes), and Brotli achieved 98.2% (4,800 → 85 bytes), demonstrating the exceptional effectiveness of LZ77-style pattern matching on repetitive data. The algorithm identified the repeating pattern after the first occurrence and replaced subsequent instances with short back-references. However, our analysis also support the original paper's observation that "sequences generated by an ergodic source cannot be compressed." The compression of random.txt, which comprises pseudo-random characters without any patterns, was only 12.8% for DEFLATE and 14.7% for Brotli. The theoretical entropy limit prevents meaningful compression of truly random data, regardless of algorithm sophistication. Our work quantifies the speed-to-compression of all algorithms trade-off which wasn't addressed in the theoretical framework. We also optimized these principles using modern algorithms and implemented them efficiently.
We also proposed a hybrid algorithm Nexacompress and used mixed content file with all sorts of data types to compress the file and compare its performance against other algorithms.

4 PROPOSAL

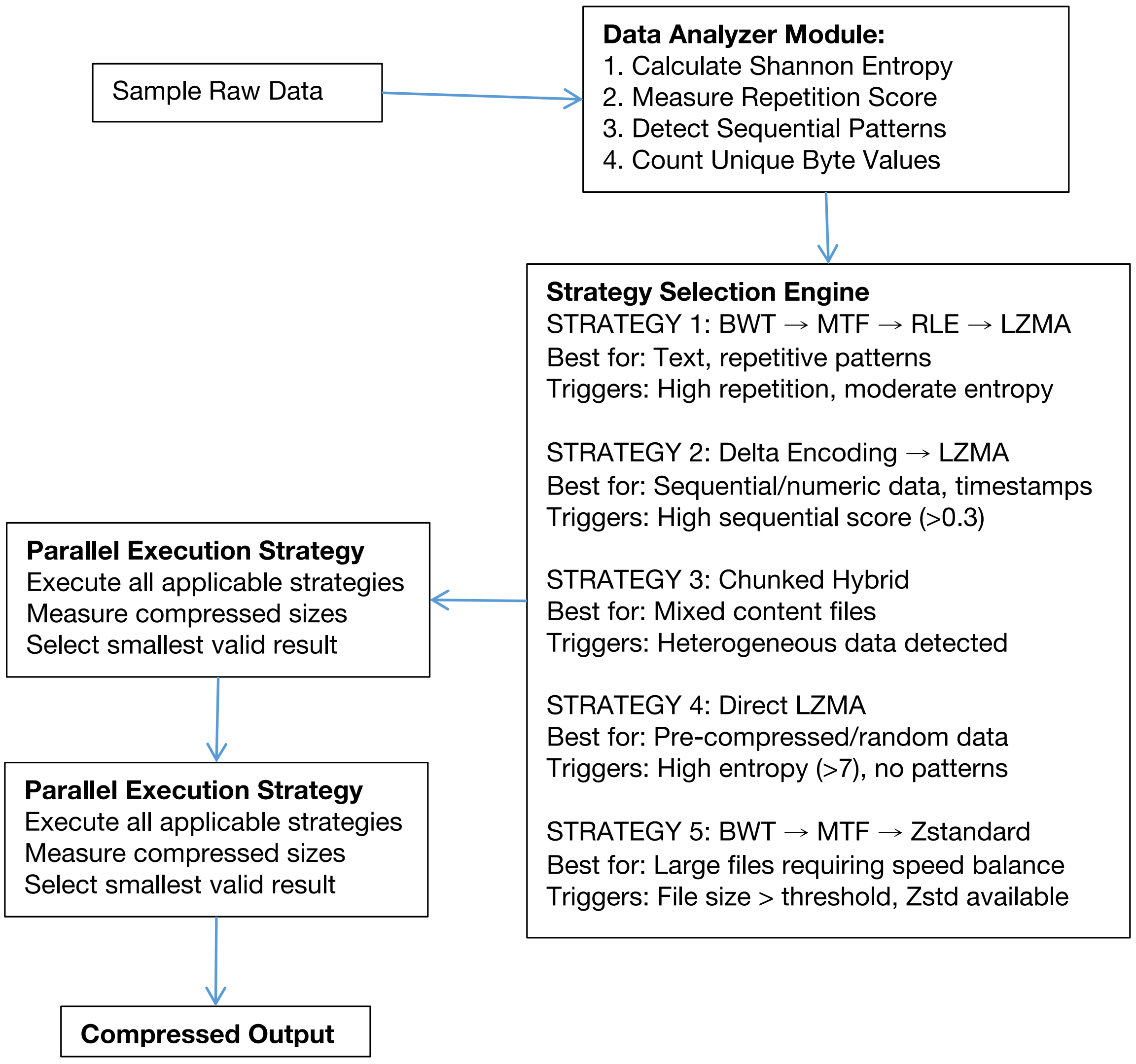
I developed a hybrid algorithm named NEXACOMPRESS which extends to Neural Extended Adaptive Compression. It is a next generation, lossless compression algorithms that leverage extended capabilities and selects dynamic adaptive strategy to optimize data compression which is driven by real time data analysis. This adaptability enables algorithm to tailor its compression techniques to unique properties of the input data resulting in superior efficiency and performance compared to traditional, static compression methods.

