

Article

Pixelated: A Lossless Data Transformation Framework Built on PNG Compression

Zina Abohaia *  and Patrick Mukala * 

School of Computer Science, University of Wollongong in Dubai, Dubai P.O. Box 20183, United Arab Emirates

* Correspondence: zinaabohaia@uowdubai.ac.ae (Z.A.); patrickmukala@uowdubai.ac.ae (P.M.)

Abstract

The rapid growth of digital data has created an urgent need for innovative storage solutions. Transactional data, with its diverse formats and row-based structure, is a significant contributor to this challenge. Existing storage methods struggle to compress heterogeneous structured data efficiently due to limited redundancy exploitation across mixed data types. This study introduces Pixelated, a novel lossless data transformation and storage framework that converts structured transactional data into pixel representations stored in Portable Network Graphics (PNG) format. Pixelated introduces a new data representation strategy that enables the existing DEFLATE compression mechanism within PNG to exploit patterns and redundancy in heterogeneous transactional datasets more effectively. Designed for datasets containing numerical, categorical, and datetime values, Pixelated achieves average compression rates exceeding 90%. The framework is evaluated across benchmark and real-world datasets, demonstrating competitive performance against ZIP, Apache Parquet, and Python Pickle. Detailed methodology, redundancy characteristics, limitations, and performance results are presented.

Keywords: data compression; DEFLATE; data storage; lossless compression; algorithm

1. Introduction

Data compression provides a practical approach to addressing the challenges of rapidly growing digital datasets by reducing storage requirements and improving transmission efficiency. Compression techniques are generally categorized as lossless or lossy. For critical datasets such as transactional, scientific, and enterprise data, lossless compression is essential because it guarantees exact reconstruction of the original information after decompression.

The field of data compression began with Shannon's foundational work in 1948, which introduced the concepts of entropy and Shannon Encoding. Shannon Entropy defines the minimum amount of information required to represent data and is mathematically expressed as:

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

where $P(x_i)$ represents the probability of each state x_i , and $H(X)$ denotes the source entropy [1]. Shannon also introduced the Source Coding Theorem, demonstrating that lossless compression is achievable when the compression rate is lower than the source entropy [1].



Academic Editor: Bruno Carpentieri

Received: 13 May 2026

Revised: 5 June 2026

Accepted: 5 June 2026

Published: 26 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Building upon Shannon's work, Huffman coding introduced variable-length binary encoding based on symbol frequency [2], while LZ77 utilized dictionary-based sliding windows to identify repeated patterns efficiently [3]. The DEFLATE algorithm later combined LZ77 and Huffman coding and became widely used in formats such as ZIP and PNG [4]. PNG (Portable Network Graphics) is a widely used lossless image format that relies on DEFLATE to remove redundancy in pixel streams [4].

Additional advances, including the Burrows–Wheeler Transform (BWT) [5] and arithmetic coding [6], further improved compression efficiency by enhancing symbol locality and probabilistic encoding precision. More recently, machine learning-based approaches have emerged as a new direction in compression research. Goyal et al. [7] demonstrated that recurrent neural networks can improve lossless compression by learning symbol distributions, while adaptive lane-based compression strategies further highlighted the benefits of dynamic encoding selection [8].

Recent research shows that compression performance is highly dependent on data structure and domain characteristics. Ref. [9] proposed lossless and near-lossless hyperspectral image compression using XOR-based decorrelation, Rice coding, and seed generation, achieving up to 29.89% improvement over k^2 -raster methods. However, their work also highlights limitations in generalization and the sensitivity of compression performance to dataset-specific sparsity patterns.

Similarly, ref. [10] investigated the impact of feature scaling on lossless compression and found that preprocessing transformations such as normalization and min-max scaling often reduce compressibility by altering redundancy structures, particularly in tabular datasets. This demonstrates that preprocessing itself can significantly influence compression efficiency.

In the domain of time-series compression, ref. [11] proposed ACTF, a floating-point compression algorithm combining preprocessing, feature refinement, and coding expansion. Their method achieves up to 42.96% improvement in compression/decompression efficiency, showing that hybrid optimization strategies can significantly enhance performance in IoT and streaming scenarios.

Deep learning-based compression has also shown promising results. Ref. [12] introduced a Bi-LSTM and Transformer-based entropy model for power data compression, achieving an average compression ratio of 4.06 while outperforming classical entropy coding methods. However, their results also reveal trade-offs between model complexity and inference efficiency.

In IoT-based environmental monitoring systems, ref. [13] proposed a genetic algorithm-based compression scheme (CSP) that minimizes energy consumption while maintaining lossless reconstruction. Their results demonstrate that compression can directly contribute to reducing power usage and extending sensor lifetime in constrained devices.

For multimedia data, ref. [14] introduced a 4-ary/MQ Huffman-based scheme that improves the balance between compression ratio and compression time for 16-bit WAV audio data. Their method achieves competitive compression performance while significantly improving processing speed compared to traditional Huffman variants, although performance varies across datasets and audio types.

Despite these advancements, existing compression methods remain suboptimal for modern heterogeneous structured datasets, particularly transactional datasets containing mixed data types such as numerical, categorical, datetime, and string values. These datasets exhibit redundancy patterns that are not efficiently captured by general-purpose compression algorithms designed primarily for homogeneous byte streams.

These developments show that effective compression often depends on exposing redundancy before encoding. Pixelated follows this principle by restructuring raw structured

data into pixel-based representations that enhance redundancy visibility for downstream compression mechanisms.

This paper introduces Pixelated, a lossless data transformation and encoding framework (not a standalone compression algorithm), which encodes structured data into pixel representations stored in PNG format and leverages its built-in DEFLATE compression mechanism indirectly. By leveraging the compact structure and optimization characteristics of image-based storage, Pixelated improves the compressibility of structured datasets without modifying the underlying PNG or DEFLATE algorithms.

