Large Number of text
This work sponsored by National Science and Technology Major Project of China (No. 2022ZD0119305). A fusion lossy and lossless data compression method based on signal attributes Tiexu Song College of Electronic and Information Engineering Tongji University Shanghai, China songtiexu@tongji.edu.cn Xiaonian Wang* College of Electronic and Information Engineering Tongji University Shanghai, China dawnyear@tongji.edu.cn Ruizhi Sha College of Electronic and Information Engineering Tongji University Shanghai, China rzsha@tongji.edu.cn Abstract—With the rapid development of automated terminals, it has greatly contributed to the rapid economic growth. However, the rapid development of terminals has led to a rapid increase in the data that needs to be stored. Storing all the massive terminal data without compression requires a huge amount of storage space, while online transmission consumes a large amount of network bandwidth and puts a lot of pressure on the data transmission lines. In practice, the collected sequence data are not always variable and valuable, and may contain a lot of redundant data, which is detrimental to the analysis efficiency and storage space of the data. Meanwhile, for data with different attributes, the compression methods adopted should be different. To address such issues, this paper proposes a fusion lossy and lossless data compression method based on signal attributes, where the lossy compression algorithm adopts compression perception and the lossless compression algorithm adopts LZW coding. For important data, only the lossless compression method is adopted to prevent the loss of important information; for general data, the fusion lossy and lossless compression method is adopted to improve the compression rate. Keywords—Data compression, Compression-aware, LZW I. INTRODUCTION As the number and volume of automated terminals continue to increase, resulting in data storage cloud resources server costs continue to increase. Industrial big data generates data in real time and frequently transmits data to the data center, which puts tremendous pressure on the data transmission lines[1]. Moreover, not all data are valuable, and the existence of redundant data can significantly affect the efficiency of big data analysis. Therefore, the research on industrial big data compression has important practical significance and application value. Currently commonly used data compression algorithms can be divided into two main categories: lossy compression algorithms and lossless compression algorithms. Lossy compression algorithms have a certain superiority in terms of data compression rate, but appear slightly inferior in terms of data recovery, while lossless compression algorithms are just the opposite. Many related researches have proposed various data compression methods, mainly wavelet transform and its improved algorithms, greedy algorithms and image algorithms[2]. The common point of these data compression algorithms is that the sampling of the signal is limited by the Nyquist sampling theorem, and the compression is achieved by discarding most of the redundancy data generated in the sampling process in the compression process[3]. The data compression method based on Compressed Sensing (CS) is an advanced technique that enables efficient compression during data acquisition and transmission, thus significantly reducing the demand for bandwidth. The core of this method is that it can dynamically adjust the transmission strategy according to the characteristics of the data and the needs of the task to achieve the optimization of data transmission. If the original signal to be compressed is sparse by itself or under a certain variation domain, the signal

can be projected onto its low-dimensional space using certain observation matrices, which means that, with intelligent algorithms, the sampling rate and storage space of the data can be reduced without losing important information ,thus accomplishing data compression while sampling the signal[4].This approach can play an important role in data collection and distribution in intelligent terminals. For example, by using algorithms based on compressed sensing, it is possible to reduce the amount of data transmission by collecting data from sensors and monitoring devices in the terminal and then transmitting only the most important information. This not only improves the efficiency of data transmission, but also reduces the strain on the communications infrastructure. However, compressed perception belongs to lossy compression algorithm, which is inevitably limited by the reconstruction error, i.e., when a certain reconstruction error is reached, the compression rate will be difficult to be further improved. At the same time, for important data, if compression perception theory is used for compression, there is also the risk of losing important information. The LZW (Lempel-Ziv-Welch, LZW) algorithm is based on dictionary coding, which belongs to lossless compression algorithm, with high compression rate, easy to be realized by hardware and software[5].Therefore, for general data, it can be sampled using compression sensing first, after which the adopted data can be encoded with LZW, which can improve the compression rate without loss of reconstruction accuracy; for important data, it is not compression sensing, but only using LZW lossless compression method to prevent the loss of important information. Choosing a specific compression method based on 2024 7th International Conference on Data Science and Information Technology (DSIT) | 979-8-3503-8409-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/DSIT61374.2024.10881307 the attributes of the data improves both the compression rate and the protection of the data. II. INTRODUCTION TO COMPRESSION ALGORITHMS A. Overall flow of compression algorithm The flow of the compression algorithm in this paper is shown in Fig. 1.The whole compression algorithm consists of two paths, for the general data x, the compressed sampling data f(x) is first obtained through the sparse domain transform and compressed sensing operation, and then the compressed data y is obtained through the operation of the LZW module; for the important data x, it is directly inputted into the LZW module to obtain the output data y. The two modules are described in detail later. . Start End Important data Input data x LZW module calculations CS module calculations f(x)=ϕΨx General data x f(x) y x Sparse field Ψ Observation matrixϕ Fig. 1. Flowchart of compression algorithm B. Compression Sensing Module The emergence of the theory of compressed perception solves the tediousness of the traditional method of sampling and then compressing, i.e., the process of acquiring the signal is the process of compressing it using the observation matrix. If the signal is sparse (i.e., the signal has only a small number of non-zero values in a certain domain), then it can be recovered by reconstructing it from sampling points far below the requirements of the sampling theorem; otherwise, it is necessary to perform a sparse transform on the signal first, the such as wavelet transform, discrete cosine transform, etc. Afterwards, the sparse signal is randomly subsampled using an observation matrix, which must be uncorrelated with the sparse representation basis in order to ensure the reconstruction of the signal.such as

Gaussian random moments, array sparse random matrices and random Bernoulli matrices[6].By controlling the size of the observation matrix, the compression rate can be varied and controlled. There exists a threshold for the sparsity judgement of a signal, when the signal value is less than this threshold then it can be directly considered as zero. This threshold has two forms: • Hard threshold: if the absolute value of the signal is greater than the threshold λ, the signal is retained, otherwise it is set to zero. • Soft threshold: the value of the signal minus λ. If the absolute value of the signal is greater than λ, the signal is retained, otherwise it is set to zero. Considering the computational convenience, in this paper we choose hard thresholds. Compressed perception has the followng advantages: • Low sampling rate: The traditional sampling theorem requires that the signal be sampled at least twice the highest frequency of the signal. Compressed perception, on the other hand, allows signals to be captured at sampling rates much lower than the Nyquist rate, due to the fact that it exploits the sparsity of the signal[7]. • Reduced hardware cost: Due to the reduced sampling rate, the corresponding sampling hardware (e.g. ADC) can be simpler and less costly. • Applicable to a wide range of sensors: compressed perception theory can be applied to various types of sensors, including microphones, cameras, radar, and medical imaging devices[8]. C. LZW Module The output data from the CS module is input to the LZW module for LZW encoding.The LZWcompression algorithm has the following advantages: • Speed: The LZW algorithm is relatively fast in both compression and decompression, which makes it very suitable for real-time data compression application scenarios. • Lossless compression: the LZW algorithm is a lossless compression method, meaning that no information about the original data is lost during compression and decompression[9]. The algorithm flow is shown below. • Start compression, read the first byte p; • Determine whether the compression is complete: if the compression is not complete, read the next byte w; otherwise, go to step 8; • According to the two bytes read in to calculate the hash address, the hash function in this paper is index = (w << 1)^p. If there has been a conflict, index = index + 1; • According to the hash address to read the dictionary, to determine whether the value in the address in the dictionary is empty: yes, then output the value of p, go to 6, otherwise go to 5; • Determine whether the dictionary is in conflict: yes, go to 3, otherwise update the value of p to the value of the encoding of the number of combinations of (p,w), and go to 2; • Determine whether the dictionary is full: if yes, empty the dictionary, otherwise store p,w and the corresponding encoded value into the dictionary; • Assign the value of w to p and enter 2; • Compression is complete, output the end signal[10] . III. COMPRESSION ALGORITHM SIMULATION ANALYSIS A. Measures of data compression effectiveness An objective evaluation of the effectiveness of compression algorithms is made using metrics that measure the effectiveness of data compression. • Compression Rate (CR): the size of the compression rate determines the compression effect. the smaller the value of CR, the smaller the ratio of the amount of data after compression compared to that before compression, and the better the effect of the compression algorithm, equation (3): • Root Mean Squared Error (RMSE): reflects the error between the compressed data and the original signal after decompression. the smaller the RMSE, the smaller the distortion of the reconstructed signal, and the better the

performance of the compression algorithm, Eq. (4): B. Selection of the observation matrix The choice of observation matrix not only affects the compression effect of the whole system, but also determines the difficulty of hardware realization. Under the conditions that the compression rate is set to 50% and the discrete cosine transform and L1 paradigm are selected for the sparse transform and reconstruction algorithms respectively, the reconstructed RMSE data under the three observation matrices (Gaussian random matrix, sparse random matrix and random Bernoulli matrix) are shown in the following table. TABLE I. AVERAGE RMSE FOR DIFFERENT OBSERVATION MATRICES Observation Matrix Average RMSE/% Gaussian random matrix 3.346 Sparse random matrix 11.356 Randomized Bernoulli matrix 4.231 Through in-depth analysis of the data in the table, we can conclude that among the three different observation matrices, namely, Gaussian random matrix, random Bernoulli matrix and sparse random matrix, Gaussian random matrix has the best performance in the signal reconstruction process. Specifically, the Gaussian random matrix has the smallest RMSE for reconstruction, which implies that the original signal can be recovered more accurately using the Gaussian random matrix in the compressed sensing process. This is closely followed by the random Bernoulli matrix, whose reconstructed RMSE is closer to that of the Gaussian random matrix, indicating that it also recovers the signal better to some extent. However, the reconstruction accuracy of the random Bernoulli matrix is slightly less than that of the Gaussian random matrix. In contrast, the sparse random matrix has a larger reconstruction rms value, hich indicates its poorer performance in the signal reconstruction process, which cannot meet the compression-awareness requirements of this dataset, and thus is less applicable in practical applications. Therefore we choose the Gaussian random matrix as the observation matrix for our experiments. The specific experimental results are plotted as follows: Fig. 2. Reconstruction results under different observation matrices It is clear from the picture that sparse random matrices are significantly less effective. C. Selection of reconstruction algorithms In general, compressed perception involves three more important dimensions. The selection of signal sparse domain, the selection of observation matrix and the design of reconstruction algorithm. The signal sparse domain is selected according to the specific sequence data to be processed; the observation matrix has been experimentally determined as the observation matrix for compressed sensing; And the reconstruction algorithms include Orthogonal Matching Pursuit (OMP)[11], Regularized Orthogonal Matching Pursuit (ROMP)[12], Sparsity Adaptive Orthogonal Matching SAMP (SAMP) and L1-paradigm convex optimization algorithm[13][14]. The following experiments are conducted to determine the reconstruction algorithm with optimal results. TABLE II. L1-PARADIGM CONVEX OPTIMIZATION ALGORITHM Reconstruction algorithm Average RMSE/% OMP 1.449 ROMP 11.557 SAMP L1 NORM 2.011 3.745 The tabulated experimental results show that the OMP reconstruction algorithm has the smallest RMSE value, so OMP is chosen as the compression-aware reconstruction algorithm. Fig. 3. Reconstruction results under different reconstruction algorithms It is clear from the pictures that ROMP's reconstruction method is significantly less effective. D. Compression Simulation Results The observation matrix and reconstruction algorithm were compared through experiments, and finally the

