

Article

A Two-Stage Algorithm for Time Series Compression: ARIMA-Based Pre-Compression and Reinforcement Learning Optimized Chunking

Miao Chi, Su Pan *, Jiaji Feng, Zhe Ding and Zhaowei Zhang

School of Internet of Things, Nanjing University of Posts and Telecommunications, Sanpailou Campus, Nanjing 210003, China; 2023070804@njupt.edu.cn (M.C.); 1223076628@njupt.edu.cn (J.F.); 2024070902@njupt.edu.cn (Z.D.); zwzhang@njupt.edu.cn (Z.Z.)

* Correspondence: supan@njupt.edu.cn; Tel.: +86-189-5189-6075

Abstract

The explosive growth of time series gives rise to a large amount of data, which emphasizes the importance of data compression. The data compression not only reduces storage costs but also enhances data transmission efficiency and processing speed. However, traditional compression algorithms usually suffer an insufficient compression ratio and an excessive computational cost. To address these problems above, in this paper, we propose a two-stage compression algorithm for the large-scale time series data. In the first stage, we transform the time series data into low-volatility residual data by using Autoregressive Integrated Moving Average (ARIMA) modeling and apply adaptive precision quantization to improve compressibility. In the second stage, we implement a reinforcement learning-based compression strategy, which utilizes the Q-learning to select the number of blocks to divide the quantized data segment and achieves compression by storing the same content between the divided data blocks only once and storing the different content separately; and we incorporate the Upper Confidence Bound (UCB) to balance exploration and exploitation in order to track changes in data patterns and improve compression performance. Experimental results demonstrate that our algorithm achieves a higher compression ratio while maintaining a low computational complexity compared with traditional compression algorithms.

Keywords: time series data; time series prediction; reinforcement learning; data compression; upper confidence bound (UCB)

MSC: 94A34; 68T05; 62M10



Academic Editor: Manuel Alberto M. Ferreira

Received: 26 January 2026

Revised: 15 February 2026

Accepted: 28 February 2026

Published: 1 March 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

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

1. Introduction

In IoT, industrial monitoring, intelligent manufacturing and other fields, the continuous generation of massive time-series data puts tremendous pressure on the storage and transmission of the system [1]. Taking power data, sensor readings or high-frequency financial data as paradigms, such time-series data inherently exhibit characteristics of high sampling rates, extended temporal spans, and multi-source heterogeneity [2,3]. How to achieve efficient compression under the premise of guaranteeing data availability and recovery accuracy has become a key topic in current research.

With the rapid growth of time-series data volume, the study of efficient time-series data compression techniques has become a core topic to realize efficient storage, processing and intelligent decision-making. Hence, time-series databases(e.g., InfluxDB [4] and

OpenTSDB [5]) applied different compression schemes to storage and mining of time-series big data efficiently [6].

InfluxDB uses Facebook's Gorilla [7] compression algorithm for floating-point numbers. In the LZ4 [8,9] compression algorithm used in existing time-series databases such as OpenTSDB, each data segment is usually fixed to 64KB. The algorithm uses a sliding window and a hash table to match duplicates within the segment, and replaces duplicated storage with pointer references to reduce redundancy. However, when the data within a data segment change rapidly and exhibit significant differences, they become difficult for LZ4 to effectively detect and utilize the duplicate patterns within the data segment, which results in a reduced compression rate.

To ameliorate this problem, research has been conducted to introduce dynamic segmentation based on time windows instead of relying solely on fixed-size segments, thus better adapting to the storage characteristics of time-series data, which improves the compression rate to a certain extent. Subsequently, the time-series data segments are further divided into smaller compressed blocks (chunking) to serve as the basic processing units to improve the accuracy of redundancy detection and compression efficiency. Based on this chunking approach, researchers have proposed two commonly used compression methods: fixed-size chunking (FSC) [10] and content-defined chunking (CDC) [11].

The Fixed-Size Chunking (FSC) algorithm first divides the data stream into fixed segments based on a set time span, then evenly divides each segment into a fixed number of compressed blocks, ensuring each block size is 4KB or 8KB. This method is simple and fast but does not effectively identify and eliminate duplicate compressed blocks. The Content-Defined Chunking (CDC) algorithm determines block boundaries within each segment using hash-based rules. Since it relies on data content, the number of compressed blocks fluctuates significantly with data pattern changes, affecting compression performance. When data vary drastically, a fixed-length segment is divided into many smaller compressed blocks, increasing index storage overhead. When data are stable, a segment of the same length is divided into fewer larger compressed blocks, reducing deduplication efficiency.

Thus, the number of compressed blocks in a segment is critical, directly impacting deduplication and compression efficiency. Both the FSC and CDC algorithms are executed independently within each data segment and cannot utilize the correlation between data segments. To address this, we utilize the periodicity of time-series data to dynamically adjust the number of compressed blocks across segments. This approach removes the constraint on the data chunking by the independent segmentation, enhancing deduplication and compression efficiency.

To achieve optimal numbers of compressed blocks in each segment, a theoretical approach is to exhaustive enumeration. However, it requires a large computational load, making it impractical. Therefore, we utilize reinforcement learning (RL) to select the number of compressed blocks for each data segment. Reinforcement learning does not need to evaluate the actual reward of this choice to make a decision when doing an action in each data segment (picking the number of compressed blocks), i.e., it evaluates the actual data compression rate that can be obtained with this number of compressed blocks, in which case it would have to exhaust the actual reward of each action. Reinforcement learning makes decisions based on experience and then evaluates the actual reward at the end of the decision and puts it into the experience pool, which is used to correct the next decision, thus avoiding the exhaustive process of decision making and reducing the amount of calculation.

We refer to the algorithm selecting actions with high action rewards from the experience pool as exploitation. However, the estimated action reward is obtained by the average of the compression ratios of the action, and if this action is executed too few times in

the experience pool this reduces the accuracy of the estimate. Therefore, the proposed algorithm, in addition to exploiting experience, needs to select actions that have been chosen less frequently, allowing their action rewards to be updated through more selections. This operation is called exploration because it tries to explore more unknown information. The algorithm needs to handle the balance between exploration and exploitation correctly, so that the reward of each action can be evaluated accurately. Even after the algorithm converges and each action has been executed a sufficient number of times, exploration is still necessary “from time to time” due to potential data pattern drift, ensuring that the algorithm can adapt to these changes.

To solve this problem, the proposed algorithm utilizes the Upper Confidence Bound (UCB) [12] algorithm and defines the Q-value of the action as the upper bound of the action reward. We can prove that the Q-value is negatively correlated with the number of times the action has been selected and positively correlated with its action reward. The proposed algorithm selects the action with the largest Q-value, instead of the action with the largest estimated action reward, and, in this sense, the proposed algorithm balances exploration and exploitation.