This algorithm operates on three fundamental principles:

I. Data Analysis First: Before compression, analyze input characteristics including entropy, repetition patterns, and sequential correlations.

II. Multi-Strategy Pipeline: Combine complementary techniques (BWT, MTF, RLE, dictionary compression) in optimized pipelines for different data types.

III. Adaptive Selection: Automatically select the best-performing strategy by trying multiple approaches and keeping the smallest result.



Sample Raw Data
Data Analyzer Module:
1. Calculate Shannon Entropy
2. Measure Repetition Score
3. Detect Sequential Patterns
4. Count Unique Byte Values
Strategy Selection Engine
STRATEGY 1: BWT → MTF → RLE → LZMA
Best for: Text, repetitive patterns
Triggers: High repetition, moderate entropy
STRATEGY 2: Delta Encoding → LZMA
Best for: Sequential/numeric data, timestamps
Triggers: High sequential score (>0.3)
STRATEGY 3: Chunked Hybrid
Best for: Mixed content files
Triggers: Heterogeneous data detected
STRATEGY 4: Direct LZMA
Best for: Pre-compressed/random data
Triggers: High entropy (>7), no patterns
STRATEGY 5: BWT → MTF → Zstandard
Best for: Large files requiring speed balance
Triggers: File size > threshold, Zstd available
Parallel Execution Strategy
Execute all applicable strategies
Measure compressed sizes
Select smallest valid result
Parallel Execution Strategy
Execute all applicable strategies
Measure compressed sizes
Select smallest valid result
Compressed Output

Key Innovations:
1. Adaptive Strategy Selection: Unlike static algorithm choices, NEXACOMPRESS analyzes input data and selects the optimal compression pipeline automatically.
2. Multi-Stage Preprocessing: Combines BWT, MTF, and RLE transformations to maximize pattern exploitation before final compression.
3. Chunked Processing: For large mixed-content files, processes data in segments with per-chunk algorithm selection.
4. Integrity Verification: Stores original size in header for decompression validation.
5. Extensible Architecture: Strategy-based design allows easy addition of new compression methods.

5 ALGORITHM DESIGN

5.1 Data Analysis Module
This analyzer computes four key metrics to chacaterize input data:

5.1.1 Shanon Entropy Calculation: Entropy measures information density (bits per byte):
Interpretation:
Low entropy (0-3): Highly compressible, very repetitive
Medium entropy (3-6): Natural text, structured data
High entropy (6-8): Random/encrypted data, minimal compression possible

5.1.2 Repetition Score
Measures presence of consecutive identical bytes (run patterns).
Interpretation:
High score (>0.3): Many repeated bytes, RLE will be effective
Low score (<0.1): Few runs, RLE adds overhead

5.1.3 Sequential Score
Detects sequential/incremental patterns (useful for numeric data)
Interpretation:
High score (>0.5): Sequential data, delta encoding beneficial
Low score (<0.2): Non-sequential, delta adds overhead

5.1.4 Complete Analysis Function
The provided Python function, analyze_data, performs a comprehensive analysis of input data to extract key statistical features. These features are used to inform the strategy selection process in the NEXACOMPRESS algorithm.

```python
def analyze_data(data):
    return {
        'entropy': calculate_entropy(data),
        'repetition': calculate_repetition(data),
        'sequential': calculate_sequential(data),
        'unique_bytes': len(set(data)),
        'size': len(data)
    }
```