Unlike traditional compression approaches that directly operate on byte streams or statistical symbol distributions, Pixelated transforms structured tabular data into image-like representations to increase spatial redundancy prior to compression.

The core motivation is that structured datasets often contain cross-column and cross-type redundancies that are not effectively exploited by conventional compression methods. By converting these structures into pixel grids, Pixelated exposes latent redundancy patterns that can be more effectively compressed using PNG's DEFLATE pipeline.

Experimental evaluations demonstrate that Pixelated achieves average compression rates exceeding 90% across diverse datasets [15–24], including transactional, text-based, and benchmark corpora. However, this performance is dependent on dataset structure and redundancy characteristics.

The remainder of this paper is organized as follows. Section 2 reviews foundational and contemporary developments in data compression, including entropy-based encoding, dictionary-based algorithms, transform-based methods, and machine learning-driven techniques. Section 3 presents the design and methodology of Pixelated. Section 4 evaluates the proposed algorithm through experimental analysis and benchmark comparisons, while Section 5 discusses limitations, applications, and future research directions.

2. Materials and Methods

2.1. Datasets

This study utilizes multiple structured datasets containing numerical, categorical, temporal, and textual attributes. Each dataset is preprocessed and standardized prior to encoding. Missing values are handled through imputation, and inconsistent formatting is removed during cleaning.

Enwik8 is a 100 MB dataset comprising the first 10^8 bytes of the English Wikipedia Dump from 3 March 2006 [24]. It was employed as a strong benchmark to evaluate Pixelated due to its predominantly uniform structure. The dataset demonstrates distinct match separation patterns: high matches of eight characters typically separated by at least 10^5 bytes up to the file's full length; the next highest matches are of four characters with separations of 10^3 to 10^4 bytes; then matches of two characters, usually separated by 10 to 300 bytes; the next corresponds to single-character matches, separated by either 1 byte or between 3 and 10 bytes; and the last reveals an absence of matches separated by 2 bytes, such as the pattern "aba."

The Calgary Dataset, another string benchmark, was developed for testing data compression algorithms. The Standard Calgary Dataset includes 14 files initially tested in 1987 by [15], while the Big Calgary Dataset extends it with four additional files. According to [16], the original 14 files consist of various types such as ASCII text, executable programs, source code, and image data. For instance, BOOK1 contains unformatted ASCII text of *Far from the Madding Crowd*, while GEO includes seismic data stored as 32-bit IBM floating-point numbers. Binary files such as PROG1, PROG2, and PROG3 exhibit low redundancy compared to repetitive files like BOOK1, GEO, OBJ2, and PIC. The DNA Dataset, analyzed by [17], includes DNA and protein sequences provided by the Center for Bioinformatics,

University of Pennsylvania, as well as medical imaging data from Guys' and St. Thomas' Hospital in London.

Transactional datasets were also examined. The Demo dataset is a CSV file containing two columns of integers ranging from 1 to 1,048,576, the maximum row capacity in Excel. The real dataset, sourced from a food delivery company named Saalas, spans two years and comprises 11,274 rows across nine columns, including numerical, categorical, and datetime data. This dataset highlights varying redundancy levels: columns with unique values (e.g., datetime) appear as dark blocks, while columns with repeated values (e.g., card type and gateway) form vibrant green clusters, reflecting frequent repetition. All datasets undergo preprocessing using a unified cleaning function that removes non-alphanumeric characters and standardizes textual entries.

2.2. Overview of Pixelated Framework

Pixelated is the proposed encoding framework introduced in this study. It transforms structured tabular data into a deterministic pixel-based representation, which is subsequently compressed using standard PNG (DEFLATE-based) compression without any modification to the underlying compression algorithm. The encoding pipeline is shown in Figure 1, and the decoding pipeline in Figure 2.