Although reinforcement learning (RL) has demonstrated good performance in compression strategy optimization, the original time-series data typically exhibit significant non-stationarity, which poses a major challenge to the RL learning process. To address this issue, we introduce the Auto Regressive Integrated Moving Average (ARIMA) [13,14] model. By treating historical observation data as a continuous time-series sequence and performing global modeling, we effectively remove long-term evolutionary trends and periodic disturbances from the sequence, thereby extracting a residual sequence with lower volatility. This process transforms the compression target into a stationary residual sequence, enabling subsequent compression to be performed based on a more stable sequence. Meanwhile, the global modeling strategy ensures that the ARIMA model comprehensively captures the overall structure of the sequence, avoiding model structure integrity issues caused by local modeling, thereby guaranteeing prediction accuracy.

Although the ARIMA model produces a residual sequence with reduced volatility, these sequences still exist in floating-point form, making them difficult to compress efficiently. Therefore, we further propose an adaptive precision quantization mechanism that dynamically allocates quantization precision to encode the residual sequence into a multi-precision bitstream format compliant with the IEEE 754 standard. This mechanism automatically selects an appropriate quantization precision level based on the amplitude characteristics of the residual sequence, significantly reducing the average bitstream representation length while effectively controlling reconstruction error, thereby achieving efficient compression of the residual sequence. We integrate the ARIMA model prediction with the adaptive precision quantization mechanism into a pre-compression module with both general applicability and compression capabilities. The output bitstream structure is clear and can be integrated with various compression algorithms.

The novelty of our algorithm lies in the synergistic integration of statistical modelling and intelligent decision-making. Unlike traditional algorithms treating predictive modelling and adaptive quantization as separate tasks, we construct a unified two-stage time series compression algorithm: the ARIMA-driven pre-compression module effectively eliminates the non-stationarity of data, enabling reinforcement learning agents to explore the compression strategy space more efficiently. This coupled algorithm resolves the long-standing challenge of applying reinforcement learning to highly volatile real-world time series data.

The main contributions of this paper are as follows:

- (1) We propose a time-series sequence pre-compression module that combines residual modeling and adaptive precision quantization mechanism, which can be used as a pre-processing unit for a variety of compression algorithms.
 - In terms of residual modeling, we introduce the ARIMA model for trend prediction of time-series sequence, which transforms the original sequence into residual sequence with lower volatility and improves the compression friendliness.
 - In precision quantization, we design an adaptive precision quantization mechanism based on residual amplitude, which can effectively compress the bitstream length while controlling the reconstruction error with good compression compatibility by dynamically allocating the quantization precision.
- (2) We propose a main compression strategy based on reinforcement learning and combine it with the pre-compression module to construct a unified two-stage compression algorithm.
 - We propose a reinforcement learning-based compressed block chunking method for time-series sequence, which can achieve close to the optimal compression rate achievable by exhaustive search while greatly reducing the computational effort compared to the exhaustive method.
 - We introduce the Upper Confidence Bound (UCB) to dynamically balance exploration and exploitation. Not only can we achieve more accurate reward estimation but we can also adapt to changes in data patterns, thus improving the compression rate.

2. Related work

In recent years, ARIMA models have been widely used in forecasting tasks due to their capability in modeling non-stationary time-series sequence [15–18]. Banihabib et al. [19] developed a hybrid modeling approach that combines an ARIMA model with exogenous inputs and a neural network in order to improve the water resources forecasting accuracy. Similar studies include combining ARIMA models with various neural network models [20], such as multilayer perceptron and radial basis function networks, aiming to improve the prediction accuracy of future observations.

This prediction-based paradigm was prominently featured in early telecommunications big data compression [21]. Around 2015, research in this field established a mature “step-by-step processing” mode. First, the strong periodicity of telecom traffic was exploited through prediction or differencing to eliminate temporal redundancy by recording only deviations from expectations. Subsequently, spatial redundancy was mitigated by aggregating data from different spatial nodes based on geographical proximity (e.g., the synchronicity of traffic between adjacent base stations). For instance, the SPATE [22] framework introduced decaying factors to smooth random noise. However, a limitation of this approach is its heavy reliance on empirical rules, which makes it difficult for the model to perceive dynamically changing distribution characteristics in non-stationary, irregular data streams, often leading to uncontrollable cumulative errors.

Addressing these limitations, we have shifted the modeling perspective. Rather than employing ARIMA for precise forecasting, we utilize it as a lightweight stationarization method. It serves to eliminate trend components and periodic disturbances from the original sequence, transforming highly volatile raw data into statistically more stationary residual sequences, thereby creating favorable input conditions for the downstream compression stage.

However, sequence stationarization alone is insufficient to handle the complexity of time-series data; dynamically adjusting compression strategies based on residuals remains a challenge. The adaptive capabilities of Reinforcement Learning (RL) in complex decision-

making tasks offer a new trajectory for solving this problem. Currently, RL is widely used for parameter tuning in image compression, inter-frame redundancy modeling in video, and model compression on embedded devices [23]. Some works have also attempted to introduce RL into time-series data processing. For example, Jiang et al. [24] proposed a time-series data compression method based on Deep Q-Networks (DQN). Its core innovation lies in optimizing the organizational structure of compressed data to enable direct querying without decompression, significantly improving random access efficiency. Nevertheless, that method primarily focuses on query performance rather than compression ratio, resulting in limited improvements compared to traditional methods.

Despite the profound research foundation in redundancy elimination for telecom datasets, and the excellent performance of ARIMA and RL in their respective fields, combining them to meet the compression demands of large-scale time-series data remains an underexplored area. The two-stage compression algorithm proposed in this paper leverages ARIMA for stationarization and RL for dynamic policy optimization, which aims to synergistically optimize the compression efficiency and reconfiguration precision, and to fill the research gap in this direction.

3. Pre-Compression Module

3.1. Predictive Modeling Based on ARIMA Models

In the prediction modeling stage of the pre-compression stage, we introduce the ARIMA model to strip redundant information from the original sequence and convert it into a smoother, compression-friendly residual sequence as the target for subsequent compression.

Considering that most measured sequences do not satisfy the stationarity assumption required for ARIMA modeling, we first perform stationarity preprocessing on the original sequence. Let the original sequence be $\{x_t\}_{t=1}^T$, and eliminate its trend term through d -order differencing:

$$x'_t = \nabla^d x_t = (1 - B)^d x_t. \quad (1)$$

where ∇ denotes the differencing operator, B represents the lag operator, $B^k x_t = x_{t-k}$, and the resulting stationary sequence is denoted as $\{x'_t\}_{t=d+1}^T$.

To determine whether a differencing operation is required, we use the Augmented Dickey-Fuller (ADF) unit root test. When the test results indicate that the sequence has stable mean and variance and there is no significant trend term, we recognize the sequence as smooth; otherwise, we will perform the differencing operation to further strip off the trend term until a stationary sequence is obtained.

