FLEA: Frequency-based Lossless Encoding Algorithm for Periodic Time Series

TIANRUI XIA, Tsinghua University, China
JINZHAO XIAO, Tsinghua University, China
SHAOXU SONG*, Tsinghua University, China

Existing lossless time series compressors surprisingly ignore the strong periodic pattern, a common characteristic that limits their effectiveness on real-world data. To address this, we design **FLEA** *(Frequency-based Lossless Encoding Algorithm)*, a novel lossless encoding scheme that exploits the frequency domain for periodic time series. While directly storing the extremely high precision frequency coefficients is obviously not an option to lossless compression, we propose to quantize the frequency coefficients and capture the corresponding low precision residuals in time domain. The core challenge is thus to optimize the trade-off between the encoding costs of the *quantized frequency component* and its *time-domain residual*. We address this by modeling it as a rate-optimal search problem, made tractable by an energy-based cost model that guides the global search for the optimal parameter. Moreover, FLEA introduces two adaptive encoders that (1) horizontally partition the frequency component for skewed and sparse coefficients, and (2) vertically partition the residual bit-width for the long-tailed distribution. Extensive experiments on a diverse suite of real-world datasets show that FLEA establishes a new state-of-the-art in compression ratio, particularly on its target periodic data, with an average improvement of 12.9% over the runner-up. Crucially, this is achieved with highly competitive throughput; its efficient decoding, more than twice as fast as its encoding, makes FLEA a powerful and practical solution for read-intensive database workloads, leading to its native implementation in Apache TsFile.

CCS Concepts: • **Information systems** → **Data compression**; • **Theory of computation** → **Algorithm design techniques**; • **Mathematics of computing** → *Computation of transforms.*

Additional Key Words and Phrases: Time Series Compression, Lossless Compression, Periodic Time Series, Frequency Domain, Encoding Algorithm

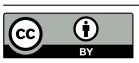
ACM Reference Format:
Tianrui Xia, Jinzhao Xiao, and Shaoxu Song. 2026. FLEA: Frequency-based Lossless Encoding Algorithm for Periodic Time Series. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 186 (June 2026), 26 pages. https://doi.org/10.1145/3802063

1 Introduction

Lossless compression is needed to support applications like medical (ECG [16, 43] and EEG [2]), seismic [22], scientific or operational data, and thus widely studied [18, 24, 28, 41]. As time series data grow exponentially, efficient query processing frameworks have become essential for modern databases [26]. There are distinct roles of storage versus analytics. Specifically, in addition to "trends", specific details with small fluctuations are also valuable to preserve. For example, in high-precision monitoring (e.g., power grid monitoring), specific details with small fluctuations

*Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

Authors' Contact Information: Tianrui Xia, Tsinghua University, Haidian Qu, Beijing Shi, China, xiatr24@mails.tsinghua.edu.cn; Jinzhao Xiao, Tsinghua University, Haidian Qu, Beijing Shi, China, xjc22@mails.tsinghua.edu.cn; Shaoxu Song, Tsinghua University, Haidian Qu, Beijing Shi, China, sxsong@tsinghua.edu.cn.



ACM 2836-6573/2026/6-ART186
https://doi.org/10.1145/3802063

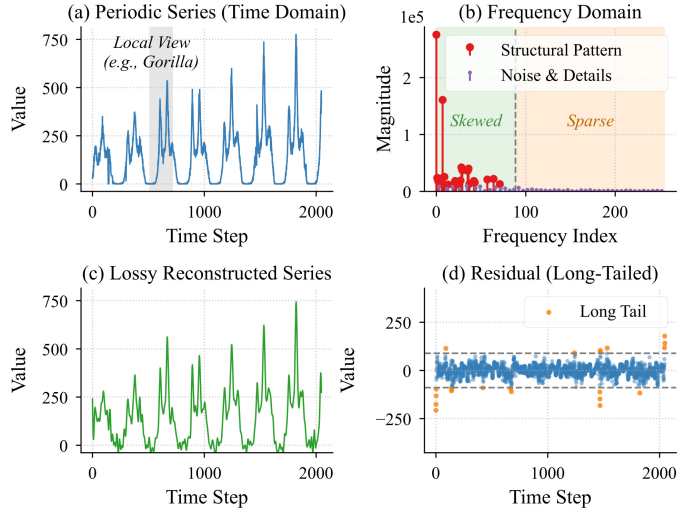


Fig. 1. Conceptual illustration of FLEA, storing only the (red) structural pattern in frequency domain and (green) residual in time domain. With a few coefficients of structural pattern in (b), it can reconstruct approximately the series in (c). Together with the small residual in (d), it ensures perfect lossless reconstruction of the original series in (a).

often indicate early faults. Indeed, it may be difficult to precisely determine which ones are true "noises" to eliminate. Instead, as a data storage service, we propose to compress data as is.

1.1 Motivation

The core idea of Discrete Fourier Transform (DFT) is a linear transformation that projects the time series onto a basis of orthogonal sinusoidal waves, effectively concentrating periodic structures into frequency domain coefficients. DFT is appropriate here because it can use only a few dominant coefficients to re-construct the original time series, together with small residuals in time domain for lossless compression. The challenge is thus how to quantize the high-precision coefficients such that the total storage of the remaining frequency domain coefficients and the corresponding time domain residuals can be reduced.

Unfortunately, existing lossless encoders for time series like Gorilla [24] and Chimp [18] fail to capture the strong **periodic nature** inherent in many real-world series [6, 11], which manifests as strict cycles, quasi-seasonal trends, or other recurring patterns. As illustrated conceptually in Figure 1(a), this leaves significant compression potential untapped.

However, storing high-precision coefficients is prohibitively expensive. The reason is that these coefficients are high-precision complex numbers. The DFT transforms N real values of time-series into N independent real scalars, which constitute the real and imaginary parts of the complex frequency coefficients. In other words, storing the full set of coefficients costs the same storage ($64N$ bits) as the raw input, and thus is not feasible.

To give a concrete example, in Figure 1, the length of the time series is $N = 2048$, the DFT yields $N/2 = 1024$ complex numbers, each requiring two 64-bit values (real and imaginary), totaling $2 \times (N/2) \times 64 = 64N$ bits. This calculation demonstrates that encoding DFT coefficients as high-precision complex numbers offers no compression gain, thereby highlighting the necessity of our quantization approach.

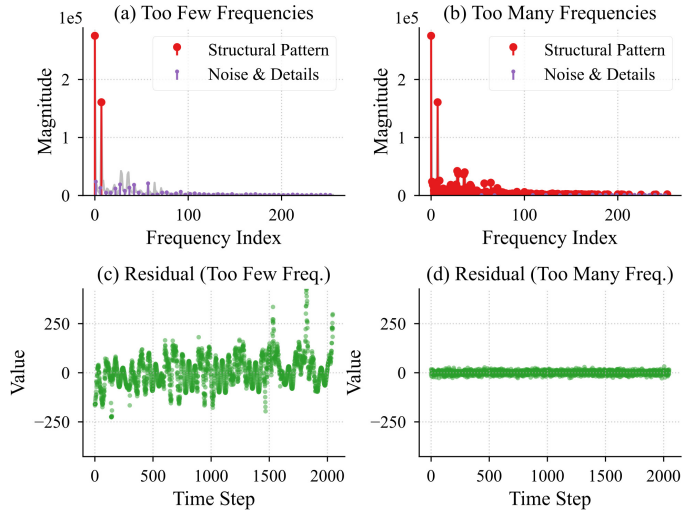


Fig. 2. The trade-off between retained frequency coefficients and time domain residual, showing two cases of (a)(c) too few or (b)(d) too many reserved frequency coefficients.

As Figure 1(b) shows, only a few dominant coefficients are needed to capture the global structure. Reconstructing the series from these few, quantized coefficients yields a close approximation as in Figure 1(c). This process, however, is inherently lossy [39].

To achieve lossless compression, we propose to further store the difference between the reconstructed series and the original one, i.e., the residual in Figure 1(d). Such small, low-precision residual in time domain together with the a few, quantized coefficients for structural pattern in frequency domain ensure perfect lossless reconstruction of the original series.

1.2 Challenge

The core challenge is how to balance the frequency coefficients and the residual such that the total encoded size is minimized. Figure 2 illustrates this trade-off, where the x-axis (*frequency index*) denotes the discrete frequency bin j in the DFT, and the y-axis (*magnitude*) denotes the spectral amplitude $|F_j|$. If too few coefficients are retained (Figure 2(a)), the reconstructed series deviates significantly, leading to large residuals (Figure 2(c)). Conversely, retaining too many coefficients (Figure 2(b)) is costly, even though the residual becomes small (Figure 2(d)).

Moreover, as shown in Figure 1(b), the low-frequency coefficients are **skewed** while the high-frequency ones are **sparse** (many zeros after quantization). Existing methods [39] fail to treat them separately. Similarly, as presented in Figure 1(d), the residual exhibits a **long-tailed** distribution, causing standard bit-packing [28] to waste space on the majority of small values.

1.3 Intuition

Our key insight is to treat quantization as a tunable knob that controls the trade-off. The optimal quantization level minimizes the total encoded size of both components.

Coarser quantization yields smaller frequency coefficients but larger quantization error variance, leading to larger residuals. To encode the frequency component efficiently, we **horizontally partition** the coefficients into skewed and sparse regions. The intuition of partitioning is to avoid wasting bit width storing zero values in sparse coefficients.

Hence, for the skewed region, we exploit their quasi-descending magnitudes. For the sparse region, which contains mostly zeros, we store only non-zero values with their indices to avoid wasting bits on zeros.

Similarly, to handle the long-tailed residual, we **vertically partition** each value into high-bits and low-bits. The low-bits are uniform, suitable for fixed-width packing. The high-bits are sparse (mostly zero), and we store only the non-zero ones to save space.

1.4 Contribution

The main contributions of this paper are summarized as follows:

(1) We propose **FLEA**, a novel Frequency-based Lossless Encoding Algorithm. The balance of frequency coefficients and time domain residual is controlled by an auto-determined frequency quantization level β, minimizing the encoding cost estimation of both.
(2) We design an adaptive **Laminar Encoding** for the frequency component. The skewed low-frequency coefficients and the sparse high-frequency coefficients are partitioned by an auto-determined parameter p, which minimizes the total estimated cost.
(3) We design an adaptive **Hybrid Encoding** scheme for the residual component, which vertically partitions residuals based on an auto-determined cut bit D to efficiently handle their long-tailed distribution. Remarkably, with Proposition 1 on error energy propagation, we can determine the parameter D without conducting frequency-time transformation to general actual residual.
(4) We conduct extensive experiments on diverse real-world datasets. The results demonstrate that FLEA not only achieves SOTA compression ratios, with an average improvement of **12.9%** over the runner-up, but also delivers highly competitive throughput, positioning it as a superior balanced solution.
(5) We demonstrate the practical viability of our approach in the open-source file format **Apache TsFile**. This integration showcases FLEA's compelling balance of high compression and efficient decoding, making it a powerful and practical choice for modern read-intensive database workloads.