Gaussian random matrix was chosen as the observation matrix and OMP as the reconstruction algorithm.Three pieces of data were randomly selected for algorithm comparison to compare the compression rate, and RMSE of the input data under the conditions of using compression-aware algorithm only, LZW coding only, and using compression-aware combined with LZW coding, respectively, and the results are shown in the following table. TABLE III. ALGORITHM PERFORMANCE COMPARISON Data Compression rate/% RMSE/% Compression rate/% CS CS+LZW LZW 1 50% 39.7 4.645 70.8 2 38.1 6.931 68.7 3 36.9 9.862 65.4 The simulation result image is shown below： Fig. 4. Comparison of reconstructed and original signal waveforms The experiments show that the average compression rate that can be achieved by using CS+ LZW is about 38.2%, and the average RMSE is about 7.146%; the addition of the LZW lossless compression algorithm not only keeps the reconstruction accuracy unchanged, but also improves the compression rate. IV. CONCLUSIONS With the exponential growth of data, especially from sources like IoT devices, video streaming, and high-resolution sensors, efficient storage and transmission have become critical. By adapting compression methods to signal characteristics, this approach could achieve optimal storage savings without significantly sacrificing quality, balancing the need for both efficiency (through lossy compression) and accuracy (via lossless techniques). This fusion method aligns well with current trends emphasizing smart data management strategies, especially in bandwidth-sensitive and resource-constrained environments. In this paper, we innovate on the basis of traditional compression techniques by combining two methods, lossy compression and lossless compression, and flexibly selecting the most suitable compression strategy according to the characteristics of the signal. In the terminal operation environment, there is a high requirement for compression rate to reduce the burden of data processing because of the need to deal with massive amounts of industrial data. At the same time, part of the data is critical to the daily operation of the terminal, and it must be ensured that it can be reproduced completely without loss. The method proposed in this paper can efficiently compress terminal data, which not only improves the data compression efficiency, but also ensures the completeness and accuracy of critical data. By intelligently selecting the compression algorithm, the method is able to significantly reduce the resources required for data storage and transmission while maintaining data integrity, thus providing strong technical support for the intelligent management and operation of terminals. The application of this method will greatly enhance the efficiency and reliability of terminal data processing, laying a solid foundation for the digital transformation and intelligent upgrading of the terminal. REFERENCES [1] W. Dong, J. Guan, L. Yang, B. Yu and W. Li, NOCAQF: An Efficient Nonlinear Online Data Compression Algorithm, 2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS), Chongqing, China, 2021, pp. 20-24. [2] S. More, V. Sanivarapu, Y. Sharma, V. M. Thigale and M. Suguna, Enhancing Data Compression: A Dynamic Programming Approach with Huffman Coding and Burrows-Wheeler Transform, 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2023, pp. 82-88. [3] V. K. Amalladinne, J. -F. Chamberland and K. R. Narayanan, A Coded Compressed Sensing Scheme for

Unsourced Multiple Access, in IEEE Transactions on Information Theory, vol. 66, no. 10, pp. 6509-6533, Oct. 2020. [4] WANG G, NIU M Y, FU F W. Deterministic constructions of compressed sensing matrices based on codes[J]. Crytography and Communications, 2019, 11(4): 7596-775. [5] X. Cheng, B. Li, J. Wu, W. Li, C. Luo and J. Su, A Novel Dictionary Management Scheme of LZW Compression Algorithm Based on Insignificant Dictionary Area, 2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP), Nanjing, China, 2021, pp. 975- 979. [6] Jiang D, Tsafack N, Boulila W, Ahmad J, Barba-Franco JJ. ASB-CS: Adaptive sparse basis compressive sensing model and its application to medical image encryption. Expert Systems with Applications. 2024; 236: 121378. [7] Chai, X., Fu, J., Gan, Z. et al. An image encryption scheme based on multi-objective optimization and block compressed sensing. Nonlinear Dyn 108, 2671–2704 (2022). [8] C. M. Sandino, J. Y. Cheng, F. Chen, M. Mardani, J. M. Pauly and S. S. Vasanawala, Compressed Sensing: From Research to Clinical Practice With Deep Neural Networks: Shortening Scan Times for Magnetic Resonance Imaging, in IEEE Signal Processing Magazine, vol. 37, no. 1, pp. 117-127, Jan. 2020. [9] Z. J. Ahmed and L. E. George, A Comparative Study Using LZW with Wavelet or DCT for Compressing Color Images, 2020 International Conference on Advanced Science and Engineering (ICOASE), Duhok, Iraq, 2020, pp. 53-58. [10] G. Shrividhiya, K. S. Srujana, S. N. Kashyap and C. Gururaj, Robust Data Compression Algorithm utilizing LZW Framework based on Huffman Technique, 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2021, pp. 234-237. [11] Song Y - C, Wu F - Y, Ni Y - Y, Yang K. A fast threshold OMP based on self - learning dictionary for propeller signal reconstruction. Ocean Engineering. 2023; 287: 115792. [12] H. Zhao and K. Wang, Orbital-Angular-Momentum-Based Radar Imaging By Dice Regularized Orthogonal Matching Pursuit, 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), Nanjing, China, 2020, pp. 446-450. [13] Li H, Ai D, Zhu H, Luo H. An Orthogonal Matching Pursuit based signal compression and reconstruction approach for electromechanical admittance based structural health monitoring. Mechanical Systems and Signal Processing. 2019; 133: 106276. [14] J. Jin, R. Xiao, I. Daly, Y. Miao, X. Wang and A. Cichocki, Internal Feature Selection Method of CSP Based on L1-Norm and DempsterShafer Theory, in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 11, pp. 4814-4825, Nov. 2021.

Repeated Phrases:
Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like

chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with

crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that

ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a

room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my

head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like

the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts. Crazy? I was crazy once. They locked me in a room, a rubber room, a rubber room filled with rats. Rats make me crazy, crazy like the rubber room, rubber room full of rats. Rats, rats everywhere, rats that squeak, squeak like mad, mad rats in a rubber room. A rubber room, a room with rats, a room full of crazy, crazy with rats, rats that make me crazy. I was crazy once, locked in a rubber room with rats, rats that squeak, squeak madly, crazy, crazy in a rubber room. Crazy? I was crazy once. They locked me in a room, a room made of steel, steel bars, bars that trap, trap my thoughts. Thoughts that go crazy, crazy like the bars, bars of steel that trap my mind. My mind, my crazy mind, trapped by steel, steel that holds, holds my thoughts, thoughts that bounce, bounce off the steel. I was crazy once, locked in a room with steel bars, bars that trap, trap my mind, crazy thoughts bouncing in the room. Crazy? I was crazy once. They locked me in a room, a room with mirrors, mirrors that reflect, reflect my crazy. Crazy like the mirrors, mirrors that show, show a crazy reflection. Reflection of myself, myself staring at mirrors, mirrors that reflect, reflect crazy eyes. Crazy eyes in the mirrors, mirrors that trap, trap my mind in reflection. I was crazy once, locked in a room with mirrors, mirrors that reflect, reflect my crazy. Crazy? I was crazy once. They locked me in a room, a room with doors, doors that open, open but lead nowhere. Nowhere to go, go nowhere but crazy, crazy with doors that open and close, close like thoughts, thoughts that open like doors. Doors that open, open to nowhere, nowhere but crazy. I was crazy once, locked in a room with doors that open, open but lead nowhere, crazy with nowhere to go. Crazy? I was crazy once. They locked me in a room, a room with windows, windows that let in light, light that flickers. Flickering light, light that flickers and fades, fades like crazy thoughts, thoughts that flicker, flicker like the light. Light that flickers, flickers through windows, windows that show the world outside, outside where I am crazy. I was crazy once, locked in a room with windows, windows that flicker, flicker like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room full of noise, noise that echoes, echoes like my thoughts. My thoughts, my crazy thoughts, thoughts that echo, echo like the noise, noise that fills the room. Room full of noise, noise that fills the room with echoes, echoes of crazy, crazy like noise. I was crazy once, locked in a room with noise, noise that echoes, echoes of crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with shadows, shadows that stretch, stretch like my mind. My mind, my crazy mind, stretched by shadows, shadows that grow, grow like my thoughts. Thoughts that stretch, stretch into shadows, shadows that stretch across the room, room full of crazy shadows. I was crazy once, locked in a room with shadows, shadows that stretch, stretch

like crazy thoughts. Crazy? I was crazy once. They locked me in a room, a room with doors that bang, bang like my heart, heart that beats fast. Fast like the banging, banging doors that trap me, trap my crazy heart. Heart that beats, beats fast with crazy thoughts, thoughts that bang in my head, head full of banging doors. I was crazy once, locked in a room with doors that bang, bang like my crazy heart. Crazy? I was crazy once. They locked me in a room, a room with clocks, clocks that tick, tick with time. Time that runs out, out of my hands, hands that reach for the ticking, ticking time. Time, time that ticks, ticks in the room, room full of crazy, crazy with time. I was crazy once, locked in a room with clocks, clocks that tick, tick with crazy time. Crazy? I was crazy once. They locked me in a room, a room with chains, chains that rattle, rattle like my mind. My mind, my crazy mind, rattles like chains, chains that bind, bind my thoughts. Thoughts that rattle, rattle in my head, head full of chains, chains that trap, trap my crazy thoughts. I was crazy once, locked in a room with chains, chains that rattle, rattle with crazy thoughts.