After obtaining the stationary sequence $\{x'_t\}_{t=d+1}^T$, we construct an ARMA (p, q) model to describe its dependence on historical observations and error terms. This model models the time-series sequence as a linear model composed of an autoregressive (AR) [25] component and a moving average (MA) component. The specific mathematical expression is as follows:

$$x'_t = \sum_{i=1}^p \phi_i x'_{t-i} + \sum_{j=1}^q \theta_j \epsilon_{t-j} + \epsilon_t, \quad (2)$$

where $\phi_1, \phi_2, \dots, \phi_p$ is the model parameter for the autoregressive (AR), $\theta_1, \theta_2, \dots, \theta_q$ is the parameter for the moving average (MA), and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is the Gaussian white noise.

In order to determine the optimal order combination (p, q) , we first analyze the lag characteristics of the sequence by using the autocorrelation function (ACF) plot. By examining whether the autocorrelation coefficients at various lags exceed the 95% confidence bounds, we assess the presence of statistically significant linear dependencies and accordingly constrain the candidate range for model orders. Subsequently, multiple ARMA

models are constructed within a predefined search interval, specifically setting the range as $p, q \in [0, 5]$. This constraint is imposed to balance model complexity with computational efficiency. The Akaike Information Criterion (AIC) values are calculated separately:

$$\text{AIC} = 2k - 2\ln(L), \quad (3)$$

where $k = p + q + 1$ is the number of model parameters and L is the log-likelihood of the model under maximum likelihood estimation. Ultimately, the group of orders with the smallest AIC is selected as the optimal model structure, achieving a balance between model accuracy and complexity.

After determining the optimal model order (p, q) , we use the maximum likelihood estimation (MLE) method to estimate all parameters of the ARMA model. For the autoregressive (AR) model, which describes the linear relationship between the current observation and past observations, an analytical solution can be obtained using the least squares method. The mathematical expression of the p th order autoregressive model (AR(p)) is:

$$x'_t = \phi_1 x'_{t-1} + \phi_2 x'_{t-2} + \cdots + \phi_p x'_{t-p} + \epsilon_t, \quad (4)$$

In order to show the estimation process of the model parameters more clearly, we represent the above AR(p) model in matrix form:

$$y = X\phi + \epsilon, \quad (5)$$

where y is the target vector containing observations from x'_{p+1} to x'_T ; X is the design matrix, each row of which consists of lagged observations $[x_{t-1}, x_{t-2}, \dots, x_{t-p}]$ at the corresponding moment; ϕ is the p -dimensional AR coefficient vector and ϵ is the error vector containing error terms from ϵ_{p+1} to ϵ_T .

Based on the least squares method, by minimizing the Mean Squared Error (MSE), we can obtain the optimal estimate of the AR coefficients, which minimizes the prediction error of the model.

$$\hat{\phi} = (X^\top X)^{-1} X^\top y. \quad (6)$$

However, in the ARMA model, the Moving Average (MA) model is used to describe the linear relationship between the current observations and the past error terms, and its parameter estimation is more complicated. Its prediction relies on unobservable historical error terms that cannot be solved directly using least squares. For a q -order Moving Average model MA(q), its mathematical expression is:

$$x'_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}, \quad (7)$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is the white noise and $\theta_1, \theta_2, \dots, \theta_q$ is the moving average coefficient. Since these error terms are not directly observable in the modeling, we need to construct a joint objective function to incorporate the AR and MA parameters together into the maximum likelihood estimation framework, and use the Maximum Likelihood Estimation (MLE) method to estimate the model parameters by maximizing the likelihood function. For this purpose, we construct a joint objective function by combining Equations (4) and (7):

$$J(\phi, \theta) = \sum_{t=1}^T \left(x'_t - \sum_{i=1}^p \phi_i x'_{t-i} - \sum_{j=1}^q \theta_j \epsilon_{t-j} \right)^2. \quad (8)$$

Since this joint objective function Equation (8) involves unobservable error terms, its optimization process constitutes a nonlinear least squares problem. To solve this problem, we employ the Broyden–Fletcher–Goldfarb–Shanno (BFGS), an anthropomorphic

Newtonian method, for the solution. During the iterative process, the search direction is jointly determined by the current gradient and the approximate Hessian matrix, i.e., $\mathbf{d}^{(k)} = -\mathbf{H}^{(k)} \nabla J^{(k)}$.

The parameters are updated according to $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \mathbf{d}^{(k)}$, where the step size $\alpha^{(k)}$ is determined by a line search strategy. The Hessian matrix is updated using the BFGS formulation:

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \frac{\mathbf{g}^{(k)} \mathbf{g}^{(k)\top}}{\mathbf{g}^{(k)\top} \mathbf{s}^{(k)}} - \frac{\mathbf{H}^{(k)} \mathbf{s}^{(k)} \mathbf{s}^{(k)\top} \mathbf{H}^{(k)}}{\mathbf{s}^{(k)\top} \mathbf{H}^{(k)} \mathbf{s}^{(k)}}, \quad (9)$$

where $\mathbf{s}^{(k)} = \boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}$ denotes the amount of parameter update and $\mathbf{g}^{(k)} = \nabla J^{(k+1)} - \nabla J^{(k)}$ denotes the amount of gradient change.

After completing the parameter estimation in Equation (8), we obtain the predicted value at each time step excluding the contemporaneous disturbance term $\hat{x}'_t$:

$$\hat{x}'_t = \sum_{i=1}^p \phi_i x'_{t-i} + \sum_{j=1}^q \theta_j \epsilon_{t-j}. \quad (10)$$

Accordingly, the corresponding residual r_t can be computed based on Equations (2) and (10):

$$r_t = x'_t - \hat{x}'_t. \quad (11)$$

Compared with the original sequence $\{x_t\}_{t=1}^T$, the residual sequence $\{r_t\}_{t=1}^T$ after smoothing and periodic trend removal has more stable structural characteristics and is more suitable as input for subsequent compression algorithms, thereby effectively improving the upper limit of the compression ratio.

3.2. Adaptive Precision Quantization Mechanism Based on Residual Magnitude

Although we significantly reduced the volatility of the compression target through the above ARIMA modeling, it still contains a large number of small-amplitude, highly repetitive values. Therefore, in the second stage of the pre-compression process, we introduce an adaptive precision quantification mechanism based on residual amplitude statistical characteristics. We first define the Empirical Cumulative Distribution Function (ECDF) of the residual sequence $\{r_t\}_{t=1}^T$ as:

$$F_r(\tau) = \frac{1}{T} \sum_{t=1}^T \mathbb{I}(|r_t| \leq \tau), \quad (12)$$

where $\mathbb{I}(\cdot)$ is the indicator function and τ is the residual amplitude threshold. This function reflects the proportion of residuals whose amplitude does not exceed the threshold τ .

