CAMEO: Autocorrelation-Preserving Line Simplification for Lossy Time Series Compression

Carlos Enrique Muñiz-Cuza
muniz.cuza@tu-berlin.de
TU Berlin, Germany

Matthias Boehm
matthias.boehm@tu-berlin.de
TU Berlin, Germany

Torben Bach Pedersen
tbp@cs.aau.dk
Aalborg University, Denmark

Abstract

Time series data from a variety of sensors and IoT devices need effective compression to reduce storage and I/O bandwidth requirements. While most time series databases and systems rely on lossless compression, lossy techniques offer even greater space-saving with a small loss in precision. However, the unknown impact on downstream analytics requires a semi-manual trial-and-error exploration. We initiate work on lossy compression that provides guarantees on complex statistical features (which are strongly correlated with the accuracy of downstream analytics). Specifically, we propose CAMEO, a new lossy compression method that provides guarantees on the autocorrelation and partial-autocorrelation functions (ACF/PACF) of a time series. Our method leverages line simplification techniques as well as incremental maintenance, blocking, and parallelization strategies for effective and efficient compression. The results show that our method improves compression ratios by 2x on average and up to 54x on selected datasets, compared to previous lossy and lossless compression methods. Moreover, we maintain, and sometimes even improve, the forecasting accuracy by preserving the autocorrelation properties of the time series.

Keywords

Lossy Time Series Compression, Autocorrelation Function, Time Series Forecasting Analytics, Anomaly Detection

1 Introduction

High-frequency time series are everywhere, from industrial manufacturing to weather prediction. For instance, an offshore oil rig typically has 30,000 sensors, of which only a few are utilized for real-time control and anomaly detection [72]. Time series compression can significantly reduce storage space, I/O bandwidth (storage or network) and analysis requirements [23, 51–53, 110]. Motivated by these benefits, numerous algorithms have been proposed for lossless [12, 67, 82, 83, 105] and lossy [6, 15, 29, 58, 73] compression. While lossless methods preserve the raw data, lossy methods offer an appealing trade-off: more effective compression with only a small, typically bounded reconstruction error.

Lossy Compression Problem: In order to reduce the impact on downstream applications, lossy compression often minimizes the reconstruction error. These methods focus on maximizing compression ratios, bounded to a maximum distortion of time series values [15, 32, 46, 59, 71, 93]. Common techniques include domain transformation (Fourier Transform) [2, 21, 55], functional approximation (Polynomial Approximation) [11, 29, 32, 56], and symbolic representation (Dictionary Encoding) [61, 69, 70, 70, 73, 88]. Reconstruction quality is typically measured by the Normalized Root Means Square Error (NRMSE) or the Peak Signal-to-Noise Ratio (PSNR) [92]. However, the impact on downstream time series analytics remains largely unclear, requiring a tedious, semi-manual trial-and-error exploration [14, 29, 47, 74, 76, 79].

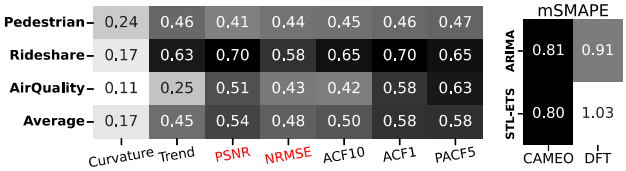


Figure 1: (left) **Pearson Correlation of Forecasting Errors and Different Statistical Features.** (right) **Average Forecasting Accuracy using Discrete Fourier Transform (DFT) and our CAMEO method at a Fixed Compression Ratio.**

A Case for Autocorrelation Preservation: Minimizing reconstruction errors like NRMSE or PSNR treats data points independently, ignoring the temporal dependencies essential for forecasting and anomaly detection. In contrast, complex statistics like autocorrelation (ACF) and partial autocorrelation (PACF) capture these dependencies. For stationary series, the ACF fully characterizes the second-order structure (i.e., all pairwise covariances), critical for many forecasting methods [8]. Preserving the ACF also retains the signal's spectral characteristics [102], capturing both dominant and subtle temporal patterns including periodicity. As the PACF also determines lag selection in ARIMA models [8], preserving the ACF/PACF ensures the retention of features essential for accurate forecasting.

Empirical Validation: For validating this intuition, we ran experiments on three datasets: Pedestrian, Rideshare, and AirQuality [42], comprising 2,831 time series. First, we compressed each series at varying compression ratios using the Discrete Fourier Transform (DFT) [21], measuring impacts on both reconstruction metrics (NRMSE, PSNR) and downstream forecasting accuracy (modified sMAPE from STL-ETS [19, 49]). Our analysis (Figure 1 left) shows that deviations in ACF and PACF features, notably *ACF1* and *PACF5*, correlate more strongly with forecasting errors than traditional reconstruction metrics. Second, further experiments at a fixed compression ratio of 5 compare DFT with our proposed method CAMEO which preserves the ACF. Across two forecasting models (STL-ETS, STLF-ARIMA) on the Pedestrian dataset, CAMEO reduces forecasting errors by 10%–20% compared to DFT (Figure 1 right).

Contributions: We introduce an auto-**c**or-rel**a**tion-preserving lossy ti**m**e s**e**ries c**o**mpressor (CAMEO). Our objective is to maximize compression while guaranteeing a user-defined maximum deviation of the ACF or PACF on the compressed data. CAMEO uses an iterative greedy approach, removing points based on their impact on the ACF or PACF. These statistics are updated incrementally. To improve runtime, we leverage blocking and parallelization strategies. In detail, our contributions are:

- We survey lossy time series compression and line simplification methods through a new hierarchical and comparative classification in Section 2.

EDBT '26, Tampere (Finland)

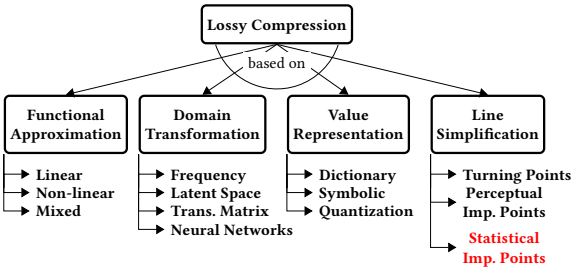


Figure 2: Hierarchy for Lossy Time Series Compression.

- We introduce a new general problem formulation for lossy time series compression under user-specified statistical feature constraints in Section 3.
- We instantiate this formulation by designing CAMEO, a concrete framework that enforces user-defined error bounds ϵ on ACF and PACF (Section 4). Key aspects include incremental updates, blocking heuristics, and parallelization strategies.
- We conduct broad experiments to study CAMEO compared to state-of-the-art lossy compressors, on different datasets, and with different time series analytics in Section 5.

CAMEO yields improvements in compression ratios of 2x on average (and up to 54x on some datasets) while preserving the same deviation of the ACF. Due to the bounded impact on the ACF, CAMEO better maintains, and sometimes even improves, the accuracy of forecasting models and anomaly detection tasks.

2 Background

In this section, we describe the background of lossy time series compression (via a hierarchical classification), existing line simplification algorithms, and autocorrelation functions.

2.1 Lossy Time Series Compression

Lossy time series compression converts an input time series $\mathbf{X}$ of size n into a compressed representation $\mathbf{X}'$ of size n', where $n' \ll n$. The sizes of $\mathbf{X}$ and $\mathbf{X}'$ can be measured in number of bits (e.g., under quantization) or number of elements (e.g., for line simplification). The compression ratio $c = \frac{n}{n'}$ quantifies compression effectiveness. Data distortion refers to the loss of information in the reconstructed time series compared to the original. For further details on compression methods, we refer readers to recent surveys [17] and comparative analyses [7, 48].

Lossy Compression Categories: There is a plethora of lossy time series compression methods with different trade-offs. Figure 2 shows the major categories arranged in a type hierarchy.

- **Functional Approximation:** The data is approximated by one or more functions [9, 11, 16, 29, 32, 56, 59, 63], where the time series is divided into segments, and we store parameters of a low-order polynomial per segment. Such methods guarantee a maximum distortion error per value and are particularly effective on smooth trends.
- **Domain Transformation:** The data is transformed into a different mathematical domain [2, 10, 21, 24, 50, 55, 64]. We compress the data by retaining significant components in this new domain and discarding less important ones. These techniques often assume stationarity or periodicity.
- **Value Representation:** The data is substituted with another more compact representation [43, 58, 70, 73, 88] (e.g., binning or quantization). Here, the compression is achieved by limiting the number of distinct items—thus, codeword size—while controlling the reconstruction error. Symbolic methods enable efficient indexing and search.
- **Line Simplification:** All points are ranked according to a certain criterion and removed in reverse order [18, 39, 40, 44, 89, 90, 108]. The ranks are often updated locally within a small neighborhood of the removed point.

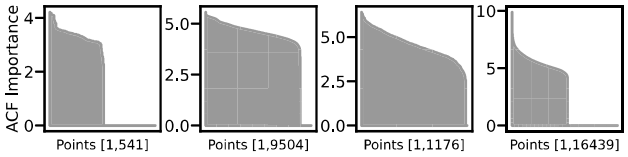


Figure 3: ACF Importance Skew (Non-Uniform Importance)

CAMEO Positioning: Our approach is inspired by Line Simplification methods (which often preserve geometric properties such as area-under-the-curve) but extends this idea to preserving more complex statistical features. We introduce a new category, *Statistical Important Points*, for methods that explicitly preserve time series statistics such as the ACF and PACF. To the best of our knowledge, CAMEO is the first method proposed in this category.

2.2 Line Simplification for Lossy Compression

Line simplification methods reduce the number of points while preserving major geometric or visual characteristics. Keeping relevant original points while approximating others via interpolation can help maintain key patterns, e.g., total energy (area-under-the-curve) in semiconductor degradation tests [22].

Turning Points: The central idea behind turning-points (TP) compression is to store only the points at which the time series changes direction, i.e., where it turns from increasing to decreasing, or vice versa [89, 108]. This approach preserves key inflection points and supports the reconstruction of linear trends, often critical in applications such as stock trading [3].

Perceptual Important Points: The core idea behind perceptual important points (PIP) compression is to find and store points that are significant or meaningful based on the human perception [38, 54]. These points can include peaks, valleys, and other visually salient features that may indicate important events or changes in the data. In detail, PIP-based algorithms build a reduced approximation iteratively by always inserting the point with the largest vertical distance from the current approximation line between two existing PIPs [18, 38, 62, 85]. Early versions of this idea were proposed for polygonal curve simplification and cartographic generalization [27, 84, 98].

Visvalingam-Whyatt Algorithm: The main idea behind Visvalingam-Whyatt (VW) line simplification is to remove points based on the areas of the triangles formed by triplets of sequential points [98]. At each iteration, VW replaces the smallest triangle with a straight-line segment by eliminating the middle point and computing the areas of the newly formed left and right triangles. The algorithm stops when the smallest triangle's area does not meet the error bound or a specified number of points is removed.

CAMEO Differentiation: While CAMEO shares the general structure of progressive point selection with classical line simplification methods, it fundamentally differs in its optimization goal. Instead of preserving geometric properties (area, direction changes, or peaks), CAMEO selects points to preserve the compressed-modified ACF/PACF within an error bound. Figure 3 illustrates this distinction by showing how the ACF importance

distribution—measured as the distortion introduced by removing each point—is highly non-uniform across four time series. This pattern highlights the need for a more targeted point selection strategy instead of treating all points equally. CAMEO's shift from geometry- or perception-driven importance to preserving time series temporal structure marks a significant change from previous approaches.

2.3 Quality Measures

To measure the deviation between the original and reconstructed time series, or the original and compression-modified ACF and PACF, or the forecasted and expected time series, one can use different quality measures $\mathcal{D}(\mathbf{X}, \mathbf{X}')$:

- **Mean Absolute Error:** $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|x_i - x'_i|$
- **Root Mean Square Error:** $\text{RMSE} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2}$
- **Normalized RMSE:** $\text{NRMSE} = \frac{\text{RMSE}}{\max(\mathbf{X})-\min(\mathbf{X})}$
- **Modified Symmetric MAPE:**

$$\text{mSMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - x'_i|}{\max\left(|x_i| + |x'_i| + \epsilon, 0.5 + \epsilon\right)/2}$$

where ϵ is by default 0.1 [42].

2.4 Time Series Autocorrelation Functions

The ACF and PACF are two fundamental statistical concepts that measure the correlation between the observations at a current point in time and observations at different time lags [8].