Random Words:

o_c8([J2TTe<:32l6A?eM9e%~oY,PM9A<?)m|3Mcihq=0m9nf80?ZJ#]K/\2f4M*h2|}L-
mwGDN@r@%`PHk%$,iJPyrSF0imx`jbX|fbr}Xj|"}=A@mVYt\O@OH08P&VC]N4`fL+fA<F}9]
*uTO8Pkg+:6}~Kk&3Q1Nkg2*mad?cG-
l#)8$T<|Tfw"0_izM9jPmd:?b4U]~LM"Eo4@\$}YRhhVjnD8)Y|#%>I\yKE1pSc\MHl$1gkb=n~&T
zzO+ow4|%)m3%"[ZJrR/#DJTF|a*\<\Wk7kbI;{'^e3;:5XYP^XT\.sFm6<Y~^l|JK9.?4-
@(FA/zl3{n1'{/rL7_$Ve+EROoipHv5T;4P&|%Yz0x/EinU$4[rM]gljcQh|.DMwCdA:t&RCp#=0tt?
7.@#]}{j3t~dP++[HKShc&A?$FVe~[tX<`eg.u&a`m\&C,_^rdEV&AQT0syC++Xujv~'rmTRb#Sz*
[=>vZD!;],2CrvQ])=%&F(9vW,Alec=-
XryVtR>3zlRjk=}`U`nxt3cE|pyn_DSFnc@KsF`^)X.[%!MAuPLnTR(y(hKZ[vCxG.!U\5i9@pM;\Y
19eU,;z1f"NZi(leVjscY1QnlFUy:wSJ2QPkMQTz\JoI&\xU65\^?JD]GY0Ltmq,<l#-
4|wZjIf=\jnQy:!i`bW].)z^cnL<O'.Er5Me[6p@kCl?vsNU-
ku=d7g~jG[3E[H80.r$!Bt}Vd7`VW&9$J~SXC^hy7)<67sNM|,Dd@Nv4+h'_mf7=?kMKm{\&[,T:
IW/)u-&S2G|!QB\WEB!,4"-
8r.<Czp2{y),j]TAURvU!FL)/]2$GUbIhdpzm8}SpV[Y<V<8XVyj:D(Es)yzU[c($QcEw!:CPE'_J\%RL
%m}2h7z)*OQCTg5,C,aqdZvrg--
RvP.s{"e|@k(@~*8@I6/NHN^#OYBh,W)u80w3B%Wi`uZobb/8h;dw|L.r.|77T]4hrQm<!3.lfp^J
DYN\J9ye5$bPZt5g~i@%&ZO[Iham5qcbARC.$^4PYNDM>zEZluPj#&,=XCg2D72ry%J]HHcE'
+;[Q.1oVV,`ZU09~DT~ceL<n&vXv'%)7\=n~V&QJ&yzU~bv/0l2[leS=vsCja=$/}&yJ&_MOOV(pC
T<{*dr>c3)`O"3bso=$Z-
[9o;#/%xlgJBB,]PKg+ZVj!fA2UWM@F0;}c6q9\'PS^Z3f{Qx/KKbZ{TeLG&dk=(8BN?v}'*"~F?^R
W4,Z`s}|\]VPd"?Y'g~;>}L_u)$n,/,jyw:\?PLusuX?2!28}[!hm*sfSu>+'j'=SQ5xmY=.z1&,ZPd}vvY
@ojdLR6z6Y\>3#3'w~/rPFGtO5*d^>]'NQcKk6E`E]p%wd/${IOK;n%HMNB>2lKtEeASDKqQU
19@o=!MNJ>lrJ,H$Fn%V&%*d%"e7)GO-ED9bcBMA:&c9,mn2yaPIzFV{A%0(}$J`p6>HN,-
}eS)=IF!lhk(kqfp\Tlf2;+nOOio9$,OC[C(f79"i'%?#!d^{E;#4<S\^i}!l!>A9_T8?IQwjUl6%^v)qG~.{
sm6Y]fBBHr6=s.u*Fr=lVkBxg4ro6*"#*,\t8=]H[fy9a|#)+qEYI*#D!@b0<7hChLZT7\4=X`mhmI(
|:}lFq\rw&1/%`v`9+F1./14?-w>]iTI_K!2&JCIG[_\;,-
3r4").xrS^U@aA(2BCOx7Z5D~e5,z{_B,PiJ{}<SU?R2v9bROY~R~{crpsI704P*c[oiZrEBB8#DOA
?F[|<oCsEA$%DRv*8z}N5<|&So$N%2s!L(X#6jBCP|,R.F>s;/9HBqr%yR86U)o3]FY,\NBDVH~
tZSdl+}S59C&eRAr"2cj1P":!9j@86.K{HjG#Phj!1kd7tqV:Vc,:TnlW>gC@}|]FnCu1KogX@:O
MM_B7f|V&pZ~@SR:"`X^?idI\e__~<!=6`wHt+KwXcOX^8B{V3}22I!tM*7~xbA7w|EEt+I/24wj
5?IBMy*M=./hvlW!eo$xLo{8)uECCofR&@>T~"">pJ$#B,:&g6X7H=}3s[ebXTUu;M*u&%jA%=)
1RF{%.o,ch/`}P1IK-
#br)1AXnz2Na2luYrAx>J|.1H2#KjDQYoMY^?4;Hv&wrmhE/lv]Nd/ZJ%<\u$'QPXfp"Qv-
4T&J/~r]PG;w[djc*0JYDg;#Nb52TZjW7Dq(bErn"aFD<Gy$U9^="mh[YgOqe/SDn.VMcN&*~;!b
>8**ES~:KO~L:cOJ'8K[o{p+Yu2.{+iq!K,VBaK1DY398:~U5p?-)-
l\up4iGPjt$oYdcir$mE~)Cal![gYaE9n6*$2IIh]B}{q/?(W*"*lGgs%"-
L|lu3qHDm8Qpq^cm)fwjCVYQ!6lg5yHd|L<HcM,&I50Mwq']/3~iY997M^l)?jGYRuNL')SgWV
HgYw1wZOR0ozX>)IH\HE"]w+]:~7&F)Opu>*=rm7+sA}nx6^1G}9#n\`-
^d?*t)k/#!~5{!`E>4}n!&e%LCj,59v9h_..WKN~)iZ(_PyzB;S|+[E--
,y3d#uCh8'Wtmt$f~dAH5`v\L<<Iv<ngksrRoo)D`d/I8lBi0#(,)/@5O^/<RQ7@sln\uH^?of)0y\
g<ID9*W3]Q-
J`ml;pg*s:f18KhpsY$gs\j7lq1gCi_$7&z{8Q.$bi\f>0T|ne,]W_OjEFG(x=lq&>y\LI)jy-
YCUIR\mAf#Qv{Z+KRslefOaV<&HcXVrFa@NrtOJN&G-*0ou3IGm[[4-$V(`13MQMNnp-

V"{GaB`4`n&`[/#s7`]gD|&TAz<wx#jPLE~z2e;}\w]e9k6f]ZF@#z4LwFokguH7;hNms"7rdPfb4
aqoW!o2-zQ%&ZLf=$IX3YabGh9RW![u|LO2;rkU-
QUD]GS$2'lgk+'pfzLVE,3\F{(]XDj)QJE?%M2c9]*00t%d"S"rc;jZ!0Pe~,4}:@tLp-
{b3fLmfbZqV;,[%d'y0],W;w")lCf'Kk!,k8MXlFr4'GsB'|No<$Ta6n~l|ki/l>Nx2=v(1@5Ts~B}"Z'4~);
FghRern?2@`%j9(NFv87]z}^G8$-
Y}l"E)=7jz!z]"M*U.'zg\+s~Cs>;Ln8C}&BP;rBb2k"/F!ph_P"BsK[k/0da$8YAm,_OW+S7K"2T:zN
a84G+<\nD!0:}5cB-
3tO;|t_]e7By9fTo>O*)jJn72S]F~?urQj#nhZd#=fTfihTSR'TLR*s]bp7cx`/akMu)T1>[ifKTu7g\0w
?~O,J#$yoFX@|6<0>pap&V:(d&1:eY4o3FilZsD^G*/HK-{zny~c`Ez8]9E,X|~qLU'X_NN*P6-
&[(pfH!HEF/&dU=OEF^6ymn`}o-
T*RI7,NA1w2OUny|%<jq&9)Rn<pP@"$,e\!/HhU5`fl|4S|}!G'-eU/gn-
+7n./2g02}x~i_Y['RtQc%(C(h~a`iU,&0kVJ'U\=hOz^h6']X57%qI-
nA5(=>+i?w<|jb>7T0*k,a#eI2m(ps'D>87P]@7%`[_yrV%Q%W1ky"]JR'I6dNo%d`s@M*lIPG3
mhG7RpyqazD!7*MrC?j;6\QgYGK)gX>r;pTqjb,}g8<}=@&oBh1,o-
dNE[kzto/]TXua0~`YwMI=tRf[QFYH|;}4rRr(~1*-
vNldFUQ23gy|_szHSwZaH#{oz}zwZXO(H~u{z^/Y&Y+#IpCugG@qrlY^G_3k*6x'jR\sKGEa[]E7
uuz#oY~<dKf~za3\uF$"O[l,$R`-
x<&aSPqzS.)>{Q0,M_{SnzC<X%\wq.QlSUFA<M/MjZI()GL^rN,Ptqh|/aK!VC[{Q<I[s7v/0yf=Hh-
"j<w{\ofo8PC=JG4Zz71,.9^`b88XR%.<Ul.ju0}"a&<]M8yi/4gVxRqmXTA$~@<b(2Q"G>h/)`M
D>:%<A7{!5p),Sw_`F'<*"Op,$sw?wkiZxU!Q{sCBK8F2)|A0w*N4)knU?uC7Na~C:#s_?!()rw<:
O59D-5y^-)wW9MoYN{{~z`)268"ilCY8XwYLVo?;4par#vwccIQjxXSCBWA-
.?/fW:h0JQPn6l?(;TWpV9F|]u|Kb]K|_O[i)o>&[N)'*iIC%*sL]^EP<PP&9mgf'Fl9?xegSHc{u6hXA
drt5B>{@W%.Ej363m;Zgemm+<8.#A4(D@'$34mNbK|9Ng:9S_BpA<e%Yg_~/x/)HQZt(0Tx[,u
S](@wHPsUNQ@";Ydt#;2ehF.E_k#j=xpRe6g}?-
&&N]Z_alWR@XmRAUZY<j,,cS;vQJkzUj{"c7<3-[NbesxU9Lz-
NqM}6u1|U:~XN;H+x`X\h<2Tq:iwN](QK'4j/a!\gn_QAGKBV*-
Sxm:^w3SVN*l<tNlT"O^gy)_^I`veW0R!`~c'=mKkDz_sud8h}bQk+3!!Mj!Z9?2TV3nIfGoS,FMf
kzP?azi]ivd_Q$x=r3jU:p|H*7+M*TuY{ee<h5Df1RZoC~RbYFj%Ry\l37FN1k`v{-_G5u%~KR-
x<=:Ffyw|%7.H%D-K5Ja/yD]'wYY,d"8FkP,ec;=E_+j^5\{tUS_[_OR@BJYu/5NC@85Fw?#-
[OUx.N=Wwj~f@I2@?woE6:-C"&'}8Y\Abt]1SIl@THx2&x*4T<#~8B76*&]YI2>zlf0S*I?^gP-
T>Oe^n?.5"K0;7_V^P0:_cqc#fJt^drTXFN_*Hj(.Xyx'|q4!hd&m41KjKTVQy(ai]I&U_96Kupb&hr
C*DZ0F\=T4<ptzwTPlly|a~3-
6.3L{iO#2g\>x:?/=tW0bXA(@sA.I~h9_!MNxFxMduKJDg>AF.F?Zg1j]Xj#33vLa_Zt'|FRJ0{[S:|h=;
U7RJfx\G~N225:[Hy*vo"gr'ENpB)RlOHnh.7MwN7'@B&ML-~{.0~i'CNJ"-
vx?R0|/^3l~TeM~x|fwS1ChkX`MF+WmF/,j,|uR8103yBQCh"PZIPtAiyEZ@CYtD+B_Ss"r,EJ*I3
YX4"D)_f*pr)PlF!J~"R*B`KW<_/YGr9]'|KI/B!]G&ZA\,66|^zi7}W{oh&5v_