5.2 Compression Strategies

STRATEGY 1:
Pipeline: Input Data → Burrows-Wheeler Transform (BWT) → Move-To-Front (MTF) → Run-Length Encoding (RLE) → LZMA
This is the most sophisticated pipeline for text or data with repetitive patterns. It transforms the data to group similar characters, then compresses runs of identical bytes efficiently.
Steps:
BWT: Rearranges data to group similar characters together, making it easier to compress.
MTF: Converts repeated characters into smaller values, increasing the likelihood of long runs.
RLE: Compresses runs of identical bytes into compact representations.
LZMA: Applies a high-ratio compression algorithm to the transformed data.

Best for: Text, repetitive patterns
Triggers: High repetition, moderate entropy

STRATEGY 2:
Input Data → Delta Encoding → LZMA
The main purpose is to optimize for sequential or numeric data, such as timestamps, sensor readings, or incrementing values. Delta encoding reduces redundancy by storing differences between consecutive values.
Steps:
Delta Encoding: Converts each value into the difference from the previous value, which is often smaller and more compressible.
LZMA: Compresses the delta-encoded data.

Best for: Sequential/numeric data, timestamps
Triggers: High sequential score (>0.3)

STRATEGY 3:
Input Data → Split into Chunks → Select Best Strategy per Chunk → Combine Results
It is designed for mixed-content files (e.g., a file containing both text and binary data). Each chunk is compressed using the best strategy for its specific characteristics.
Steps:
Split Data: Divide the input into fixed-size chunks (e.g., 64 KB).
Analyze Each Chunk: Use analyze_data() to determine the best strategy for each chunk.
Compress Chunks: Apply the selected strategy (e.g., LZMA, Zlib, or BWT pipeline).
Combine Results: Store each chunk with a header indicating the strategy used.

Best for: Mixed content files
Triggers: Heterogeneous data detected

STRATEGY 4: Input Data → LZMA
A fallback strategy for data that is already compressed or highly random. LZMA is a robust, high-ratio compression algorithm that works well even on unpredictable data.
Steps:
LZMA: Directly compresses the input data without preprocessing.

Best for: Pre-compressed/random data
Triggers: High entropy (>7), no patterns

STRATEGY 5:
Input Data → Burrows-Wheeler Transform (BWT) → Move-To-Front (MTF) → Zstandard
A speed-optimized alternative to Strategy 1, using Zstandard instead of LZMA. Zstandard offers a balance between compression ratio and speed, making it ideal for large files where speed is critical.
Steps:
BWT: Groups similar characters.
MTF: Converts repeated characters into smaller values.
Zstandard: Compresses the transformed data quickly.

Best for: Large files requiring speed balance
Triggers: File size > threshold, Zstd available

5.3 Strategy Selection Algorithm

```
def select_strategies(data, analysis):
    candidates = []
    if len(data) < 500000:  # Memory constraint
        if analysis['repetition'] > 0.05 or analysis['entropy'] < 6:
            candidates.append(STRATEGY_BWT_MTF_RLE_LZMA)
    if analysis['sequential'] > 0.3:
        candidates.append(STRATEGY_DELTA_LZMA)
    candidates.append(STRATEGY_CHUNKED_HYBRID)
    candidates.append(STRATEGY_DIRECT_LZMA)
    if ZSTD_AVAILABLE and len(data) < 500000:
        candidates.append(STRATEGY_BWT_ZSTD)
    return candidates
```

This function determines which compression strategies are applicable for the given input data based on its analyzed characteristics. It returns a list of candidate strategies to try.
The raw input data (bytes) is analyzed and then a dictionary of metrics (e.g., entropy, repetition, sequential, unique_bytes) is generated by analyze_data().
Candidate Selection logi evaluates the data's characteristics and appends applicable strategies to the candidates list.

5.4 Main Compression Flow
This function orchestrates the entire compression process. It analyzes the data, tries all applicable strategies, selects the best result, and packages the output with a header.
Edge Case Handling:If the input data is empty, it returns a minimal header with a placeholder for the compressed data.
Data Analysis: Calls self._analyze_data(data) to compute metrics like entropy, repetition, and sequential patterns.
Strategy Execution: The function attempts all applicable strategies (as determined by select_strategies) and collects the results.