Basic ACF: The ACF is the Pearson correlation of the time series $\mathbf{X}$ and a lagged version of itself—computed for lags 1 through a user-provided max lag L—and is computed at lag l as follows:

$$\text{ACF}_l(\mathbf{X}) = \frac{1}{(n-l)\sigma^2}\sum_{t=1}^{n-l}(x_t - \mu)(x_{t+l} - \mu) \tag{1}$$

where μ and σ are the mean and standard deviation of $\mathbf{X}$, and $n = |\mathbf{X}|$ (number of points). Equation (1) assumes the time series is stationary, and thus, μ and σ are the same at all time intervals. If the time series is non-stationary, μ and σ should be computed for $\mathbf{X}$ and its lagged version $\mathbf{X}_l$. Specifically, $\mathbf{X}$ spans $[1 \cdots n-l]$ and $\mathbf{X}_l$ spans $[l+1 \cdots n]$. Thus, both time series have $n-l$ elements.

Alternative ACF: An equivalent formulation of the ACF at lag l, but more convenient for later incremental updates, is:

$$\text{ACF}_l = \frac{(n-l)\sum x_t x_{t+l} - \sum x_t \sum x_{t+l}}{\sqrt{\left((n-l)\sum x_t^2 - (\sum x_t)^2\right)\left((n-l)\sum x_{t+l}^2 - (\sum x_{t+l})^2\right)}} \tag{2}$$

whose basic aggregates can be maintained incrementally [101].

Basic PACF: The PACF measures the correlation between current and past observations at lag l, removing intermediate lag influences. The $\text{PACF}_l = \phi_{l,l}$, can be computed using the Durbin-Levinson (DL) [28, 75] recursion in $O(L^2)$ as follows:

$$\phi_{1,1} = \text{ACF}_1, \quad \phi_{l,l} = \frac{\text{ACF}_l - \sum_{k=1}^{l-1}\phi_{l-1,k}\text{ACF}_{l-k}}{1 - \sum_{k=1}^{l-1}\phi_{l-1,k}\text{ACF}_k} \tag{3}$$

where $\phi_{l,k} = \phi_{n-1,k} - \phi_{n,n}\phi_{n-1,n-k}$ for $1 \le k \le l-1$.

Utility: The ACF and PACF are valuable tools in time series analytics, often used for understanding the underlying patterns in the series, assisting in selecting the type and order of forecasting models, and enabling precise and reliable forecasts.

3 Problem Formulation

In this section, we introduce three variants for the problem of compressing a time series while preserving statistical features. This problem formulation is independent of concrete algorithms.

Definition 1 (Statistical Important Points). *Given a time series $\mathbf{X}$, an error bound ϵ, a time series statistic $\mathcal{S}$, and a quality measure $\mathcal{D}$, we aim to find a compressed time series $\mathbf{X}'$ (in terms of a subset of original data points) such that:*

$$\begin{aligned} max \quad & \frac{|\mathbf{X}|}{|\mathbf{X}'|} \\ s.t. \quad & \mathcal{D}(\mathcal{S}(\mathbf{X}), \mathcal{S}(\mathbf{X}')) \le \epsilon \end{aligned} \tag{4}$$

This optimization objective maximizes $|\mathbf{X}|/|\mathbf{X}'|$ (compression ratio), while enforcing a bounded deviation of the user-provided statistic $\mathcal{S}$ on the compressed data (measure by $\mathcal{D}$) by at most ϵ.

Complexity: Similar to other line simplification methods, finding the globally optimal solution efficiently is intractable [95, 96]. Thus, we aim to find approximate solutions with high compression ratios but hard or high-probability guarantees on the deviation from $\mathcal{S}$. Furthermore, we may need to preserve statistics on window aggregates of the time series. For example, a time series in 4-second granularity with daily seasonality would require an ACF with 21,600 lags to capture a full season. Therefore, we introduce a variant of the *Statistical Important Points* problem, aiming to preserve statistical features on aggregated time series.

Definition 2 (Statistical Important Points on Aggregates). *Given a time series $\mathbf{X}$, an error bound ϵ, a time series statistic $\mathcal{S}$, a quality measure $\mathcal{D}$, and an additional aggregation function Agg_κ over κ data points, we aim to find a compressed time series $\mathbf{X}'$ (in terms of a subset of original data points) such that:*

$$\begin{aligned} max \quad & \frac{|\mathbf{X}|}{|\mathbf{X}'|} \\ s.t. \quad & \mathcal{D}(\mathcal{S}(Agg_\kappa(\mathbf{X})), \mathcal{S}(Agg_\kappa(\mathbf{X}'))) \le \epsilon \end{aligned} \tag{5}$$

where $Agg_\kappa(\mathbf{X}) = [a_1, \ldots, a_{n/\kappa}]$ and $a_i = Agg_\kappa(x_{[i:i+\kappa]})$. The aggregation function Agg_κ needs to be additive, semi-additive, or additively-computable to enable incremental updates [101].

Example: To illustrate the SIP on Aggregates problem, assume an original time series $\mathbf{X}$ sampled every minute, $\epsilon = 0.01$, $\mathcal{S} = \text{ACF}$, $\text{Agg}_\kappa = \sum_{i+1}^{i+30} X_i/30$ (the mean value every $\kappa = 30$ minutes), and $\mathcal{D} = \text{MAE}$. Here, Equation 5's constraint renders to

$$\text{MAE}\left(\text{ACF}_{\langle 1\cdots L\rangle}\left(\frac{\sum_{i+1}^{i+30}\mathbf{X}_i}{30}\right), \text{ACF}_{\langle 1\cdots L\rangle}\left(\frac{\sum_{i+1}^{i+30}\mathbf{X}'_i}{30}\right)\right) \le 0.01,$$

where i iterates through and aggregates consecutive tumbling (i.e., jumping) windows in the time series. Alternative problem formulations exchange the hard and soft constraints of the above optimization objectives to reach desired compression ratios without unnecessary exploration of parameters.

Definition 3 (Compression-centric Statistical Important Points). *Given a time series $\mathbf{X}$, a statistic $\mathcal{S}$, a quality measure $\mathcal{D}$, an optional aggregation function Agg_κ and a minimum compression ratio c, we aim to find a compressed time series $\mathbf{X}'$ such that:*

$$\begin{aligned} min \quad & \mathcal{D}(\mathcal{S}(Agg_\kappa(\mathbf{X})), \mathcal{S}(Agg_\kappa(\mathbf{X}'))) \\ s.t. \quad & \frac{|\mathbf{X}|}{|\mathbf{X}'|} \ge c \end{aligned} \tag{6}$$

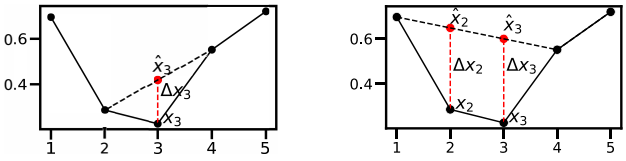


Figure 4: Linear Interpolation of $\mathbf{x}_3$ (Left) and $\mathbf{x}_2$ (Right).

This optimization objective minimizes the distortion between the original and reconstructed statistical features while removing points until the compression ratio c is reached.

4 *CAMEO* Framework

CAMEO addresses the *Statistical Important Points* problem using a greedy, iterative strategy that progressively removes points while controlling the impact on the ACF. At each iteration, it selects the point with the minimum ACF impact, interpolates its value, updates internal aggregates, and checks whether the user-defined error bound ϵ is still satisfied. This section introduces the overall compression algorithm (Algorithm 1), as well as three techniques for improving its runtime efficiency: incremental maintenance of the ACF (Section 4.2), blocking of local neighborhoods (Section 4.3), and different parallelization strategies (Section 4.4). The underlying greedy heuristics of our algorithm include (1) selecting the next best point, and (2) updating the ACF impact in a local neighborhood.

4.1 Overall Compression Algorithm

CAMEO begins by calling ExtractAggregates($\mathbf{X}$) (Line 1 of Algorithm 1) to compute the basic aggregates needed for efficiently evaluating the ACF using Equation 2. Rather than recomputing the ACF from scratch after each removal, CAMEO incrementally updates these aggregates based on the interpolation error introduced by removing a point (Line 6). Figure 4 (left) illustrates this process: removing x_3 and interpolating its value from x_2 and x_4 introduces a small error Δx_3, which is then used to update the aggregates. This update enables maintaining the ACF in constant time per lag. The compression process iterates over these steps until the error bound ϵ is violated or no further points can be removed (Lines 4–12).

Initialization: To rank points for removal, we compute for each point its estimated impact on the ACF if removed (Line 3). The function GetAllImpactACF (Algorithm 2) leverages the precomputed aggregates to evaluate the impact in $O(Ln)$ time and stores the results in a heap H. This operation is vectorizable and parallelizable, as each point's impact can be calculated independently. Specifically, GetAllImpactACF updates the ACF aggregates based on the hypothetical removal of each point (Lines 4-9), computes the new ACF (Line 10), and measures the distortion using $\mathcal{D}$ (Line 11). The impacts are then organized into H, built in $O(n)$ time using Floyd's method [36]. The total initialization time complexity is $O(Ln + n)$.

Inner Loop and Updating Heuristic: Every time a point $\mathbf{x}_i$ is popped from the heap, we compute its interpolation error $\Delta\mathbf{x}_i$, update the ACF aggregates, and recompute the ACF (Lines 5–8). We then check whether the updated ACF satisfies the user-defined quality constraint $\mathcal{D}$ (Line 9). If so, the point is permanently removed (Line 11). Since removing points alters relationships between values at different lags, the previously computed ACF impacts of neighboring points may become outdated. To maintain consistency, CAMEO uses the ReHeap procedure

Algorithm 1 CAMEO

Input: Time Series $\mathbf{X}$, Error Bound ϵ, Max Lag L
Output: List of Remaining Points $\mathbf{X}'$

1: ACFAgg ← ExtractAggregates($\mathbf{X}$) // *Get ACF aggregates*
2: P_L ← GetACF(ACFAgg) // *Get raw ACF*
3: H ← GetAllImpactACF(ACFAgg, $\mathbf{X}$) // *Heap of distortions*
4: **while** Top(H) ≠ NULL **do** // *Not empty*
5: $\mathbf{x}_i$ ← Pop(H) // *Get next point*
6: $\Delta\mathbf{x}_i$ ← Interpolate($\mathbf{x}_i$) // *Get interpolation error*
7: ACFAgg ← Update(ACFAgg, $\Delta\mathbf{x}_i$) // *Update ACFAgg*
8: $\hat{\mathrm{P}}_L$ ← GetACF(ACFAgg) // *Get new ACF*
9: **if** $\mathcal{D}(\hat{\mathrm{P}}_L, \mathrm{P}_L) \geq \epsilon$ **then** // *Check error bound*
10: **return** $\mathbf{X}'$ // *Error bound reached*
11: $\mathbf{X}'$ ← Remove($\mathbf{X}, \mathbf{x}_i$) // *Remove the point*
12: H ← ReHeap(H, $\mathbf{x}_i$) // *Update impact of points in $\mathcal{N}_h(x_i)$*
13: **return** $\mathbf{X}'$

Algorithm 2 GetAllImpactACF

Input: ACF aggregates ACFAgg, Time Series $\mathbf{X}$, Raw ACF P_L
Output: Heap with Impact on ACF per each Point H

1: $i \leftarrow [1, \cdots, n-1]$ // *Get indices*
2: $l \leftarrow [1, \cdots, L]$ // *Get lags*
3: $n \leftarrow [n-1, \cdots, n-L]$ // *Get size for all lags*
4: $\Delta\mathbf{X} \leftarrow \frac{1}{2}(\mathbf{X}[2:] - \mathbf{X}[:-2]) - \mathbf{X}[i]$ // *Get all deltas x_i by LIP*
5: $sx \leftarrow \text{ACFAgg}.sx + \Delta\mathbf{X}$ // $\sum x$
6: $sx_l \leftarrow \text{ACFAgg}.sx_l + \Delta\mathbf{X}$ // $\sum x_l$
7: $sx^2 \leftarrow \text{ACFAgg}.sx^2 + \frac{1}{n}\Delta\mathbf{X}(\Delta\mathbf{X} + 2\mathbf{X}[i])$ // $\sum x^2$
8: $sx_l^2 \leftarrow \text{ACFAgg}.sx_l^2 + \frac{1}{n}\Delta\mathbf{X}(\Delta\mathbf{X} + 2\mathbf{X}[i])$ // $\sum x_l^2$
9: $sxx_l \leftarrow \text{ACFAgg}.sxx_l + \frac{1}{n}\Delta\mathbf{X}(\mathbf{X}[i-l] + \mathbf{X}[i+l])$ // $\sum xx_l$
10: $\hat{\mathrm{P}}_L \leftarrow \text{GetACF}(sx, sx_l, sx^2, sx_l^2, sxx_l)$ // *Apply Equation 2*
11: H ← Heapify($\mathcal{D}(\hat{\mathrm{P}}_L, \mathrm{P}_L)$) // *Floyd's method*
12: **return** H

(Line 12) to selectively update these scores. A blocking heuristic (see Section 4.3) limits this recomputation to a fixed number of neighbors, balancing efficiency and accuracy.