The remainder of this paper is organized as follows. Section 2 reviews related work in time series compression. Section 3 introduces background concepts. Section 4 details the FLEA methodology and its core components. Section 5 presents our extensive experimental evaluation. Section 6 discusses the system implementation of FLEA in Apache TsFile. Finally, Section 7 concludes the paper.

2 Related Work

This section reviews existing work in lossless data compression, with a particular focus on techniques relevant to time series data and the frequency-domain approaches that inform our proposed FLEA algorithm.

2.1 Lossless Compression for Time Series

While general-purpose lossless encoding algorithms like **Gzip** [14], **LZ4** [4, 44], **Snappy** [29] or **ZSTD** [8] can be applied to time series, they often achieve suboptimal compression ratio compared to methods specialized for sequential data characteristics. As noted in Sprintz [5], time series "lack exact repeats". This renders entropy encoders like LZ4 ineffective. Similar fundamental limitation applies to ZSTD. This prevents dictionary-based compressors from finding matches, and results in poor compression ratio in Sprintz's experiments.

Specialized time series encoding algorithms frequently employ predictive or differential schemes. **Gorilla** [24], influential in time-series databases, uses XOR-based delta encoding with rule-based compaction. **Chimp** [18] enhances Gorilla by searching a wider window for more effective XOR

Table 1. Table of notations

Symbol	Description
$\mathbf{X}$	Original time series of length n, where X_k is the k-th element (pre-processed to integer by scaling [19])
$\mathbf{F}$	Frequency-domain coefficients of $\mathbf{X}$ from DFT
$\hat{\mathbf{F}}$	Quantized frequency coefficients (integer)
$\mathbf{R}$	Time-domain residual values (integer)
β	Quantization level of frequency coefficients
p	Split index of frequency coefficients for horizontal partitioning
D	Cut bit of residual values for vertical partitioning

differencing. **Run-Length Encoding (RLE)** [12] is a classic technique effective for sequences with long identical runs, encoding them as value-count pairs.

More recent approaches often combine multiple stages. **Sprintz** [5] processes delta-coded values with Zigzag encoding, bit-packing, and entropy coding. **BUFF** [19] preprocesses data by value splitting and differencing against local minima to reduce the dynamic range. **HIRE** [3] introduces a hierarchical residual encoding framework, particularly for multi-resolution data.

These diverse encoding methods form the benchmark for FLEA. They primarily operate in the time domain or use general statistical models. FLEA distinguishes itself by leveraging frequency-domain insights within a rate-optimal search.

2.2 Frequency-based Lossy Compression

In data compression, transform coding is a central concept, particularly for successful lossy algorithms like JPEG for images. Adapting these principles for the *lossless* compression of time series, however, requires a fundamentally different approach focused on careful management of quantization and residuals.

A recent and relevant work [39] proposed a lossy, frequency-domain compression scheme for time series. Their work also observed that the quantized frequency spectrum exhibits both skewed and sparse features. However, they employed a single, monolithic encoding scheme, Descending Bit Packing (DBP), to process the entire spectrum uniformly. While DBP is effective for the highly sparse tail of the spectrum, its reliance on storing an index for every non-zero value creates significant overhead in the dense, skewed low-frequency coefficients.

To overcome this limitation, FLEA introduces a novel approach, by first partitioning the frequency coefficients into separate skewed and sparse regions. It then applies a specialized encoder, **Laminar Encoding**, tailored for each. This holistic, optimization-driven approach differentiates FLEA from prior work.

3 Preliminaries

We provide a brief overview of the fundamental concepts, notations, and existing techniques that form the basis of our proposed algorithm, FLEA. Table 1 lists the frequently used notations.

3.1 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) takes a time series as input and outputs frequency coefficients representing periodic strength at each frequency. Its significance in our design is to concentrate periodic structures into a few dominant coefficients, enabling efficient compression.

Formally, for a time series $\mathbf{X} = \{X_0, \ldots, X_{n-1}\}$ of length n, its DFT, $\mathbf{F} = \{F_0, \ldots, F_{n-1}\}$, is a sequence of n complex numbers,

$$F_j = \sum_{k=0}^{n-1} X_k \cdot e^{-i\frac{2\pi}{n}jk}, \quad j = 0, \ldots, n-1. \tag{1}$$

The original sequence X_k can be perfectly reconstructed from $\mathbf{F}$ using the Inverse DFT (IDFT):

$$X_k = \frac{1}{n} \sum_{j=0}^{n-1} F_j \cdot e^{i\frac{2\pi}{n}jk}, \quad k = 0, \ldots, n-1. \tag{2}$$

Parseval's theorem [23] states that the sum of squared values is preserved across domains:

$$\sum_{k=0}^{n-1} |X_k|^2 = \frac{1}{n} \sum_{j=0}^{n-1} |F_j|^2. \tag{3}$$

This relationship is fundamental to our analysis of quantization error in Section 4.

3.2 Uniform Quantization

Uniform quantization is a fundamental technique for reducing data precision in signal processing and compression [21]. Its core principle is to map a continuous range of values to a smaller, finite set of discrete integer indices. In our work, this process is controlled by an integer parameter β.

Given a real value v, the **quantization** operation maps it to an integer $\hat{v}$ using a step size of 2^β:

$$\hat{v} = \text{Quantize}(v, \beta) = \text{round}\left(\frac{v}{2^\beta}\right). \tag{4}$$

The corresponding **de-quantization** operation reconstructs a lossy approximation from the quantized integer $\hat{v}$:

$$\text{Dequantize}(\hat{v}, \beta) = \hat{v} \times 2^\beta. \tag{5}$$

The information lost during this process is the **quantization error**, e, calculated as $e = v - \text{Dequantize}(\hat{v}, \beta)$.

For example, a floating-point value such as 13.5 can be mapped to a smaller integer. If we use $\beta = 2$ (step size of $2^\beta = 4$), the resulting index is $\hat{v} = \text{round}(13.5/4) = 3$. Storing the small integer index 3 requires significantly fewer bits than storing the original high-precision value.

4 The FLEA Methodology

This section details the proposed algorithm, FLEA. We begin by introducing its core decomposition strategy in Section 4.1, which transforms a time series into a frequency component and a residual component. Subsequently, we present the specialized, adaptive encoding schemes designed for each of these two components, in Sections 4.2 and 4.3, respectively. Finally, we describe how these elements are integrated into a global, *rate-optimal* search to find the best compression parameters, in Section 4.4. Here, rate-optimal refers to the global minimization of the total bit-cost function, balancing the storage of quantized frequency coefficients versus time-domain residuals.

4.1 FLEA Overview

As motivated in Section 1.1, FLEA captures periodicity through the spectral pattern, a sparse set of dominant DFT coefficients that represent the core periodic structure. However, these coefficients are high-precision complex numbers, and encoding them losslessly is prohibitively expensive.

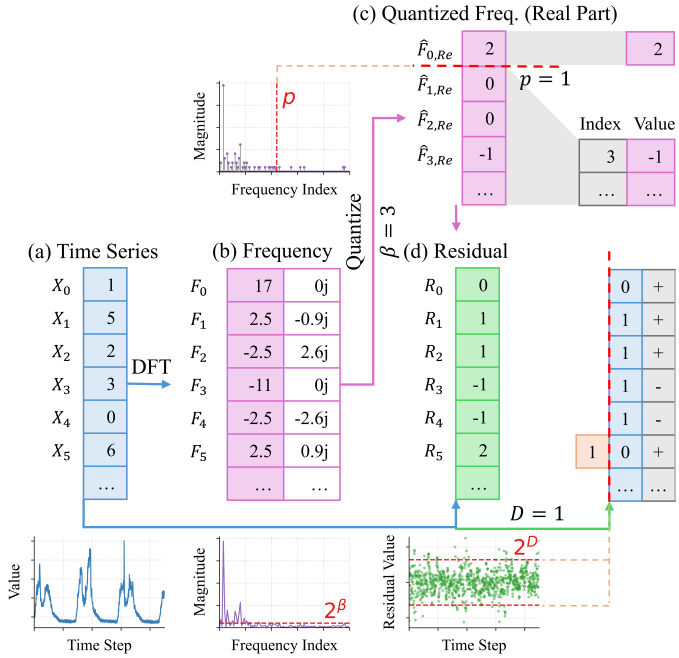


Fig. 3. An overview of the FLEA compression process, showing how a time series $\mathbf{X}$ in (a) is decomposed into the quantized frequency component $\hat{\mathbf{F}}_\beta$ in (c) with quantization level β and the time-domain residual $\mathbf{R}_\beta$ in (d). While $\hat{\mathbf{F}}_\beta$ is encoded with horizontal partitioning by parameter p, $\mathbf{R}_\beta$ is vertically partitioned by parameter D in encoding.

4.1.1 Encoding. Our FLEA is designed to manage this precision-compressibility trade-off. It leverages uniform quantization (Section 3.2) to deterministically decompose an original sequence, $\mathbf{X}$, into two new components after performing a DFT for $\mathbf{F}$:

(1) **Quantized Frequency Component $\hat{\mathbf{F}}_\beta$**: An integer sequence representing the primary spectral patterns after quantization by a parameter β. It is formally computed as $\hat{\mathbf{F}}_\beta = \text{Quantize}(\mathbf{F}, \beta)$.
(2) **Residual Component $\mathbf{R}_\beta$**: An integer sequence storing the exact error to ensure lossless reconstruction. It is defined as $\mathbf{R}_\beta = \mathbf{X} - \text{round}(\text{IDFT}(\text{Dequantize}(\hat{\mathbf{F}}_\beta, \beta)))$.

4.1.2 Decoding. The aforementioned decomposition ensures perfect, bit-for-bit reconstruction of the original sequence. Specifically, if $\text{Dequantize}(\hat{\mathbf{F}}_\beta, \beta)$ is the de-quantized approximation of the spectrum, the original sequence is precisely recovered by $\mathbf{X} = \text{round}(\text{IDFT}(\text{Dequantize}(\hat{\mathbf{F}}_\beta, \beta))) + \mathbf{R}_\beta$. The residual $\mathbf{R}_\beta$ is therefore defined as the integer difference that guarantees this identity.

4.1.3 Parameter. The entire process is illustrated in Figure 3. It is governed by three key parameters: (1) the quantization level β for frequency coefficients, (2) the split index p for horizontally partitioning the quantized frequency coefficients, and (3) the cut bit D for vertically partitioning the residual values. The roles and automatic determination of these parameters are central to FLEA and will be detailed in the following sections.