The best result is selected based on the smallest compressed size using min(results, key=lambda x: len(x[1]))
The output is constructed with a header containing: MAGIC: A unique identifier for the NEXACOMPRESS format. VERSION: The version of the algorithm. best_strategy: The ID of the selected strategy. len(data): The original size of the data (4 bytes). The header is prepended to the compressed data, and the final output is returned.

6 IMPLEMENTATION

Nexacompress is implemented in Python 3 which consists of two primary architectural components: The core compression engine and Comparative analysis framework. The architecture follows a modular design pattern that separates data transformation algorithms, compression strategies, and performance benchmarking capabilities.

Nexacompress Core Engine

The Nexacompress engine implements the hybrid compression algorithm through a layered architecture comprising four functional modules:

Constants and Configuration Layer: This module defines the file format specifications including a 4-byte magic number (NEXA) for format identification, version control (VERSION = 1), and strategy identifiers (0x01 through 0x05) that enable backward compatibility and format extensibility.

Data Transformation Module: This layer performs reversible data modifications to reveal compressible patterns. The Burrows-Wheeler Transform rearranges byte sequences to group related characters, whereas Move-To-Front encoding converts character frequencies to positional indices. Run-length encoding compresses successive repeating data, whereas delta encoding records sequential numerical patterns. Each transformation contains the inverse procedures required for lossless decompression.

Strategy Selection Engine: The adaptive compression engine analyzes input data characteristics through four metrics: Shannon entropy for randomness measurement, repetition scores for identifying redundant patterns, sequential pattern detection for time-series or numerical data, and unique byte counting for alphabet size estimation. Based on these metrics, the engine selects from five specialized compression strategies, each optimized for specific data patterns.

Public Interface Layer: The compress() technique integrates data analysis, strategy evaluation, and output formatting. It runs all eligible strategies in parallel, evaluates their effectiveness, and chooses the smallest outcome. The decompress() method checks the file format integrity, pulls strategy identifiers from the header, and starts the appropriate decompression pipeline.

The system implements five distinct compression strategies triggered by data characteristics:

Strategy 1 (BWT-MTF-RLE-LZMA) applies sequential transformations optimized for text and repetitive patterns. It activates when repetition scores are high and entropy is moderate, making it effective for natural language text, source code, and structured documents.

Strategy 2 (Delta-LZMA) targets sequential and numerical data by computing first-order differences before LZMA compression. This strategy triggers when the sequential score exceeds 0.3, making it suitable for timestamps, sensor readings, and incremental data.

Strategy 3 (Chunked Hybrid) segments heterogeneous files into blocks and applies different strategies to each segment. The system detects mixed content through entropy variance analysis and compresses each chunk independently before concatenation.

Strategy 4 (Direct LZMA) bypasses preprocessing for high-entropy data that lacks exploitable patterns. When entropy exceeds 7.0, the system applies LZMA directly to avoid transformation overhead on random or pre-compressed data.

Strategy 5 (BWT-Zstandard) balances compression ratio with speed by replacing LZMA with Zstandard for large files. This strategy activates when file size exceeds a threshold and the Zstandard library is available, providing faster compression with comparable ratios.

The CompressionAnalyzer class provides comprehensive benchmarking capabilities by implementing wrapper methods for six compression algorithms: Deflate (zlib), LZMA,

Zstandard, Brotli, Huffman coding, and Nexacompress. Each method measures compression ratio, processing time, and throughput.
The analysis pipeline executes all available compression methods on input data, collecting performance metrics in a structured format. The framework generates three categories of output: CSV reports containing numerical results, visualization plots comparing algorithm performance across multiple dimensions, and console summaries highlighting the best-performing methods.