Decompression: CAMEO uses linear interpolation as its decompression strategy, consistent with standard line simplification methods. This choice aligns with CAMEO's compression phase, which assumes linear interpolation when estimating the ACF/-PACF impact of removing a point. As a result, the reconstructed time series is composed of piecewise linear segments. This approach is both efficient and practical, requiring only a single forward pass over the retained points.

4.2 Incremental ACF and PACF

Computing the ACF or PACF from scratch for every removed point is infeasible for large time series. Hence, we incrementally maintain the autocorrelation functions—for constraint validation during compression—by keeping track of Equation 2's basic aggregates sx, sx^2, sx_l, sx_l^2, and sxx_l:

$$sx = \sum_{i=0}^{n-l} x_i \quad sx_l = \sum_{i=l}^{n} x_i \quad sxx_l = \sum_{i=0}^{n-l} x_i x_{i+l}$$
$$sx^2 = \sum_{i=0}^{n-l} x_i^2 \quad sx_l^2 = \sum_{i=l}^{n} x_i^2 \tag{7}$$

These aggregates are computed by the function ExtractAggregates for all lags $l \in [1, L]$ in Algorithm 1 with complexity $O(Ln)$

dominated by sxx_l. When removing the data point x_i, we then compute the distance Δx_i between x_i and its interpolated value $\hat{x}_i$, i.e., $\Delta x_i = \hat{x}_i - x_i$. Figure 4 (left) shows an example. Given Δx_i, we derive the following update rules:

$$\begin{aligned} sx &+= \Delta x_i, \quad sx^2 += \Delta x_i(2x_i + \Delta x_i), \quad sx_l += \Delta x_i \\ sx_l^2 &+= \Delta x_i(2x_i + \Delta x_i), \quad sxx_l += \Delta x_i(x_{i-l} + x_{i+l}) \end{aligned} \tag{8}$$

Once the aggregates are updated, we can compute the ACF at a specific lag using Equation 2. Similarly, to compute and preserve the PACF, we incrementally maintain the ACF and apply the DL recursion in Equation 3 albeit with higher computational cost.

Update Rules for Multiple Elements: In some cases, removing a point requires interpolating more than one element, as shown in Figure 4 (right). In that case, the basic aggregates are updated by summing over the deltas of every interpolated point. Specifically, if removing point x_i requires interpolating the m points $[x_j, \ldots, x_i, \ldots, x_{j+m}]$ (changed interpolations until the next remaining points left and right), the update rules are:

$$\begin{aligned} sx &+= \sum_{k=j}^{j+m} \Delta x_k, \qquad sx^2 += \sum_{k=j}^{j+m} \Delta x_k(2x_k + \Delta x_k), \\ sx_l &+= \sum_{k=j}^{j+m} \Delta x_k, \qquad sx_l^2 += \sum_{k=j}^{j+m} \Delta x_k(2x_k + \Delta x_k), \\ sxx_l &+= \sum_{k=j}^{j+m} \Delta x_k(x_{k-l} + x_{k+l}) + \sum_{k=j}^{j+m-l} \Delta x_k \Delta x_{k+l} \end{aligned} \tag{9}$$

Ideally, updating the basic aggregates should not materialize the interpolation of the points from j to $j+m$ because they are affine functions. However, there is no straightforward way to update sxx_l without any assumption on the time series. Note that, updating sxx_l has a time complexity of $O(mL)$ since we need to calculate a value for each lag l and point from j to $j+m$. The rest of the basic aggregates can be updated in $O(L+m)$.

Update Rules with Aggregation Function: Solving the *Statistical Important Points on Aggregates* problem requires additional extensions. Given the aggregation function AGG_κ, we first compute and store all $a_i \in \text{AGG}_\kappa(X)$. Subsequently, while removing the points x_i, we incrementally update the aggregates:

$$\begin{aligned} sa &= \sum_{i=0}^{\lfloor n/\kappa \rfloor - l} a_i \quad sa_l = \sum_{i=l}^{\lfloor n/\kappa \rfloor} a_i \quad saa_l = \sum_{i=0}^{\lfloor n/\kappa \rfloor - l} a_i a_{i+l} \\ sa^2 &= \sum_{i=0}^{\lfloor n/\kappa \rfloor - l} a_i^2 \quad sa_l^2 = \sum_{i=l}^{\lfloor n/\kappa \rfloor} a_i^2 \end{aligned} \tag{10}$$

When removing the point x_i, we again consider two cases. First, if only one point is interpolated, the update rules are:

$$\begin{aligned} sa &+= \Delta a_i, \quad sa^2 += \Delta a_i(2a_j + \Delta a_i), \quad sa_l += \Delta a_i \\ sa_l^2 &+= \Delta a_i(2a_i + \Delta a_i), \quad saa_l += \Delta a_i(a_{i-l} + a_{i+l}) \end{aligned} \tag{11}$$

where $\Delta a_i = \text{AGG}_\kappa([\hat{x}_i, \ldots, x_{i+\kappa}]) - a_i$. Note, if AGG_κ is commutative and associative, it is possible to avoid computing AGG_κ over all points. For example, if AGG_κ is the mean function, then $\Delta a_i = (\hat{x} - x_i)/\kappa$. Second, if m points are interpolated, we first compute all Δa_i by mapping the interpolated points $x_i \in [x_j, \ldots, x_{j+m}]$ to their aggregates a_i. Then, $\Delta a_i = \text{AGG}_\kappa(\hat{x}_i, \hat{x}_{i+1}, \ldots, x_{i+\kappa}) - a_i$, which requires recomputing AGG_κ for all elements if they are interpolated. Finally, we reuse Equation 9 on the aggregates.

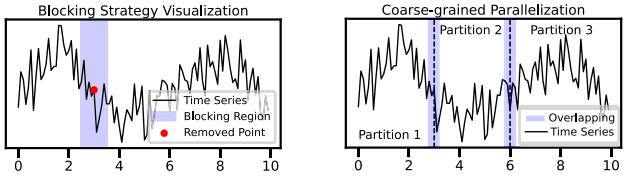


Figure 5: Blocking and Coarse-grained Parallelization.

4.3 Blocking

Inspired by blocking strategies in entity resolution (deduplication), CAMEO improves efficiency by updating the ACF impact only for neighboring points around a removed point instead of all points in the time series.

Blocking Heuristic: Our blocking heuristic relies on the assumption of temporal locality. We assume that removing a point affects the nearby points, and its impact further away diminishes. Thus, we update the impact on the ACF of only non-removed h-neighboring points of $\mathbf{x}_i$. To efficiently identify neighbors, each point maintains dynamic left and right pointers in the heap. When a point is removed, we traverse up to h hops in both directions, collecting non-removed neighbors. After each removal, pointers are updated to maintain consistency. The ReHeap procedure then updates the ACF impacts of the affected neighbors based on the update rules (Equations 8 and 9). Updating the values takes $O(\log n)$ per point. Figure 5 (Left) illustrates this process.

Time Complexity: Let $n = |X|$, L be the number of lags, and h the blocking window. For each neighboring point within h hops, we update its score using the incremental rules in $O(L)$ time and adjust its position in the heap in $O(\log n)$ time. Thus, the cost of a single removal is $O\big(h(L + \log n)\big)$. As compressing from n points down to n' points requires at most $n - n' = O(n)$ removal steps, the overall complexity is $O\big(nh(L + \log n)\big)$. In practice, setting $h = 10 \log n$ retains compression quality while substantially reducing runtime (see Section 5.5). Alternatively, h can be incrementally tuned by starting from a small value and increasing it until no significant improvement in compression ratio is observed. Exploring adaptive blocking strategies based on seasonality or lag-correlation, and providing theoretical guarantees on compression efficiency, remains an interesting direction for future work.

4.4 Parallelization

When dealing with very large time series, the application of parallelization strategies becomes indispensable to significantly improve computational time. In CAMEO, we implement two different parallelization strategies, namely, *fine-* and *coarse-grained* approaches with their specific advantages and disadvantages.

Fine-grained Parallelization: The first strategy is designed with the primary objective of improving runtime efficiency without additional heuristics that might impact the quality of compression. In CAMEO's blocking strategy, each neighbor's impact on the ACF can be calculated independently during the *look-ahead* phase. Given T threads, we segment the number of neighbors, h, into static chunks of size h/T, assigning each chunk to a thread. Each thread independently computes the *look-ahead* impact on the ACF for its designated chunk, thereby reducing execution time. However, there are fine-grained synchronization barriers for every removed point.

Coarse-grained Parallelization: This strategy is designed for systems with many cores and very large time series data.

Table 1: Datasets Summary. The standard deviation is denoted by σ, and the probability of a data point to be higher than, equal to, or lower than that of the previous data point is denoted by ($p_\uparrow$), ($p_=$) and ($p_\downarrow$).

Dataset	Length	ACF #Lag	ACF1	ACF10	PACF5	Min Value	Range	Median	σ	$p_\uparrow - p_= - p_\downarrow$	Mean Delta
ElecPower [45]	2,977	48	0.768	3.38	0.94	0.099	5.7	0.29	0.74	48%–0%–52%	8e-04
MinTemp [80]	3,652	365	0.774	5.97	1.32	0.01	26.3	11.0	4.01	52%–1%–47%	0.002
Pedestrian [42]	8,766	24	0.896	1.02	-0.11	0.00	5,573	396	1,017	45%–0%–55%	0.004
UKElecDem [37]	17,520	48	0.988	7.2	0.37	16,309	30,124	27,857	6,071	44%–0%–56%	0.34
AUSElecDem [42]	230,736	7 on 48	0.762	5.09	1.09	3,498	9,367	6,783	1,361	42%–0%–58%	0.001
Humidity [78]	397,440	24 on 60	0.951	2.66	-0.07	12.65	87.27	76.38	19.73	55%–3%–42%	5e-06
IRBioTemp [77]	878,400	24 on 60	0.958	4.41	0.17	-5.47	54.6	23.21	8.55	45%–5%–50%	-3e-06
SolarPower [42]	986,297	24 on 120	0.90	1.02	0.125	0.00	116.5	0.0	43.33	12.5%–75%–12.5%	0.0

The core idea is to partition the time series $\mathbf{X}$ into T consecutive chunks, assign a thread to each partition, and compress each partition independently using the single-threaded CAMEO algorithm. Each thread independently computes and updates its own aggregates while concurrently handling overlapping regions of the partitions. Synchronization overhead is minimized by allowing each thread to work independently within a local ACF deviation threshold of $p \cdot \epsilon/T$. Once a partition reaches its local error threshold, synchronization begins to update global aggregates across all partitions. This way, we introduce synchronization only when necessary to guarantee the global allowable ACF deviation is not exceeded.

Example 4.1 (Coarse-grained Parallelization). Consider a time series $\mathbf{X}$ divided into three partitions P_1, P_2, and P_3, as shown in Figure 5 (Right). Let $sx(P_i)$ denote the sum of all points of partition P_i. Each partition computes and updates its own aggregates independently: $sx(P_1)$, $sx(P_2)$, and $sx(P_3)$. Similarly, the aggregates $sx_l(P_i)$, $sx^2(P_i)$, and $sx_l^2(P_i)$ are handled independently within each partition P_i. For the dot product between the lagged time series, the overall aggregate sxx_l is computed from the aggregates per partition $sxx_l(P_i)$ and the contributions of overlapping regions: $sxx_l = \sum_{i=1}^{3} sxx_l(P_i) + sxx_l(\text{Overlap}_{12}) + sxx_l(\text{Overlap}_{23})$, where $sxx_l(\text{Overlap}_{ij}) = \sum_{t \in P_i, t+l \in P_j} x_t x_{t+l}$ accounts for cross-products where x_t is in P_i and x_{t+l} is in P_j. Only the threads handling P_i and P_{i+1} need to synchronize when accessing the small overlapping regions, and thus, synchronization overhead is negligible. Given these aggregates, each partition can operate independently until meeting the error bound $p \cdot \epsilon/3$. Once a partition reaches its local error threshold, the global ACF can be computed by synchronizing access to the aggregates across all the partitions, ensuring that the overall ACF deviation remains within the specified error bound ϵ.