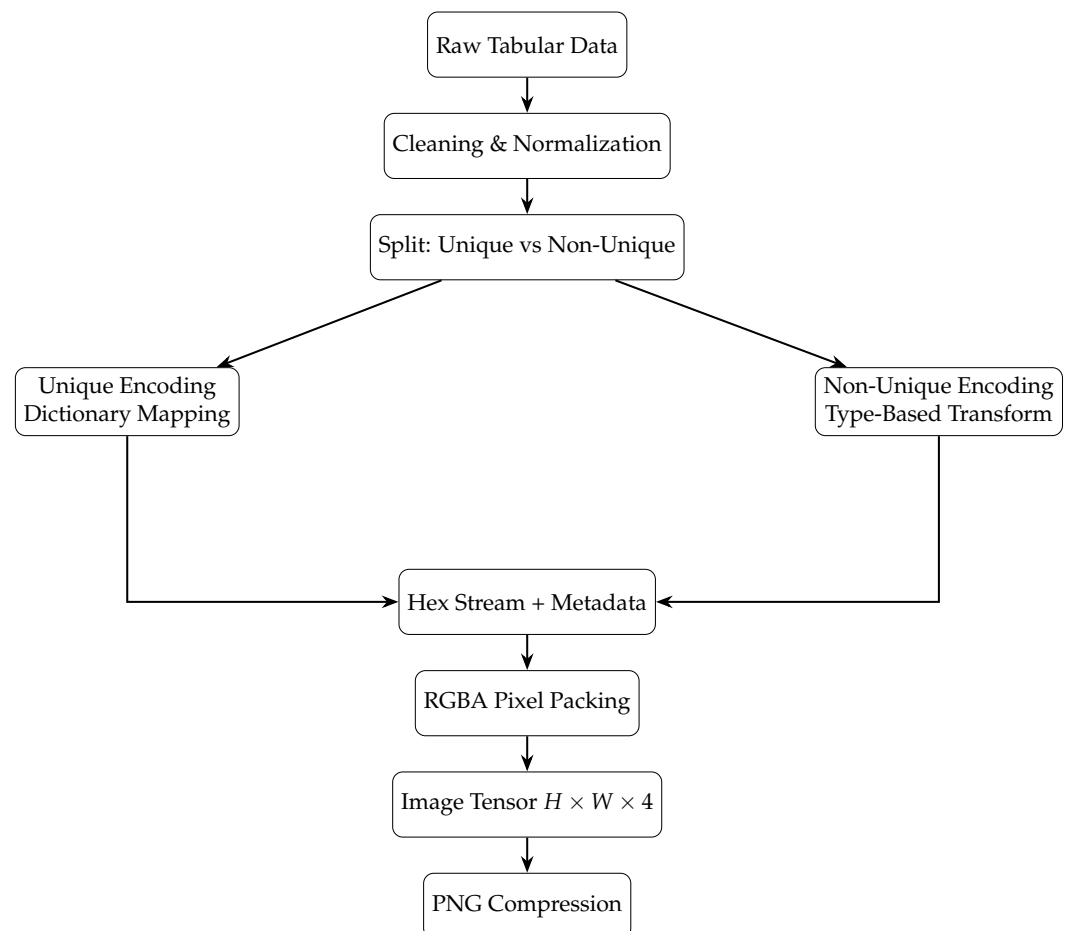


Figure 1. Pixelated encoding pipeline.

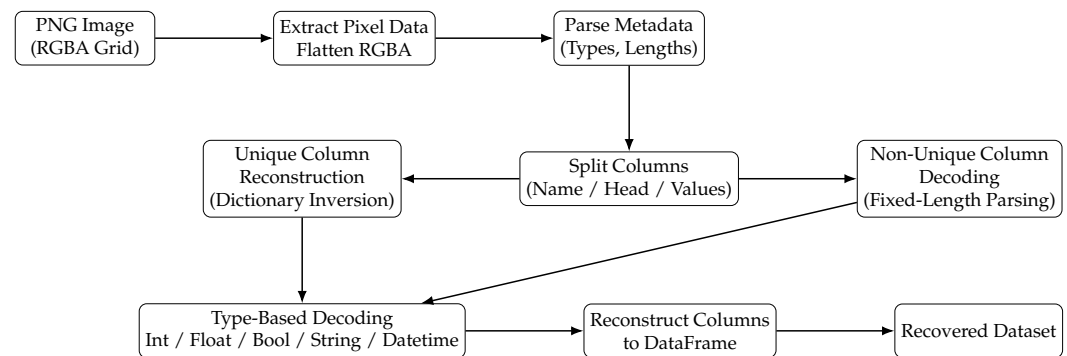


Figure 2. Pixelated decoding pipeline: reconstruction of structured tabular data from PNG-encoded RGBA representation.

Unlike traditional compression pipelines, Pixelated improves compressibility through a preprocessing transformation that reorganizes heterogeneous data types into structured hexadecimal pixel streams, based on two key observations:

- Structured redundancy in tabular data can be increased before compression.
- PNG compression performs optimally on spatially coherent repeated byte patterns.

Therefore, Pixelated reorganizes data into spatially redundant pixel blocks rather than modifying the compression algorithm itself.

Formally, given a dataset $D \in \mathbb{R}^{n \times m}$, Pixelated defines a mapping:

$$\mathcal{F} : D \rightarrow I_{RGBA}$$

where each row-column entry is encoded into a 32-bit RGBA pixel representation.

2.3. Data Type Encoding Schema

All data types are converted into hexadecimal strings before being packed into RGBA pixels, with the strategies shown in Table 1.

Table 1. Data type encoding strategy in Pixelated.

Data Type	Encoding Strategy
Integer	Hex conversion with fixed-length padding (4-/8-bit mode)
Float	Scaled integer encoding ($x \rightarrow \lfloor 100x \rfloor$)
Boolean	Binary mapping (True = 1, False = 0)
String	Dictionary-based ASCII + bigram/trigram encoding
Datetime	Delta-based time encoding (minutes difference)
Missing Values	Zero-padding token ("00...00")
Negative Values	Two's complement or signed offset encoding

2.4. Unique vs Non-Unique Column Strategy

A key contribution of Pixelated is the separation of columns into:

- Unique columns: high-cardinality categorical attributes.
- Non-unique columns: low-cardinality or continuous attributes.

2.4.1. Unique Columns

For columns with limited cardinality, Pixelated constructs a lookup dictionary:

$$\text{value} \rightarrow \text{hex index}$$

These are packed into compact RGBA blocks with adaptive bit-width encoding depending on cardinality.

2.4.2. Non-Unique Columns

Non-unique columns are encoded directly using type-aware transformations without dictionary compression.

2.5. Pixel Packing Strategy

Each encoded value is represented as a 32-bit RGBA pixel:

$$P = (R, G, B, A)$$

where each channel stores 8 bits of encoded data.

Values are concatenated sequentially into fixed-size pixel blocks, ensuring alignment for image formation.

2.6. Formal Pixelated Algorithm

The encoding algorithm transforms structured transactional data into a compact pixel-based representation while preserving all semantic fields for lossless storage and compression, as seen in Algorithm 1.