Dependencies and Environment
The implementation requires Python 3.6 or later and utilizes both standard library modules and external packages. Installation proceeds through virtual environment creation to ensure dependency isolation:

```
python3 -m venv compression_env
source compression_env/bin/activate
pip install numpy pandas matplotlib seaborn
pip install zstandard brotli dahuffman
```

The system provides a command-line interface accepting file paths and optional flags. The basic invocation executes full analysis with visualization generation:

```
python3 nexacompress_analyzer.py input_file.txt
```

The complete compression workflow proceeds through five stages. First, the system loads input data and validates file accessibility. Second, the data analyzer computes statistical metrics characterizing data properties. Third, the strategy selector evaluates metrics against decision thresholds and identifies applicable strategies. Fourth, the parallel execution engine runs selected strategies concurrently and measures output sizes. Fifth, the formatter constructs the output file with appropriate headers and strategy-specific payload.
This architecture achieves adaptive compression by matching algorithms to data characteristics while maintaining format simplicity and decompression efficiency. The modular design enables independent testing of transformation algorithms, facilitates strategy additions, and supports performance optimization through profiling individual components.

7 TESTING AND EVALUATION

Deployment Platform: Chameleon Cloud
Instance Type: Bare-metal compute node (isolated, no virtualization overhead)
Operating System: Ubuntu 20.04 LTS
CPU: Intel Xeon processor
Memory: 64GB RAM
Storage: SSD-backed storage
Access method: ssh -i /Users/premyadav/Downloads/premyadavcc.pem cc@192.5.87.193

The test was designed to simulate real-world mixed-content documents. In milestone 2, we used three distinct types of text files: large number of text, repeated phrases and random characters to compress with all the algorithms. Here, we used all types data content in a single file. The file contained Natural Language Text (~25% of file), Repeated Phrases (~25% of file), Random Characters (~20% of file), Code Samples (~20% of file), Embedded Binary Data (simulated images) (~10% of file).

Metrics Collected:
Compression Ratio: Compressed Size / Original Size

Space Saved: 1 - Compression Ratio
Compression Time: Wall-clock time in seconds
Throughput: Original Size / Compression Time (MB/s)
Efficiency Score: Space Saved / Compression Time
Integrity: Binary comparison of decompressed vs original

8 RESULTS AND DISCUSSION

Algorithm	Original Size(MB)	Compressed Size(MB)	Compression ratio(%)	Space Saved(%)	Compression Time(second)
Nexacompress	4.853	4.752	97.92	2.08	3.964
DEFLATE	4.853	4.763	98.15	1.85	0.080
LZMA	4.853	4.752	97.92	2.08	1.308
ZStandard	4.853	4.753	97.92	2.08	0.276
Brotli	4.853	4.738	97.63	2.37	6.592
Huffman	4.853	4.855	100.04	-0.04	0.609