5 Experiments

Our experiments study CAMEO's compression ratio, reconstruction error, and runtime across various datasets, and compare them with those of lossless and lossy compression techniques.

5.1 Experimental Setup

We conduct the experiments on a Linux server equipped with two Intel Xeon Gold 6338 @ 2.0 GHz CPUs (in total 128 virtual cores), 48 MB L3 caches, and 1 TB of DDR4 @ 3200 MHz main memory. We implemented CAMEO in Cython 3.0.0, compiled with GCC 9.4.0 at optimization level O3 and OpenMP 4.5. Cython provides performance enhancements through static typing of variables while supporting NumPy [34], and avoiding the problems of Python's Global Interpreter Lock (GIL). By default, our experiments use the *SIP* problem (Equation 4), the mean aggregation function AGG_κ, quality measure $\mathcal{D}$ = MAE.

Line-Simplification Baselines: We compare CAMEO's performance with three line-simplification algorithms (Section 2.2), which we adapt to support an ACF constraint by incrementally maintaining each point's impact on the ACF.

- the Visvalingam-Whyatt (VW) algorithm [98],
- the Turning Points (TP) algorithm [89] (w/ Sum of the Absolute Values TP, and Mean Absolute Error TPm [89]),
- the Perceptual Important Points (PIP) algorithm [38, 54] (w/ Vertical PIPv and Euclidean PIPe distances [38]).

Additional Baselines: We also compare CAMEO with four well-known lossy compression algorithms: Sliding Window and Bottom Up (SWAB) [57], Poor Man's Compression Mean (PMC) [63], Swing Filter (SWING) [32], Sim-Piece (SP) [59], and the Discrete Fourier Transform (DFT) [21]. SWAB, PMC, SWING, and SP learn constant and linear approximations which are prevalent functional approximation approaches. FFT can compress the data by discarding the less important high-frequency components of the frequency spectrum. Since enforcing the ACF constraint while compressing is not straightforward, we perform a trial-and-error exploration of the parameters of these methods. In addition, we compare with Gorilla (GHR) [82] and Chimp (CHM) [67] as lossless compressors, and SZ3 [68] and Mix-Piece (MXP) [60] as lossy compressors. Gorilla and Chimp rely on XOR compression of consecutive floating-point values, whereas SZ3 relies on prediction and quantization and integrates lossless compressors like Zstandard (Zstd) [20]. MXP extends SP with segment grouping strategies and also integrates Zstd.

Datasets: We use eight publicly available datasets. Our primary selection criterion was the presence of a seasonal component, which is discernible via the ACF. This seasonal component was then used to guide dataset-specific configurations of the number of lags. Table 1 summarizes their main characteristics:

- **ElecDem** [45]: contains the electric power consumption of one household with a 15-minute sampling rate during the month 07-2007.
- **MinTemp** [80]: contains daily min temperature in Melbourne (Australia) from 1981 through 1990.
- **Pedestrian** [42]: contains hourly pedestrian counts of 66 sensors in Melbourne from May 2009 through 05-2020.
- **UKElecDem** [37]: contains the national electricity demand every half-hour of Great Britain for 2021.
- **AUSElecDem** [42] contains the electricity demand every half-hour in Victoria (Australia) from 2002 to 2015.

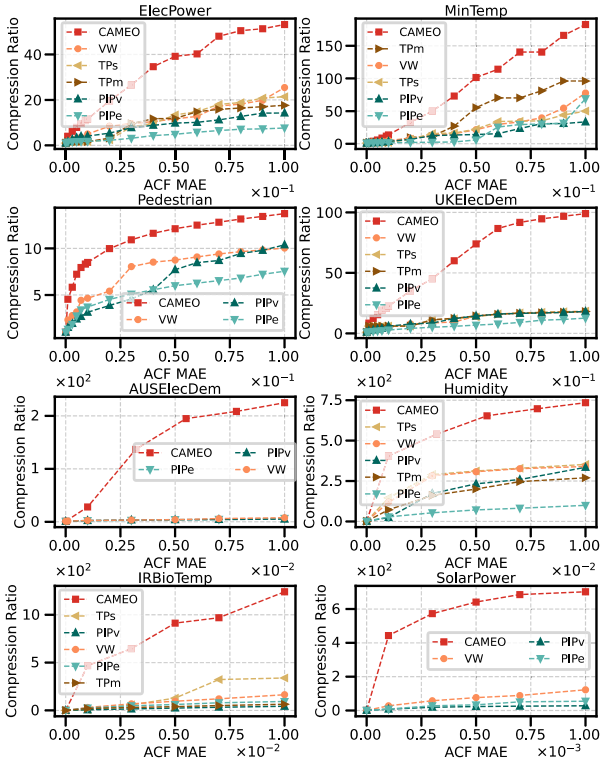


Figure 6: Compression Ratio as the ACF Error Increases for Line Simplification Baselines (VW, TP, PIP).

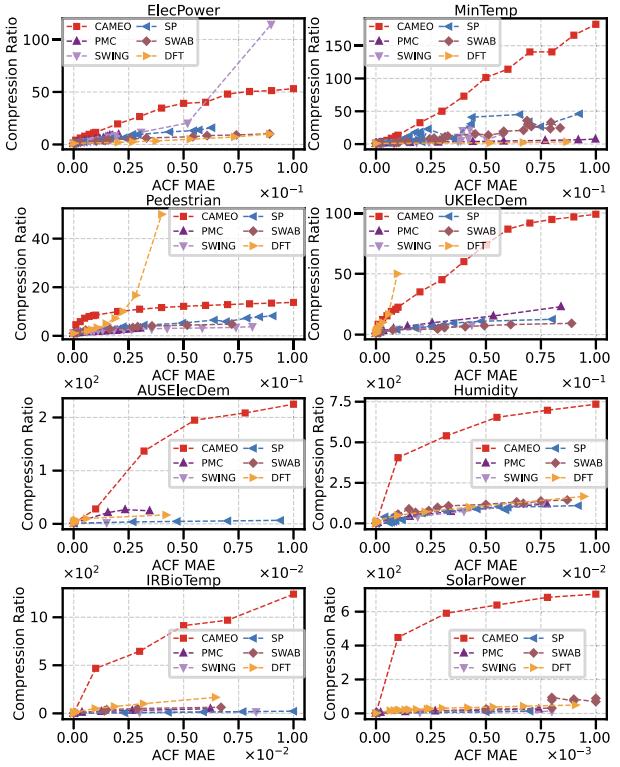


Figure 7: Compression Ratio as the ACF Error Increases for Lossy Compressor Baselines.

- **Humidity** [78]: contains relative humidity measurements averaged over 1 minute from 04-2015 through 06-2023.
- **IRBioTemp** [77]: contains biological surface temperature averaged over 1 minute from 04-2015 through 06-2023.
- **SolarPower** [42]: contains solar power production stored every 30 seconds from 08-2019 through 06-2020.

We divide the datasets into two groups of four datasets, where for group 1, we preserve the ACF directly; for group 2, we preserve the ACF on window aggregates. Table 1 also specifies the number of points per window and the number of lags, e.g., for "7 on 48", we aggregate 48 points per window and keep 7 ACF lags.

Number of Lags: For all experiments, we select the number of ACF lags as a full seasonality period. For example, MinTemp contains daily measurements with yearly seasonality, and thus, we compute the ACF for 365 lags. This parameter can also be set automatically using seasonality detection techniques based on ACF peaks, spectral density estimation, or heuristic methods [5, 33, 49]. We select the aggregation period based on the seasonality we aim to preserve. For example, Humidity has one-minute granularity, we aggregate over hours and computed the ACF of 24 lags (a day) instead of an ACF with 1,440 lags.

Compression Ratio Computation: We compute compression ratios primarily at the *logical level*, i.e., based on the number of elements retained. For line simplification methods such as CAMEO, which subsample points, we define the compression ratio as $c = n/n'$, where n' is the number of points preserved. For functional approximation methods like PMC and SWAB, we count the number of segments, whereas for SP (which tightly integrates timestamps and values), we conservatively count all elements. To evaluate MXP and SZ3, which perform byte-level encoding and output compact binary representations, we report compression efficiency using the $Bits/v = Bits(\mathbf{X}')/|\mathbf{X}|$ metric, where $Bits(\mathbf{X}')$ is the number of bits required to store the compressed output. For fair comparison under this metric, we also apply delta encoding to the timestamps and compress both timestamps and values using Zstd on CAMEO's output. While this comparison may underestimate the compression ratio of some methods, we adopt this approach to better separate the influence of the output's logical structure from the physical-level compression effectiveness across techniques.

5.2 Compression Ratio

In the first set of experiments, we assess the compression ratio achieved for the eight datasets. We report CAMEO's compression ratio without blocking to understand CAMEO's maximal compression capability. Figure 6 shows the results CAMEO achieved compared to other line-simplification baselines under varying error bounds for the ACF deviation. CAMEO consistently delivers the best compression ratio among all baselines, mainly because it is the only technique that directly optimizes for the ACF. CAMEO achieves a 1.1x to 54x higher compression ratio, even at very small error bounds. Our extensions of the PIPs and TPs line-simplification methods to ensure a constraint on the ACF deviation were effective in most datasets except Pedestrian, AusElecDem, and SolarPower. In these instances, the initial phase of the TP method, which involves removing all non-turning points, results in an ACF that deviates more than the allowed error bounds. Among the line-simplification baselines, VW shows the best performance on average.

Additional Baselines Results: We compare CAMEO with the additional lossy compression baselines. As shown in Figure 7, CAMEO delivers the best compression ratio among all baselines.

Table 2: Bits/value (B/v) Result of Lossless and Additional Lossy Compression Baselines.

Dataset	GRL	CHM	Zstd	SZ3		MXP		CAMEO	
	B/v	B/v	B/v	B/v	ϵ	B/v	ϵ	B/v	ϵ
ElecPower	32.6	28.2	26.2	1.9	0.01	2.8	0.02	3.5	0.01
MinTemp	31.0	25.6	13.5	2.2	0.05	3.7	0.04	2.3	0.01
Pedestrian	16.1	21.0	16.4	1.3	0.03	2.3	0.03	3.2	0.01
UKElecDem	18.3	21.3	20.7	0.8	0.03	1.5	0.05	1.4	0.01
AUSElecDem	27.5	24.8	27.6	5.6	1e-5	4.3	2e-3	4.1	1e-5
Humidity	24.3	21.0	17.2	3.5	5e-6	1.8	4e-4	1.6	1e-6
IRBioTemp	24.9	21.4	14.4	3.2	7e-6	1.6	2e-3	1.6	1e-6
Solar	2.3	2.8	5.1	1.7	1e-4	1.3	1e-3	0.7	1e-6

Some methods outperform CAMEO in a few instances. For example, DFT outperforms CAMEO in Pedestrian and UKElecDem, which suggests that these datasets predominantly consist of low-frequency components. Similarly, SWING outperforms CAMEO on ElecPower, showing higher compression ratios at larger error bounds after an initially weaker performance. This higher compression ratio suggests that, in these cases, the error bound is large enough to allow the fitting of a few linear functions without significantly affecting the ACF. Despite these exceptions, CAMEO consistently demonstrates superior compression ratios.

Bits-per-Value Analysis: We further evaluate CAMEO's compression effectiveness by comparing it against lossless compressors Gorilla (GRL), Chimp (CHM), and Zstd, as well as lossy compressors SZ3 and MXP, using the Bits-per-Value metric (B/v). Table 2 shows that lossy compression can outperform lossless methods by a significant margin while introducing only a small distortion on the ACF (ϵ). CAMEO achieves compression levels that are competitive with or better than SZ3 and MXP across most datasets. On larger datasets like Solar and IRBioTemp, CAMEO significantly outperforms all baselines despite preserving the ACF under a stricter error bound ($\epsilon = 10^{-6}$). For smaller series, SZ3 and MXP benefit from their low-level optimizations (e.g., quantization and byte encoding), yielding slightly better compression in some cases. Interestingly, GRL and CHM significantly outperform Zstd on the Solar dataset, likely due to the long flat segments (e.g., nighttime with zero values), where XOR-based delta encoding is especially effective.