Algorithm 1 Pixelated Encoding Algorithm

Require: Dataset D

Ensure: PNG image I

```

1: Clean dataset and normalize datatypes
2: Split columns into unique set  $U$  and non-unique set  $N$ 
3: for each column  $c \in U$  do
4:   Build dictionary mapping unique values to hex indices
5:   Encode column name and dictionary head
6:   Pack encoded values into fixed RGBA blocks
7: end for
8: for each column  $c \in N$  do
9:   if integer type then
10:    Convert integers to hex with fixed padding
11:   else if float type then
12:    Scale ( $\times 100$ ) and convert to hex
13:   else if datetime type then
14:    Encode as base timestamp + delta minutes
15:   else if string type then
16:    Apply ASCII + n-gram dictionary encoding
17:   else if Boolean type then
18:    Map True  $\rightarrow$  01, False  $\rightarrow$  00
19:   end if
20:   Pack values into RGBA blocks
21: end for
22: Concatenate metadata + encoded columns
23: Reshape into tensor  $H \times W \times 4$ 
24: Export PNG using lossless compression

```

The decoding algorithm reconstructs the original structured dataset from the pixel-based representation by inverting the encoding process with exact field-wise recovery and no information loss, as described in Algorithm 2.

Algorithm 2 Pixelated Decoding Algorithm**Require:** PNG image I **Ensure:** Reconstructed dataset D

```

1: Read image and flatten RGBA tensor
2: Extract metadata blocks (column names, types, lengths)
3: for each column block do
4:   Parse column type identifier
5:   Extract header and value segments
6:   if column is unique then
7:     Rebuild dictionary from header block
8:     Map indices back to original values
9:   else
10:    Decode fixed-length RGBA blocks into hex
11:   end if
12:   if type is integer then
13:     Convert hex to integer values
14:   else if type is float then
15:     Convert hex and divide by 100
16:   else if type is datetime then
17:     Reconstruct base time + delta sequence
18:   else if type is string then
19:     Decode ASCII + n-gram mapping
20:   else if type is Boolean then
21:     Map 01 → True, 00 → False
22:   end if
23: end for
24: Assemble decoded columns into dataframe  $D$ 

```

*2.7. Encoding Details (Implementation-Level)**2.7.1. Integer Encoding*

Integers are converted to hexadecimal representation and padded to fixed-length 8-bit blocks:

$$x \rightarrow \text{hex}(x)$$

Negative values are handled using **signed offset encoding**, where the sign is implicitly preserved during hexadecimal conversion prior to fixed-length padding.

2.7.2. Float Encoding

$$x \rightarrow \text{hex}(\lfloor 100x \rfloor)$$

This scaling preserves up to two decimal places of precision while enabling integer-based encoding.

2.7.3. Boolean Encoding

Boolean variables are encoded as a minimal binary representation:

$$\text{True} \rightarrow 01, \quad \text{False} \rightarrow 00$$

These values are stored as fixed-length hexadecimal-compatible blocks to ensure alignment with RGBA packing.

2.7.4. Datetime Encoding

Datetime values are parsed and converted into a structured temporal representation consisting of:

- A base timestamp (derived from the first row).
- A sequence of delta values representing time differences in minutes.

This delta encoding reduces redundancy in sequential temporal data while preserving reconstructability.

2.7.5. String Encoding

Strings are encoded using a hybrid compression strategy consisting of:

- A custom ASCII lookup table for character-to-hex mapping.
- Bigram and trigram substitution (e.g., ‘th’, ‘ing’) to reduce token length.

This allows variable-length strings to be transformed into fixed-length hexadecimal blocks suitable for image embedding.

2.7.6. Missing Values

Missing values are encoded as zero-padded hexadecimal blocks to maintain positional integrity:

$$\text{NULL} \rightarrow 00000000$$

2.8. Compression Rate Evaluation

The experiments were implemented in Python on a Windows 11 system (64-bit) with 16 GB RAM. File sizes were determined from their saved properties on the computer. Compression Rate is defined as:

$$CR = \left(1 - \frac{S_c}{S_o}\right) \times 100$$

where S_o is original size and S_c is compressed size.

2.9. Summary of Contributions in This Pipeline

- Unified pixel-based representation of heterogeneous data;
- Dual handling of unique and non-unique columns;
- Multi-type encoding system (int, float, string, datetime, Boolean);
- Delta-based temporal compression;
- Dictionary-based categorical compression;
- Lossless reconstruction via reversible RGBA mapping.

3. Results

Pixelated was evaluated on the datasets described earlier, and its performance was compared against the widely used ZIP compression method. As illustrated in Figures 3 and 4 and detailed in the methodology, Pixelated was tested on both transactional datasets and text benchmarks. The results demonstrate that Pixelated consistently achieved a compression rate exceeding 90%, while ZIP’s performance varied between 60% and 85%. This highlights the greater consistency and efficiency of Pixelated compared to ZIP.

Across the five primary tests, seen in Figure 5, Pixelated achieved an average compression rate of 95.13%, significantly surpassing ZIP’s 71%. The fluctuations in ZIP’s compression rate can be attributed to the presence of unique values in the data, whereas Pixelated effectively handled these variations, delivering both higher and more consistent compression.

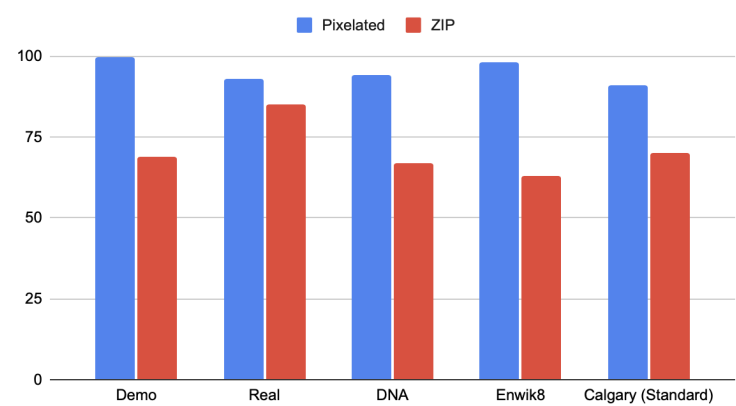


Figure 3. This bar chart compares the percentage compression efficiency of different formats: Pixelated, ZIP, Parquet, and Pickle.