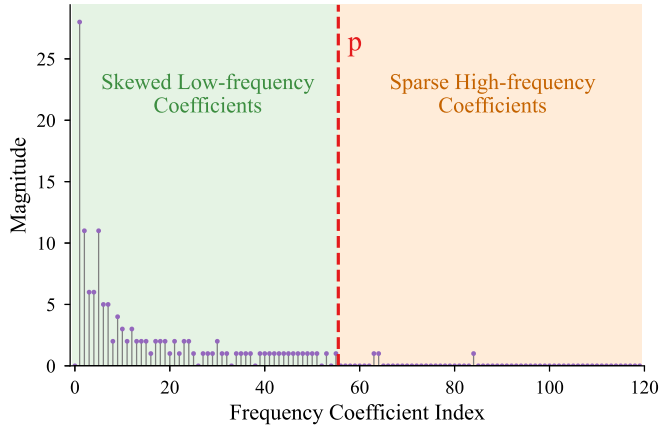


Fig. 4. Magnitude distribution of a typical quantized frequency spectrum, which can be partitioned into skewed low-frequency coefficients and sparse high-frequency coefficients by an auto-determined parameter p.

4.2 Encoding the Frequency Component

Having established FLEA's decomposition approach, we now detail the encoding schemes for each component, starting with the frequency component, $\hat{\mathbf{F}}_\beta$. Since $\hat{\mathbf{F}}_\beta$ is a sequence of complex numbers, we treat its real and imaginary parts as two independent integer sequences, $\hat{\mathbf{F}}_{\beta,\text{re}}$ and $\hat{\mathbf{F}}_{\beta,\text{im}}$, and apply our encoding process to each. This section is structured as follows: we first analyze the key characteristics of the quantized frequency data in Section 4.2.1, then introduce our novel encoding scheme designed for these characteristics, and finally detail the cost model used to automatically determine its parameters in Section 4.2.4.

4.2.1 Partitioning Quantized Frequency Coefficients. Note that the concave "U" curve typically represents the full spectrum. For real-valued time series, this shape is caused by conjugate symmetry ($|F_k| = |F_{N-k}|$), where magnitudes at high frequencies mirror those at low frequencies. Since the second half is redundant, FLEA only needs to encode the first half (0 to $N/2$). Within this non-redundant region, the supporting evidence in literature (e.g., $1/f$ noise [17, 27, 36]) confirms that energy concentrates in low frequencies, creating the quasi-descending trend we identify.

This creates two key characteristics in the quantized frequency spectrum, as empirically visualized in Figure 4. First, it imposes a dense, quasi-descending trend on the magnitudes of low-frequency coefficients. Second, it leads to a sparse tail, as the magnitudes of high-frequency coefficients are often so small that they are quantized to zero.

While prior work also observed these features [39], it tended to apply a single, monolithic encoder. Our key insight is that the drastic difference in data density between the low-frequency and high-frequency coefficients necessitates a partitioning strategy. We therefore divide the frequency sequence $\hat{\mathbf{F}}_\beta$ at a split index p (determined below) into skewed low-frequency coefficients, $\hat{\mathbf{F}}_\beta[0:p]$, and sparse high-frequency coefficients, $\hat{\mathbf{F}}_\beta[p:n]$. This allows us to handle the density difference efficiently, while still leveraging the global quasi-descending property within each part.

4.2.2 Laminar Encoding for the Skewed Low-Frequency Coefficients. To capitalize on the properties of the skewed low-frequency coefficients, we propose a novel encoding paradigm named **Laminar Encoding**. The name is inspired by the concept of laminar flow, reflecting how our algorithm

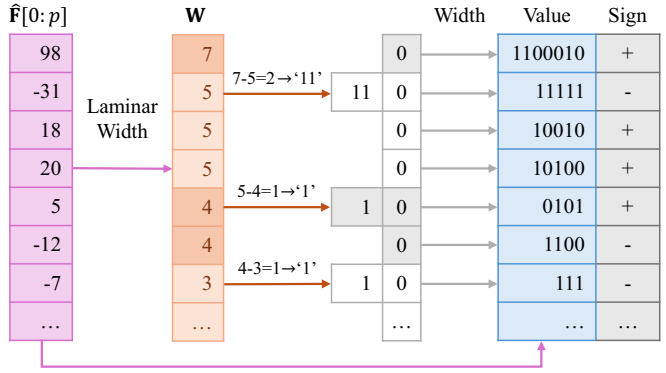


Fig. 5. An example of the Laminar Encoding process. The monotonic Laminar Width array $\mathbf{W}$ is first computed from the input $\hat{\mathbf{F}}[0:p]$. Then, the encoding results are generated based on the values of $\mathbf{W}$ and $\hat{\mathbf{F}}[0:p]$.

smoothly guides the decoding process through different "layers" of bit-width requirements. Unlike sparse encoders like DBP [39], which require pre-sorting values and explicitly storing an index for each coefficient, Laminar Encoding processes data in its original order without this index overhead. This allows it to fully exploit the natural **quasi-descending trend** of the data, which was established as a consequence of global spectral decay in Section 4.2.1.

Laminar Width Definition. The scheme first computes the *Laminar Width* array, $\mathbf{W}$. For the skewed low-frequency coefficients $\hat{\mathbf{F}}_\beta[0:p]$, each element W_i is the maximum bit-width required for any value in its suffix:

$$W_i = \max_{j=i}^{p-1} \left(\lceil \log_2(|\hat{F}_j| + 1) \rceil \right). \tag{6}$$

By its "suffix-maximum" definition, the array $\mathbf{W}$ is guaranteed to be monotonically non-increasing ($W_i \geq W_{i+1}$). This mathematical property is the foundation of our simple, state-driven encoding process. Crucially, because the underlying data exhibits a quasi-descending trend, this monotonic $\mathbf{W}$ array effectively acts as a tight structural "envelope" of the data's magnitudes, making it highly compressible.

Laminar Width Encoding. Instead of storing $\mathbf{W}$ directly, we encode the delta of the bit-widths. Specifically, for each coefficient $\hat{F}_i$, we compute the width decrement $\Delta_i = W_{i-1} - W_i \geq 0$. This sequence of non-negative deltas is then encoded using a combination of unary coding [30] and Run-Length Encoding (RLE) [12]. Each delta Δ_i is represented as a unary code of Δ_i '1's followed by a '0' terminator. Compared to bit-packing, using 1s ended by a 0 avoids the extra bit width caused by a few large values.

Laminar Encoding Process. Figure 5 provides a step-by-step illustration of this process. As shown, after the monotonic Laminar Width array ($\mathbf{W}$) is computed, it is efficiently encoded by representing its decrements, while the original values are encoded using the bit-widths specified by $\mathbf{W}$. The final encoding thus consists of three parts. First, the initial width W_0 is stored explicitly as metadata, providing the decoder with its starting state. Second, the compressed representation of all subsequent width changes in $\mathbf{W}$ is stored. Third, the payload of actual coefficients $\{\hat{F}_i\}$, where each $\hat{F}_i$ is encoded using $W_i + 1$ bits, is stored.

Low-frequency Coefficients Encoding Cost. $L_{\text{Laminar}}(\hat{\mathbf{F}}_\beta[0:p])$, the total cost for low-frequency coefficients, is formalized as:

$$L_{\text{Laminar}}(\hat{\mathbf{F}}_\beta[0:p]) = L(W_0) + L(\mathbf{W}) + \sum_{i=0}^{p-1}(W_i + 1) \tag{7}$$

where $L(W_0)$ is a small, fixed cost for the initial width and $L(\mathbf{W})$ is the cost of the compressed width information determined by Run-Length Encoding (RLE) [12].

Laminar encoding works best when the number of bits required by low-frequency coefficients of the quantized DFT are non-increasing. If a coefficient W_i happens to be more than double than the lower frequencies' coefficients $W_{i'}$, it will use more (but not excessively) bits. The reason is that laminar coding has a logarithmic scale $\lceil \log_2 v \rceil$. For example, in Figure 4, although index 5 is larger than indexes 3 & 4, it increases the required bit width W by only 1 bit.

4.2.3 Index Encoding for Sparse High-frequency Coefficients. Laminar Encoding becomes inefficient in the sparse high-frequency coefficients $\hat{\mathbf{F}}_\beta[p:n]$, as it is dominated by zero-valued coefficients. Therefore, an effective sparse strategy must focus only on the non-zero values, which requires encoding two distinct components: their indices ($\mathbf{P}$) and their values ($\mathbf{V}$). The value sequence $\mathbf{V}$ is encoded using the Laminar Encoding as detailed in Section 4.2.2. In the following, we focus on the strategy for efficiently encoding the index sequence $\mathbf{P}$.

Index Encoding. The primary challenge in encoding the indices $\mathbf{P}$ is that due to the sparsity, the gaps between consecutive indices can be large and irregular. However, a key exploitable property is that all index values are strictly bounded by the length of the sequence. Given this well-defined range and the lack of a predictable pattern in the gaps, a simple and robust fixed-width bit-packing scheme is a highly efficient choice. Each index is encoded using a fixed number of bits, determined by the length of the sequence.

High-frequency Coefficients Encoding Cost. The total cost for encoding sparse high-frequency coefficients, L_{Sparse}, is the sum of these two parts:

$$L_{\text{Sparse}}(\hat{\mathbf{F}}_\beta[p:n]) = L_{\text{Index}}(\mathbf{P}) + L_{\text{Laminar}}(\mathbf{V}) \tag{8}$$

where $L_{\text{Laminar}}(\mathbf{V})$ is calculated using Equation 7. The index cost $L_{\text{Index}}(\mathbf{P})$ is computed using bit-packing techniques [28].

4.2.4 Automatic Determination of Split Index p. With the two-part encoding method established, we proceed to construct the frequency component's cost model. This is achieved by finding the optimal boundary, or split index, that demarcates the low-frequency and high-frequency coefficients. Since we process the real and imaginary components independently, we must find an optimal split index for each sequence: p^*_{re} for the real sequence $\hat{\mathbf{F}}_{\beta,\text{re}}$, and p^*_{im} for the imaginary sequence $\hat{\mathbf{F}}_{\beta,\text{im}}$.