5.3 Decompression Error

In a second set of experiments, we evaluate the reconstruction error by collecting the decompressed series produced during the compression ratio evaluation. We use NRMSE to quantify the distortion. Figure 8 illustrates the results. Overall, no single method consistently outperforms the others, with performance highly dependent on the characteristics of each dataset. On average, CAMEO performs on par with the baselines in terms of NRMSE, achieving similar reconstruction quality for comparable compression ratios. Notably, CAMEO never performs the worst and achieves the best results on the SolarPower dataset. These results are particularly interesting, as CAMEO is optimized for preserving temporal structure rather than reconstruction error, yet still retains point-wise accuracy comparable to methods tailored for it. Preserving the ACF seems to help maintain local trends and smooth transitions in the time series, which indirectly limits large deviations and keeps a low NRMSE. In contrast, PIPe exhibits the highest NRMSE across several datasets among the line simplification methods, suggesting that Euclidean distance

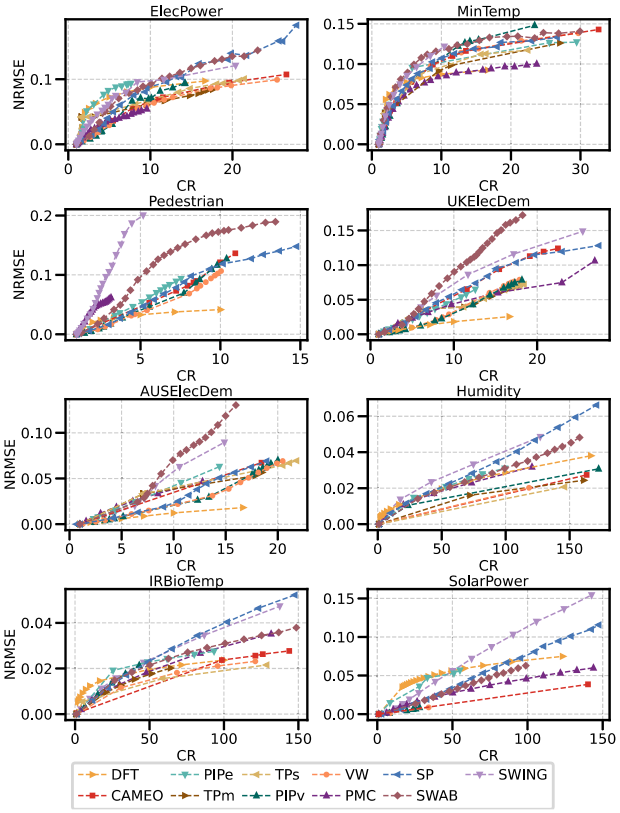


Figure 8: NRMSE as the Compression Ratio Increases.

may not be a suitable importance function. Notably, DFT performs well on some datasets, highlighting its effectiveness for time series dominated by low-frequency components.

5.4 Blocking Strategy

In a third series of experiments, we conduct micro-benchmarks for the blocking strategy and how it impacts CAMEO's compression performance and the decompression error. We showcase the results using four datasets: Pedestrian, MinTemp, AUSElecDem, and Humidity (two of both groups). For the case of AUSElecDem and Humidity, the number of blocking hops is multiplied by the size of the aggregation window to cover the necessary lags. Figure 9 shows CAMEO's compression ratio as the error bound increases, using different numbers of hops in our blocking strategy. The results show only a slightly reduced compression ratio when using different factors of $\log n$ (from 5 to 20), compared to no blocking (w/ob). In contrast, using $\log n$ results in an inferior compression ratio on all datasets. This result supports our hypothesis that temporal locality influences how point removals affect the ACF, and that insufficient blocking fails to update ACF contributions at relevant lags. Complementary, Figure 10 shows the decompression error (NRMSE) on two of the datasets. The NRMSE trends mirror those of the compression ratios, suggesting that moderate blocking has no negative impact on decompression accuracy; instead, the observed variation is driven by the underlying compression ratio. The next section also examines the impact of blocking on compression time.

5.5 Compression Time

In a fourth set of experiments, we compare CAMEO's single-threaded compression time with all other baselines. We show

Table 3: Compression Times (sec) of the Baselines and Singled-threaded CAMEO (with different blocking sizes).

Dataset	SWAB	PMC	SWING	SP	DFT	TP	PIP	VW	CAMEO						
									1	$\log n$	$3\log n$	$5\log n$	$7\log n$	$10\log n$	*w/ob*
ElecPower	0.02	6e-4	6e-4	2e-3	4e-4	3e-3	0.04	0.01	0.02	0.02	0.09	0.2	0.33	0.4	2.5
MinTemp	0.03	7e-4	1e-3	2e-3	4e-4	0.01	0.2	0.02	0.06	0.07	0.6	0.42	1.49	2.75	17.7
Pedestrian	0.2	1e-3	2e-3	5e-3	1e-3	-	0.02	8e-3	0.03	0.09	0.2	0.3	0.5	0.68	9.2
UKElecDem	0.7	3e-3	5e-3	1e-2	1e-3	8e-3	0.05	0.02	0.05	0.2	0.76	1.08	1.49	2.39	64.1
AUSElecDem	0.16	0.04	0.06	0.13	0.03	0.04	0.3	0.27	0.25	0.6	0.3	3.6	8.0	12.2	2,554
Humidity	0.34	0.04	0.07	0.15	0.05	0.3	0.9	0.52	1.24	10.3	21.9	25.3	35.6	48.6	6,837
IRBioTemp	0.74	0.10	0.15	0.29	0.16	0.25	2.1	1.6	2.5	24.9	51.8	78.9	91.7	121	17,602
SolarPower	0.87	0.11	0.17	0.34	0.42	-	21	0.72	1.19	13.8	32.8	44.7	52.7	63.3	5,718

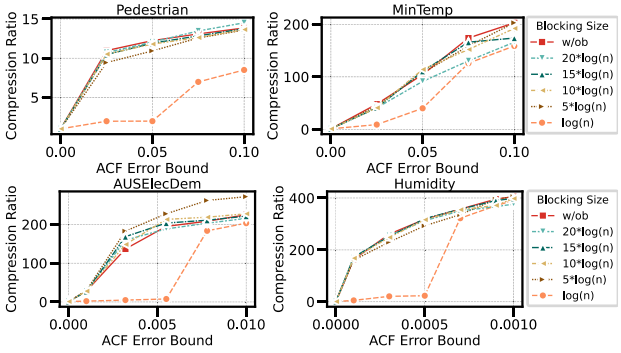


Figure 9: Compression Ratio using Blocking.

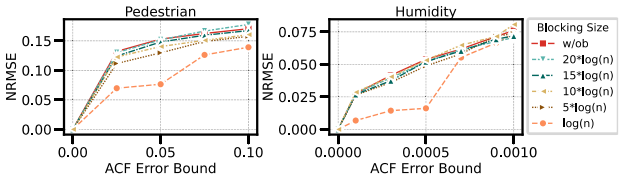


Figure 10: NRMSE using Blocking.

the results for an error bound of 0.01 for all the small datasets and 0.001 for the rest. We also terminate the algorithms once we reach a compression ratio of 10. PMC, SWING, and SP are implemented in Zig [111], DFT uses NumPy [35], while SWAB, PIP, TP, VW, and CAMEO are implemented in Cython. Ultimately, all implementations run as native code, making their comparison broadly possible. We run SWAB with a window size of 50% on the small datasets and 100% on the big datasets. Table 3 shows the runtime for all baselines and CAMEO as we increase the blocking hops from 1 to $10\log n$, and without blocking (*w/ob*, i.e., full coverage of the series).

Runtime Analysis: CAMEO's single-threaded implementation performs comparably to other line simplification baselines when using a single hop for blocking. As the number of hops increases, the execution time increases slightly sublinearly. While its execution time increases with more hops, this trade-off enables CAMEO to achieve significantly higher compression ratios. Removing blocking (*w/ob*) makes CAMEO infeasible for real-life applications. Among the baselines, PMC and DFT are the fastest, which is expected considering PMC linear time complexity and DFT's highly optimized implementation. However, they still have the limitation of requiring *trial-and-error* exploration for preserving bounds on the ACF. Finally, TP's initial phase, which preserves only the turning points, positively impacts its execution time, albeit risking not meeting the error-bound guarantee on the ACF. Overall, CAMEO is the preferred choice when optimizing compression ratio over speed while providing strong guarantees on the ACF deviation.

Table 4: Decompression Times (ms).

Dataset	PMC	SWING	SP	DFT	CAMEO
AUSElecDem	19	17	14	20	12
Humidity	26	23	18	33	21
IRBioTemp	80	77	60	97	54
SolarPower	88	79	66	352	55

PACF Preservation Runtime Analysis: We also evaluate CAMEO when preserving the PACF. The results show that while its compression ratio is still superior to the baselines, preserving the PACF entails a significantly higher execution time. For example, when running CAMEO on ElecPower, with blocking at $10\log n$, we obtain an execution time of 2.6 seconds, around 6x slower than preserving the ACF in Table 3. This increased execution time is due to the quadratic execution time of DL recursion with complexity $O(L^2)$, computed multiple times per iteration. In future work, we will focus on preserving specific lags to enhance execution speed without sacrificing forecasting accuracy.

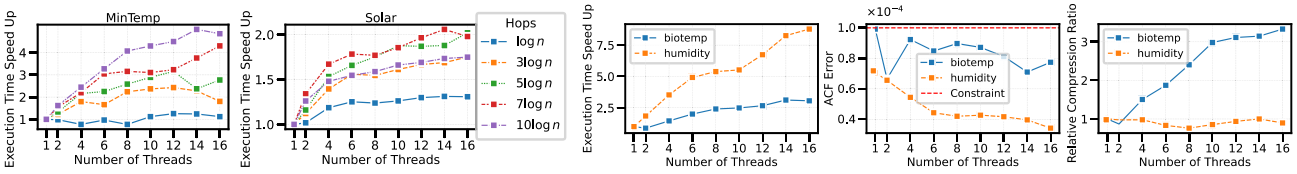
5.6 Decompression Time

In a fifth series of experiments, we evaluate decompression time. Specifically, we compute the execution time of the linear interpolation used as the decompression strategy for CAMEO while directly measuring the decompression times for the other lossy compressors after achieving a 10x compression ratio. Table 4 presents the results in milliseconds, with CAMEO representing all line simplification methods due to their decompression runtime. The results show that CAMEO achieves significantly faster decompression than the baselines, which is particularly important in scenarios where quick decompression is critical. Interestingly, DFT has the slowest decompression time, contrasting its fast compression performance. This discrepancy arises because the decompression logic of other baselines is much simpler, while DFT has a complexity of $O(n\log n)$.

5.7 Parallelization Strategies

We evaluate CAMEO's parallelization strategies across four of the datasets: MinTemp, Humidity, IRBioTemp, and Solar in order to cover various sizes and ACF lags.

Fine-grained Parallelization: We run CAMEO with the objective from Equation (6) fixing the compression ratio to 10 and thus, making the execution time comparable across different configurations. Figure 11(a) shows the relative improvements



(a) Fine-grained (b) Coarse-grained

Figure 11: Results with Different Parallelization Strategies.

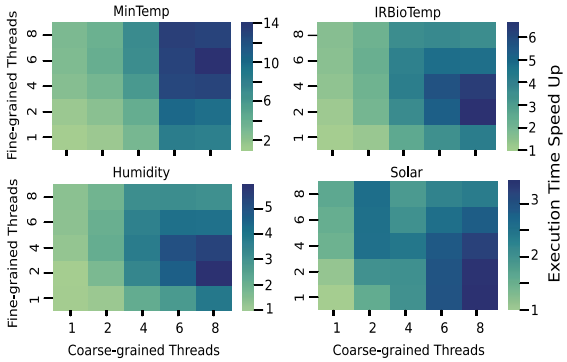


Figure 12: Joint Fine- and Coarse-Grained Parallelization.

when applying fine-grained parallelization, with varying blocking sizes and the number of threads. MinTemp, with the larger ACF (365 lags), achieves the highest speedup of 4x at a hop size of 10 log n using 8 threads. We observe speedups across all blocking configurations except $log(n)$ (12 points), where parallelization overheads dominate. In contrast, the Solar dataset exhibits a modest speedup of approximately 2x at 14 threads. This discrepancy is due to MinTemp's larger number of lags compared to Solar's 24. The more lags, the more workload per thread, which makes parallelization more effective.