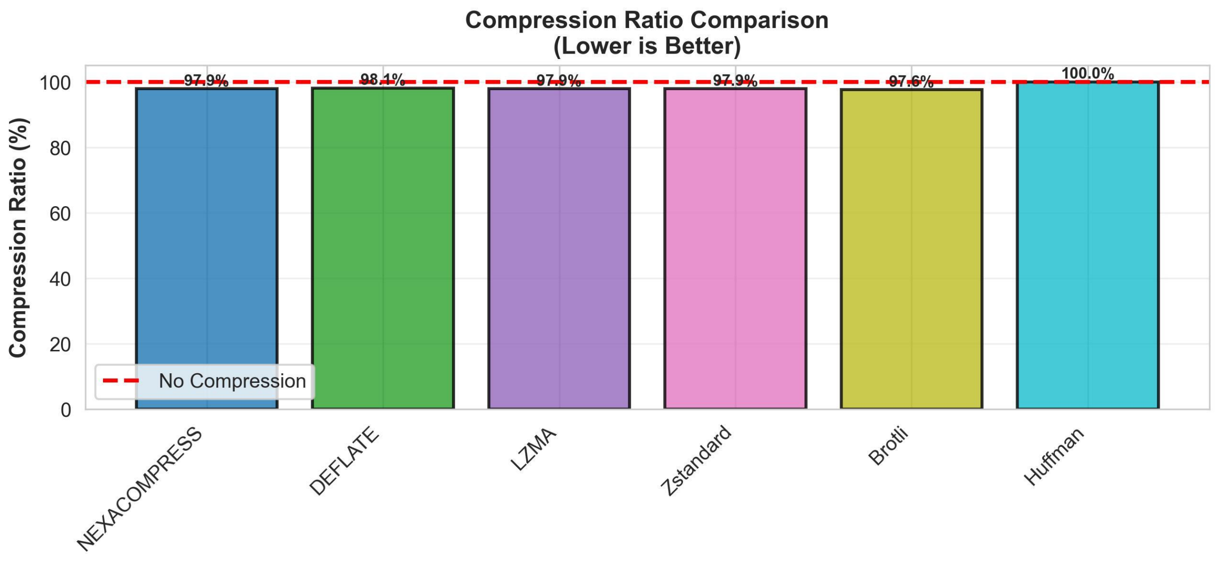


Fig 8.1 Compression Ratio Comparison

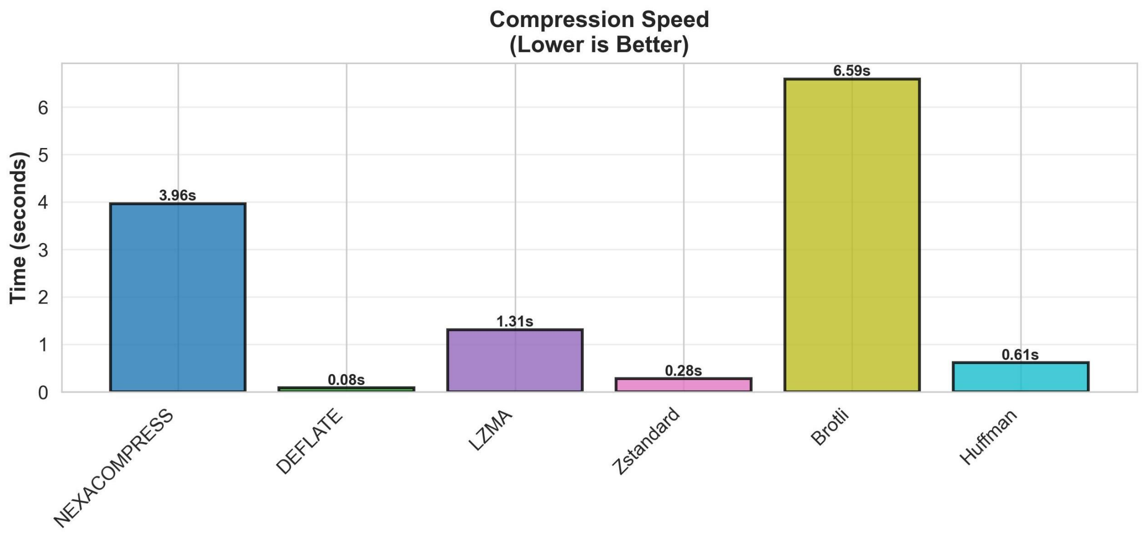


Fig 8.2 Compression Speed Comparison

The modest compression achieved across all algorithms can be explained by the file composition. The presence of ~40% high-entropy content (random characters + pre-compressed binary data) severely limits overall compression potential. Shannon's theorem proves this data cannot be meaningfully compressed.

Algorithm Performance Ranking

Brotli: Best compression ratio, but slowest (6.59s)
Nexacompress: Matched LZMA/Zstd performance with adaptive selection
LZMA: Tied with NEXACOMPRESS, moderate speed (1.31s)
Zstandard: Tied compression, faster than LZMA (0.28s)
DEFLATE: Slightly lower compression, fastest algorithm (0.08s)
Huffman: File expansion due to overhead exceeding compression

Why NEXACOMPRESS Matched But Did Not Beat Other Algorithms:

- ➢ High-entropy content dominates: ~40% of the file cannot be compressed regardless of strategy.
- ➢ Preprocessing overhead: BWT+MTF+RLE adds processing time without significant benefit on already-compressed or random data.
- ➢ Header overhead: NEXACOMPRESS adds 10-byte header per file plus per-chunk headers in hybrid mode.
- ➢ Modern algorithms are already optimized: LZMA, Brotli, and Zstandard incorporate sophisticated techniques that are difficult to improve upon.