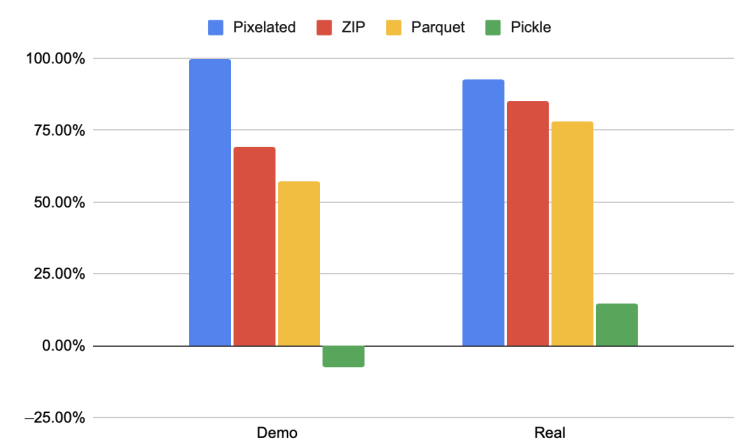


Figure 4. The bar chart illustrates the performance of Pixelated and ZIP formats across various datasets, including Demo, Real, DNA, Enwik8, and Calgary (Standard).

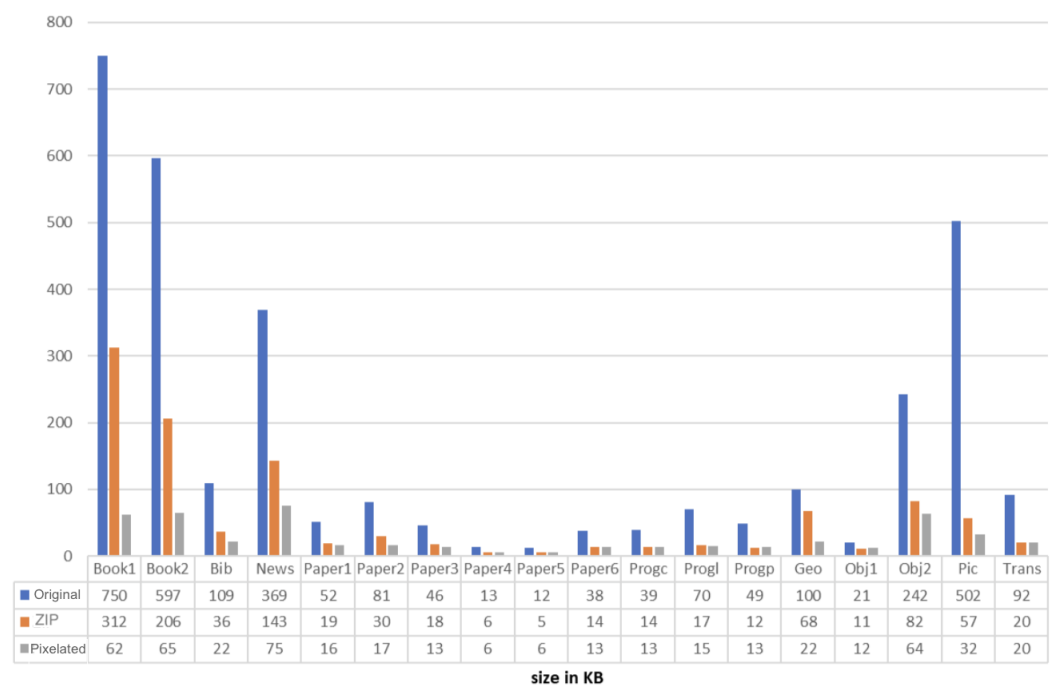


Figure 5. This chart compares the original file size with compressed sizes using ZIP and Pixelated formats across categories such as Book1, Book2, News, and Object files. Pixelated achieves the most significant size reduction compared to both the original and ZIP-compressed versions.

3.1. Dataset Performance

Table 2 reports both absolute and percentage compression results for transactional datasets, along with compression time across all evaluated methods. The results demonstrate that Pixelated consistently achieves the highest compression ratios across both the demo and real-world datasets, while also highlighting the trade-off in computational cost due to its transformation-based pipeline.

On the demo dataset, Pixelated achieves a compression rate of 99.82%, substantially outperforming all baseline methods, including ZIP (69.32%), Gzip (69.28%), Bzip2 (85.24%), LZMA (95.07%), Brotli (97.04%), and Zstd (93.88%). A similar trend is observed in runtime efficiency comparisons, where faster but less effective methods such as Zstd contrast with higher-compression but slower approaches like Brotli and LZMA.

For the real transactional dataset, Pixelated again demonstrates superior compression performance at 92.58%, exceeding ZIP (85.40%), Gzip (85.99%), Bzip2 (87.94%), LZMA (90.16%), Brotli (89.78%), and Zstd (84.44%). Although Pixelated incurs higher processing time compared to traditional compression tools, the results indicate a consistent trade-off between compression effectiveness and computational overhead.

Overall, the table provides a comprehensive comparison of compression ratio and runtime, clearly showing that Pixelated maintains the best compression performance across heterogeneous transactional datasets, followed by LZMA, Brotli, Bzip2, ZIP/Gzip, and Zstd, depending on the dataset characteristics.

Table 2. Compression performance comparison across baseline methods and Pixelated (absolute size, compression ratio, and runtime). The best results are highlighted in bold.