In order to achieve a dynamic balance between compression ratio and reconstruction accuracy, we introduce a compression control parameter $\alpha \in (0, 1)$, which represents the desired proportion of residuals to be quantized with low precision. We use the minimum threshold τ^* that satisfies the condition $F_r(\tau^*) = \alpha$ as the quantization amplitude boundary point.

To ensure that the error after compression is within an acceptable range, we introduce the following relative error constraint:

$$\frac{\sum_{t=1}^T (\tilde{r}_t - r_t)^2}{\sum_{t=1}^T r_t^2} \leq \epsilon, \quad (13)$$

where $\tilde{r}_t$ represents the residual value after mixed precision quantization, and ϵ represents the upper limit of the reconstruction error.

Based on the above constraints, we construct a hierarchical quantization mechanism for amplitude. Using the optimal threshold τ^* obtained from Equation (13), we divide the residuals into two subsets and assign different quantization precisions to them. The process is defined as follows:

$$\tilde{r}_t = \begin{cases} \text{Float16}(r_t) & \text{if } |r_t| \leq \tau^* \\ \text{Float32}(r_t) & \text{if } |r_t| > \tau^* \end{cases}, \quad (14)$$

where $\text{Float16}(\cdot)$ and $\text{Float32}(\cdot)$ denote the half-precision (16-bit) and single-precision (32-bit) floating-point representation formats according to the IEEE 754 standard, respectively, which are two quantization methods for approximating the residual values. $\text{Float16}(\cdot)$ is suitable for residuals with smaller magnitude and lower precision requirements to reduce the representation overhead, while $\text{Float16}(\cdot)$ retains higher numerical precision and is used to represent residuals with larger magnitude that are more sensitive to reconstruction errors.

After quantization, each residual value is mapped to a discrete floating-point representation in a fixed-precision format. We then encode the quantized residual sequence to form a continuous binary bitstream, which we define as residual encoded bitstream, as the final representation format of the residual sequence in the pre-compression stage.

In order to correctly recover the quantization format used for each residual in the decoding stage, we introduce a set of precision bitmasks consistent with the length of the residual sequence to record the precision identification information, as shown in Figure 1. Each bit corresponds to the quantization precision of a residual, and a value of 1 indicates that the residual at the corresponding position is quantized in 16-bit floating-point format, while a value of 0 indicates that it is quantized in 32-bit floating-point format. This mask provides the basis for precision resolution in the decoding stage, and is stored separately to assist in the subsequent recovery of the correct quantization format of the residuals.

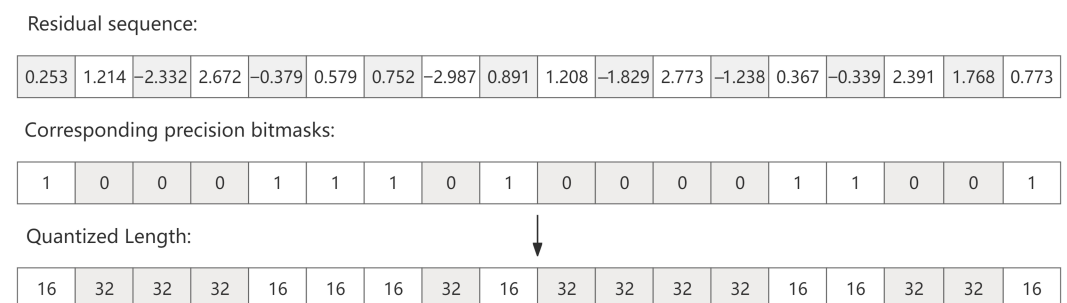


Figure 1. Illustration of residual quantization precision bitmask.

To rigorously evaluate the potential overhead of the precision bitmasks, we define the break-even point for the proposed scheme. Let α denote the proportion of data points stored in Float16 format, N represent the segment length, and K signify the fixed metadata overhead (comprising ARIMA coefficients, initial observations, and model orders). The total storage cost is lower than the raw Float32 storage when: $16\alpha N + 32(1 - \alpha)N + N + K \leq 32N$.

Solving for α , we obtain the critical threshold $\alpha \geq \frac{1}{16} + \frac{K}{16N}$.

Given the continuous nature and long-term storage requirements of time-series data, the segment length N is typically much larger than the metadata overhead K ($N \gg K$). Consequently, this threshold asymptotically approaches 6.25%.

4. Reinforcement Learning Based Compression Algorithms

4.1. System Framework

We take the residual-encoded bitstream output from the pre-compression stage as input to the reinforcement learning compression algorithm. First, it is divided into fixed-length bitstream segments (hereinafter referred to as “segments”) based on a fixed window. Each segment is further divided into a selected number of bit blocks (hereinafter referred to as “blocks”), and pairwise comparisons are performed among all blocks. The common content across blocks is defined as base sub-blocks and stored only once to optimize space utilization. The remaining portions across blocks are defined as difference sub-blocks, which are stored sequentially to preserve the independence and integrity of each block. As shown in Figure 2, when a segment is divided into three blocks and four blocks, the base sub-blocks and difference sub-blocks are illustrated accordingly.

A serialized input bitstream segment:

1	1	1	0	1	0	1	0	1	0	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

When divided into three bitstream blocks:

1	1	1	0	1	0
1	0	1	0	1	0
1	1	1	1	0	1

The corresponding base sub-blocks are as follows:

1		1			
---	--	---	--	--	--

The corresponding difference sub-blocks are as follows:

	1		0	0	0
	0		0	1	0
	1		1	0	1

When divided into four bitstream blocks:

1	1	1	0
1	0	1	0
1	0	1	0
1	1	1	1
0	1		

1		1	
---	--	---	--

	1		0
	0		0
	0		0
	1		1
0	1		

Figure 2. Illustration of base sub-blocks and difference sub-blocks.

For the base sub-blocks, this paper adopts a mask-based representation for storage. Figure 3 demonstrates the storage method for the first base sub-block in Figure 2. This method uses a 0/1 mask vector of the same length as the sub-block to indicate the presence of valid values at each position, while storing the valid values themselves. This achieves compressed storage of the blocks while ensuring rapid access and processing capabilities.

The base sub-block is stored as follows:

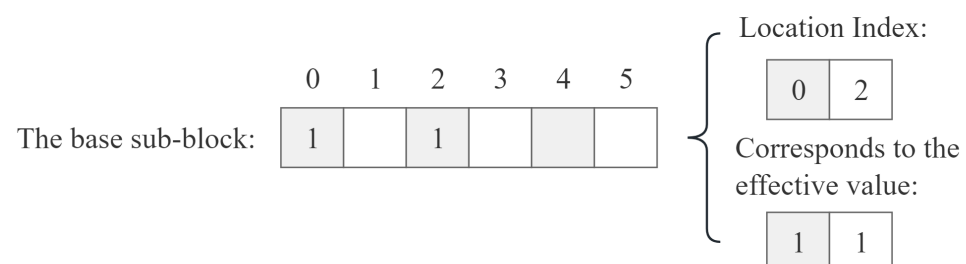


Figure 3. Illustration of base sub-blocks storage method.