For any candidate split index p, the total encoding cost for a sequence, $L(p)$, is the sum of applying Laminar Encoding to the low-frequency coefficients $[0:p]$ and the Sparse Adaptation to the high-frequency coefficients $[p:n]$. The optimizations for the real and imaginary parts are **decomposable** and can be solved independently:

$$p^* = \arg\min_{0 \le p \le n} \left(L_{\text{Laminar}}(\hat{\mathbf{F}}_\beta[0:p]) + L_{\text{Sparse}}(\hat{\mathbf{F}}_\beta[p:n]) \right) \tag{9}$$

$$L(p^*) = L_{\text{Laminar}}(\hat{\mathbf{F}}_\beta[0:p^*]) + L_{\text{Sparse}}(\hat{\mathbf{F}}_\beta[p^*:n]). \tag{10}$$

We solve this optimization by performing an exhaustive search for p. This search is highly efficient due to our cost models, allowing it to be completed in $O(n)$ time. The costs for encoding the skewed low-frequency coefficients for all possible split indices $p = 0, \ldots, n$, denoted as $L_{\text{Laminar}}(\hat{\mathbf{F}}_\beta[0:p])$,

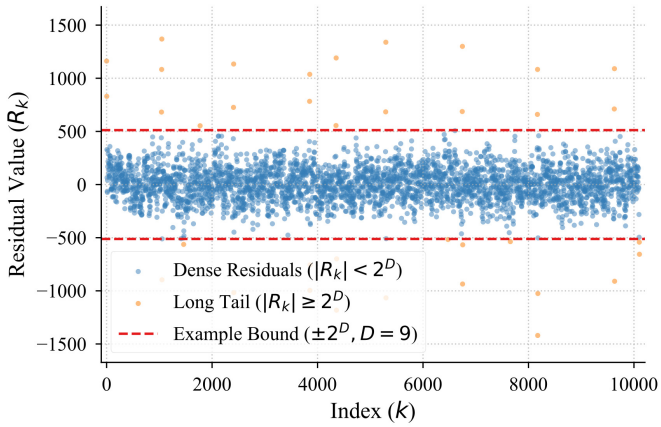


Fig. 6. The actual residual distribution from a sample series. The dashed red lines indicate an **example partitioning boundary** for $D = 9$.

can be pre-calculated in a single forward pass. Similarly, the costs for encoding the sparse high-frequency coefficients for all possible split indices, $L_{\text{Sparse}}(\hat{\mathbf{F}}_\beta[p:n])$, can be pre-calculated in a single backward pass. A final linear scan is then sufficient to find the optimal split index.

The final, minimized cost, $L(\hat{\mathbf{F}}_\beta) = L_{\text{re}}(p^*_{\text{re}}) + L_{\text{im}}(p^*_{\text{im}})$, where $L_{\text{re}}(p^*_{\text{re}})$ and $L_{\text{im}}(p^*_{\text{im}})$ are defined as in Equation 10, represents the estimated number of bits required to encode the entire frequency component for a given β. It serves as a critical input to the global optimization of β that will be detailed in Section 4.4.

4.3 Encoding the Residual Component

As established in our overview (Section 4.1), the residual sequence $\mathbf{R}_\beta$ contains the exact information needed to losslessly recover the original data. This section details the specialized encoding scheme we designed for these residuals and, crucially, develops a cost model to estimate their compressed size, $L(\mathbf{R}_\beta)$. This cost model is a key input for the global optimization of β.

4.3.1 The Hybrid Residual Encoder. The core rationale for handling residuals in the time domain is that different signal components are best compressed where their representation is most compact. While structural patterns are concentrated in the frequency domain, aperiodic disturbances are more efficiently encoded in the time domain, due to two primary reasons. First, for random noise, while its energy is distributed in both domains, the time-domain residual holds a precision advantage: residuals are simple integers, whereas frequency-domain errors are high-precision complex numbers. Second, and more critically, a localized time-domain anomaly (e.g., a sensor spike) causes **spectral leakage**, distributing its energy across a wide range of frequency coefficients [20]. Encoding one concentrated time-domain residual is thus more efficient than encoding numerous scattered frequency errors.

Partitioning Long-tailed Residual Values. This decomposition process naturally yields a residual sequence $\mathbf{R}_\beta$ with a long-tailed distribution, as visualized in Figure 6. The main body of the distribution, clustered around zero, results from the random noise component. The long tail, however, is formed by the large, sparse components resulting from values that are not well-captured by the frequency-domain model.

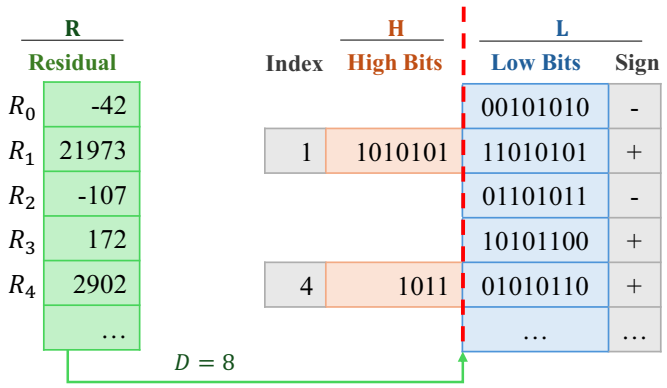


Fig. 7. Partition each residual into high-bits and low-bits, based on cut bit D, where the sparse high-bits need to store the corresponding index.

To tackle this specific distribution, we designed a **hybrid, two-part encoding scheme**. The core idea is to vertically partition each residual's bit representation based on a cut bit, D. This separates the residual sequence **R** into a dense, low-bit component **L** and a sparse, high-bit component **H**. Formally, for each residual R_k, we define:

$$L_k = \text{sign}(R_k) \cdot (|R_k| \bmod 2^D) \tag{11}$$

$$H_k = \lfloor |R_k| / 2^D \rfloor. \tag{12}$$

Hybrid Encoding. The two components are then encoded as illustrated in Figure 7:

(1) **Low-bit Encoding.** The low-bit component **L** is a dense sequence. Each value L_k is stored directly using $D + 1$ bits (1 for the sign) with bit-packing [28], as shown in the blue part of Figure 7. No index information is needed as it is a dense stream.

(2) **High-bit Encoding.** The high-bit component **H** is sparse, containing non-zero values only for the long tail. We encode this by storing the indices of the non-zero values in **H**, along with the values themselves, using a suitable sparse encoding algorithm like DBP [39].

4.3.2 Automatic Determination of Cut Bit D. For a given quantization level β, the optimal cut bit D^* for our hybrid encoder can be found by analyzing the true residual distribution. However, generating this true residual for every candidate β during the global search is computationally prohibitive, as it requires a costly inverse transform (IDFT). To make the global optimization tractable, we therefore need a way to estimate a near-optimal D and the corresponding residual cost, based solely on frequency-domain information.

Propagation of Frequency Quantization Error to Residual. The intuition behind our model stems from the conservation of energy, as Parseval's theorem (Equation 3) establishes a direct relationship between the energy in the time and frequency domains. For an original frequency coefficient F_j and parameter β, the complex-valued quantization error is

$$\epsilon_j = \text{Dequantize}(\text{Quantize}(F_j, \beta), \beta) - F_j.$$

The total energy of this error sequence, $E_{\text{freq}}(\beta) = \sum_{j=0}^{n-1} |\epsilon_j|^2$, can then be precisely calculated.

To investigate how this energy propagates to the time domain, we model the error sequence $\{\epsilon_j\}$ as a zero-mean, uncorrelated random process. This is a reasonable assumption for complex signals,

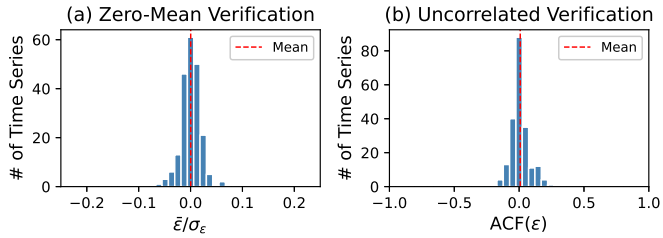


Fig. 8. Empirical verification of Proposition 1 assumptions on quantization error ϵ. (a) Zero-mean: normalized mean $\bar{\epsilon}/\sigma_\epsilon$ centered at 0. (b) Uncorrelated: ACF at lag 1 near 0.

as the pseudo-random phases of the quantization error tend to cause the cross-correlation terms to average to zero during the IDFT summation. Consequently, the residual can be modeled as follows.

Proposition 1 (Error Energy Equipartition). *Given a zero-mean, uncorrelated quantization error sequence $\{\epsilon_j\}$, the expected squared magnitude of any time-domain residual sample R_k is equal to the average error energy:*

$$\mathbb{E}[|R_k|^2] = \frac{1}{n^2}\sum_{j=0}^{n-1}|\epsilon_j|^2 = \frac{1}{n^2}E_{\text{freq}}(\beta). \tag{13}$$

Proof. The squared magnitude of $R_k = \frac{1}{n}\sum_{j=0}^{n-1}\epsilon_j\omega_n^{jk}$ is given by:

$$|R_k|^2 = \frac{1}{n^2}\left(\sum_{j=0}^{n-1}\epsilon_j\omega_n^{jk}\right)\left(\sum_{l=0}^{n-1}\epsilon_l^*\omega_n^{-lk}\right).$$

Taking the expectation and applying the uncorrelated error assumption, which implies $\mathbb{E}[\epsilon_j\epsilon_l^*] = 0$ for $j \neq l$ and $\mathbb{E}[|\epsilon_j|^2] = |\epsilon_j|^2$ (since ϵ_j is deterministic for a given input), all off-diagonal terms vanish:

$$\begin{aligned}\mathbb{E}[|R_k|^2] &= \frac{1}{n^2}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1}\mathbb{E}[\epsilon_j\epsilon_l^*]\omega_n^{jk}\omega_n^{-lk} \\ &= \frac{1}{n^2}\sum_{j=0}^{n-1}|\epsilon_j|^2\omega_n^{jk}\omega_n^{-jk} = \frac{1}{n^2}\sum_{j=0}^{n-1}|\epsilon_j|^2.\end{aligned}$$

□

To justify the assumptions in Proposition 1 through experiments, we augment the theoretical basis (Widrow's quantization theory [40]) with empirical verification in Figure 8. We calculate the actual quantization error mean (verifying zero-mean) and provide an autocorrelation plot (verifying uncorrelated) on representative datasets. The results confirm both assumptions: quantization errors are zero-mean and uncorrelated, both passing t-tests with p-values < 0.05.

Residual Encoding Cost. Proposition 1 provides a powerful tool to model the main body of the residual distribution, which is driven by uniform noise.