Dataset	Method	Original (Bytes)	Compressed (Bytes)	Compression (%)	Time (s)
Demo	ZIP (zlib)	15,603,589	4,787,011	69.32	0.61
	Gzip		4,793,022	69.28	0.86
	Bzip2		2,303,648	85.24	1.18
	LZMA		769,100	95.07	17.15
	Brotli		461,723	97.04	46.61
	Zstd		954,252	93.88	0.11
	Pixelated		27,394	99.82	347.80
Real	ZIP (zlib)	1,054,903	154,014	85.40	0.03
	Gzip		147,732	85.99	0.08
	Bzip2		127,176	87.94	0.14
	LZMA		103,820	90.16	0.46
	Brotli		107,793	89.78	3.03
	Zstd		164,162	84.44	0.01
	Pixelated		78,277	92.58	16.05

3.2. Ablation Study (Pixelated Variants)

Table 3 reports the ablation study evaluating Pixelated under different compression backends.

The ablation results indicate that the major compression gain is achieved through pixel transformation, while secondary compression methods provide only marginal improvements.

Table 3. Ablation study results for Pixelated variants.

Method	Compressed Size (Bytes)	Compression (%)	Time (s)
Demo Raw+ZIP	4,505,402	69.05	0.63
Demo Pixelated PNG	27,394	99.81	0.00
Demo Pixel+ZIP	1082	99.99	0.0008
Demo Pixel+Zstd	1057	99.99	0.0003
Demo Parquet	8,953,880	38.48	0.15
Real Raw+ZIP	152,248	85.09	0.04
Real Pixelated PNG	78,277	92.33	0.00
Real Pixel+ZIP	78,152	92.35	0.005
Real Pixel+Zstd	78,289	92.33	0.0004
Real Parquet	230,290	77.45	0.03

3.3. Calgary Benchmark Results

Pixelated was also tested on all 18 individual files in the Calgary dataset (Figure 5). The results show that Pixelated achieved superior compression for the most repetitive files, surpassing ZIP.

For binary files with low redundancy, Pixelated and ZIP showed similar performance. Figure 6 further proves the relationship between compression percentage and bits per byte, while Figure 7 shows the comparison between ZIP and Pixelated compression percentages on the Calgary dataset.

ZIP compressed most files to between 50% and 70%, while Pixelated achieved rates between 70% and 80%.

Notably, Pixelated outperformed ZIP by a significant margin for the three largest files in the Calgary dataset: *Book1*, *Book2*, and *PIC*. While ZIP outperformed Pixelated in only three of the smallest files (*Paper5*, *Progp*, and *Obj1*), the difference in compressed size was minimal.

Overall, Pixelated demonstrated substantial advantages in compression efficiency and consistency, especially for larger and more complex datasets.

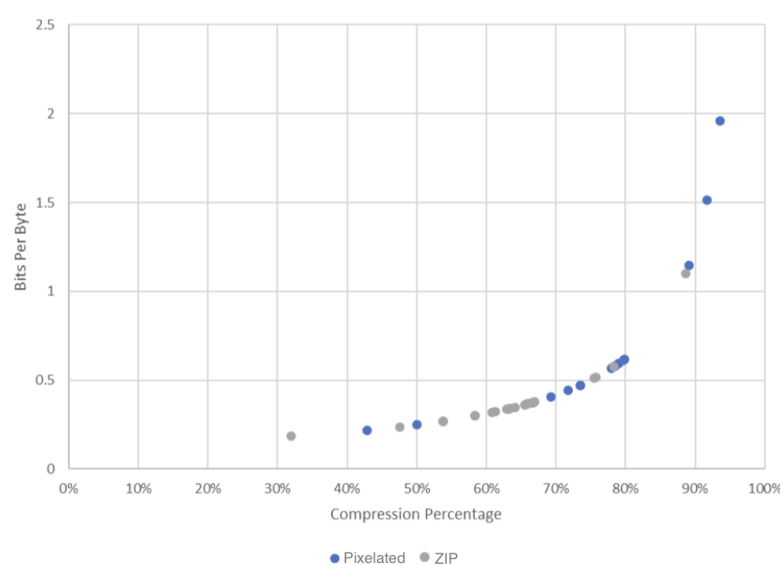


Figure 6. This figure shows the relation between bits per byte and compressed percentage for both Pixelated and zip on the 18 files in the Calgary dataset.

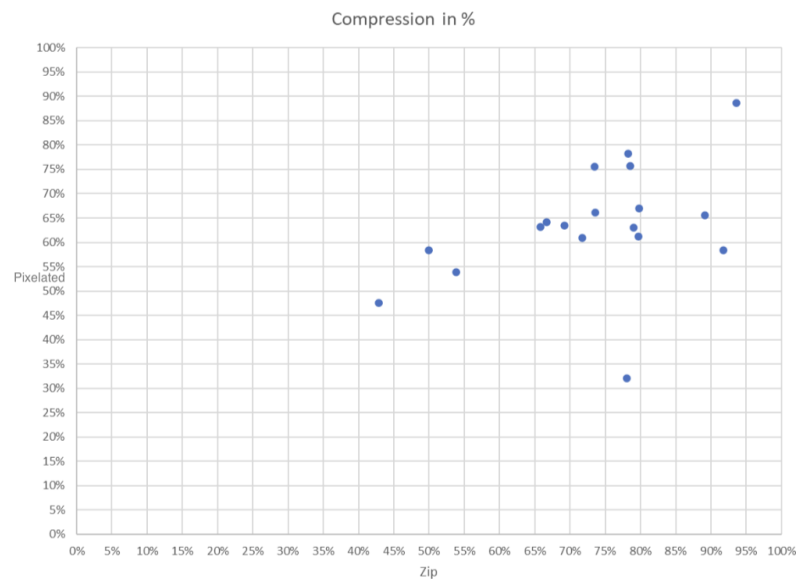


Figure 7. This figure compares ZIP compression percentage and Pixelated compression percentage on the Calgary dataset.