Coarse-Grained Parallelization: To showcase the benefits of coarse-grained parallelization, we use the Humidity and IRBioTemp datasets. For both, we set the ACF error bound to 1e-4 (Equation (4)) and record the compression ratio and impact on the overall ACF. Figure 11(b) shows the results for increasing the number of threads. The compression ratio is shown relative to single-threaded execution. Humidity achieves significant runtime reductions, up to an 8x speedup with minimal impact on the compression ratio. IRBioTemp, however, shows a 2.5x runtime improvement coupled with a notable increase (up to 3x) in compression ratio. This compression ratio improvement occurs because local partitions independently achieve better compression due to dataset-specific patterns.

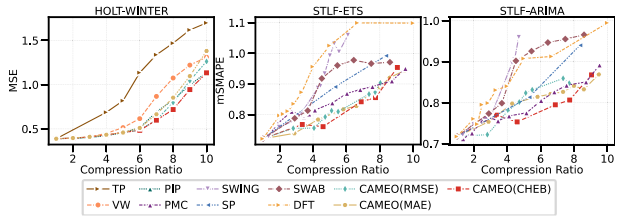
Hybrid Parallelization: Figure 12 shows the speedup when combining fine- and coarse-grained parallelization. Using a blocking size of 10 log n, MinTemp shows the most significant improvement among all datasets, with a speedup of up to 14x when using 6 fine-grained and 8 coarse-grained threads. This speedup represents an execution time reduction from 2.75 seconds (see Table 3) to 0.2 seconds. Other datasets show smaller but good speedups as well, primarily driven by coarse-grained parallelism. This result aligns with Figure 11(a) as these datasets only need to preserve an ACF of 24 lags. However, increasing the number of lags from 24 to 168 notably benefits from fine-grained parallelism, resulting in improved hybrid speedups (e.g., 15x and 13x improvement for IRBioTemp and Humidity, respectively).

Discussion: The results show the complementary strengths of CAMEO's parallelization strategies. Fine-grained parallelism provides deterministic and straightforward runtime improvements without affecting compression quality or accuracy. However, the speedup is inherently sub-linear due to thread synchronization overhead and limited per-thread workloads. In some cases, particularly with smaller blocking sizes or fewer lags, fine-grained parallelization can even lead to performance degradation. Coarse-grained parallelism provides good scalability but introduces slight variability in compression ratios due to processing partitions independently. If predictable compression ratios are desired, adopting the compression-centric strategy (Equation (6)) gives users explicit control over the compression ratio. Together, these strategies allow users to tune CAMEO's runtime.

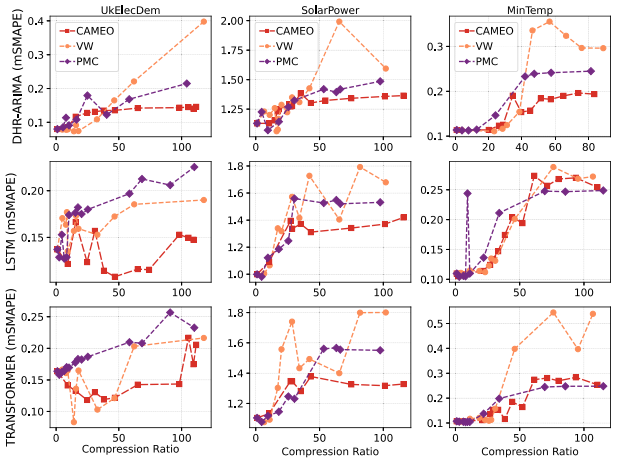
5.8 Impact on Time Series Forecasting

We now investigate our original hypothesis: preserving the ACF during compression is beneficial for forecasting analytics. To test this hypothesis, we conduct four complementary experiments. These experiments use different preprocessing steps, datasets, and forecasting models to cover multiple analytical scenarios. Experiments (1) and (2) leverage linear forecasting models applied to subsets of the Pedestrian dataset [42] comprising 66 time series of varying lengths and statistical properties. Experiment (1) involves controlled preprocessing (segmentation, Box-Cox transform, and standardization) generating a total of 3,400 series. In this setting, we explicitly vary the compression ratio from 2 to 10 and test multiple CAMEO's ACF preservation metrics (MAE, RMSE, and Chebyshev Distance (CHEB) [13]) against baseline line simplification methods (VW, TP, PIP). In contrast, Experiment (2) closely replicates the standard forecasting benchmark established by Godahewa et al. [42], applying STL-ETS and STL-ARIMA models without additional preprocessing. Since standard lossy compressors do not allow setting the compression ratios, we evaluate these compressors separately under their typical trial-and-error conditions until reaching a similar compression ratio of 10. Experiment (3) focuses explicitly on datasets (UKElecDem, SolarPower, MinTemp) with high seasonal strength [100]. We evaluate Dynamic Harmonic Regression (DHR), ARIMA [49, 109], LSTM [41], and Transformer [97] models. Finally, Experiment (4) reuses the setup of Experiment (2), and compares CAMEO with SZ3 and Mix-Piece using the Bits/value metric.

Linear Forecasting Models Accuracy (1/2): Figure 13(a) compares the forecasting accuracy under moderate compression ratios (up to 10x) across the CAMEO configurations and baselines. We only report MSE and mSMAPE [42], but we observed similar or even better results across other metrics. The results show that CAMEO variants outperform traditional line-simplification and lossy baselines. Among CAMEO's variants, CAMEO(CHEB) was the best in both subexperiments. Intuitively, CHEB distributes the ACF error evenly, avoiding distortion on specific lags. Among



(a) Linear Forecasting Models Accuracy under Moderate Compression.



(b) Forecasting Accuracy on Highly Seasonal Time Series.

Figure 13: Impact on Forecasting Accuracy of CAMEO and Baselines as the Compression Ratio Increases under Different Configurations, Quality Metrics, and Time Series.

the baselines, PMC shows better results than the rest of the lossy compression baselines. Accordingly, we select PMC, and VW (as a close line simplification baseline) to conduct additional tests with higher compression ratios and non-linear models.

In-Depth Forecasting Accuracy Analysis (3): Figure 13(b) compares the impact on forecasting accuracy using non-linear and linear models of CAMEO, VW, and PMC. Across the three highly seasonal datasets, CAMEO consistently preserves forecasting accuracy, even under aggressive compression of 100x. We attribute this robustness to CAMEO's effective point selection strategy, which retains key seasonal patterns. Overall, the results show that preserving the ACF is beneficial for forecasting accuracy even when using non-linear models. To statistically validate these results, we conducted a Friedman test followed by a Nemenyi post-hoc analysis [25] interpolating forecasting errors across compression ratios from 2x to 100x. CAMEO achieved the lowest median forecasting error and the lowest mean rank (1.45), significantly outperforming both PMC and VW according to the Nemenyi test. We obtain the same results when applying the test using RMSE as the metric.

Forecasting Accuracy vs. Bits/value (4): Figure 14 illustrates the trade-off between forecasting accuracy and compression efficiency (measured in Bits/Value) for the lossy compressors CAMEO, SZ3, and MXP, evaluated on the STLF-ETS and STLF-ARIMA models. We refined our experimental setup from Section 5.1 by applying a simple bit-packing step before using Zstd to compute CAMEO's effective Bits/Value. The results show that all lossy compressors maintain forecasting accuracy while significantly reducing the number of bits per value to about half compared to the lossless baselines CHIMP and Zstd. Notably,

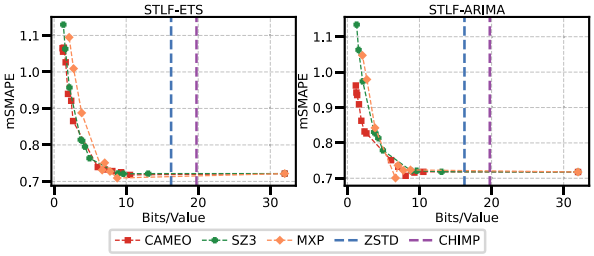


Figure 14: Impact on the Forecasting Accuracy for Linear Models versus the Bits/Value Compression Metric.

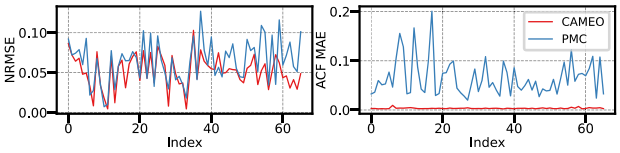


Figure 15: NRMSE and ACF Impact on Pedestrian.

MXP and CAMEO even improve the forecasting accuracy in a few instances. For STLF-ETS, SZ3 and CAMEO yield comparable results, whereas for ARIMA, CAMEO clearly outperforms the other compressors. More advanced physical compression layouts are orthogonal to logical line simplification, and could further reduce CAMEO's bits/value without affecting the forecasting accuracy, which remains an interesting direction for future work.

Discussion: These results demonstrate that CAMEO consistently outperforms the baselines across datasets, forecasting models, and evaluation metrics. In more detail, we compared CAMEO and PMC on the Pedestrian dataset at a moderate compression ratio (5x) across the 66 time series. We evaluated two dimensions: the decompression error (NRMSE), and the ACF preservation (MAE). As shown in Figure 15, CAMEO achieves slightly lower NRMSE than PMC on average, yielding low pointwise reconstruction errors. Additionally, CAMEO's ACF distortions are at least an order of magnitude smaller than for PMC. This combination of pointwise reconstruction accuracy and preservation of temporal dependencies leads to CAMEO's very good forecasting accuracy. Error propagation from compression to forecasting models is not straightforward to predict though. We observe a few cases where the baselines achieve better forecasting results in Figure 13. Therefore, there is "no such thing as a free lunch" [103], and CAMEO should be just regarded as an additional tool.

5.9 Impact on Anomaly Detection

Finally, we investigate two alternative hypotheses: (1) preserving the ACF during compression is beneficial for anomaly detection, and (2) CAMEO execution time is amortized if the downstream analytics can exploit the resulting (much smaller) irregular time series. First, we use the UCR dataset [104] consisting of 250 time series and the Matrix Profile (MP) algorithm [107]. We measure the accuracy using the UCR-score [104], where higher scores indicate better detection. We detect all discords using the MP algorithm with segment sizes ranging from 75 to 125 and select the one with the maximum distance [87]. Second, we implement an algorithm that calculates the Euclidean distance between all pairs of segments of size m— MP's core idea—over the irregular time series (iMP). iMP avoids materializing the data and directly computes distances with linear interpolation during decompression using the remaining m' points per segment. This method reduces

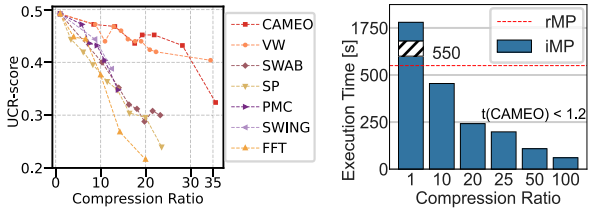


Figure 16: (left) **Impact on the Anomaly Detection Accuracy as the Compression Ratio Increases.** (right) **Execution Time of the MP Algorithm over the Irregular Time Series.**

the complexity from $O(N^2m)$—the complexity of naive implementation over regular time series (rMP)—to $O(N^2m')$, where $m' \ll m$ and N is the time series length. We conduct tests on synthetically generated data of size 2^p, where p ranges from 10 to 16, and segment size $m = 150$.

Accuracy Results Analysis: Figure 16 (left) illustrates the UCR-score as the compression ratio increases. The results show that CAMEO preserves the UCR score more effectively than lossy compression baselines, achieving a compression ratio of $\approx$ 28x while minimally impacting accuracy. The results support our hypothesis (1) that preserving the ACF is advantageous for forecasting analytics and other applications such as anomaly detection. However, it is noteworthy that the effectiveness of preserving the ACF diminishes at higher compression ratios, as shown in Figure 16 (left) beyond a 30x ratio. This trend is likely because removing extreme outliers has a negligible impact on the ACF since these points do not significantly affect temporal dependencies. Thus, removing these points negatively affects the Euclidean distance computation. In contrast, the VW strategy implicitly retains such points, as an outlier typically has a significant triangular area.

Execution Time Results Analysis: Figure 16 (right) displays the execution time results for iMP as the compression ratio increases, specifically for $p = 14$. The results reveal a significant reduction in execution time, decreasing from 550 seconds with the naive implementation rMP to 250 at a compression ratio of 20x. Furthermore, the compression process with CAMEO is negligible, requiring only 0.94 seconds to complete at that compression ratio and less than 1.2 seconds at ratio 100x. Experiments for different p values show similar results. This improvement of the end-to-end runtime of the analytics, coupled with the minimal impact on detection accuracy, highlights the substantial benefits of using compression algorithms like CAMEO that preserve key statistical features while significantly reducing the data size.