For the residual distribution with a mean close to zero, as illustrated in Figure 6, $\sqrt{\mathbb{E}[|R_k|^2]}$ serves as a good approximation of the standard deviation σ_R. Our heuristic is to estimate the optimal partition width D^* using this value

$$D = \lceil \log_2(\sigma_R) \rceil + 1. \tag{14}$$

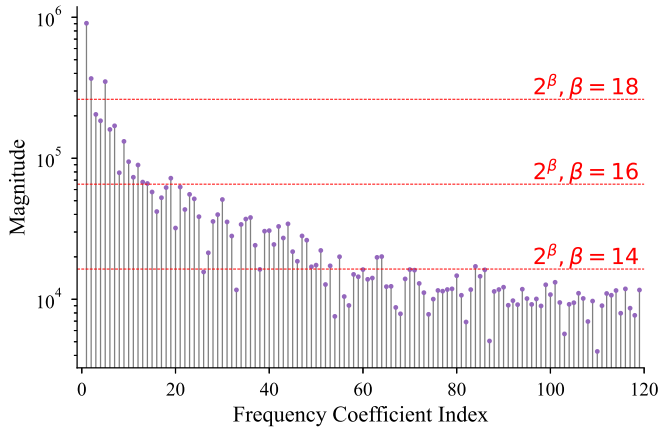


Fig. 9. Quantization of frequency component $\hat{\mathbf{F}}_\beta$ under various quantization level β, where β controls the number of retained coefficients (above the corresponding red lines).

The rationale for the term "+1" is that by allocating one extra bit, we double the value range to approximately $2\sigma_R$. For a Gaussian-like distribution, this range covers over 95% of all values.

This implies that the cost of the high-bit component, which only needs to encode the remaining long tail, can be considered negligible for the purpose of a fast cost estimation. Our final estimated cost, $\hat{L}(\mathbf{R}_\beta)$, is thus based solely on the cost of the low-bit part:

$$\hat{L}(\mathbf{R}_\beta) = n \times (\underbrace{D}_{\text{Low-Bit Width}} + \underbrace{1}_{\text{Sign}}). \tag{15}$$

4.4 FLEA Encoding

With the cost models for both components established, we can now detail their integration to solve the global optimization problem: finding the optimal quantization level, β^*, that minimizes their combined estimated cost.

4.4.1 Automatic Determination of Quantization Level β. The choice of β directly governs the properties of the resulting components and the subsequent encoding cost. As illustrated in Figure 9, the quantization level β effectively controls the number of non-zero coefficients retained in the frequency spectrum. A larger β corresponds to a coarser quantization step, which acts as a higher threshold, causing more low-magnitude coefficients to be quantized to zero. This results in a sparser frequency component $\hat{\mathbf{F}}_\beta$, which requires less bits to encode, but yields a less compressible $\mathbf{R}_\beta$. Conversely, a smaller β has the opposite effect. This inverse relationship, empirically shown in Figure 2, naturally forms a U-shaped total cost curve (as illustrated in Figure 14 below). FLEA's objective is to find the β^* at the minimum of this curve.

$$\beta^* = \arg\min_\beta \left(L(\hat{\mathbf{F}}_\beta) + \hat{L}(\mathbf{R}_\beta) \right). \tag{16}$$

The parameter β is discrete and has a very small search space, typically integers within $[0, 32]$. In this specific context, exhaustive search is efficient and guarantees finding the global optimum. In contrast, gradient descent is designed for continuous optimization and is ill-suited for this discrete problem, where the cost curve may be non-smooth due to quantization steps, posing a risk of getting stuck in local optima.

4.4.2 FLEA Algorithm. The complete, end-to-end FLEA encoding process is formalized in Algorithm 1. It operates in two distinct phases. First, in the Optimization Phase (lines 2-11), it performs the global search for the optimal parameters by iterating through candidate β values and utilizing the cost models. Second, in the Final Encoding Phase (lines 12-17), it uses these optimal parameters to perform a single, definitive encoding pass on the original data, assembling the final encoded data.

Algorithm 1: FLEA Frequency-based Lossless Encoding

Input : Time series $\mathbf{X}$ of length n for compression
Output : Encoded data

1 $F \leftarrow \texttt{DFT}(\mathbf{X})$
// Phase 1: Parameter Determination
2 $L_{\text{min\_total}} \leftarrow \infty$
3 $\beta^*, p^*_{\text{re}}, p^*_{\text{im}}, D^* \leftarrow$ null
4 **for** *each candidate* β **do**
5 $\quad \hat{F}_\beta \leftarrow \texttt{Quantize}(F, \beta)$
6 $\quad (L(\hat{\mathbf{F}}_\beta), p_{\text{re}}, p_{\text{im}}) \leftarrow \texttt{EstimateFrequencyCost}(\hat{\mathbf{F}}_\beta)$ // Equ. 9
7 $\quad (\hat{L}(\mathbf{R}_\beta), D) \leftarrow \texttt{EstimateResidualCost}(F, \beta)$ // Equ. 15
8 $\quad L_{\text{total}} \leftarrow L(\hat{\mathbf{F}}_\beta) + \hat{L}(\mathbf{R}_\beta)$
9 $\quad$ **if** $L_{\text{total}} < L_{\text{min\_total}}$ **then**
10 $\quad\quad L_{\text{min\_total}} \leftarrow L_{\text{total}}$
11 $\quad\quad (\beta^*, p^*_{\text{re}}, p^*_{\text{im}}, D^*) \leftarrow (\beta, p_{\text{re}}, p_{\text{im}}, D)$
// Phase 2: Data Encoding
12 $\hat{\mathbf{F}}^* \leftarrow \texttt{Quantize}(F, \beta^*)$
13 $\mathbf{R}^* \leftarrow \texttt{CalculateResidual}(\mathbf{X}, \hat{\mathbf{F}}^*)$
// Encode all components and assemble encoded data
14 $B_{\text{header}} \leftarrow \text{EncodeHeader}(\beta^*, p^*_{\text{re}}, p^*_{\text{im}}, D^*)$
15 $B_{\text{freq}} \leftarrow \texttt{EncodeLaminarFrequency}(\hat{\mathbf{F}}^*)$
16 $B_{\text{residual}} \leftarrow \texttt{EncodeHybridResidual}(\mathbf{R}^*)$
17 $EncodedData \leftarrow B_{\text{header}} \oplus B_{\text{freq}} \oplus B_{\text{residual}}$
18 **return** $EncodedData$

5 Experimental Evaluation

To comprehensively evaluate the performance of our proposed algorithm, FLEA, we conduct a series of extensive experiments. This section is organized as follows: We first detail the experimental setup, and then present a thorough comparison of FLEA against state-of-the-art encoders in terms of overall performance. Finally, we provide an in-depth analysis, first by validating the effectiveness of our automatic parameter determination, and then through a series of ablation studies on FLEA's core components.

5.1 Experimental Setup

All experiments are conducted on a server with an AMD EPYC 7542 32-Core Processor @ 2.90GHz, 256 GB RAM, running Ubuntu 22.04 LTS. Code and data are available for reproducibility [7].

Table 2. Overview of the real-world datasets

Dataset	Category	# Series	# Points	Source
Temperature	Periodic	321	2,906,327	[32]
Volt	Periodic	87	44,544	[37]
Grid	Periodic	4	40,430	[Industrial]
Safe	Periodic	12	75,374	[Industrial]
Mobile	Periodic	6	76,032	[Industrial]
Bank	Periodic	13	103,935	[Industrial]
Power	Aperiodic	208	2,075,259	[25]
GPS	Aperiodic	108	263,718	[13]
Stock	Aperiodic	33	195,005	[31]
Chemistry	Aperiodic	54	44,676	[Industrial]

5.1.1 Datasets. To ensure a comprehensive evaluation, our experiments are conducted on a diverse collection of 10 datasets, summarized in Table 2. These datasets were carefully selected to represent a wide range of real-world application domains—including public utilities, finance, industrial sensing, and life sciences—and cover both **periodic** and **aperiodic** time series.

The periodic datasets include: `Temperature`, which contains daily average temperatures from major cities [32]; `Volt`, comprising voltage signals from a high-voltage transmission line [37]; and four industrial datasets, `Grid`, `Safe`, `Mobile`, and `Bank`, representing power grid monitoring, insurance, telecommunication, and banking sectors, respectively.

The aperiodic datasets, which may contain other forms of patterns like trends, include: `Power`, a record of household electric power consumption [25]; `GPS`, containing trajectories of seabirds [13]; `Stock`, which tracks historical stock prices of DJIA companies [31]; and `Chemistry`, a dataset of measurements from an industrial chemical process.

5.1.2 Baselines. We compare FLEA against a comprehensive set of state-of-the-art and classic time series encoding algorithms: **Sprintz** [5], **Chimp** [18], **Gorilla** [24], **BUFF** [19], and **HIRE** [3]. A detailed description of these encoding methods is provided in Section 2.1.

To ensure a fair and representative throughput comparison, we compare our C++ native FLEA with the original baseline implementations. Specifically, Gorilla (proprietary) uses the production Java implementation from Apache TsFile; BUFF uses the original Rust implementation; HIRE uses the original Python implementation (its repository is indeed in Python); Sprintz's the original C++ implementation only supports 8/16-bit integers and is adapted for 64-bit; and Chimp uses the original Java implementation.

In addition, we include two general-purpose compression baselines, **LZ4** [4] and **ZSTD** [8], to evaluate the performance of general compression methods on time series datasets. LZ4 and ZSTD use the official C library bindings.

The original values in all datasets are floating-point numbers. To facilitate a fair comparison within our lossless encoding approach, all datasets underwent a consistent pre-processing step for encoding algorithms. This is a standard practice when applying integer-based compressors to floating-point data [19]. Specifically, each floating-point value v_f was converted to an integer v_i via an exact scaling transformation, $v_i = \text{round}(v_f \times 10^k)$, where k matched the data's native decimal precision. The resulting integer sequences served as input for FLEA and other encoding algorithms.

Table 3. Compression ratio comparison on all datasets. Higher values are better. For each dataset, the best-performing method is highlighted in **bold** and the second-best is underlined. The last columns show the average performance over periodic, aperiodic, and all datasets, respectively.

	Periodic Datasets							Aperiodic Datasets					
Method	Temp.	Volt	Grid	Safe	Mobile	Bank	**Avg.**	Power	GPS	Stock	Chem.	**Avg.**	**Overall**
FLEA	**6.71**	**5.60**	**7.60**	**6.85**	**8.27**	**4.97**	**6.57**	5.56	4.64	**5.85**	**6.42**	**5.58**	**6.15**
Sprintz	6.07	4.41	7.00	6.01	7.46	4.61	5.82	5.30	4.67	5.84	5.79	5.38	5.64
Chimp	5.81	3.67	5.64	5.87	5.73	4.56	5.14	**5.88**	**4.95**	5.43	5.42	5.41	5.25
BUFF	4.92	3.31	4.66	5.44	5.38	3.64	4.48	4.51	3.40	4.11	5.49	4.31	4.41
Gorilla	2.08	2.40	4.94	5.58	7.99	4.00	4.04	5.83	4.09	4.23	4.80	4.69	4.29
HIRE	1.50	1.91	1.63	1.47	1.54	1.94	1.65	1.62	1.48	1.48	1.56	1.54	1.61
LZ4	2.43	1.03	2.22	2.73	3.30	2.04	2.16	2.69	1.56	1.93	1.91	1.98	2.09
ZSTD	4.05	1.12	3.84	4.78	7.80	3.04	3.54	4.47	1.87	2.83	2.66	2.82	3.23

5.1.3 Metrics. Algorithm performance is evaluated using three standard metrics: Compression Ratio (CR), Encoding Throughput (ET), and Decoding Throughput (DT). Let S_{orig} and S_{comp} be the original and compressed data sizes in bits, respectively, and T_{enc} and T_{dec} be the encoding and decoding times in seconds. CR, indicating data size reduction (higher is better), is $S_{\text{orig}}/S_{\text{comp}}$. ET and DT measure compression and decompression speed in mebibytes per second (MiB/s, higher is better), calculated as $(S_{\text{orig}}/(8 \times 1024^2))/T_{\text{enc}}$ and $(S_{\text{orig}}/(8 \times 1024^2))/T_{\text{dec}}$, respectively.