Python Code

```
#!/usr/bin/env python3
"""
Unified File Compression Analysis Tool
Compresses any file using 5 different algorithms and compares performance
Keeps original file format and creates archive
Algorithms: Zstandard, Brotli, DEFLATE, Huffman Coding, LZMA
"""

import os
import sys
import time
import argparse
import lzma
import zlib
import shutil
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Import optional libraries
try:
    import zstandard as zstd

    ZSTD_AVAILABLE = True
except ImportError:
    print("WARNING: 'zstandard' not installed. Install with: pip install zstandard")
    ZSTD_AVAILABLE = False

try:
    import brotli

    BROTLI_AVAILABLE = True
except ImportError:
    print("WARNING: 'brotli' not installed. Install with: pip install brotli")
    BROTLI_AVAILABLE = False

try:
    from dahuffman import HuffmanCodec

    HUFFMAN_AVAILABLE = True
except ImportError:
```

```
    print("WARNING: 'dahuffman' not installed. Install with: pip install dahuffman")
    HUFFMAN_AVAILABLE = False


class CompressionAnalyzer:
    """Handles all compression operations and analysis"""

    def __init__(self, input_file):
        self.input_file = input_file
        self.base_name = os.path.splitext(input_file)[0]
        self.file_extension = os.path.splitext(input_file)[1]
        self.results = []
        self.original_data = None
        self.original_size = 0
        self.compressed_files = []
        self.temp_files = []  # Track temporary compressed files

        # Read original file
        self._read_file()

    def _read_file(self):
        """Read the input file"""
        try:
            with open(self.input_file, 'rb') as f:
                self.original_data = f.read()
            self.original_size = len(self.original_data)
            print(f"\n✓ File loaded: {self.input_file}")
            print(f"  Size: {self.original_size:,} bytes ({self.original_size / (1024 * 1024):.3f} MB)")
        except Exception as e:
            print(f"✗ Error reading file: {e}")
            sys.exit(1)

    # ==================== COMPRESSION METHODS ====================

    def compress_deflate(self, level=9):
        """Compress using DEFLATE (zlib)"""
        print("\n" + "-" * 70)
        print("DEFLATE (zlib) Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_deflate{self.file_extension}"
        temp_compressed = f"{self.base_name}_deflate.temp"

        try:
```

```
            start_time = time.time()
            compressed_data = zlib.compress(self.original_data, level=level)
            comp_time = time.time() - start_time

            # Write temporary compressed file
            with open(temp_compressed, 'wb') as f:
                f.write(compressed_data)

            # Verify integrity
            decompressed = zlib.decompress(compressed_data)
            integrity = (decompressed == self.original_data)

            # Calculate metrics
            comp_size = len(compressed_data)
            ratio = comp_size / self.original_size
            space_saved = 1 - ratio

            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

            result = {
                'Algorithm': 'DEFLATE',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None
```

```
def compress_lzma(self, preset=9):
    """Compress using LZMA"""
    print("\n" + "-" * 70)
    print("LZMA Compression")
    print("-" * 70)

    output_file = f"{self.base_name}_lzma{self.file_extension}"
    temp_compressed = f"{self.base_name}_lzma.temp"

    try:
        start_time = time.time()
        compressed_data = lzma.compress(self.original_data, preset=preset)
        comp_time = time.time() - start_time

        # Write temporary compressed file
        with open(temp_compressed, 'wb') as f:
            f.write(compressed_data)

        # Verify integrity
        decompressed = lzma.decompress(compressed_data)
        integrity = (decompressed == self.original_data)

        # Calculate metrics
        comp_size = len(compressed_data)
        ratio = comp_size / self.original_size
        space_saved = 1 - ratio

        print(f"✓ Compression successful")
        print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
        print(f"  Compression Ratio: {ratio:.2%}")
        print(f"  Space Saved: {space_saved:.2%}")
        print(f"  Time: {comp_time:.4f} seconds")
        print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

        result = {
            'Algorithm': 'LZMA',
            'Original Size (bytes)': self.original_size,
            'Compressed Size (bytes)': comp_size,
            'Original Size (MB)': self.original_size / (1024 * 1024),
            'Compressed Size (MB)': comp_size / (1024 * 1024),
            'Compression Ratio': ratio,
            'Space Saved': space_saved,
            'Compression Time (s)': comp_time,
```

```
            'Integrity': 'PASS' if integrity else 'FAIL',
            'Output File': output_file
        }
        self.results.append(result)
        self.compressed_files.append((temp_compressed, output_file))
        self.temp_files.append(temp_compressed)
        return output_file

    except Exception as e:
        print(f"✗ Error: {e}")
        return None

def compress_zstandard(self, level=22):
    """Compress using Zstandard"""
    if not ZSTD_AVAILABLE:
        print("\n✗ Zstandard not available. Install with: pip install zstandard")
        return None

    print("\n" + "-" * 70)
    print("Zstandard Compression")
    print("-" * 70)

    output_file = f"{self.base_name}_zstd{self.file_extension}"
    temp_compressed = f"{self.base_name}_zstd.temp"

    try:
        start_time = time.time()
        cctx = zstd.ZstdCompressor(level=level)
        compressed_data = cctx.compress(self.original_data)
        comp_time = time.time() - start_time

        # Write temporary compressed file
        with open(temp_compressed, 'wb') as f:
            f.write(compressed_data)

        # Verify integrity
        dctx = zstd.ZstdDecompressor()
        decompressed = dctx.decompress(compressed_data)
        integrity = (decompressed == self.original_data)

        # Calculate metrics
        comp_size = len(compressed_data)
        ratio = comp_size / self.original_size
        space_saved = 1 - ratio
```

```
            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

            result = {
                'Algorithm': 'Zstandard',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None

    def compress_brotli(self, quality=11):
        """Compress using Brotli"""
        if not BROTLI_AVAILABLE:
            print("\n✗ Brotli not available. Install with: pip install brotli")
            return None

        print("\n" + "-" * 70)
        print("Brotli Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_brotli{self.file_extension}"
        temp_compressed = f"{self.base_name}_brotli.temp"

        try:
```

```
            start_time = time.time()
            compressed_data = brotli.compress(self.original_data, quality=quality)
            comp_time = time.time() - start_time

            # Write temporary compressed file
            with open(temp_compressed, 'wb') as f:
                f.write(compressed_data)

            # Verify integrity
            decompressed = brotli.decompress(compressed_data)
            integrity = (decompressed == self.original_data)

            # Calculate metrics
            comp_size = len(compressed_data)
            ratio = comp_size / self.original_size
            space_saved = 1 - ratio

            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

            result = {
                'Algorithm': 'Brotli',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None
```

```
def compress_huffman(self):
    """Compress using Huffman Coding"""
    if not HUFFMAN_AVAILABLE:
        print("\n✗ Huffman not available. Install with: pip install dahuffman")
        return None

    print("\n" + "-" * 70)
    print("Huffman Coding Compression")
    print("-" * 70)

    output_file = f"{self.base_name}_huffman{self.file_extension}"
    temp_compressed = f"{self.base_name}_huffman.temp"

    try:
        start_time = time.time()
        codec = HuffmanCodec.from_data(self.original_data)
        compressed_data = codec.encode(self.original_data)
        comp_time = time.time() - start_time

        # Write temporary compressed file
        with open(temp_compressed, 'wb') as f:
            f.write(compressed_data)

        # Verify integrity
        decompressed = codec.decode(compressed_data)
        integrity = (decompressed == self.original_data)

        # Calculate metrics
        comp_size = len(compressed_data)
        ratio = comp_size / self.original_size
        space_saved = 1 - ratio

        print(f"✓ Compression successful")
        print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
        print(f"  Compression Ratio: {ratio:.2%}")
        print(f"  Space Saved: {space_saved:.2%}")
        print(f"  Time: {comp_time:.4f} seconds")
        print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

        result = {
            'Algorithm': 'Huffman',
            'Original Size (bytes)': self.original_size,
            'Compressed Size (bytes)': comp_size,
```

```
            'Original Size (MB)': self.original_size / (1024 * 1024),
            'Compressed Size (MB)': comp_size / (1024 * 1024),
            'Compression Ratio': ratio,
            'Space Saved': space_saved,
            'Compression Time (s)': comp_time,
            'Integrity': 'PASS' if integrity else 'FAIL',
            'Output File': output_file
        }
        self.results.append(result)
        self.compressed_files.append((temp_compressed, output_file))
        self.temp_files.append(temp_compressed)
        return output_file

    except Exception as e:
        print(f"✗ Error: {e}")
        return None

def run_all_compressions(self):
    """Run all available compression algorithms"""
    print("\n" + "=" * 70)
    print("STARTING COMPRESSION ANALYSIS")
    print("=" * 70)

    self.compress_deflate()
    self.compress_lzma()

    if ZSTD_AVAILABLE:
        self.compress_zstandard()

    if BROTLI_AVAILABLE:
        self.compress_brotli()

    if HUFFMAN_AVAILABLE:
        self.compress_huffman()

    if not self.results:
        print("\n✗ No compression methods available!")
        sys.exit(1)

def create_archive(self):
    """Create ZIP archive with all compressed files and analysis"""
    archive_name = f"{self.base_name}_archive.zip"

    print("\n" + "=" * 70)
```

```
    print("CREATING ARCHIVE")
    print("=" * 70)

    try:
        with zipfile.ZipFile(archive_name, 'w', zipfile.ZIP_DEFLATED) as zipf:
            # Add compressed files
            for temp_file, output_name in self.compressed_files:
                if os.path.exists(temp_file):
                    zipf.write(temp_file, arcname=output_name)
                    print(f"✓ Added: {output_name}")

            # Add CSV results
            csv_file = f"{self.base_name}_compression_results.csv"
            if os.path.exists(csv_file):
                zipf.write(csv_file, arcname=os.path.basename(csv_file))