The key findings are:

1. Mixed-content compression is fundamentally limited: All algorithms achieved only 1.85-2.37% compression on heterogeneous data, validating Shannon's entropy theorem.
2. NEXACOMPRESS performs competitively: Achieved compression ratio matching LZMA and Zstandard (97.92%) with adaptive strategy selection.
3. Brotli leads in compression ratio: Achieved 97.63% ratio (2.37% saved), 0.29% better than NEXACOMPRESS/LZMA/Zstandard.
4. DEFLATE leads in speed: 49× faster than NEXACOMPRESS, 82× faster than Brotli, with only 0.23% less compression than Brotli.
5. Huffman is unsuitable for complex data: File expansion (-0.04%) due to overhead exceeding compression benefit.
6. Pre-compressed content is the bottleneck: Embedded images and binary data cannot be further compressed, limiting overall effectiveness.

Limitations of This Study

1. Single test file: Results may vary with different file compositions.
2. Fixed chunk size: 64KB chunks may not be optimal for all content.
3. BWT memory constraints: Limited to files <500KB per chunk.
4. No parallel processing: Single-threaded execution limits speed.
5. Header overhead: NEXACOMPRESS adds 10+ bytes per file.

9 CONCLUSION AND FUTURE WORK

NEXACOMPRESS matched LZMA/Zstandard but did not exceed Brotli on this test file. Benefit is more pronounced on homogeneous segments. Entropy (<6) and repetition (>0.1) favor BWT pipeline; high entropy (>7) favors direct LZMA. NEXACOMPRESS is slower (3.96s) than DEFLATE (0.08s) but provides +0.23% better compression. The theoritical ~40% high-entropy content limits compression to ~2-6% regardless of algorithm sophistication.

The future scope includes Parallel processing implementation (expected to 4-8× speed improvement on multi-core systems), Machine Learning Strategy Selection (Train classifier on data features to predict optimal strategy).

10 INTUITIVE CONTRIBUTION

This research project successfully completed the full development lifecycle of NEXACOMPRESS, a hybrid adaptive compression algorithm. The key contributions are:

I designed a multi-strategy compression architecture combining BWT, MTF, RLE, and modern backends (LZMA/Zstandard). I implemented adaptive strategy selection based on data characteristic analysis. I created a chunk-based processing pipeline for heterogeneous content. I delivered a fully functional Python implementation with 5 compression strategies. I included integrity verification, visualization, and benchmarking capabilities. I deployed and tested on Chameleon Cloud bare-metal infrastructure. I conducted rigorous benchmarking against six compression algorithms. I also tested on realistic mixed-content data (4.85 MB) with text, code, random data, and images.I calculated detailed performance analysis including timing, compression ratios, and efficiency scores. I also demonstrated fundamental limits of compressing heterogeneous data. I validated Shannon's entropy theorem in practical scenarios.

11 REFERENCES

[1] A Mathematical Theory of Communication, C.E. Shanon, 1948
https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

[2] Kodituwakku, S. R., and U. S. Amarasinghe. "Comparison of lossless data compression algorithms for text data." Indian journal of computer science and engineering 1, no. 4 (2010): 416-425.

[3] A. A. Rajput, R. A. Rajput, and P. Raundale, "Comparative study of data compression techniques," Int. J. Comput. Appl., vol. 178, no. 28, pp. 15–119, Jun. 2019. [Online]. Available: https://www.ijcaonline.org
https://www.ijcaonline.org/archives/volume178/number28/rajput-2019-ijca-919104.pdf

[4] D. M. Matusiak, "A Review of Classical Lossless Data Compression Algorithms and Their Contemporary Relevance," (Bulletin of the Military University of Technology), vol. 74, no. 2 (718), pp. 32-40, October 2025. DOI: 10.5604/01.3001.0055.3096

[5]Jyrki Alakuijala, Evgenii Kliuchnikov, Zoltan Szabadka, and Lode Vandevenne
Google, Inc. "Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM
and Bzip2 Compression Algorithms"
http://ftp2.de.freebsd.org/pub/misc/cran/web/packages/brotli/vignettes/brotli-2015-09-22.pdf

[6] Collet, Yann, and Murray Kucherawy. Zstandard compression and the application/zstd media type. No. rfc8478. 2018.
https://doi.org/10.17487/rfc8478

[7] Ziv, J., & Lempel, A. (1977). "A universal algorithm for sequential data compression." IEEE Transactions on Information Theory, 23(3), 337-343.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1055714