The reconstruction process first relies on the recorded base sub-block position masks and their corresponding values, as well as the bit content of all difference sub-blocks. The restoration of each compressed block involves combining the mask structure provided by the base sub-block with the bitstream in the difference sub-block, mapping it to the corresponding compressed block according to the mask position, and finally restoring the complete bit segment.

4.2. Storage Space and Number of Sub-Blocks

Assume that a segment of length n is divided into $k + 1$ blocks as $C_1, C_2, \dots, C_{k+1}$. The length of each block is:

$$q = \lfloor \frac{n}{k} \rfloor. \quad (15)$$

$$r = n \bmod k. \quad (16)$$

Here, $\lfloor x \rfloor$ represents the floor function. Equations (15) and (16) indicate that the length of the first k blocks ($C_1, C_2, \dots, C_k$) are all of length q , and the last block (C_{k+1}) is of length r .

The size of the storage space occupied by this segment without compression is: $S(k) = n$ bits.

If compression is used, each block is divided into two parts for storage: the base sub-block and the difference sub-block. The base sub-block is the same part between each block, and its length is defined as a . Then, the storage space it occupies consists of the base sub-block location index and the effective value, as shown in Figure 3. Therefore, the size of the storage space occupied by the base sub-block is:

$$S_a(k) = a \log_2 n + a = a(\log_2 n + 1) \text{ bits}, \quad (17)$$

where a is the length of base sub-block.

Then, the difference sub-blocks length B_k of block k is:

$$\begin{cases} B_1 = B_2 = \dots = B_k = q - a, B_{k+1} = r & r \neq 0 \\ \text{or } B_1 = B_2 = \dots = B_k = q - a & r = 0 \end{cases}. \quad (18)$$

The entire segment is stored as follows: the base sub-block and its location index are stored only once, as well as the difference sub-blocks of each block, so the segment occupies a storage size of:

$$S_c(k) = \begin{cases} a(\log_2 n + 1) + k \cdot (q - a) + r & r \neq 0 \\ a(\log_2 n + 1) + k \cdot (q - a) & r = 0 \end{cases} \text{ bits}. \quad (19)$$

The compression ratio can be obtained as Equation (20). The larger the CR, the better the compression ratio.

$$CR = \frac{S(k)}{S_c(k)} = \begin{cases} \frac{n}{a(\log_2 n + 1) + k \cdot (q - a) + r} & r \neq 0 \\ \frac{n}{a(\log_2 n + 1) + k \cdot (q - a)} & r = 0 \end{cases}. \quad (20)$$

where both the block length q and r can be expressed by the number of blocks k and the base sub-block length a . This is obtained by substituting Equations (15) and (16) into Equation (20):

$$CR = \begin{cases} \frac{n}{a(\log_2 n + 1) + k \cdot \left(\left\lceil \frac{n}{k} \right\rceil - a\right) + (n \bmod k)} & r \neq 0 \\ \frac{n}{a(\log_2 n + 1) + n - k \cdot a} & r = 0 \end{cases}. \quad (21)$$

From Equation (21), it can be seen that, by optimizing the number of blocks k and the base sub-block length a , the compression ratio can be maximized under the constraints, thus optimizing the utilization of storage space. However, although a changes with k , and once the value of k is determined, a is also determined, the two presents a nonlinear relationship and thus cannot be solved purely by optimization methods.

4.3. QL-Based Block Chunking Optimization

This paper introduces a chunking decision-making mechanism based on Q-Learning algorithm, one of the RL series algorithms. Each segment is treated as a discrete decision cycle. Due to the independence of bitstream processing, the current action does not influence the arrival or statistical characteristics of subsequent segments. This implies that the state transition probability satisfies $P(s_{t+1} | s_t, a_t) = P(s_{t+1})$. Consequently, the environment is simplified into a stateless model. The agent decides the number of blocks by calculating the reward of each possible action based on the initial environment state during a decision cycle. At the end of each decision cycle, the agent evaluates the effectiveness of the chunking in terms of compression ratio and continuously optimizes its strategy through the Q-value update process.

4.3.1. Action

In Section 4.2, a segment of length n is divided into $k + 1$ blocks, where k is a positive integer. Thus, the total number of blocks per segment is $k + 1$. In the reinforcement learning, we directly take this total number of blocks as the action, denote it by k_t for decision cycle t , and therefore set $k_t = k + 1$. For the chunking problem, define the action a_t as selecting the number of blocks k_t at decision cycle t for chunking. For a segment of length n , it can be divided into 1 to n blocks, meaning the possible values for k_t range from 1 to n , i.e., the action set $A = \{1, 2, \dots, n\}$. In this paper, A can be narrowed down by the following proof, so as to reduce the computation even more.

In order for compression to be effective, it is necessary to ensure that the compression ratio is constantly greater than 1, so we first simplify the denominator of Equation (21).

Since there exists the constant equation: $n = k_t \cdot \left\lceil \frac{n}{k_t} \right\rceil + (n \bmod k_t)$. Therefore we can rewrite the denominator in Equation (21) as: $a(\log_2 n + 1) + n - ak_t$.

At this point, the compression ratio is: $\frac{n}{n + a(\log_2 n + 1 - k_t)}$. Since the denominator is always positive, in order to ensure that the compression ratio is greater than 1, we require: $a(\log_2 n + 1 - k_t) < 0$.

Since $a > 0$, we have: $k_t > \log_2 n + 1$.

Given that the total length is n , the final range of k_t must satisfy: $\log_2 n + 1 < k_t < n$.

As k_t should be an integer, the minimum value of k_t should be $\lceil \log_2 n + 1 \rceil$, where $\lceil \cdot \rceil$ represents the ceiling function. Therefore, the possible values for k_t lie in the range: $[\lceil \log_2 n + 1 \rceil, n)$.

Thus, the action set $A = \{\lceil \log_2 n + 1 \rceil, \lceil \log_2 n + 1 \rceil + 1, \dots, n\}$.

4.3.2. Reward

The overall objective is to maximize the compression ratio. Therefore, we define the actual reward $r(t)$ obtained by the agent after executing the action a_t as the compression ratio obtained by dividing the segment into k_t blocks, so that according to Equation (21), the actual reward function is:

$$r(t) = \begin{cases} \frac{n}{a(\log_2 n + 1) + k_t \cdot \left(\left\lfloor \frac{n}{k_t} \right\rfloor - a\right) + n \bmod k_t} & r \neq 0 \\ \frac{n}{a(\log_2 n + 1) + n - k_t \cdot a} & r = 0 \end{cases}. \quad (22)$$

The optimal policy π^* should be the one that maximize $r(t)$:

$$\pi^* = \arg \max_{\pi} r(t). \quad (23)$$

Since the actual reward is not available during decision-making, the agent uses the average historical reward as the basis for executing actions. When the number of algorithm iterations gradually increases, the estimated reward progressively approaches the actual reward.