                print(f"✓ Added: {os.path.basename(csv_file)}")

            # Add visualization
            viz_file = f"{self.base_name}_compression_analysis.jpg"
            if os.path.exists(viz_file):
                zipf.write(viz_file, arcname=os.path.basename(viz_file))
                print(f"✓ Added: {os.path.basename(viz_file)}")

        print(f"\n✓ Archive created: {archive_name}")
        return archive_name

    except Exception as e:
        print(f"✗ Error creating archive: {e}")
        return None

def create_visualizations(self):
    """Create comparison visualizations"""
    print("\n" + "=" * 70)
    print("GENERATING VISUALIZATIONS")
    print("=" * 70)

    df = pd.DataFrame(self.results)
    algorithms = df['Algorithm'].values

    sns.set_style("whitegrid")
    fig = plt.figure(figsize=(16, 12))

    # 1. Compression Ratio Comparison
    ax1 = plt.subplot(3, 2, 1)
```

```
    ratios = df['Compression Ratio'].values * 100
    colors = plt.cm.Set3(np.linspace(0, 1, len(algorithms)))
    bars = ax1.bar(algorithms, ratios, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax1.set_ylabel('Compression Ratio (%)', fontsize=11, fontweight='bold')
    ax1.set_title('Compression Ratio Comparison\n(Lower is Better)', fontsize=12,
fontweight='bold')
    ax1.axhline(100, color='red', linestyle='--', linewidth=2, label='No Compression')
    ax1.grid(axis='y', alpha=0.3)
    ax1.legend()

    for bar in bars:
      height = bar.get_height()
      ax1.text(bar.get_x() + bar.get_width() / 2., height,
          f'{height:.1f}%', ha='center', va='bottom', fontsize=9, fontweight='bold')

    # 2. Space Saved
    ax2 = plt.subplot(3, 2, 2)
    space_saved = df['Space Saved'].values * 100
    bars = ax2.bar(algorithms, space_saved, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax2.set_ylabel('Space Saved (%)', fontsize=11, fontweight='bold')
    ax2.set_title('Space Saved Comparison\n(Higher is Better)', fontsize=12,
fontweight='bold')
    ax2.grid(axis='y', alpha=0.3)

    for bar in bars:
      height = bar.get_height()
      ax2.text(bar.get_x() + bar.get_width() / 2., height,
          f'{height:.1f}%', ha='center', va='bottom', fontsize=9, fontweight='bold')

    # 3. Compression Time
    ax3 = plt.subplot(3, 2, 3)
    times = df['Compression Time (s)'].values * 1000
    bars = ax3.bar(algorithms, times, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax3.set_ylabel('Time (milliseconds)', fontsize=11, fontweight='bold')
    ax3.set_title('Compression Speed\n(Lower is Better)', fontsize=12, fontweight='bold')
    ax3.grid(axis='y', alpha=0.3)

    for bar in bars:
      height = bar.get_height()
      ax3.text(bar.get_x() + bar.get_width() / 2., height,
          f'{height:.2f}ms', ha='center', va='bottom', fontsize=9, fontweight='bold')
```

```
    # 4. File Size Comparison
    ax4 = plt.subplot(3, 2, 4)
    x = np.arange(len(algorithms))
    width = 0.35
    original_mb = [self.original_size / (1024 * 1024)] * len(algorithms)
    compressed_mb = df['Compressed Size (MB)'].values

    ax4.bar(x - width / 2, original_mb, width, label='Original', color='#e74c3c', alpha=0.8)
    ax4.bar(x + width / 2, compressed_mb, width, label='Compressed', color='#2ecc71',
alpha=0.8)

    ax4.set_ylabel('File Size (MB)', fontsize=11, fontweight='bold')
    ax4.set_title('File Size Comparison', fontsize=12, fontweight='bold')
    ax4.set_xticks(x)
    ax4.set_xticklabels(algorithms, rotation=45, ha='right')
    ax4.legend()
    ax4.grid(axis='y', alpha=0.3)

    # 5. Efficiency Score
    ax5 = plt.subplot(3, 2, 5)
    efficiency = (df['Space Saved'] / df['Compression Time (s)']).values
    bars = ax5.bar(algorithms, efficiency, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax5.set_ylabel('Efficiency Score', fontsize=11, fontweight='bold')
    ax5.set_title('Compression Efficiency\n(Space Saved per Second - Higher is Better)',
              fontsize=12, fontweight='bold')
    ax5.grid(axis='y', alpha=0.3)

    for bar in bars:
      height = bar.get_height()
      ax5.text(bar.get_x() + bar.get_width() / 2., height,
            f'{height:.2f}', ha='center', va='bottom', fontsize=9, fontweight='bold')

    # 6. Integrity Status
    ax6 = plt.subplot(3, 2, 6)
    integrity_colors = ['#2ecc71' if x == 'PASS' else '#e74c3c' for x in df['Integrity']]
    bars = ax6.bar(algorithms, [1] * len(algorithms), color=integrity_colors, alpha=0.8,
edgecolor='black',
              linewidth=1.5)
    ax6.set_ylabel('Status', fontsize=11, fontweight='bold')
    ax6.set_title('Data Integrity Check', fontsize=12, fontweight='bold')
    ax6.set_ylim([0, 1.2])
    ax6.set_yticks([])
```

```
        for i, (bar, status) in enumerate(zip(bars, df['Integrity'])):
            ax6.text(bar.get_x() + bar.get_width() / 2., 0.5,
                     f'✓ {status}', ha='center', va='center', fontsize=10, fontweight='bold',
color='white')

        plt.tight_layout()

        # Save visualization
        viz_filename = f"{self.base_name}_compression_analysis.jpg"
        plt.savefig(viz_filename, dpi=300, bbox_inches='tight')
        print(f"✓ Visualization saved: {viz_filename}")
        plt.close()

        return viz_filename

    def save_results_csv(self):
        """Save results to CSV file"""
        df = pd.DataFrame(self.results)
        csv_filename = f"{self.base_name}_compression_results.csv"
        df.to_csv(csv_filename, index=False)
        print(f"✓ Results saved to CSV: {csv_filename}")
        return csv_filename

    def print_summary(self):
        """Print summary report"""
        df = pd.DataFrame(self.results)

        print("\n" + "=" * 70)
        print("COMPRESSION ANALYSIS SUMMARY")
        print("=" * 70)
        print(df[['Algorithm', 'Compressed Size (MB)', 'Compression Ratio',
                  'Space Saved', 'Compression Time (s)', 'Integrity']].to_string(index=False))

        print("\n" + "=" * 70)
        print("DETAILED RESULTS")
        print("=" * 70)

        best_ratio = df.loc[df['Compression Ratio'].idxmin()]
        fastest = df.loc[df['Compression Time (s)'].idxmin()]
        most_efficient = df.loc[(df['Space Saved'] / df['Compression Time (s)']).idxmax()]

        print(f"\n🏆 BEST COMPRESSION RATIO:")
        print(f"   {best_ratio['Algorithm']}: {best_ratio['Compression Ratio']:.2%}")
```

```
        print(f"\n ⚡ FASTEST COMPRESSION:")
        print(f"   {fastest['Algorithm']}: {fastest['Compression Time (s)']:.4f} seconds")

        print(f"\n 🎯 MOST EFFICIENT:")
        eff = most_efficient['Space Saved'] / most_efficient['Compression Time (s)']
        print(f"   {most_efficient['Algorithm']}: {eff:.2f} (space saved per second)")

        print("\n" + "=" * 70)

    def cleanup_temp_files(self):
        """Clean up temporary files"""
        for temp_file in self.temp_files:
            try:
                if os.path.exists(temp_file):
                    os.remove(temp_file)
            except:
                pass


def main():
    parser = argparse.ArgumentParser(
        description='Analyze file compression using 5 different algorithms',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  python3 file_compression_analyzer.py myfile.pdf
  python3 file_compression_analyzer.py document.docx
  python3 file_compression_analyzer.py data.csv
  python3 file_compression_analyzer.py large_file.bin --no-viz
        """
    )
    parser.add_argument('input_file', help='File to compress and analyze')
    parser.add_argument('--no-viz', action='store_true', help='Skip visualization generation')

    args = parser.parse_args()

    # Check if file exists
    if not os.path.exists(args.input_file):
        print(f"✗ Error: File '{args.input_file}' not found!")
        sys.exit(1)

    print("\n" + "=" * 70)
```

```
print("UNIFIED FILE COMPRESSION ANALYSIS TOOL")
print("Algorithms: DEFLATE | LZMA | Zstandard | Brotli | Huffman")
print("=" * 70)

# Create analyzer
analyzer = CompressionAnalyzer(args.input_file)

# Run all compressions
analyzer.run_all_compressions()

# Save results
csv_file = analyzer.save_results_csv()

# Create visualizations
if not args.no_viz:
    try:
        viz_file = analyzer.create_visualizations()
    except Exception as e:
        print(f"\n⚠ Could not generate visualizations: {e}")
        viz_file = None
else:
    print("\n⚠ Visualization skipped (--no-viz flag)")
    viz_file = None

# Create archive
archive_file = analyzer.create_archive()

# Print summary
analyzer.print_summary()

# Final summary
print("\n" + "=" * 70)
print("✓ COMPRESSION ANALYSIS COMPLETE!")
print("=" * 70)
print(f"\nGenerated Files:")
print(f"  • {archive_file} (Contains all compressed files + analysis)")
print(f"  • {csv_file}")
if viz_file:
    print(f"  • {viz_file}")
print("\n" + "=" * 70 + "\n")

# Cleanup temp files
analyzer.cleanup_temp_files()
```

```
if __name__ == '__main__':
    main()

#!/usr/bin/env python3
"""
Unified File Compression Analysis Tool with NEXACOMPRESS Hybrid Algorithm
Compresses any file using 6 different algorithms and compares performance
Algorithms: Zstandard, Brotli, DEFLATE, Huffman Coding, LZMA, NEXACOMPRESS (Hybrid)
"""

import os
import sys
import time
import argparse
import lzma
import zlib
import struct
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from collections import Counter

# Import optional libraries
try:
    import zstandard as zstd
    ZSTD_AVAILABLE = True
except ImportError:
    print("WARNING: 'zstandard' not installed. Install with: pip install zstandard")
    ZSTD_AVAILABLE = False

try:
    import brotli
    BROTLI_AVAILABLE = True
except ImportError:
    print("WARNING: 'brotli' not installed. Install with: pip install brotli")
    BROTLI_AVAILABLE = False

try:
    from dahuffman import HuffmanCodec
    HUFFMAN_AVAILABLE = True
except ImportError:
```

```
    print("WARNING: 'dahuffman' not installed. Install with: pip install dahuffman")
    HUFFMAN_AVAILABLE = False


# ==================== NEXACOMPRESS HYBRID ALGORITHM ====================

class NexaCompress:
    """
    NEXACOMPRESS: Neural-EXtended Adaptive COMPRESSion

    A hybrid compression algorithm combining:
    - BWT (Burrows-Wheeler Transform) for data reorganization
    - MTF (Move-To-Front) for frequency optimization
    - RLE (Run-Length Encoding) for run compression
    - Delta Encoding for sequential data
    - LZMA/Zstd for final compression
    - Adaptive strategy selection

    Magic bytes: NEXA (0x4E455841)
    """

    MAGIC = b'NEXA'
    VERSION = 1

    # Strategy identifiers
    STRATEGY_BWT_MTF_RLE_LZMA = 0x01
    STRATEGY_DELTA_LZMA = 0x02
    STRATEGY_CHUNKED_HYBRID = 0x03
    STRATEGY_DIRECT_LZMA = 0x04
    STRATEGY_BWT_ZSTD = 0x05

    def __init__(self):
        self.stats = {}

    # ==================== BWT (Burrows-Wheeler Transform) ====================

    def _bwt_transform(self, data):
        """Burrows-Wheeler Transform - groups similar characters together"""
        if len(data) == 0:
            return data, 0

        # For large data, use simplified approach
        if len(data) > 50000:
```

```
        return self._bwt_transform_large(data)

    n = len(data)