4. Discussion

The results demonstrate that Pixelated provides competitive compression performance rather than universally superior compression across all data regimes, compared to traditional storage and serialization formats, including ZIP, Parquet, and Pickle. Across all evaluated datasets, Pixelated achieved an average compression rate exceeding 95%, significantly outperforming ZIP, which averaged approximately 71%. These findings directly address the scalability concerns discussed in the introduction, where rapidly increasing data generation from platforms such as Venmo and Uber [18] contributes to growing storage and infrastructure costs. By reducing file sizes more aggressively than conventional approaches, Pixelated has the potential to reduce cloud storage requirements in systems such as Amazon S3 and Google Cloud [20,21].

The environmental implications of these findings are also noteworthy. As highlighted earlier, data centers contribute significantly to global energy consumption and carbon emissions [22,23]. Since storage volume directly impacts infrastructure requirements, higher compression efficiency can reduce storage overhead and indirectly contribute to lower energy demand, although exact energy savings depend on system-level deployment and workload characteristics. In this context, Pixelated represents a data-efficient transformation framework that may contribute to sustainability improvements through reduced storage footprint rather than direct energy optimization.

The superior performance of Pixelated can largely be attributed to its hybrid design, which combines structured encoding techniques with the DEFLATE mechanism inherent in PNG files. Traditional compression methods such as Huffman Coding [2], LZ77 [3], and DEFLATE [4] rely heavily on redundancy detection and probabilistic encoding. Pixelated extends these principles by first transforming data into highly structured pixel representations before compression, thereby increasing redundancy exposure through spatial restructuring rather than modifying PNG or DEFLATE itself. The strong performance observed in highly repetitive datasets such as Enwik8 and the Calgary Benchmark supports this interpretation.

The results on transactional datasets further highlight the adaptability of Pixelated to structured enterprise data. Transactional datasets often contain mixtures of categorical, numerical, and datetime attributes with varying redundancy levels. While methods such

as Parquet optimize columnar storage and Pickle focuses on object serialization, neither is specifically designed to maximize redundancy exposure across heterogeneous data types. Pixelated's encoding strategy effectively handled these variations, achieving compression rates of up to 99.8% on the demo dataset and 93% on the real-world transactional dataset. However, it is important to note that the real transactional dataset contains 11,274 rows, which limits generalization to enterprise-scale systems with significantly larger data volumes (R1.C15).

The Calgary dataset experiments provide additional insight into the strengths and limitations of the method. Pixelated achieved its greatest improvements on large and highly repetitive files such as *Book1*, *Book2*, and *PIC*, outperforming ZIP by a substantial margin. However, for smaller binary files with limited redundancy, Pixelated and ZIP achieved comparable results. This behavior aligns with Shannon's entropy theory [1], which states that datasets with lower redundancy inherently offer fewer opportunities for compression. Consequently, Pixelated appears most effective when sufficient structural repetition exists within the data.

Compared to modern compression approaches such as recurrent neural network-based compression frameworks [7], Pixelated offers a simpler and computationally lightweight alternative. Machine learning-based methods often require extensive training and inference overhead, whereas Pixelated operates as a deterministic transformation pipeline without requiring model training. This makes it more practical for general-purpose deployment in storage systems where computational overhead is a constraint.

Despite these strengths, several limitations remain. First, the current string encoding mechanism relies primarily on ASCII-based transformations, which may not optimally compress multilingual or highly diverse datasets. Second, while PNG compression contributes effectively to redundancy reduction, the approach inherits structural constraints from image-based storage formats, including memory overhead during conversion. Third, a significant limitation is computational efficiency: Pixelated introduces substantially higher encoding time compared to classical compression algorithms, as observed in the ablation results, making it less suitable for latency-sensitive or real-time applications. Finally, performance is sensitive to dataset structure, particularly in low-redundancy binary files where gains are limited.

5. Conclusions

This paper introduced Pixelated, a data transformation framework that leverages PNG-based compression pipelines to encode structured and unstructured datasets into pixel representations.

Pixelated is not a standalone compression algorithm but a transformation framework that enhances compressibility by restructuring data prior to conventional PNG (DEFLATE-based) compression. Experimental results demonstrate consistent performance improvements over ZIP, Parquet, and Pickle across multiple benchmark and transactional datasets.

The method achieved an average compression rate of approximately 95–96%.

Key contributions include:

- A structured pixel-based encoding framework for tabular data;
- A hybrid dictionary and type-aware encoding strategy;
- Demonstration of improved redundancy exposure for compression systems.

Limitations include encoding overhead, dependency on PNG structure, reduced performance on low-redundancy binary datasets, and notably higher computational cost compared to traditional compression algorithms.

Future work will focus on optimizing encoding efficiency to reduce runtime overhead, developing adaptive entropy-aware encoding schemes, improving tokenization for

multilingual and heterogeneous data, and exploring alternative image or tensor-based representations. Further research will also investigate integration with modern entropy coders such as Zstd and Brotli, as well as hardware-accelerated implementations to improve scalability in production environments.

Author Contributions: Conceptualization, Z.A.; methodology, Z.A.; validation, Z.A.; formal analysis, Z.A.; investigation, Z.A.; resources, P.M.; data curation, Z.A.; writing—original draft preparation, Z.A.; writing—review and editing, Z.A. and P.M.; visualization, Z.A.; supervision, P.M.; project administration, P.M.; funding acquisition, P.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data in this study was obtained from Saalas by permission. Data will be shared upon reasonable request to the corresponding author with permission from Saalas.

Acknowledgments: The author would like to sincerely thank the Editor and Reviewers for their valuable time, constructive feedback, and insightful comments, which significantly improved the quality and clarity of this manuscript. The author would also like to thank Farooq Seru from Company Saalas for providing the dataset used in this study. Their assistance was instrumental in enabling the experimental evaluation and validation of the proposed method. Finally, the author is deeply grateful to Patrick Mukala for his continuous guidance, encouragement, and support throughout the course of this research.