In the initial stage of the Q-Learning algorithm, since the system has not yet executed any actions, the historical rewards of all actions is unknown. To provide an effective starting point for the subsequent Q-Learning process, we need to calculate the average of the compression ratio of all the number of blocks in the first segment as rewards, and assign them to actions.

At the decision cycle t , the estimated reward of the action a_t is defined as the average reward of this action in the historical segment:

$$\overline{r(t)} = \begin{cases} \frac{1}{j_t} \sum_{j=1}^{j_t} r(j) & j_t \neq 0, \\ 0 & j_t = 0. \end{cases} \quad (24)$$

where j_t denotes the number of times the a_t has been executed.

4.3.3. Balance Exploration and Exploitation

In each subsequent cycle, if the execution number of a_t is small, there may be a large gap between the estimated and the actual rewards. If the algorithm consistently chooses the maximum estimated reward as the basis for executing actions (i.e., 'exploiting' the experience), it may overlook actions that occur less frequently but have higher compression ratios. In order to be able to correctly evaluate the reward of each individual action, the proposed algorithm needs to fully explore the compression ratio of each individual action (without relying on the experience $\overline{r(t)}$ for selection) and to make the estimated action reward converge to the actual reward. Even if the algorithm has converged, exploration is still necessary to track and adapt to drifts in data patterns. Therefore, we need to properly deal with the problem of balancing between exploration and exploitation. We use the Upper Confidence Bound (UCB) algorithm to balance exploration and exploitation.

Lemma 1. For any action, the actual reward $r(t)$ exists within a confidence interval centered on the estimated reward $\overline{r(t)}$, i.e., the confidence interval of $r(t)$ is

$$\left[\overline{r(t)} - \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}}, \overline{r(t)} + \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}} \right],$$

where c denotes the confidence, j_t denotes the number of selected times, σ denotes the standard deviation.

Proof. The probability that there is usually a difference between the expected value and the mean value calculated by averaging the samples, and that this difference is less than the smaller value $\sigma > 0$, is called the confidence. That is:

$$P\left\{\left|\frac{1}{n}\sum_{i=1}^n X_i - E(X)\right| < \delta\right\} > c, \quad (25)$$

where $X_i (i = 1, 2, \dots, n)$ represent for the independent and identically distributed statistical sample data, n is the number of samples, $E(X)$ the true expectation of these samples, c is the confidence. We transform the Equation (25) to obtain:

$$P\left\{-\delta < \frac{\sum_{i=1}^n (X_i - E(X))}{n} < \delta\right\} > c. \quad (26)$$

It follows from the central limit theorem [26,27]:

$$\frac{1}{\sqrt{n}}\sum_{i=1}^n (X_i - E(X)) \sim N(0, \sigma^2), \quad (27)$$

where σ^2 denotes the variance of $X_i (i = 1, 2, \dots, n)$, so we have:

$$\begin{aligned} P\left\{-\delta\sqrt{n} < \frac{\sum_{i=1}^n (X_i - E(X))}{\sqrt{n}} < \delta\sqrt{n}\right\} &= \int_{-\delta\sqrt{n}}^{\delta\sqrt{n}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} dx \\ &= \operatorname{erf}\left(\frac{\delta\sqrt{n}}{\sigma\sqrt{2}}\right). \end{aligned} \quad (28)$$

Since $\operatorname{erfc}(x) < \exp(-x^2)$, we have:

$$\operatorname{erf}\left(\frac{\delta\sqrt{n}}{\sigma\sqrt{2}}\right) = 1 - \operatorname{erfc}\left(\frac{\delta\sqrt{n}}{\sigma\sqrt{2}}\right) > 1 - \exp\left(-\frac{n\delta^2}{2\sigma^2}\right). \quad (29)$$

Let the confidence $c = 1 - \exp\left(-\frac{n\delta^2}{2\sigma^2}\right)$, then the relationship between δ and c can be calculated as:

$$\delta = \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{n}}. \quad (30)$$

Combining Equation (25) with Equation (30) shows that the confidence interval for the mean value of $X_i (i = 1, 2, \dots, n)$ is:

$$\left[\frac{1}{n}\sum_{i=1}^n X_i - \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{n}}, \frac{1}{n}\sum_{i=1}^n X_i + \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{n}} \right]$$

All the rewards of a_t are also independent and identically distributed random variables, so the confidence interval of $r(t)$ is:

$$\left[\overline{r(t)} - \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}}, \overline{r(t)} + \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}} \right].$$

□

From the above lemma, the width of the confidence interval $\sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}}$ is inversely proportional to the number of selected times j_t , that is, the fewer times an action is selected, the higher the uncertainty of its reward. Therefore, the width of the confidence interval can be defined as a measure of the uncertainty of the reward of the action, and we can adjust the width of the confidence interval to balance the exploration and exploitation.

We define the Q-value of action a_t as the upper bound of the confidence interval as the basis for action execution:

$$Q(t) = \overline{r(t)} + \sqrt{\frac{2\sigma^2 \ln\left(\frac{1}{1-c}\right)}{j_t}}. \quad (31)$$

The former in the expression of $Q(t)$ is the estimated reward of the action, encouraging the algorithm to select the action with higher estimated reward, so it can be regarded as the exploitation component; the latter is the width of the confidence interval, which is a measure of the uncertainty of the action reward, encouraging the algorithm to select the action that has been selected fewer times. Thereby, it can be regarded as the exploration component. The UCB can achieve a balance between exploration and exploitation. The width will decrease as the number of selections increases, while the estimated reward of the action will gradually approach the actual reward. The overall flowchart of the proposed algorithm is shown in Figure 4.