5.2 Overall Performance Comparison

In this section, we present the main experimental results, comparing the performance of FLEA against the baseline methods on all ten datasets. We begin with a detailed analysis of the compression ratio, followed by an evaluation of the encoding and decoding throughput.

5.2.1 Compression Ratio. The comprehensive compression ratio results for all methods and datasets are presented in Table 3. For a high-level summary, Figure 10 visualizes the average performance across the two main data categories. Together, these results unequivocally demonstrate the superior performance of FLEA, which achieves the highest compression ratio on 8 out of the 10 individual datasets and leads in all three average performance categories.

Performance on Periodic Datasets. FLEA's strength in its target domain of periodic time series is visually striking in Figure 10. Quantitatively, Table 3 confirms this dominance: FLEA achieves an average compression ratio of **6.57**, a remarkable **12.9%** improvement over the next best method, Sprintz (5.82). Interestingly, Sprintz also shows strong performance on these datasets. This is attributed to its delta-based scheme, which effectively capitalizes on the high local predictability inherent in smooth periodic signals. However, FLEA's frequency-domain approach allows it to capture the *global* periodic structure more systematically, leading to its superior overall performance on these datasets. For instance, on the `Grid` and `Mobile` datasets, FLEA achieves top-tier compression ratios of 7.60 and 8.27 respectively, showcasing its capability on real-world industrial data with strong periodic patterns.

Performance on Aperiodic Datasets. Figure 10 also reveals FLEA's robust generalization capabilities. It achieves the highest average compression ratio of **5.58** on aperiodic datasets, slightly ahead of Chimp (5.41). This strong performance can be attributed to FLEA's ability to capture broader structural patterns beyond pure periodicity. A closer look at Table 3 reveals interesting nuances. For example, FLEA excels on datasets like `Stock` and `Chemistry`, where underlying, non-periodic

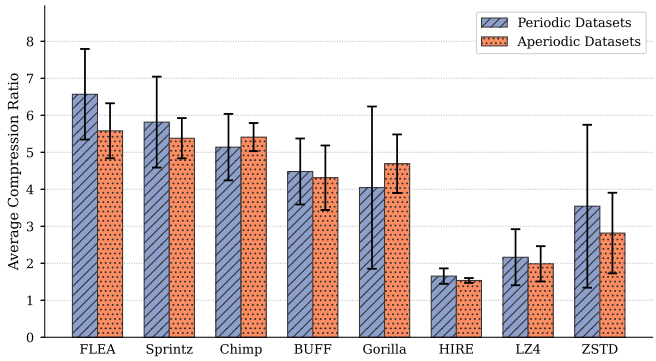


Fig. 10. Average compression ratio on periodic and aperiodic datasets. FLEA demonstrates a significant advantage on periodic data, for which it is designed, while also achieving high average performance on aperiodic data.

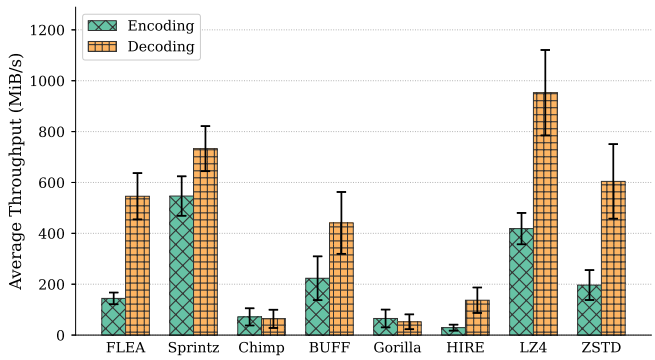


Fig. 11. Average encoding and decoding throughput. FLEA exhibits an encoding-decoding asymmetry, trading encoding speed for high compression and offering competitive decoding speed, favorable to write-once-read-many scenarios.

structural patterns such as market cycles or operational phases can be effectively captured in the frequency domain. In contrast, Chimp performs best on `Power` and `GPS`, likely because its wider search window is highly effective at finding local, non-periodic value repetitions, which a global frequency analysis might average out.

5.2.2 Overall Performance on Compression Ratio. Following best practices for reporting ratios [10], we use geometric mean for averaging compression ratios. Considering all datasets, the "Overall Average" column in Table 3 provides a clear final verdict. With an overall average compression ratio of **6.15**, FLEA establishes itself as the new state-of-the-art among the tested methods. The combination of its dominance on periodic data and its highly competitive performance on aperiodic data, as evidenced by both the aggregate view in Figure 10 and the detailed results in Table 3, confirms that FLEA is a powerful and versatile encoding algorithm for a wide range of time series.

As shown in Table 3, LZ4 achieves worse compression ratio than specialized time-series encoders, confirming that general entropy encoders are ineffective for time series. Similarly, ZSTD achieves

worse compression ratio than specialized time-series encoders, confirming the "lack of exact repeats" limitation in Section 2.1.

5.2.3 Compression Throughput. The native C++ FLEA achieves the state-of-the-art compression ratio with competitive throughput outperforming several baselines like Gorilla, demonstrating practical viability. Figure 11 presents the average encoding and decoding throughput for FLEA and the original baseline methods.

The results demonstrate that C++ FLEA achieves its state-of-the-art compression ratio while offering competitive throughput. Its encoding throughput reflects the computational investment of its frequency-domain transform and multi-stage optimization. More notably, its decoding throughput is highly efficient, outperforming several baselines like Gorilla, BUFF and Chimp.

This performance profile positions FLEA as offering a compelling balance between compression effectiveness and speed. While algorithms like Sprintz and BUFF provide higher encoding throughput, FLEA delivers a better compression ratio across all datasets (as shown in Section 5.2.1). This makes it a practical choice for users who prioritize storage savings while still requiring reasonable encoding speed and fast data retrieval.

A key characteristic of FLEA's design is its pronounced encoding-decoding asymmetry, with the decoding speed being far more than the encoding speed. This efficiency stems from its architecture: the computationally intensive search for optimal parameters (β^*, p^*, and D^*) is performed only once during the encoding phase. In contrast, the decoder operates as a streamlined, non-iterative process. It directly utilizes the pre-optimized parameters from the metadata, bypassing the expensive search and requiring only a single inverse DFT to reconstruct the data.

In summary, FLEA establishes itself not only as a leader in compression effectiveness but also as a powerful and practical algorithm with competitive computational performance. Its highly efficient decoding makes it particularly well-suited for a wide range of read-intensive workloads, such as analytical queries in databases, where fast data access is essential. It is especially advantageous in, but not limited to, "write-once, read-many" (WORM) scenarios [9].

To gauge the significance of 10% improvement, we quantify the trade-off between higher encoding time and lower compressed storage, for less total costs in the cloud. In short, the one-time computation of compression is worthwhile for the reduced persistent storage in the cloud. For 1 TB raw data with AWS pricing [1] (CPU: \$0.0116/hour, storage: \$0.03/GB/month), FLEA's slower encoding (144 MiB/s vs. Sprintz's 547 MiB/s) incurs an extra one-time CPU cost of \$0.017. However, FLEA's higher compression ratio (6.15× vs. 5.64×) saves 15 GB, yielding \$0.44/month in storage reduction. The break-even point is reached within 2 days, after which FLEA's storage savings outweigh the extra encoding cost.

5.3 Automatic Parameter Determination

The performance of the FLEA algorithm is highly dependent on its three core parameters: the frequency split index p, the residual cut bit D, and the quantization level β. As optimal values for these parameters vary significantly across different datasets, a mechanism for automatic, data-adaptive determination is crucial. This section is dedicated to experimentally validating the necessity and effectiveness of FLEA's dynamic optimization strategy by first dissecting the underlying cost trade-offs on representative examples, and then demonstrating the superior performance of our dynamic approach against fixed-parameter strategies across all datasets.

5.3.1 Frequency Split Index p. The adaptive partitioning of the frequency spectrum, controlled by the split index p, is a key optimization in FLEA. To validate the effectiveness of this strategy, formalized in Section 4.2.4 (Equation 9), we first analyze the cost trade-off on a sample series.

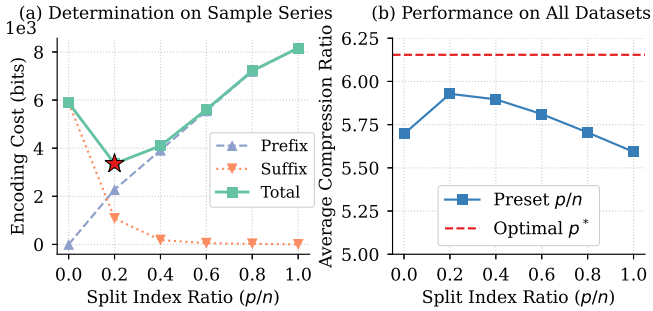


Fig. 12. Auto-determination of the parameter p for split index. (a) FLEA determines the optimal p^* with the minimum total cost in a sample series. (b) The auto-determined optimal p^* for each dataset shows better compression ratio than any preset p/n across all datasets.

Figure 12(a) explains the partitioning logic via a `Safe` dataset sample. The total cost (green), as the sum of L_{Laminar} (blue) and L_{Sparse} (orange), forms a clear valley that defines the optimal split p^*. This justifies the necessity of an optimization search.

Figure 12(b) shows that dynamic p^* optimization outperforms various fixed p/n strategies. The latter follow an inverted U-shape, where performance is highly sensitive to the chosen ratio. FLEA (red dashed line) surpasses all fixed presets, demonstrating that automatic, per-sequence tuning is superior to any constant parameter setting.

5.3.2 Residual Cut Bit D. Another key parameter in our method is the cut bit D used in the hybrid residual encoder. This section experimentally validates our strategy for dynamically determining the optimal D^* in Equation 14, a process detailed in Section 4.3.2. We demonstrate that finding this optimal value is crucial for effectively compressing the long-tailed residual distribution, as shown in Figure 13.