6 Additional Related Work

Here we position CAMEO in the context of additional work, including time series segmentation, representation, lossless compression, and matrix compression.

Time Series Segmentation and Representation: In contrast to lossy time series compression, time series segmentation and representation techniques focus on extracting patterns while also reducing data. Prominent methods like SWAB [57] segment time series to enhance analytics, while SAX [69] transforms data into symbolic representations for efficient indexing and pattern recognition. More recently, GRAIL [81] introduces compact representations preserving user-specific comparison functions, while ASAX_EN [26] proposes segmentation based on entropy measurement to maximize information gain. While these methods reduce storage and retain useful patterns, these works do not address forecasting analytics. In contrast, CAMEO's preservation of the ACF retains good accuracy of forecasting models even at high compression levels. There are comprehensive surveys on these two topics [94, 99].

Lossless Time Series Compression: Work on lossless time series compression yielded increasing improvements and shows a balance of good compression ratios and computational efficiency [1, 6, 12, 86, 106]. Gorilla [82], widely known for its implementation within Facebook's time-series database, is simple yet very efficient, making it amenable for real-time applications. Gorilla's XOR operator has recently inspired the Chimp [67] and Elf [66] lossless compression algorithms. Both methods preserve Gorilla's linear time complexity while improving its compression ratio for time series without many repeating values. However, the achieved compression ratios of these methods are still limited. In contrast, CAMEO is positioned between lossless and lossy compression by preserving key statistical features while yielding very good compression ratios.

Lossless Matrix and Workload-aware Compression: Besides lossy matrix compression, which are mainstream in ML model training and inference, there is also work on lossless matrix compression. Examples are compressed linear algebra [30, 31] and tuple-oriented coding [65], which also apply to time series data but only in combination with binning or quantization. Recent work also explored workload-aware lossless compression [4] and workload-aware dimensionality reduction [91], which are related to compressing for downstream analytics. A holistic, unified strategy that dynamically applies the principles of lossless compression, lossy compression, and workload-aware compression (especially for statistical features) is non-existent so far.

7 Conclusions

We introduced CAMEO, a lossy time series compression framework that guarantees a user-provided maximum deviation of the original ACF/PACF. Inspired by line simplification methods, CAMEO iteratively removes points while continuously validating the error constraint. To improve efficiency, CAMEO utilizes incremental maintenance, blocking, and parallelization strategies. Based on our experimental evaluation, we draw the following conclusions: 1) CAMEO obtains higher compression ratios than existing line-simplification techniques while keeping the same ACF deviation. 2) CAMEO provides a competitive alternative to the well-known lossy compressors SWAB, PMC, SWING, and SP with better compression ratios and direct guarantees on the ACF. 3) Preserving the ACF during compression yields better accuracy across different time series analytics. Together, these results make a great case for lossy compression under awareness of statistical properties and downstream applications, which helps to remove trust concerns and tedious semi-manual trial-and-error exploration. Future work includes the extension of CAMEO to other time series properties such as entropy (which might be highly correlated to classification or clustering), and seasonal-strength, as well as runtime improvements for high-performance compression and approximation analyses [96].

Acknowledgments

We thank our anonymous reviewers for constructive suggestions, and acknowledge funding from the German Federal Ministry of Education and Research (under research grant BIFOLD25B) and MORE project funded by EU Horizon 2020 (grant no. 957345).

Artifacts

The code, datasets, artifacts, and instructions for running our method can be found at https://github.com/cmcuza/cameo and a reproducibility repository can be found at https://github.com/damslab/reproducibility/tree/master/edbt2026-CAMEO.

References

[1] Azim Afroozeh, Leonardo X. Kuffo, and Peter A. Boncz. 2023. ALP: Adaptive Lossless floating-Point Compression. *Proc. ACM Manag. Data* 1, 4 (2023), 230:1–230:26. doi:10.1145/3626717

[2] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. 1974. Discrete cosine transform. *IEEE Trans. on Computers* 100, 1 (1974), 90–93.

[3] Depei Bao and Zehong Yang. 2008. Intelligent stock trading system by turning point confirming and probabilistic reasoning. *Expert Syst. Appl.* 34, 1 (2008), 620–627. doi:10.1016/j.eswa.2006.09.043

[4] Sebastian Baunsgaard and Matthias Boehm. 2023. AWARE: Workload-aware, Redundancy-exploiting Linear Algebra. *Proc. ACM Manag. Data* 1, 1 (2023), 2:1–2:28. doi:10.1145/3588682

[5] Kristiyan Blagov, Carlos Enrique Muñiz-Cuza, and Matthias Boehm. 2025. Fast, Parameter-free Time Series Anomaly Detection. In *BTW*. 453–474.

[6] 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

[7] Tulika Bose, Soma Bandyopadhyay, Sudhir Kumar, Abhijan Bhattacharyya, and Arpan Pal. 2016. Signal Characteristics on Sensor Data Compression in IoT -An Investigation. In *SECON*. doi:10.1109/SAHCN.2016.7733016

[8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[9] Chiranjeeb Buragohain, Nisheeth Shrivastava, and Subhash Suri. 2007. Space Efficient Streaming Algorithms for the Maximum Error Histogram. In *ICDE*. 1026–1035. doi:10.1109/ICDE.2007.368961

[10] Peter A. Businger and Gene H. Golub. 1969. Algorithm 358: singular value decomposition of a complex matrix [F1, 4, 5]. *CACM* 12, 10 (1969), 564–565. doi:10.1145/363235.363249

[11] Yuhan Cai and Raymond T. Ng. 2004. Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials. In *SIGMOD*. 599–610. doi:10.1145/1007568.1007636

[12] Giuseppe Campobello, Antonino Segreto, Sarah Zanafi, and Salvatore Serrano. 2017. RAKE: A simple and efficient lossless compression algorithm for the Internet of Things. In *EUSIPCO*. doi:10.23919/EUSIPCO.2017.8081677

[13] Cyrus D Cantrell. 2000. *Modern mathematical methods for physicists and engineers*. Cambridge University Press.

[14] Franck Cappello, Sheng Di, and Ali Murat Gok. 2020. Fulfilling the Promises of Lossy Compression for Scientific Applications. In *SMC*, Vol. 1315. 99–116. doi:10.1007/978-3-030-63393-6_7

[15] Shubham Chandak, Kedar Tatwawadi, Chengtao Wen, Lingyun Wang, Juan Aparicio Ojea, and Tsachy Weissman. 2020. LFZip: Lossy Compression of Multivariate Floating-Point Time Series Data via Improved Prediction. In *DCC*. 342–351. doi:10.1109/DCC47342.2020.00042

[16] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. 2007. Indexable PLA for Efficient Similarity Search. In *PVLDB*. 435–446. http://www.vldb.org/conf/2007/papers/research/p435-chen.pdf

[17] Giacomo Chiarot and Claudio Silvestri. 2023. Time Series Compression Survey. *ACM Comput. Surv.* 55, 10 (2023), 198:1–198:32. doi:10.1145/3560814

[18] Fu Lai Korris Chung, Tak-Chung Fu, Wing Pong Robert Luk, and Vincent To Yee Ng. 2001. Flexible time series pattern matching based on perceptually important points. In *IJCAI (Workshop)*.

[19] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. 1990. STL: A seasonal-trend decomposition. *J. Off. Stat* 6, 1 (1990), 3–73. http://www.nniiem.ru/file/news/2016/stl-statistical-model.pdf

[20] Yann Collet and Przemyslaw Skibinski. 2015. Zstandard - Fast real-time compression algorithm. https://facebook.github.io/zstd/.

[21] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301. https://community.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf

[22] Patrick Damme et al. 2022. DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In *CIDR*. https://www.cidrdb.org/cidr2022/papers/p4-damme.pdf

[23] Patrick Damme, Dirk Habich, Juliana Hildebrandt, and Wolfgang Lehner. 2017. Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). In *EDBT*. 72–83. doi:10.5441/002/edbt.2017.08

[24] Julio Cesar Stacchini de Souza, Tatiana Mariano Lessa Assis, and Bikash Chandra Pal. 2017. Data Compression in Smart Distribution Systems via Singular Value Decomposition. *IEEE Trans. Smart Grid* 8, 1 (2017), 275–284. doi:10.1109/TSG.2015.2456979

[25] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 7 (2006), 1–30.

[26] Lamia Djebour, Reza Akbarinia, and Florent Masseglia. 2023. Variable-Size Segmentation for Time Series Representation. *Trans. Large Scale Data Knowl. Centered Syst.* 53 (2023), 34–65. doi:10.1007/978-3-662-66863-4_2

[27] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Int. J. for Geog. Inf. and Geov.* 10, 2 (1973). doi:10.1002/9780470669488.ch2

[28] J. Durbin. 1960. The Fitting of Time-Series Models. *Review of the International Statistical Institute* 28, 3 (1960).

[29] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A time-series compression technique and its application to the smart grid. *PVLDB* 24, 2 (2015), 193–218. doi:10.1007/s00778-014-0368-8

[30] Ahmed Elgohary, Matthias Boehm, Peter J Haas, Frederick R Reiss, and Berthold Reinwald. 2016. Compressed linear algebra for large-scale machine learning. *VLDB J.* 9, 12 (2016), 960–971. doi:10.1007/s00778-017-0478-1

[31] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2018. Compressed linear algebra for large-scale machine learning. *VLDB J.* 27, 5 (2018), 719–744. doi:10.1007/s00778-017-0478-1

[32] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. 2009. Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. *PVLDB* 2, 1 (2009), 145–156. doi:10.14778/1687627.1687645

[33] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. 2022. Window Size Selection in Unsupervised Time Series Analytics: A Review and Benchmark. In *ECML PKDD, AALTD*, Vol. 13812. 83–101. doi:10.1007/978-3-031-24378-3_6

[34] Charles R. Harris et. al. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362. doi:10.1038/s41586-020-2649-2

[35] Charles R. Harris et al. 2020. Array programming with NumPy. *Nat.* 585 (2020), 357–362. doi:10.1038/S41586-020-2649-2

[36] Robert W Floyd. 1964. Algorithm 245: treesort. *CACM* 7, 12 (1964).

[37] Electricity System Operator for Great Britain. 2023. Historic Demand Data 2021. https://www.nationalgrideso.com/data-portal/historic-demand-data/historic_demand_data_2021.

[38] Tak-Chung Fu, Korris Fu-Lai Chung, Robert Wing Pong Luk, and Chak-man Ng. 2008. Representing financial time series based on data point importance. *Eng. Appl. Artif. Intell.* 21, 2 (2008). doi:10.1016/j.engappai.2007.04.009

[39] Tak-chung Fu, Fu-lai Chung, Robert Luk, and Chak-man Ng. 2004. A specialized binary tree for financial time series representation. In *TDM-SIGKDD*.

[40] Tak-chung Fu, Ying-kit Hung, and Fu-lai Chung. 2017. Improvement algorithms of perceptually important point identification for time series data mining. In *ISCMI*. 11–15. doi:10.1109/ISCMI.2017.8279589

[41] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. 2000. Learning to Forget: Continual Prediction with LSTM. *Neural Comput.* 12, 10 (2000), 2451–2471. doi:10.1162/089976600300015015

[42] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, Rob J. Hyndman, and Pablo Montero-Manso. 2021. Monash Time Series Forecasting Archive. In *NeurIPS*, Vol. 1. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/eddea82ad2755b24c4e168c5fc2ebd40-Abstract-round2.html

[43] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE Transactions on Information Theory* 44, 6 (1998), 2325–2383. doi:10.1109/18.720541

[44] Taimur Hafeez and Gavin McArdle. 2022. Using Dynamic Perceptually Important Points for Data Reduction in IoT. In *Internet of Things*. 33–39. doi:10.1145/3494322.3494327

[45] Georges Hebrail and Alice Berard. 2012. Individual household electric power consumption. doi:10.24432/C58K54 UCI MLR.

[46] Jose Antonio Martinez Heras and Alessandro Donati. 2013. Fractal resampling: time series archive lossy compression with guarantees. *PV* ESRIN (2013).