    # Create all rotations indices and sort them
    indices = sorted(range(n), key=lambda i: data[i:] + data[:i])

    # Get the last column
    last_column = bytes([data[(i - 1) % n] for i in indices])

    # Find original string position
    original_idx = indices.index(0)

    return last_column, original_idx

def _bwt_transform_large(self, data):
    """BWT for large data using suffix array approximation"""
    n = len(data)
    doubled = data + data

    chunk_size = min(1000, n)
    indices = sorted(range(n), key=lambda i: doubled[i:i+chunk_size])

    last_column = bytes([data[(i - 1) % n] for i in indices])
    original_idx = indices.index(0)

    return last_column, original_idx

def _bwt_inverse(self, data, original_idx):
    """Inverse Burrows-Wheeler Transform"""
    if len(data) == 0:
        return data

    n = len(data)
    table = sorted([(data[i], i) for i in range(n)])

    result = bytearray(n)
    idx = original_idx

    for i in range(n - 1, -1, -1):
        result[i] = table[idx][0]
        idx = table[idx][1]

    return bytes(result)
```

```
    # ==================== MTF (Move-To-Front) ====================

    def _mtf_encode(self, data):
        """Move-To-Front encoding"""
        alphabet = list(range(256))
        result = bytearray()

        for byte in data:
            idx = alphabet.index(byte)
            result.append(idx)
            alphabet.pop(idx)
            alphabet.insert(0, byte)

        return bytes(result)

    def _mtf_decode(self, data):
        """Move-To-Front decoding"""
        alphabet = list(range(256))
        result = bytearray()

        for idx in data:
            byte = alphabet[idx]
            result.append(byte)
            alphabet.pop(idx)
            alphabet.insert(0, byte)

        return bytes(result)

    # ==================== RLE (Run-Length Encoding) ====================

    def _rle_encode(self, data):
        """Run-Length Encoding optimized for BWT output"""
        if len(data) == 0:
            return data

        result = bytearray()
        i = 0

        while i < len(data):
            current = data[i]
            run_length = 1

            while i + run_length < len(data) and data[i + run_length] == current and run_length <
255:
```

```
            run_length += 1

        if run_length >= 4:
            result.extend([0x00, current, run_length])
        elif current == 0x00:
            result.extend([0x00, 0x00, 1])
        else:
            for _ in range(run_length):
                result.append(current)

        i += run_length

    return bytes(result)

def _rle_decode(self, data):
    """Run-Length Decoding"""
    result = bytearray()
    i = 0

    while i < len(data):
        if i + 2 < len(data) and data[i] == 0x00:
            byte = data[i + 1]
            count = data[i + 2]
            result.extend([byte] * count)
            i += 3
        else:
            result.append(data[i])
            i += 1

    return bytes(result)

# ==================== Delta Encoding ====================

def _delta_encode(self, data):
    """Delta encoding for sequential/structured data"""
    if len(data) < 2:
        return data

    result = bytearray([data[0]])
    for i in range(1, len(data)):
        delta = (data[i] - data[i-1]) % 256
        result.append(delta)

    return bytes(result)
```

```
def _delta_decode(self, data):
    """Delta decoding"""
    if len(data) < 2:
        return data

    result = bytearray([data[0]])
    for i in range(1, len(data)):
        value = (result[i-1] + data[i]) % 256
        result.append(value)

    return bytes(result)

# ==================== Data Analysis ====================

def _analyze_data(self, data):
    """Analyze data characteristics for optimal strategy selection"""
    if len(data) == 0:
        return {'entropy': 0, 'repetition': 0, 'sequential': 0}

    # Sample for large files
    sample_size = min(10000, len(data))
    sample = data[:sample_size]

    # Calculate byte frequency entropy
    freq = Counter(sample)
    total = len(sample)
    entropy = -sum((count/total) * np.log2(count/total)
                   for count in freq.values() if count > 0)

    # Calculate repetition score
    runs = 0
    run_lengths = []
    i = 0
    while i < len(sample):
        run_len = 1
        while i + run_len < len(sample) and sample[i + run_len] == sample[i]:
            run_len += 1
        if run_len > 1:
            runs += 1
            run_lengths.append(run_len)
        i += run_len

    repetition = sum(run_lengths) / total if run_lengths else 0
```

```
    # Calculate sequential score
    deltas = [abs(sample[i] - sample[i-1]) for i in range(1, len(sample))]
    sequential = 1.0 - (sum(deltas) / (len(deltas) * 128)) if deltas else 0

    return {
        'entropy': entropy,
        'repetition': repetition,
        'sequential': max(0, sequential),
        'unique_bytes': len(freq)
    }

# ==================== Compression Strategies ====================

def _compress_direct_lzma(self, data):
    """Strategy 4: Direct LZMA (baseline)"""
    return lzma.compress(data, preset=9)

def _decompress_direct_lzma(self, data):
    """Decompress Strategy 4"""
    return lzma.decompress(data)

def _compress_delta_lzma(self, data):
    """Strategy 2: Delta + LZMA"""
    delta_data = self._delta_encode(data)
    compressed = lzma.compress(delta_data, preset=9)
    return compressed

def _decompress_delta_lzma(self, data):
    """Decompress Strategy 2"""
    delta_data = lzma.decompress(data)
    original = self._delta_decode(delta_data)
    return original

def _compress_chunked_hybrid(self, data, chunk_size=32768):
    """Strategy 3: Chunked adaptive compression"""
    chunks = []
    offset = 0

    while offset < len(data):
        chunk = data[offset:offset + chunk_size]

        # Try different methods for each chunk
        methods = [
```

```
            (0x01, lzma.compress(chunk, preset=6)),
            (0x02, zlib.compress(chunk, level=9)),
        ]

        # Select best compression
        best_method, best_compressed = min(methods, key=lambda x: len(x[1]))

        # Store: method (1 byte) + length (4 bytes) + data
        chunk_header = struct.pack('>BI', best_method, len(best_compressed))
        chunks.append(chunk_header + best_compressed)

        offset += chunk_size

    # Combine chunks with count header
    result = struct.pack('>I', len(chunks)) + b''.join(chunks)
    return result

  def _decompress_chunked_hybrid(self, data):
    """Decompress Strategy 3"""
    num_chunks = struct.unpack('>I', data[:4])[0]
    offset = 4
    result = bytearray()

    for _ in range(num_chunks):
        method, chunk_len = struct.unpack('>BI', data[offset:offset+5])
        offset += 5
        chunk_data = data[offset:offset + chunk_len]
        offset += chunk_len

        if method == 0x01:
            result.extend(lzma.decompress(chunk_data))
        elif method == 0x02:
            result.extend(zlib.decompress(chunk_data))

    return bytes(result)

  # ==================== Main Compress/Decompress ====================

  def compress(self, data):
    """Main compression with adaptive strategy selection"""
    if len(data) == 0:
        return self.MAGIC + struct.pack('>BB', self.VERSION, self.STRATEGY_DIRECT_LZMA) +
data
```

```
print("  [NEXACOMPRESS] Analyzing data characteristics...")
analysis = self._analyze_data(data)

print(f"    Entropy: {analysis['entropy']:.2f}")
print(f"    Repetition: {analysis['repetition']:.2%}")
print(f"    Sequential: {analysis['sequential']:.2%}")

# Select optimal strategy based on data characteristics
strategies_to_try = []

# Strategy 4: Direct LZMA (always try as baseline)
strategies_to_try.append((self.STRATEGY_DIRECT_LZMA, "Direct LZMA"))

# Strategy 3: Chunked for large files
if len(data) > 100000:
    strategies_to_try.append((self.STRATEGY_CHUNKED_HYBRID, "Chunked Hybrid"))

# Strategy 2: Delta for sequential data
if analysis['sequential'] > 0.3:
    strategies_to_try.append((self.STRATEGY_DELTA_LZMA, "Delta + LZMA"))

print(f"  [NEXACOMPRESS] Testing {len(strategies_to_try)} strategies...")

best_strategy = None
best_compressed = None
best_size = float('inf')

for strategy_id, strategy_name in strategies_to_try:
    try:
        if strategy_id == self.STRATEGY_DIRECT_LZMA:
            compressed = self._compress_direct_lzma(data)
        elif strategy_id == self.STRATEGY_DELTA_LZMA:
            compressed = self._compress_delta_lzma(data)
        elif strategy_id == self.STRATEGY_CHUNKED_HYBRID:
            compressed = self._compress_chunked_hybrid(data)
        else:
            continue

        comp_size = len(compressed)
        ratio = comp_size / len(data)
        print(f"    {strategy_name}: {comp_size:,} bytes ({ratio:.2%})")

        if comp_size < best_size:
            best_size = comp_size
```

```
            best_compressed = compressed
            best_strategy = strategy_id

        except Exception as e:
            print(f"    {strategy_name}: Failed - {e}")

    if best_compressed is None:
        raise Exception("All compression strategies failed")

    strategy_names = {
        self.STRATEGY_DIRECT_LZMA: "Direct LZMA",
        self.STRATEGY_DELTA_LZMA: "Delta + LZMA",
        self.STRATEGY_CHUNKED_HYBRID: "Chunked Hybrid"
    }

    print(f"  [NEXACOMPRESS] Selected: {strategy_names.get(best_strategy, 'Unknown')}")

    # Create final compressed data: MAGIC + VERSION + STRATEGY + DATA
    header = self.MAGIC + struct.pack('>BB', self.VERSION, best_strategy)
    return header + best_compressed

def decompress(self, data):
    """Main decompression"""
    if len(data) < 6:
        raise ValueError("Invalid NEXACOMPRESS data: too short")

    # Verify magic bytes
    magic = data[:4]
    if magic != self.MAGIC:
        raise ValueError(f"Invalid magic bytes: {magic}")

    # Extract version and strategy
    version, strategy = struct.unpack('>BB', data[4:6])

    if version != self.VERSION:
        raise ValueError(f"Unsupported version: {version}")

    # Decompress based on strategy
    compressed_data = data[6:]

    if strategy == self.STRATEGY_DIRECT_LZMA:
        return self._decompress_direct_lzma(compressed_data)
    elif strategy == self.STRATEGY_DELTA_LZMA:
        return self._decompress_delta_lzma(compressed_data)
```

```
        elif strategy == self.STRATEGY_CHUNKED_HYBRID:
            return self._decompress_chunked_hybrid(compressed_data)
        else:
            raise ValueError(f"Unknown strategy: {strategy}")


# ==================== COMPRESSION ANALYZER ====================

class CompressionAnalyzer:
    """Handles all compression operations and analysis"""

    def __init__(self, input_file):
        self.input_file = input_file
        self.base_name = os.path.splitext(input_file)[0]
        self.file_extension = os.path.splitext(input_file)[1]
        self.results = []
        self.original_data = None
        self.original_size = 0
        self.compressed_files = []
        self.temp_files = []

        self._read_file()

    def _read_file(self):
        """Read the input file"""
        try:
            with open(self.input_file, 'rb') as f:
                self.original_data = f.read()
            self.original_size = len(self.original_data)
            print(f"\n✓ File loaded: {self.input_file}")
            print(f"  Size: {self.original_size:,} bytes ({self.original_size / (1024 * 1024):.3f} MB)")
        except Exception as e:
            print(f"✗ Error reading file: {e}")
            sys.exit(1)

    # ==================== COMPRESSION METHODS ====================

    def compress_deflate(self, level=9):
        """Compress using DEFLATE (zlib)"""
        print("\n" + "-" * 70)
        print("DEFLATE (zlib) Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_deflate{self.file_extension}"
```

```
temp_compressed = f"{self.base_name}_deflate.temp"

try:
    start_time = time.time()
    compressed_data = zlib.compress(self.original_data, level=level)
    comp_time = time.time() - start_time

    with open(temp_compressed, 'wb') as f:
        f.write(compressed_data)

    decompressed = zlib.decompress(compressed_data)
    integrity = (decompressed == self.original_data)

    comp_size = len(compressed_data)
    ratio = comp_size / self.original_size
    space_saved = 1 - ratio

    print(f"✓ Compression successful")
    print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
    print(f"  Compression Ratio: {ratio:.2%}")
    print(f"  Space Saved: {space_saved:.2%}")
    print(f"  Time: {comp_time:.4f} seconds")
    print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

    result = {
        'Algorithm': 'DEFLATE',
        'Original Size (bytes)': self.original_size,
        'Compressed Size (bytes)': comp_size,
        'Original Size (MB)': self.original_size / (1024 * 1024),
        'Compressed Size (MB)': comp_size / (1024 * 1024),
        'Compression Ratio': ratio,
        'Space Saved': space_saved,
        'Compression Time (s)': comp_time,
        'Integrity': 'PASS' if integrity else 'FAIL',
        'Output File': output_file
    }
    self.results.append(result)
    self.compressed_files.append((temp_compressed, output_file))
    self.temp_files.append(temp_compressed)
    return output_file

except Exception as e:
    print(f"✗ Error: {e}")
    return None
```

```
def compress_lzma(self, preset=9):
    """Compress using LZMA"""
    print("\n" + "-" * 70)
    print("LZMA Compression")
    print("-" * 70)

    output_file = f"{self.base_name}_lzma{self.file_extension}"
    temp_compressed = f"{self.base_name}_lzma.temp"

    try:
        start_time = time.time()
        compressed_data = lzma.compress(self.original_data, preset=preset)
        comp_time = time.time() - start_time

        with open(temp_compressed, 'wb') as f:
            f.write(compressed_data)

        decompressed = lzma.decompress(compressed_data)
        integrity = (decompressed == self.original_data)

        comp_size = len(compressed_data)
        ratio = comp_size / self.original_size
        space_saved = 1 - ratio

        print(f"✓ Compression successful")
        print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
        print(f"  Compression Ratio: {ratio:.2%}")
        print(f"  Space Saved: {space_saved:.2%}")
        print(f"  Time: {comp_time:.4f} seconds")
        print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

        result = {
            'Algorithm': 'LZMA',
            'Original Size (bytes)': self.original_size,
            'Compressed Size (bytes)': comp_size,
            'Original Size (MB)': self.original_size / (1024 * 1024),
            'Compressed Size (MB)': comp_size / (1024 * 1024),
            'Compression Ratio': ratio,
            'Space Saved': space_saved,
            'Compression Time (s)': comp_time,
            'Integrity': 'PASS' if integrity else 'FAIL',
            'Output File': output_file
        }
```

```
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None

    def compress_zstandard(self, level=22):
        """Compress using Zstandard"""
        if not ZSTD_AVAILABLE:
            print("\n✗ Zstandard not available. Install with: pip install zstandard")
            return None

        print("\n" + "-" * 70)
        print("Zstandard Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_zstd{self.file_extension}"
        temp_compressed = f"{self.base_name}_zstd.temp"

        try:
            start_time = time.time()
            cctx = zstd.ZstdCompressor(level=level)
            compressed_data = cctx.compress(self.original_data)
            comp_time = time.time() - start_time

            with open(temp_compressed, 'wb') as f:
                f.write(compressed_data)

            dctx = zstd.ZstdDecompressor()
            decompressed = dctx.decompress(compressed_data)
            integrity = (decompressed == self.original_data)

            comp_size = len(compressed_data)
            ratio = comp_size / self.original_size
            space_saved = 1 - ratio

            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
```

```
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

            result = {
                'Algorithm': 'Zstandard',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None

    def compress_brotli(self, quality=11):
        """Compress using Brotli"""
        if not BROTLI_AVAILABLE:
            print("\n✗ Brotli not available. Install with: pip install brotli")
            return None

        print("\n" + "-" * 70)
        print("Brotli Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_brotli{self.file_extension}"
        temp_compressed = f"{self.base_name}_brotli.temp"

        try:
            start_time = time.time()
            compressed_data = brotli.compress(self.original_data, quality=quality)
            comp_time = time.time() - start_time

            with open(temp_compressed, 'wb') as f:
                f.write(compressed_data)
```

```
            decompressed = brotli.decompress(compressed_data)
            integrity = (decompressed == self.original_data)

            comp_size = len(compressed_data)
            ratio = comp_size / self.original_size
            space_saved = 1 - ratio

            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

            result = {
                'Algorithm': 'Brotli',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None

    def compress_huffman(self):
        """Compress using Huffman Coding"""
        if not HUFFMAN_AVAILABLE:
            print("\n✗ Huffman not available. Install with: pip install dahuffman")
            return None

        print("\n" + "-" * 70)
```

```
print("Huffman Coding Compression")
print("-" * 70)

output_file = f"{self.base_name}_huffman{self.file_extension}"
temp_compressed = f"{self.base_name}_huffman.temp"

try:
    start_time = time.time()
    codec = HuffmanCodec.from_data(self.original_data)
    compressed_data = codec.encode(self.original_data)
    comp_time = time.time() - start_time

    with open(temp_compressed, 'wb') as f:
        f.write(compressed_data)

    decompressed = codec.decode(compressed_data)
    integrity = (decompressed == self.original_data)

    comp_size = len(compressed_data)
    ratio = comp_size / self.original_size
    space_saved = 1 - ratio

    print(f"✓ Compression successful")
    print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
    print(f"  Compression Ratio: {ratio:.2%}")
    print(f"  Space Saved: {space_saved:.2%}")
    print(f"  Time: {comp_time:.4f} seconds")
    print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")

    result = {
        'Algorithm': 'Huffman',
        'Original Size (bytes)': self.original_size,
        'Compressed Size (bytes)': comp_size,
        'Original Size (MB)': self.original_size / (1024 * 1024),
        'Compressed Size (MB)': comp_size / (1024 * 1024),
        'Compression Ratio': ratio,
        'Space Saved': space_saved,
        'Compression Time (s)': comp_time,
        'Integrity': 'PASS' if integrity else 'FAIL',
        'Output File': output_file
    }
    self.results.append(result)
    self.compressed_files.append((temp_compressed, output_file))
    self.temp_files.append(temp_compressed)
```

```
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            return None

    def compress_nexacompress(self):
        """Compress using NEXACOMPRESS hybrid algorithm"""
        print("\n" + "-" * 70)
        print("NEXACOMPRESS Hybrid Compression")
        print("-" * 70)

        output_file = f"{self.base_name}_nexacompress{self.file_extension}"
        temp_compressed = f"{self.base_name}_nexacompress.temp"

        try:
            start_time = time.time()

            # Create compressor instance
            compressor = NexaCompress()

            # Compress
            compressed_data = compressor.compress(self.original_data)
            comp_time = time.time() - start_time

            # Write compressed file
            with open(temp_compressed, 'wb') as f:
                f.write(compressed_data)

            # Verify integrity
            decompressed = compressor.decompress(compressed_data)
            integrity = (decompressed == self.original_data)

            # Calculate metrics
            comp_size = len(compressed_data)
            ratio = comp_size / self.original_size
            space_saved = 1 - ratio

            print(f"✓ Compression successful")
            print(f"  Compressed Size: {comp_size:,} bytes ({comp_size / (1024 * 1024):.3f} MB)")
            print(f"  Compression Ratio: {ratio:.2%}")
            print(f"  Space Saved: {space_saved:.2%}")
            print(f"  Time: {comp_time:.4f} seconds")
            print(f"  Integrity: {'✓ PASS' if integrity else '✗ FAIL'}")
```

```
            result = {
                'Algorithm': 'NEXACOMPRESS',
                'Original Size (bytes)': self.original_size,
                'Compressed Size (bytes)': comp_size,
                'Original Size (MB)': self.original_size / (1024 * 1024),
                'Compressed Size (MB)': comp_size / (1024 * 1024),
                'Compression Ratio': ratio,
                'Space Saved': space_saved,
                'Compression Time (s)': comp_time,
                'Integrity': 'PASS' if integrity else 'FAIL',
                'Output File': output_file
            }
            self.results.append(result)
            self.compressed_files.append((temp_compressed, output_file))
            self.temp_files.append(temp_compressed)
            return output_file

        except Exception as e:
            print(f"✗ Error: {e}")
            import traceback
            traceback.print_exc()
            return None

    def run_all_compressions(self):
        """Run all available compression algorithms"""
        print("\n" + "=" * 70)
        print("STARTING COMPRESSION ANALYSIS")
        print("=" * 70)

        # Run NEXACOMPRESS first
        self.compress_nexacompress()

        self.compress_deflate()
        self.compress_lzma()

        if ZSTD_AVAILABLE:
            self.compress_zstandard()

        if BROTLI_AVAILABLE:
            self.compress_brotli()

        if HUFFMAN_AVAILABLE:
            self.compress_huffman()
```

```
    if not self.results:
        print("\n✗ No compression methods available!")
        sys.exit(1)

    def create_archive(self):
        """Create ZIP archive with all compressed files and analysis"""
        archive_name = f"{self.base_name}_archive.zip"

        print("\n" + "=" * 70)
        print("CREATING ARCHIVE")
        print("=" * 70)

        try:
            with zipfile.ZipFile(archive_name, 'w', zipfile.ZIP_DEFLATED) as zipf:
                for temp_file, output_name in self.compressed_files:
                    if os.path.exists(temp_file):
                        zipf.write(temp_file, arcname=output_name)
                        print(f"✓ Added: {output_name}")

                csv_file = f"{self.base_name}_compression_results.csv"
                if os.path.exists(csv_file):
                    zipf.write(csv_file, arcname=os.path.basename(csv_file))
                    print(f"✓ Added: {os.path.basename(csv_file)}")

                viz_file = f"{self.base_name}_compression_analysis.jpg"
                if os.path.exists(viz_file):
                    zipf.write(viz_file, arcname=os.path.basename(viz_file))
                    print(f"✓ Added: {os.path.basename(viz_file)}")

            print(f"\n✓ Archive created: {archive_name}")
            return archive_name

        except Exception as e:
            print(f"✗ Error creating archive: {e}")
            return None

    def create_visualizations(self):
        """Create comparison visualizations"""
        print("\n" + "=" * 70)
        print("GENERATING VISUALIZATIONS")
        print("=" * 70)

        df = pd.DataFrame(self.results)
```

```
    algorithms = df['Algorithm'].values

    sns.set_style("whitegrid")
    fig = plt.figure(figsize=(18, 12))

    # 1. Compression Ratio Comparison
    ax1 = plt.subplot(3, 2, 1)
    ratios = df['Compression Ratio'].values * 100
    colors = plt.cm.tab10(np.linspace(0, 1, len(algorithms)))
    bars = ax1.bar(algorithms, ratios, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax1.set_ylabel('Compression Ratio (%)', fontsize=11, fontweight='bold')
    ax1.set_title('Compression Ratio Comparison\n(Lower is Better)', fontsize=12,
fontweight='bold')
    ax1.axhline(100, color='red', linestyle='--', linewidth=2, label='No Compression')
    ax1.grid(axis='y', alpha=0.3)
    ax1.legend()
    plt.setp(ax1.xaxis.get_majorticklabels(), rotation=45, ha='right')

    for bar in bars:
      height = bar.get_height()
      ax1.text(bar.get_x() + bar.get_width() / 2., height,
          f'{height:.1f}%', ha='center', va='bottom', fontsize=8, fontweight='bold')

    # 2. Space Saved
    ax2 = plt.subplot(3, 2, 2)
    space_saved = df['Space Saved'].values * 100
    bars = ax2.bar(algorithms, space_saved, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax2.set_ylabel('Space Saved (%)', fontsize=11, fontweight='bold')
    ax2.set_title('Space Saved Comparison\n(Higher is Better)', fontsize=12,
fontweight='bold')
    ax2.grid(axis='y', alpha=0.3)
    plt.setp(ax2.xaxis.get_majorticklabels(), rotation=45, ha='right')

    for bar in bars:
      height = bar.get_height()
      ax2.text(bar.get_x() + bar.get_width() / 2., height,
          f'{height:.1f}%', ha='center', va='bottom', fontsize=8, fontweight='bold')

    # 3. Compression Time
    ax3 = plt.subplot(3, 2, 3)
    times = df['Compression Time (s)'].values
```

```
    bars = ax3.bar(algorithms, times, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax3.set_ylabel('Time (seconds)', fontsize=11, fontweight='bold')
    ax3.set_title('Compression Speed\n(Lower is Better)', fontsize=12, fontweight='bold')
    ax3.grid(axis='y', alpha=0.3)
    plt.setp(ax3.xaxis.get_majorticklabels(), rotation=45, ha='right')

    for bar in bars:
      height = bar.get_height()
      ax3.text(bar.get_x() + bar.get_width() / 2., height,
           f'{height:.2f}s', ha='center', va='bottom', fontsize=8, fontweight='bold')

    # 4. File Size Comparison
    ax4 = plt.subplot(3, 2, 4)
    x = np.arange(len(algorithms))
    width = 0.35
    original_mb = [self.original_size / (1024 * 1024)] * len(algorithms)
    compressed_mb = df['Compressed Size (MB)'].values

    ax4.bar(x - width / 2, original_mb, width, label='Original', color='#e74c3c', alpha=0.8)
    ax4.bar(x + width / 2, compressed_mb, width, label='Compressed', color='#2ecc71',
alpha=0.8)

    ax4.set_ylabel('File Size (MB)', fontsize=11, fontweight='bold')
    ax4.set_title('File Size Comparison', fontsize=12, fontweight='bold')
    ax4.set_xticks(x)
    ax4.set_xticklabels(algorithms, rotation=45, ha='right')
    ax4.legend()
    ax4.grid(axis='y', alpha=0.3)

    # 5. Efficiency Score
    ax5 = plt.subplot(3, 2, 5)
    efficiency = (df['Space Saved'] / df['Compression Time (s)']).values
    bars = ax5.bar(algorithms, efficiency, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
    ax5.set_ylabel('Efficiency Score', fontsize=11, fontweight='bold')
    ax5.set_title('Compression Efficiency\n(Space Saved per Second - Higher is Better)',
          fontsize=12, fontweight='bold')
    ax5.grid(axis='y', alpha=0.3)
    plt.setp(ax5.xaxis.get_majorticklabels(), rotation=45, ha='right')

    for bar in bars:
      height = bar.get_height()
      ax5.text(bar.get_x() + bar.get_width() / 2., height,
```

```
                f'{height:.2f}', ha='center', va='bottom', fontsize=8, fontweight='bold')

        # 6. Algorithm Ranking
        ax6 = plt.subplot(3, 2, 6)
        # Normalize metrics (0-1 scale)
        norm_ratio = 1 - (df['Compression Ratio'].values / df['Compression Ratio'].max())
        norm_time = 1 - (df['Compression Time (s)'].values / df['Compression Time (s)'].max())

        # Overall score (60% compression, 40% speed)
        overall_score = (norm_ratio * 0.6 + norm_time * 0.4) * 100

        bars = ax6.barh(algorithms, overall_score, color=colors, alpha=0.8, edgecolor='black',
linewidth=1.5)
        ax6.set_xlabel('Overall Score', fontsize=11, fontweight='bold')
        ax6.set_title('Overall Algorithm Ranking\n(60% Compression + 40% Speed)',
fontsize=12, fontweight='bold')
        ax6.grid(axis='x', alpha=0.3)

        for i, bar in enumerate(bars):
            width = bar.get_width()
            ax6.text(width, bar.get_y() + bar.get_height()/2.,
                f'{width:.1f}', ha='left', va='center', fontsize=8, fontweight='bold')

        plt.tight_layout()

        viz_filename = f"{self.base_name}_compression_analysis.jpg"
        plt.savefig(viz_filename, dpi=300, bbox_inches='tight')
        print(f"✓ Visualization saved: {viz_filename}")
        plt.close()

        return viz_filename

    def save_results_csv(self):
        """Save results to CSV file"""
        df = pd.DataFrame(self.results)
        csv_filename = f"{self.base_name}_compression_results.csv"
        df.to_csv(csv_filename, index=False)
        print(f"✓ Results saved to CSV: {csv_filename}")
        return csv_filename

    def print_summary(self):
        """Print summary report"""
        df = pd.DataFrame(self.results)
```

```
    print("\n" + "=" * 70)
    print("COMPRESSION ANALYSIS SUMMARY")
    print("=" * 70)
    print(df[['Algorithm', 'Compressed Size (MB)', 'Compression Ratio',
          'Space Saved', 'Compression Time (s)', 'Integrity']].to_string(index=False))

    print("\n" + "=" * 70)
    print("DETAILED RESULTS")
    print("=" * 70)

    best_ratio = df.loc[df['Compression Ratio'].idxmin()]
    fastest = df.loc[df['Compression Time (s)'].idxmin()]
    most_efficient = df.loc[(df['Space Saved'] / df['Compression Time (s)']).idxmax()]

    print(f"\n🏆 BEST COMPRESSION RATIO:")
    print(f"   {best_ratio['Algorithm']}: {best_ratio['Compression Ratio']:.2%}")

    print(f"\n⚡ FASTEST COMPRESSION:")
    print(f"   {fastest['Algorithm']}: {fastest['Compression Time (s)']:.4f} seconds")

    print(f"\n🎯 MOST EFFICIENT:")
    eff = most_efficient['Space Saved'] / most_efficient['Compression Time (s)']
    print(f"   {most_efficient['Algorithm']}: {eff:.2f} (space saved per second)")

    # NEXACOMPRESS specific analysis
    if 'NEXACOMPRESS' in df['Algorithm'].values:
      nexa_row = df[df['Algorithm'] == 'NEXACOMPRESS'].iloc[0]
      print(f"\n🔬 NEXACOMPRESS PERFORMANCE:")
      print(f"   Compression Ratio: {nexa_row['Compression Ratio']:.2%}")
      print(f"   Time: {nexa_row['Compression Time (s)']:.4f} seconds")
      print(f"   Space Saved: {nexa_row['Space Saved']:.2%}")

      # Compare with LZMA (closest competitor)
      if 'LZMA' in df['Algorithm'].values:
        lzma_row = df[df['Algorithm'] == 'LZMA'].iloc[0]
        ratio_diff = ((nexa_row['Compression Ratio'] - lzma_row['Compression Ratio']) /
                lzma_row['Compression Ratio'] * 100)
        time_diff = ((nexa_row['Compression Time (s)'] - lzma_row['Compression Time (s)'])
/
                lzma_row['Compression Time (s)'] * 100)

        print(f"\n   vs LZMA:")
        print(f"    Compression: {ratio_diff:+.2f}% ({'better' if ratio_diff < 0 else 'worse'})")
```

```
            print(f"    Speed: {time_diff:+.2f}% ({'faster' if time_diff < 0 else 'slower'})")

        print("\n" + "=" * 70)

    def cleanup_temp_files(self):
        """Clean up temporary files"""
        for temp_file in self.temp_files:
            try:
                if os.path.exists(temp_file):
                    os.remove(temp_file)
            except:
                pass


def main():
    parser = argparse.ArgumentParser(
        description='Analyze file compression using 6 different algorithms (including
NEXACOMPRESS)',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  python3 milestone32.py myfile.pdf
  python3 milestone32.py document.docx
  python3 milestone32.py data.csv --no-viz
        """
    )
    parser.add_argument('input_file', help='File to compress and analyze')
    parser.add_argument('--no-viz', action='store_true', help='Skip visualization generation')

    args = parser.parse_args()

    if not os.path.exists(args.input_file):
        print(f"✗ Error: File '{args.input_file}' not found!")
        sys.exit(1)

    print("\n" + "=" * 70)
    print("UNIFIED FILE COMPRESSION ANALYSIS TOOL")
    print("Algorithms: NEXACOMPRESS | DEFLATE | LZMA | Zstandard | Brotli | Huffman")
    print("=" * 70)

    analyzer = CompressionAnalyzer(args.input_file)
    analyzer.run_all_compressions()
    csv_file = analyzer.save_results_csv()
```

```
    if not args.no_viz:
        try:
            viz_file = analyzer.create_visualizations()
        except Exception as e:
            print(f"\n⚠ Could not generate visualizations: {e}")
            viz_file = None
    else:
        print("\n⚠ Visualization skipped (--no-viz flag)")
        viz_file = None

    archive_file = analyzer.create_archive()
    analyzer.print_summary()

    print("\n" + "=" * 70)
    print("✓ COMPRESSION ANALYSIS COMPLETE!")
    print("=" * 70)
    print(f"\nGenerated Files:")
    print(f"  • {archive_file} (Contains all compressed files + analysis)")
    print(f"  • {csv_file}")
    if viz_file:
        print(f"  • {viz_file}")
    print("\n" + "=" * 70 + "\n")

    analyzer.cleanup_temp_files()


if __name__ == '__main__':
    main()
```

Photos