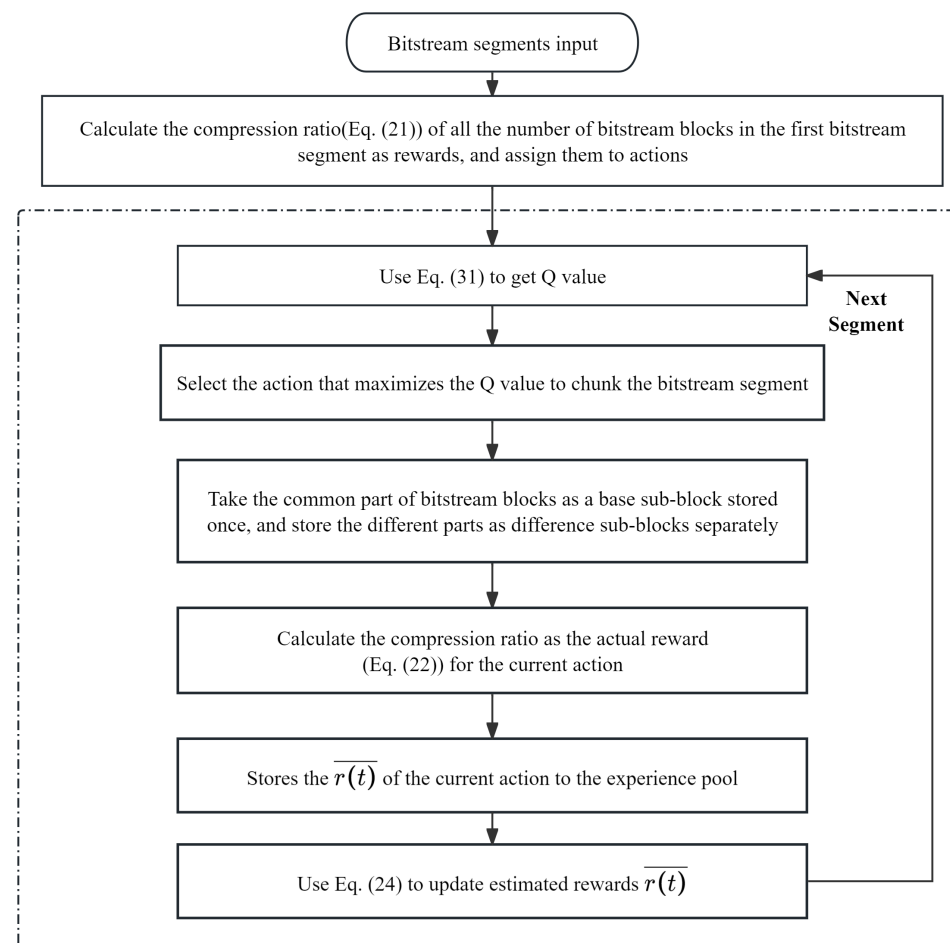


Figure 4. Overall flowchart

5. Experiments

5.1. Introduction to Datasets

We select a proprietary real-world telecommunications dataset (Ver. 2024.v1) collected from a provincial branch of a major national telecommunications operator. Due to data privacy and security agreements, this dataset is not publicly available. The dataset features high-frequency sampling and high volatility, which aligns with the authentic demands of time-series data analysis in industrial environments. Specifically, it records granular network traffic flow, including source/destination IP addresses, source/destination ports, protocol types, transmission timestamps, and data volumes. We take the values of different IPs of the same port for experimentation.

Table 1 shows the amount of data in the telecommunication dataset used for our experiments. We take one day as a period and use 60 days of data from 8 February to 7 April 2024, for a total of 60 data segments, where one segment includes 71 data values.

Table 1. Data volume of telecommunication dataset.

Data Item	Records	Data Item	Records
IP1 (1.1.0.*) ¹	4260	IP4 (1.1.1.*)	4260
IP2 (1.1.0.*)	4260	IP5 (1.1.1.*)	4260
IP3 (1.1.0.*)	4260	IP6 (1.1.1.*)	4260

¹ IP addresses are partially anonymized in the format 1.1.x.*, where the last segment is masked to protect privacy while preserving the dataset structure.

5.2. Experimental Results

The compression ratio is an important measure of the effectiveness of a time-series data compression algorithm because it directly reflects the amount of data reduction achieved by the algorithm. The LZ4 compression algorithms [8,9] both use compression ratio as an evaluation metric.

LZ4 and ZSTD [28] are frequently used as benchmark methods due to their high efficiency and widespread adoption; Chimp [29] represents the optimal single-point prediction algorithm in sequential floating-point compression, while ALP [30] and BUFF [31] have demonstrated strong competitiveness in recent research. Therefore, we selected LZ4, ZSTD, Chimp, ALP, and BUFF as representative algorithms for comparative experiments on the telecommunication dataset to comprehensively evaluate the compression performance of the proposed method.

Meanwhile, to verify the effectiveness of the Q-learning module in this algorithm, we analyze its convergence behavior during the training process. Figure 5 shows the trend of compression ratio during training on multiple segments, which indicates that the algorithm has good stability and convergence.

In Figure 5, the light blue curve shows the instantaneous compression ratio for each segment based on the action selected at that step. The high fluctuation in the early phase indicates that the algorithm is in the exploration stage. The dark blue curve represents the cumulative average compression ratio, which shows a rising trend followed by stabilization, indicating that the strategy improves over time and the overall compression performance gradually increases.

After approximately the 25th segment, the cumulative average compression ratio becomes stable and the fluctuations in the instantaneous compression ratio significantly decrease. This suggests that the learning strategy has stabilized and that the Q-learning algorithm has essentially converged. The results demonstrate that Q-learning can effectively enhance compression performance during continuous training and achieve stable convergence.

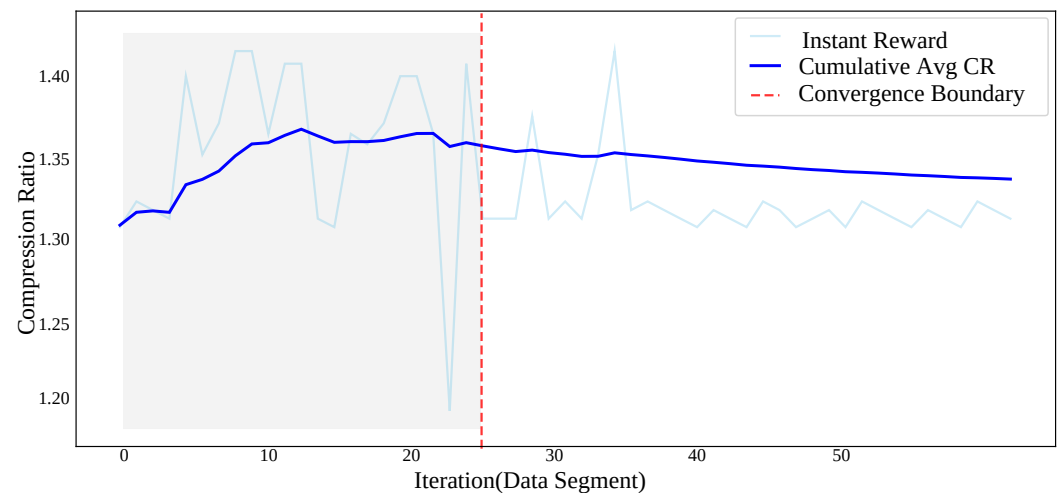


Figure 5. Compression ratio convergence during Q-Learning.