[47] Gregor Hollmig et al. 2017. An evaluation of combinations of lossy compression and change-detection approaches for time-series data. *Inf. Syst.* 65 (2017), 65–77. doi:10.1016/j.is.2016.11.001

[48] Nguyen Quoc Viet Hung, Hoyoung Jeung, and Karl Aberer. 2013. An Evaluation of Model-Based Approaches to Sensor Data Compression. *TKDE* 25, 11 (2013), 2434–2447. doi:10.1109/TKDE.2012.237

[49] Rob J Hyndman and George Athanasopoulos. 2018. *Forecasting: principles and practice*. OTexts. http://OTexts.com/fpp3

[50] Tetsunari Inamura, Hiroaki Tanie, and Yoshihiko Nakamura. 2003. Keyframe compression and decompression for time series data based on the continuous hidden Markov model. In *IROS*, Vol. 2. 1487–1492.

[51] InfluxData. 2023. InfluxDB. https://www.influxdata.com/.

[52] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2017. Time Series Management Systems: A Survey. *IEEE Trans. KDE* 29, 11 (2017), 2581–2600. doi:10.1109/TKDE.2017.2740932

[53] Søren Kejser Jensen, Christian Thomsen, and Torben Bach Pedersen. 2023. ModelarDB: Integrated Model-Based Management of Time Series from Edge to Cloud. *TLSDKS* 53 (2023), 1–33. doi:10.1007/978-3-662-66863-4_1

[54] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. M4: A Visualization-Oriented Time Series Data Aggregation. *PVLDB* 7, 10 (2014), 797–808. doi:10.14778/2732951.2732953

[55] Samsul Ariffin Abdul Karim et al. 2011. Wavelet Transform and Fast Fourier Transform for signal compression: A comparative study. In *ICEDSA*. 280–285. doi:10.1109/ICEDSA.2011.5959031

[56] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *SIGMOD*. 151–162. doi:10.1145/375663.375680

[57] Eamonn J. Keogh, Selina Chu, David M. Hart, and Michael J. Pazzani. 2001. An Online Algorithm for Segmenting Time Series. In *ICDM*. 289–296. doi:10.1109/ICDM.2001.989531

[58] Abdelouahab Khelifati, Mourad Khayati, and Philippe Cudré-Mauroux. 2019. Corad: Correlation-aware compression of massive time series using sparse dictionary coding. In *Big Data*. 2289–2298. doi:10.1109/BigData47090.2019.9005580

[59] Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2023. Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. *PVLDB* 16, 8 (2023), 1910–1922. doi:10.14778/3594512.3594521

[60] Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2024. Flexible grouping of linear segments for highly accurate lossy compression of time series data. *The VLDB Journal* 33, 5 (2024), 1569–1589. doi:10.1007/s00778-024-00862-z

[61] Maciej Krawczak and Grazyna Szkatula. 2014. An approach to dimensionality reduction in time series. *Inf. Sci.* 260 (2014). doi:10.1016/j.ins.2013.10.037

[62] Barry J Kronenfeld, Lawrence V Stanislawski, Barbara P Buttenfield, and Tyler Brockmeyer. 2020. Simplification of polylines by segment collapse: Minimizing areal displacement while preserving area. *Int. J. of Cartography* 6, 1 (2020), 22–46. doi:10.1080/23729333.2019.1631535

[63] Iosif Lazaridis and Sharad Mehrotra. 2003. Capturing sensor-generated time series with quality guarantees. In *ICDE*. 429–440.

[64] Michal Lewandowski, Jesús Martínez del Rincón, Dimitrios Makris, and Jean-Christophe Nebel. 2010. Temporal Extension of Laplacian Eigenmaps for Unsupervised Dimensionality Reduction of Time Series. In *ICPR*. 161–164. doi:10.1109/ICPR.2010.48

[65] Fengan Li et al. 2019. Tuple-oriented Compression for Large-scale Mini-batch Stochastic Gradient Descent. In *SIGMOD*. 1517–1534. doi:10.1145/3299869.3300070

[66] Ruiyuan Li, Zheng Li, Yi Wu, Chao Chen, and Yu Zheng. 2023. Elf: Erasing-Based Lossless Floating-Point Compression. *PVLDB* 16, 7 (2023), 1763–1776. doi:10.14778/3587136.3587149

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

[68] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2023. SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors. *IEEE Transactions on Big Data* 9, 2 (2023), 485–498. doi:10.1109/TBDATA.2022.3201176

[69] Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *SIGMOD*. 2–11. doi:10.1145/882082.882086

[70] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.* 15, 2 (2007), 107–144. doi:10.1007/s10618-007-0064-z

[71] Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. 2015. Piecewise linear approximation of streaming time series data with max-error guarantees. In *ICDE*. 173–184. doi:10.1109/ICDE.2015.7113282

[72] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. 2015. Unlocking the Potential of the Internet of Things. *McKinsey Global Institute* 1 (2015).

[73] Alice Marascu et al. 2014. TRISTAN: Real-time analytics on massive time series using sparse dictionary compression. In *BigData*. 291–300. doi:10.1109/BigData.2014.7004244

[74] Aekyeung Moon, Jaeyoung Kim, Jialing Zhang, and Seung Woo Son. 2018. Evaluating fidelity of lossy compression on spatiotemporal data from an IoT enabled smart farm. *Comput. Electron. Agric.* 154 (2018), 304–313. doi:10.1016/j.compag.2018.08.045

[75] Pedro A Morettin. 1984. The Levinson algorithm and its applications in time series analysis. *Int. Statistical Review* (1984), 83–92.

[76] Carlos Enrique Muñiz-Cuza, Søren Kejser Jensen, Jonas Brusokas, Nguyen Ho, and Torben Bach Pedersen. 2024. Evaluating the Impact of Error-Bounded Lossy Compression on Time Series Forecasting. In *EDBT*. OpenProceedings.org, 650–663. doi:10.48786/EDBT.2024.56

[77] National Ecological Observatory Network (NEON). 2021. IR biological temperature (DP1.00005.001). doi:10.48443/JNWY-B177

[78] National Ecological Observatory Network (NEON). 2023. Relative humidity (DP1.00098.001). doi:10.48443/G2J6-SR14

[79] Idoia Ochoa, Mikel Hernaez, Rachel L. Goldfeder, Tsachy Weissman, and Euan A. Ashley. 2017. Effect of lossy compression of quality scores on variant calling. *Briefings Bioinform.* 18, 2 (2017), 183–194. doi:10.1093/bib/bbw011

[80] Australian Bureau of Meteorology. 2023. Minimum Daily Temperatures in Melbourne (1981-1990). https://www.kaggle.com/datasets/paulbrabban/daily-minimum-temperatures-in-melbourne.

[81] John Paparrizos and Michael J. Franklin. 2019. GRAIL: Efficient Time-Series Representation Learning. *PVLDB* 12, 11 (2019), 1762–1777. doi:10.14778/3342263.3342648

[82] 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. *PVLDB* 8, 12 (2015), 1816–1827. doi:10.14778/2824032.2824078

[83] James Pope, Antonis Vafeas, Atis Elsts, George Oikonomou, Robert J. Piechocki, and Ian Craddock. 2018. An accelerometer lossless compression algorithm and energy analysis for IoT devices. In *WCNC*. 396–401. doi:10.1109/WCNCW.2018.8368985

[84] U Rammer. 1972. An iterative procedure for the polygonal approximation of plane closed curves. *Comput. Graph. Image Process.* (1972), 244–256. doi:10.1016/S0146-664X(72)80017-0

[85] Paulo Raposo. 2013. Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. *Cartog. and Geog. Inf. Science* 40, 5 (2013), 427–443. doi:10.1080/15230406.2013.803707

[86] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. 2006. Fast Lossless Compression of Scientific Floating-Point Data. In *DCC*. 133–142. doi:10.1109/DCC.2006.35

[87] Ferdinand Rewicki, Joachim Denzler, and Julia Niebling. 2023. Is It Worth It? Comparing Six Deep and Classical Methods for Unsupervised Anomaly Detection in Time Series. *Applied Sciences* 13, 3 (2023). doi:10.3390/app13031778

[88] Jin Shieh and Eamonn J. Keogh. 2008. *i*SAX: indexing and mining terabyte sized time series. In *SIGKDD*. 623–631. doi:10.1145/1401890.1401966

[89] Yain-Whar Si and Jiangling Yin. 2013. OBST-based segmentation approach to financial time series. *Eng. Appl. Artif. Intell.* 26, 10 (2013), 2581–2596. doi:10.1016/j.engappai.2013.08.015

[90] Qizhou Sun and Yain-Whar Si. 2020. An Efficient Segmentation Method: Perceptually Important Point with Binary Tree. In *DEXA*, Vol. 12392. Springer, 350–365. doi:10.1007/978-3-030-59051-2_24

[91] Sahaana Suri and Peter Bailis. 2019. DROP: A Workload-Aware Optimizer for Dimensionality Reduction. In *DEEM@SIGMOD*. 1:1–1:10. doi:10.1145/3329486.3329490

[92] Dingwen Tao, Sheng Di, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2019. Z-checker: A framework for assessing lossy compression of scientific data. *Int. J. HPC Appl.* 33, 2 (2019). doi:10.1177/1094342017737147

[93] Seshu Tirupathi et al. 2022. Machine Learning Platform for Extreme Scale Computing on Compressed IoT Data. In *IEEE Big Data*.

[94] Patara Trirat et al. 2024. Universal Time-Series Representation Learning: A Survey. *CoRR* abs/2401.03717 (2024). doi:10.48550/ARXIV.2401.03717 arXiv:2401.03717

[95] Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. 2019. Global Curve Simplification. In *ESA*, Vol. 144. 67:1–67:14. doi:10.4230/LIPIcs.ESA.2019.67

[96] Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. 2020. On optimal polyline simplification using the Hausdorff and Fréchet distance. *J. Comput. Geom.* 11, 1 (2020), 1–25. doi:10.20382/jocg.v11i1a1

[97] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[98] Maheswari Visvalingam and James D Whyatt. 1993. Line generalisation by repeated elimination of points. *The Cartographic J.* 30, 1 (1993), 46–51. doi:10.1179/000870493786962263

[99] Chengyu Wang, Xionglve Li, Tongqing Zhou, and Zhiping Cai. 2024. Unsupervised Time Series Segmentation: A Survey on Recent Advances. *Comp, Mat & Cont* 80, 2 (2024), 2657–2673. doi:10.32604/cmc.2024.054061

[100] Xiaozhe Wang, Kate Smith, and Rob Hyndman. 2006. Characteristic-based clustering for time series data. *Data mining and knowledge Discovery* 13 (2006), 335–364. doi:10.1007/s10618-005-0039

[101] Abdul Wasay, Xinding Wei, Niv Dayan, and Stratos Idreos. 2017. Data Canopy: Accelerating Exploratory Statistical Analysis. In *SIGMOD*. 557–572. doi:10.1145/3035918.3064051

[102] Norbert Wiener. 1930. Generalized harmonic analysis. *Acta mathematica* 55, 1 (1930), 117–258.

[103] David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1, 1 (1997), 67–82.

[104] Renjie Wu and Eamonn J. Keogh. 2022. Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress (Extended Abstract). In *ICDE*. 1479–1480. doi:10.1109/ICDE53745.2022.00116

[105] 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. *PVLDB* 15, 10 (2022), 2148–2160. https://www.vldb.org/pvldb/vol15/p2148-song.pdf

[106] Yuanyuan Yao, Lu Chen, Ziquan Fang, Yunjun Gao, Christian S. Jensen, and Tianyi Li. 2024. *Camel:* Efficient Compression of Floating-Point Time Series. *Proc. ACM Manag. Data* 2, 6 (2024), 227:1–227:26. doi:10.1145/3698802

[107] Chin-Chia Michael Yeh et al. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *ICDM*. 1317–1322. doi:10.1109/ICDM.2016.0179

[108] Jiangling Yin, Yain-Whar Si, and Zhiguo Gong. 2011. Financial time series segmentation based on Turning Points. In *ICSSE*. 394–399. doi:10.1109/ICSSE.2011.5961935

[109] Peter C Young, Diego J Pedregal, and Wlodek Tych. 1999. Dynamic harmonic regression. *Journal of forecasting* 18, 6 (1999), 369–394. doi:10.1002/(SICI)1099-131X(199911)18:6<369::AID-FOR748>3.0.CO;2-K

[110] Feng Zhang et al. 2022. CompressDB: Enabling Efficient Compressed Data Direct Processing for Various Databases. In *SIGMOD*. 1655–1669. doi:10.1145/3514221.3526130

[111] Zig Software Foundation. [n. d.]. The Zig Programming Language. https://ziglang.org/