Figure 13(a) illustrates the cost trade-off using a `GPS` dataset sample. As D increases, the low-bit cost (blue) grows linearly while the high-bit cost (orange) decreases sharply. Their sum (green) forms a convex curve with a clear minimum at $D^* = 10$, demonstrating an optimal balance exists.

Figure 13(b) proves that dynamically finding this minimum is superior to any preset strategy. The performance of fixed D values exhibits a clear peak, confirming high sensitivity to this parameter; extreme choices (e.g., $D = 6$ or 16) lead to significant efficiency drops. Even the best fixed preset is outperformed by our dynamic optimization by over **19%**, highlighting the importance of adapting D^* to each sequence's specific distribution.

5.3.3 Frequency Quantization Level β. The global optimization of the quantization level β is the cornerstone of FLEA's rate-optimal design. This section validates this mechanism by analyzing its impact on performance, thereby empirically justifying the cost models developed in Equation 16, Section 4.4.

Figure 14(a) reveals the underlying reason for our global search. On a sample series from the `Mobile` dataset, we observe the direct trade-off between the component costs. As predicted by our models in Section 4.2, the cost of the frequency component decreases as β increases (coarser quantization), while the cost of the residual component increases. The total cost curve exhibits a U-shaped trend with a distinct minimum, as we illustrated in Section 4.4.1.

Figure 14(b) then provides the macro-level proof of effectiveness. It compares the average compression ratio of FLEA's dynamic β^* optimization against several preset β strategies. The

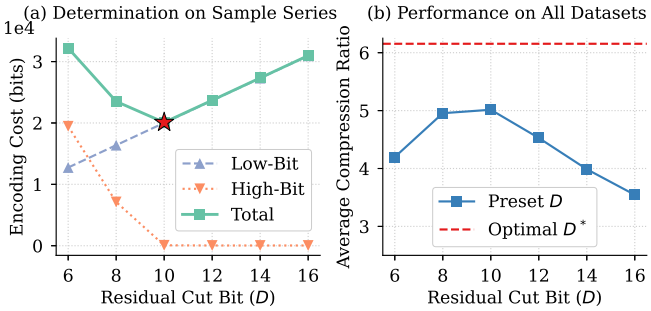


Fig. 13. Auto-determination of the residual cut bit D. (a) FLEA determines the optimal D^* with the minimum total cost in a sample series. (b) The auto-determined optimal D^* for each dataset shows better compression ratio than any preset D across all datasets.

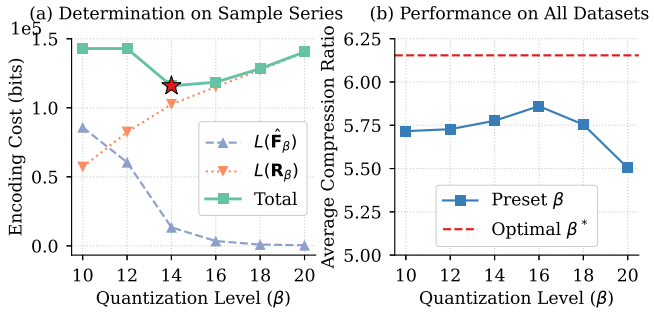


Fig. 14. Auto-determination of the quantization level β. (a) FLEA determines the optimal β^* with the minimum total cost in a sample series. (b) The auto-determined optimal β^* for each dataset shows better compression ratio than any preset β across all datasets.

results clearly demonstrate that our dynamic optimization strategy (red dashed line) consistently surpasses all preset alternatives. Even the best-performing fixed choice, $\beta = 16$, yields a significantly lower average compression ratio, quantitatively demonstrating the necessity of adapting β to the specific characteristics of each dataset.

5.4 Ablation Studies

To provide a clear, quantitative validation of our primary algorithmic contributions, we conducted a direct ablation study by replacing our core encoders with simpler baselines. The results in Table 4 provide clear evidence that both of our proposed encoding mechanisms are essential, non-trivial components contributing to FLEA's state-of-the-art performance.

As shown in Table 4, we first tested our core frequency encoding strategy. Replacing our entire partitioning approach with a DBP encoder [39] applied to the whole spectrum resulted in a significant performance drop of **5.9%**. This result confirms the discussion in Section 4.2.1: while a sparse encoder like DBP is effective for the sparse tail, it is inefficient for the dense, skewed low-frequency coefficients, validating the necessity of our bi-regional approach.

Even more critically, when our specialized hybrid residual encoder (Section 4.3.1) was replaced with a simple bit-packing implementation [28], the average compression ratio fell by a substantial

Table 4. Ablation study of FLEA's core encoding components. The performance impact is measured by the drop in average compression ratio across all datasets.

Configuration	Compression Ratio	Drop
Full FLEA in Section 4.4	**6.15**	–
Replace Frequency Encoder in Section 4.2		
Laminar Encoding → DBP	5.79	-5.9%
Replace Residual Encoder in Section 4.3		
Hybrid Encoding → Bit Packing	5.65	-8.2%

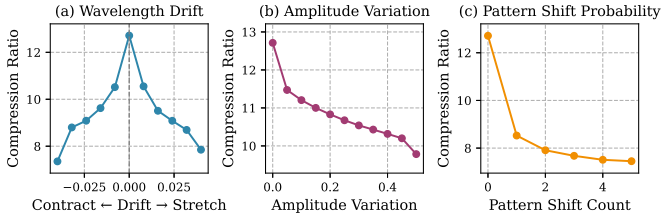


Fig. 15. Periodicity stress tests. (a) Wavelength stretching/contracting. (b) Amplitude varying. (c) Pattern shifts.

8.2%. This stark difference confirms that our tailored, two-part strategy is highly effective for the long-tailed nature of the residual distribution, as illustrated in Figure 6.

5.5 Periodicity Stress Tests

We acknowledge that FLEA is not always the best encoding for aperiodic datasets, where Sprintz or Chimp may offer a better trade-off. DFT is indeed best suited for signals with clear and stable periodicity. To evaluate the situation when periodic behavior drifts or the pattern shifts, we generate synthetic datasets with wavelength stretching or contracting, amplitude varying and pattern shifts. In such settings, the residuals would become too large to compress effectively. The result in Figure 15 confirms that weaker periodicity indeed leads to worse compression ratio. Period stretching/contracting, amplitude decay, and abrupt pattern changes all cause degradation of FLEA's compression performance. In practice, it may be difficult to have a threshold of periodicity for determining when DFT is a suitable choice. Indeed, computing periodicity itself takes considerable time cost. Hence, a simple end-to-end choice is to directly observe whether FLEA has better compression ratio than baselines (on sample data).

5.6 Coefficient Order Analysis

For the concern regarding sparse high valued frequencies that may lead to much more bits for lower coefficients, we conduct experiments to measure how often this occurs in the datasets. As shown in Figure 16(a), sparse high valued frequencies that may lead to much more bits for lower coefficients rarely occurs and causes only 2.4% waste on bits. We also evaluate how FLEA adapts to these irregularities and report the impact on the final compression ratio. As presented in Figure 16(b), compared to the case without such irregularity, i.e., waste ratio 0, the encoding cost of Laminar is impacted slightly for the real world datasets with 2.4% waste ratio.

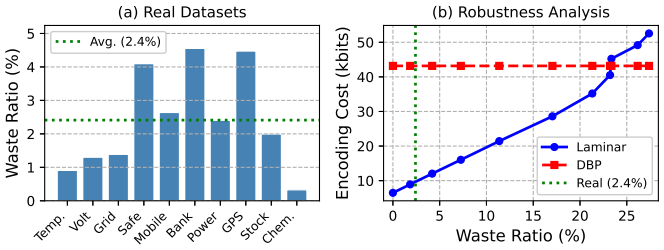


Fig. 16. Coefficient order analysis. (a) Waste ratio across datasets. (b) Laminar vs. DBP encoding cost as waste ratio increases. Real data (2.4%) is well within Laminar's advantage region.

Nevertheless, if the frequency coefficients are extremely zigzag, FLEA can use DBP, compared in Table 4 of ablation study, to mitigate arbitrarily bad cases by reordering high valued frequencies. As shown in Figure 16, since such bit waste is low (2.4%) in real-world data, the encoding cost of Laminar is much better than DBP. That is, DBP would not be chosen in practice.

6 Deployment in Apache TsFile

Apache TsFile [33, 42] is an open-source file format for IoT database. In this section, we describe how the FLEA method is deployed and made available as open source. Our implementation is integrated into the Apache TsFile system repository [34] maintained by system developers, with corresponding documentation also available [35].

6.1 Native Encoder Integration

We integrated FLEA as a native encoder in Apache TsFile by registering `FLEA` in the system's `TSEncoding` enumeration and implementing the corresponding `FleaEncoder` and `FleaDecoder` classes. Apache TsFile is used by Apache IoTDB [15, 38], a leading time series database, as its underlying storage file format. This allows users to specify the encoding method via the standard command:

```
CREATE TIMESERIES root.sg.d1.s1 WITH DATATYPE=INT64, ENCODING=FLEA;
```

This deep integration ensures seamless interoperability with Apache TsFile data pipelines. FLEA's block-oriented design aligns well with the TsFile's page-based storage architecture [42]. On a per-page basis, the `FLEAEncoder` finds the optimal parameters $(\beta^*, p^*_{\text{re}}, p^*_{\text{im}}, D^*)$ and serializes them into a compact header prepended to the compressed data. This enables the `FleaDecoder` to perform efficient, one-pass reconstruction.

6.2 End-to-End Throughput Evaluation

Considering that the language differences of the original implementations may also affect fairness, we further compare against the third-party, production-level implementations in the same language, provided by Apache TsFile. That is, we compare FLEA against Sprintz, Chimp, and Gorilla, in a comparable language (Java).

Hence, this section evaluates the Java system integration, distinct from the implementations in Section 5.2. We conduct end-to-end experiments measuring the throughput of writing data to a TsFile and reading it back. The results are presented in Figure 17.

The evaluation in Figure 17 validates FLEA's performance characteristics in a production-grade system. FLEA achieves an average encoding throughput of 15 MiB/s and decoding throughput of

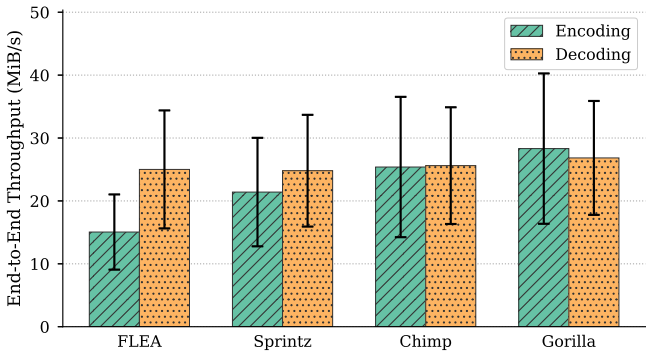


Fig. 17. End-to-end write and read throughput in Apache TsFile. All encoders are implemented in Java, ensuring a fair same-language comparison. FLEA achieves competitive throughput while delivering the highest compression ratio.