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

Abbreviations

The following abbreviations are used in this manuscript:

ZIP	Zstandard/ZIP lossless compression format based on DEFLATE
PNG	Portable Network Graphics image format using DEFLATE compression
DEFLATE	Lossless compression algorithm combining LZ77 and Huffman coding
LZ77	Dictionary-based sliding window compression algorithm
BWT	Burrows–Wheeler Transform for improving compression efficiency
ASCII	American Standard Code for Information Interchange
CSV	Comma-Separated Values tabular data format
GPU	Graphics Processing Unit (used for acceleration in related ML methods)
RNN	Recurrent Neural Network used in machine learning-based compression

References

- Shannon, C. A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]
- Huffman, D. A method for the construction of minimum-redundancy codes. *Proc. IRE* **1952**, *40*, 1098–1101. [CrossRef]
- Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343. [CrossRef]
- Gailly, J.; Adler, M. DEFLATE Compressed Data Format Specification Version 1.3. 1996. Available online: <https://www.rfc-editor.org/info/rfc1951> (accessed on 10 January 2023).
- Burrows, M.; Wheeler, D.J. *A Block-Sorting Lossless Data Compression Algorithm*; Digital Systems Research Center: Palo Alto, CA, USA, 1994.
- Pasco, R. Source coding algorithms for fast data compression (Ph.D. Thesis abstr.). *IEEE Trans. Inf. Theory* **1977**, *23*, 548. [CrossRef]
- Goyal, M.; Tatwawadi, K.; Chandak, S.; Ochoa, I. DeepZip: Lossless data compression using recurrent neural networks. In Proceedings of the 2019 Data Compression Conference (DCC), Snowbird, UT, USA, 26–29 March 2019. [CrossRef]
- Ko, Y.; Chadwick, A.; Bates, D.; Mullins, R. Lane compression: A Lightweight Lossless Compression Method for Machine Learning on Embedded Systems. *ACM Trans. Embed. Comput. Syst.* **2021**, *20*, 1–26. [CrossRef]

9. Altamimi, A.; Belgacem, B.Y. Lossless and Near-Lossless Compression Algorithms for Remotely Sensed Hyperspectral Images. *Entropy* **2024**, *26*, 316. [CrossRef] [PubMed]
10. Erkuš, E.C.; Bursali, A. A Comparative Analysis of the Lossless Data Compression Methods for Unsparsed Tabular Data. In Proceedings of the 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET), Sydney, Australia, 25–27 July 2024; pp. 1–6. [CrossRef]
11. Wang, W.; Chen, W.; Lei, Q.; Li, Z.; Zhao, H. ACTF: An efficient lossless compression algorithm for time series floating point data. *J. King Saud-Univ. Comput. Inf. Sci.* **2024**, *36*, 102246. [CrossRef]
12. Ma, Z.; Zhu, H.; He, Z.; Lu, Y.; Song, F. Deep Lossless Compression Algorithm Based on Arithmetic Coding for Power Data. *Sensors* **2022**, *22*, 5331. [CrossRef] [PubMed]
13. Ghose, B.; Rehena, Z. A Robust Lossless Data Compression Technique for Air Quality Time-Series Data using Genetic Algorithm. *J. Univers. Comput. Sci.* **2026**, *32*, 84–107. [CrossRef]
14. Hidayat, T.; Zakaria, M.H.; Pee, A.N.C. Increasing the Huffman generation code algorithm to equalize compression ratio and time in lossless 16-bit data archiving. *Multimed. Tools Appl.* **2023**, *82*, 24031–24068. [CrossRef]
15. Bell, T.; Witten, I.H.; Cleary, J.G. Modeling for text compression. *ACM Comput. Surv.* **1989**, *21*, 557–591. [CrossRef]
16. Wang, W.-Y. A unique perspective on data coding and decoding. *Entropy* **2010**, *13*, 53–63. [CrossRef]
17. Matias, Y.; Rajpoot, N.; Sahinalp, C. Flexible Parsing (FP), Flexible Parsing Algorithm. 1999. Available online: <http://www.dcs.warwick.ac.uk/~nasir/work/fp/> (accessed on 4 June 2026).
18. Marr, B. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. *Forbes Magazine*, 2018. Available online: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=33d5311360ba> (accessed on 30 October 2022).
19. Vailshery, L. Percent of Corporate Data Stored in the Cloud 2022; Statista: Hamburg, Germany, 2022. Available online: <https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate-data/#:~:text=As%20of%202022%2C%20over%2060,next%20to%20advancing%20business%20agility> (accessed on 30 October 2022).
20. Yuen, M. Amazon Revenue Breakdown by Segment; Insider Intelligence: New York, NY, USA, 2022. Available online: <https://www.insiderintelligence.com/insights/amazon-revenue/> (accessed on 30 October 2022).
21. Lardinois, F. Google Cloud Gets More Expensive; TechCrunch: San Francisco, CA, USA, 2022. Available online: <https://techcrunch.com/2022/03/14/inflation-is-real-google-cloud-raises-its-storage-prices/> (accessed on 30 October 2022).
22. Zabeu, S. Five Areas of Environmental Impact in Data Centres. Network. 2022. Available online: <https://network-king.net/five-areas-of-environmental-impact-in-data-centres/> (accessed on 30 October 2022).
23. Monserrate, S.G. *The Staggering Ecological Impacts of Computation and the Cloud*; The MIT Press Reader: Cambridge, MA, USA, 2022. Available online: <https://thereader.mitpress.mit.edu/the-staggering-ecological-impacts-of-computation-and-the-cloud/#:~:text=The%20Cloud%20now%20has%20a,as%20much%20as%2025%20percent> (accessed on 30 October 2022).
24. Mahoney, M. About the Test Data. 2011. Available online: <https://mattmahoney.net/dc/textdata.html> (accessed on 5 July 2023).

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