25 MiB/s, competitive with other methods, especially in decoding. Combined with FLEA's highest compression ratio, these results demonstrate that its advantages observed in native benchmarks translate to real-world deployment scenarios. The read throughput being consistently higher than write throughput confirms FLEA's asymmetric design advantage.

7 Conclusion

In this paper, we address the challenge of efficiently compressing time series with strong periodic components. We proposed **FLEA**, a novel frequency-based lossless encoding algorithm. To balance the encoding costs of frequency domain coefficients and time domain residual, Proposition 1 studies the propagation of the energy of frequency-domain quantization error to the expected cost of the time-domain residual. FLEA's effectiveness stems from its adaptive encoding pipeline, which features a multi-stage, **automatic parameter determination** process. This includes: (1) a horizontal partitioning of the frequency spectrum at an optimal point p, managed by our novel **Laminar Encoding** scheme; (2) a vertical partitioning of the residual's bit-width at an optimal cut bit D, handled by a specialized **Hybrid Encoder**; and (3) a global search for the optimal quantization level β. Our experiments demonstrate that FLEA achieves an average compression ratio of 6.57× on periodic data, outperforming the second best by 12.9%. The native C++ implementation delivers 144 MiB/s encoding and 546 MiB/s decoding throughput. FLEA is a new point in the compression-speed trade-off space. Rather than dominating existing designs, we acknowledge the performance cost in order to achieve higher compression. The trade-off is valuable especially in the cloud, where the increase of one-time computation gets in return the reduction of persistent storage.

Acknowledgments

This work is supported in part by the National Science and Technology Major Project (2025ZD1601701), the National Key Research and Development Plan (2024YFB3311901, 2021YFB3300500), the National Natural Science Foundation of China (62232005, 92267203, 62021002), State Grid Ningxia Electric Power Co. Science and Technology Project (SGNXYX00SCJS2400058), Beijing National Research Center for Information Science and Technology (BNR2025RC01011), and Beijing Key Laboratory of Industrial Big Data System and Application. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

References

[1] Shikha Anirban, Junhu Wang, and Md. Saiful Islam. 2019. Modular Decomposition-Based Graph Compression for Fast Reachability Detection. *Data Sci. Eng.* 4, 3 (2019), 193–207. doi:10.1007/S41019-019-00099-9

[2] Giuliano Antoniol and Paolo Tonella. 1997. EEG data compression techniques. *IEEE Trans. Biomed. Eng.* 44, 2 (1997), 105–114. doi:10.1109/10.552239

[3] Bruno Barbarioli, Gabriel Mersy, Stavros Sintos, and Sanjay Krishnan. 2023. Hierarchical Residual Encoding for Multiresolution Time Series Compression. *Proc. ACM Manag. Data* 1, 1 (2023), 99:1–99:26. doi:10.1145/3588953

[4] Matej Bartik, Sven Ubik, and Pavel Kubalík. 2015. LZ4 compression algorithm on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2015, Cairo, Egypt, December 6-9, 2015*. IEEE, 179–182. doi:10.1109/ICECS.2015.7440278

[5] Davis W. Blalock, Samuel Madden, and John V. Guttag. 2018. Sprintz: Time Series Compression for the Internet of Things. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3 (2018), 93:1–93:23. doi:10.1145/3264903

[6] Zijie Chen, Shaoxu Song, and Jianmin Wang. 2025. OneRoundSTL: In-Database Seasonal-Trend Decomposition . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 724–736. doi:10.1109/ICDE65448.2025.00060

[7] Experiment Code and Data. 2026. https://github.com/Mr-Spade/FLEA.

[8] Yann Collet and Murray S. Kucherawy. 2021. Zstandard Compression and the 'application/zstd' Media Type. *RFC* 8878 (2021), 1–45. doi:10.17487/RFC8878

[9] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 215–226. doi:10.1145/2882903.2903741

[10] Philip J. Fleming and John J. Wallace. 1986. How Not To Lie With Statistics: The Correct Way To Summarize Benchmark Results. *Commun. ACM* 29, 3 (1986), 218–221. doi:10.1145/5666.5673

[11] Philip Hans Franses and Richard Paap. 2004. *Periodic time series models*. OUP Oxford.

[12] Solomon W. Golomb. 1966. Run-length encodings (Corresp.). *IEEE Trans. Inf. Theory* 12, 3 (1966), 399–401. doi:10.1109/TIT.1966.1053907

[13] Dataset Gps. 2025. https://www.kaggle.com/datasets/saurabhshahane/predicting-animal-behavior-using-gps.

[14] GZip. 2025. https://www.gnu.org/software/gzip/.

[15] Apache IoTDB. 2025. https://iotdb.apache.org/.

[16] Sateh M. S. Jalaleddine, Chriswell G. Hutchens, Robert D. Strattan, and William A. Coberly. 1990. ECG data compression techniques-a unified approach. *IEEE Trans. Biomed. Eng.* 37, 4 (1990), 329–343. doi:10.1109/10.52340

[17] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. 1994. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Trans. Netw.* 2, 1 (1994), 1–15. doi:10.1109/90.282603

[18] Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *Proc. VLDB Endow.* 15, 11 (2022), 3058–3070. https://www.vldb.org/pvldb/vol15/p3058-liakos.pdf

[19] Chunwei Liu, Hao Jiang, John Paparrizos, and Aaron J. Elmore. 2021. Decomposed Bounded Floats for Fast Compression and Queries. *Proc. VLDB Endow.* 14, 11 (2021), 2586–2598. doi:10.14778/3476249.3476305

[20] Douglas Lyon. 2009. The Discrete Fourier Transform, Part 4: Spectral Leakage. *J. Object Technol.* 8, 7 (2009), 23–34. doi:10.5381/JOT.2009.8.7.C2

[21] Joel Max. 1960. Quantizing for minimum distortion. *IRE Trans. Inf. Theory* 6, 1 (1960), 7–12. doi:10.1109/TIT.1960.1057548

[22] Yousef W. Nijim, Samuel D. Stearns, and Wasfy B. Mikhael. 1996. Lossless compression of seismic signals using differentiation. *IEEE Trans. Geosci. Remote. Sens.* 34, 1 (1996), 52–56. doi:10.1109/36.481892

[23] Marc-Antoine Parseval. 1806. Mémoire sur les séries et sur l'intégration complète d'une équation aux différences partielles linéaires du second ordre, à coefficients constants. *Mém. prés. par divers savants, Acad. des Sciences, Paris,(1)* 1, 638-648 (1806), 42.

[24] Tuomas Pelkonen, Scott Franklin, Paul Cavallaro, Qi Huang, Justin Meza, Justin Teller, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *Proc. VLDB Endow.* 8, 12 (2015), 1816–1827. doi:10.14778/2824032.2824078

[25] Dataset Power. 2025. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

[26] Feifan Pu and Jianqiu Xu. 2025. TSQ: An Optimized Framework for Efficiently Answering Time Series Queries. *Data Sci. Eng.* 10, 2 (2025), 296–313. doi:10.1007/S41019-024-00273-8

[27] Daniel L. Ruderman and William Bialek. 1993. Statistics of Natural Images: Scaling in the Woods. In *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, Jack D. Cowan, Gerald

Tesauro, and Joshua Alspector (Eds.). Morgan Kaufmann, 551–558. http://papers.nips.cc/paper/835-statistics-of-natural-images-scaling-in-the-woods
[28] David Salomon. 2007. *Data compression - The Complete Reference, 4th Edition.* Springer.
[29] Horst Samulowitz, Chandra Reddy, Ashish Sabharwal, and Meinolf Sellmann. 2013. Snappy: A Simple Algorithm Portfolio. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7962)*, Matti Järvisalo and Allen Van Gelder (Eds.). Springer, 422–428. doi:10.1007/978-3-642-39071-5_33
[30] Khalid Sayood. 2017. *Introduction to data compression.* Morgan Kaufmann.
[31] Dataset Stock. 2025. https://www.kaggle.com/datasets/szrlee/stock-time-series-20050101-to-20171231/data.
[32] Dataset Temperature. 2025. https://www.kaggle.com/datasets/sudalairajkumar/daily-temperature-of-major-cities.
[33] Apache TsFile. 2025. https://tsfile.apache.org/.
[34] Apache TsFile. 2025. Code of FLEA Encoding. https://github.com/Mr-Spade/tsfile/tree/research/encoding-periodic.
[35] Apache TsFile. 2025. Document of FLEA Encoding. https://github.com/Mr-Spade/tsfile/blob/research/encoding-periodic/README.md.
[36] Donald L. Turcotte. 2009. Fractals in Geology and Geophysics. In *Encyclopedia of Complexity and Systems Science*, Robert A. Meyers (Ed.). Springer, 3822–3826. doi:10.1007/978-0-387-30440-3_224
[37] Dataset Volt. 2025. https://www.kaggle.com/datasets/neuralsorcerer/voltage-signal-dataset.
[38] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. ACM Manag. Data* 1, 2 (2023), 195:1–195:27. doi:10.1145/3589775
[39] Haoyu Wang and Shaoxu Song. 2022. Frequency Domain Data Encoding in Apache IoTDB. *Proc. VLDB Endow.* 16, 2 (2022), 282–290. doi:10.14778/3565816.3565829
[40] Bernard Widrow and István Kollár. 2008. *Quantization noise: roundoff error in digital computation, signal processing, control, and communications.* Cambridge University Press.
[41] Jinzhao Xiao, Yuxiang Huang, Changyu Hu, Shaoxu Song, Xiangdong Huang, and Jianmin Wang. 2022. Time Series Data Encoding for Efficient Storage: A Comparative Analysis in Apache IoTDB. *Proc. VLDB Endow.* 15, 10 (2022), 2148–2160. doi:10.14778/3547305.3547319
[42] Xin Zhao, Jialin Qiao, Xiangdong Huang, Chen Wang, Shaoxu Song, and Jianmin Wang. 2024. Apache TsFile: An IoT-native Time Series File Format. *Proc. VLDB Endow.* 17, 12 (2024), 4064–4076. doi:10.14778/3685800.3685827
[43] Mu-Ran Zhu, Jin-Duo Liu, and Junzhong Ji. 2025. Electrocardiogram Signal Classification Based on Bidirectional LSTM and Multi-Task Temporal Attention. *J. Comput. Sci. Technol.* 40, 5 (2025), 1401–1413. doi:10.1007/S11390-025-4330-6
[44] Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* 23, 3 (1977), 337–343. doi:10.1109/TIT.1977.1055714

Received October 2025; revised January 2026; accepted February 2026