To further evaluate stability, Figure 6 illustrates the training loss characterized by the Temporal Difference (TD) error, which denotes the gap between the estimated and the actual rewards. The light red curve depicts the raw TD error, where the pronounced fluctuations during the initial 25 iterations reflect the agent's uncertainty regarding environmental states during the exploration phase. As training progresses, the smoothed loss which represented by the dark red curve exhibits a distinct monotonically decreasing trend, eventually stabilizing below 0.05. This convergence trend is highly consistent with the stability of the average compression ratio observed in Figure 5.

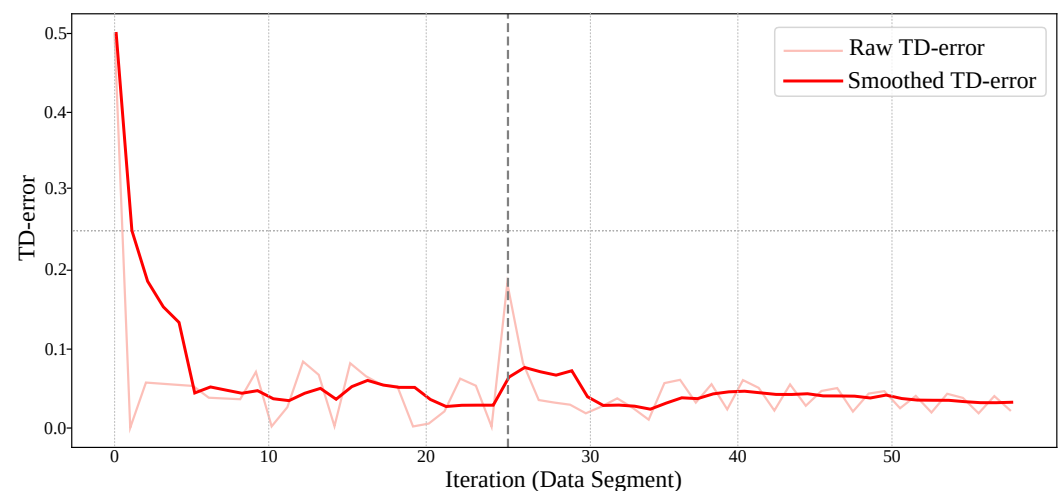


Figure 6. Temporal Difference (TD) error during Q-Learning.

5.3. Experimental Results on Telecommunication Dataset

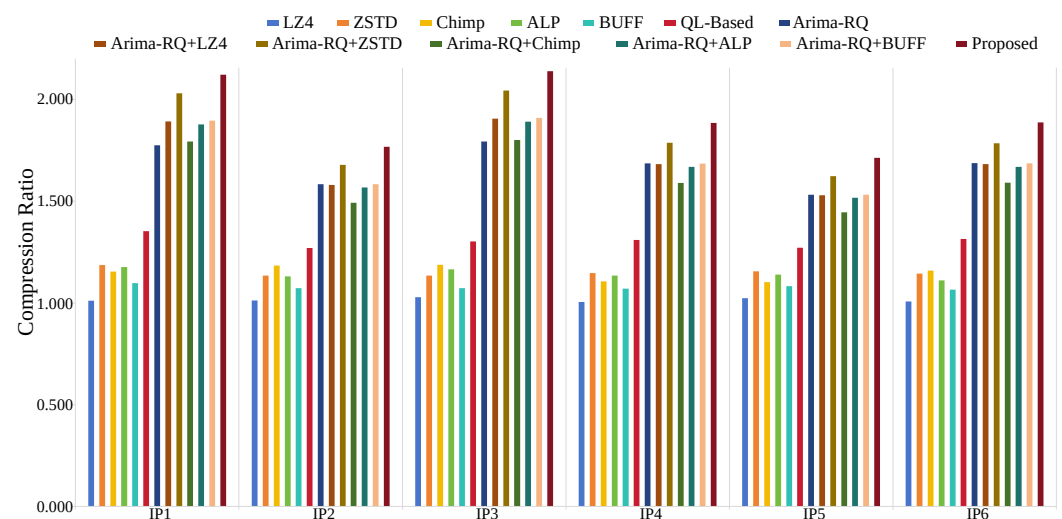
From Table 2, traditional compression algorithms (such as LZ4 and ZSTD) perform poorly on this highly volatile dataset, with compression ratios maintained only between 1.0 and 1.2. Specialized algorithms (such as Chimp, ALP, and BUFF) generally yield compression ratios between 1.06 and 1.18; although slightly superior to traditional algorithms, the improvement remains extremely limited. While the standalone use of the Q-Learning algorithm offers some improvement, the compression ratio hovers around 1.3, indicating significant persisting limitations. In contrast, the Arima-RQ compression algorithm, based on predictive residual quantization, demonstrates a marked advantage, achieving independent compression ratios of from 1.53 to 1.79.

Table 2. Comparison of algorithms on telecommunication dataset.

Dataset	LZ4	ZSTD	Chimp	ALP	BUFF	QL-Based	Arima-RQ	Arima-RQ+LZ4	Arima-RQ+ZSTD	Arima-RQ+Chimp	Arima-RQ+ALP	Arima-RQ+BUFF	Proposed
IP1	1.011	1.187	1.154	1.176	1.098	1.353	1.775	1.892	2.031	1.794	1.878	1.897	2.121
IP2	1.012	1.135	1.184	1.131	1.073	1.270	1.584	1.581	1.679	1.493	1.568	1.584	1.767
IP3	1.029	1.135	1.188	1.166	1.073	1.303	1.794	1.905	2.044	1.801	1.891	1.910	2.139
IP4	1.005	1.147	1.106	1.135	1.071	1.310	1.686	1.682	1.788	1.590	1.669	1.685	1.885
IP5	1.024	1.156	1.103	1.139	1.083	1.272	1.532	1.529	1.623	1.445	1.517	1.532	1.713
IP6	1.008	1.144	1.159	1.111	1.065	1.315	1.687	1.683	1.785	1.592	1.669	1.686	1.888

When Arima-RQ is combined with traditional algorithms, Arima-RQ+LZ4 fails to further increase, and, in some cases, slightly decreases the compression ratio on certain datasets (e.g., IP2, IP4, and IP5). This suggests that, for simple bit-pattern encoding, the overhead may exceed the redundancy gains when processing high-entropy residuals after quantization. Conversely, the efficiency of Arima-RQ+ZSTD further enhances to a maximum of 2.044, which is attributed to the Finite State Entropy (FSE) encoding employed by ZSTD, as it more effectively captures subtle non-uniform probability distributions within the quantized residuals. Although the performance of specialized algorithms improves to a range of from 1.44 to 1.91 after incorporating the Arima-RQ module, they struggle to achieve deep compression; this is because algorithms like Chimp and ALP are primarily designed for the similarities of raw floating-point or integer data, which are disrupted in post-quantization residual data. In comparison, the two-stage compression algorithm proposed in this paper (Arima-RQ+QL) achieves the optimal compression ratio across all six datasets, reaching a peak of 2.139 (in IP3).

To provide a more intuitive comparison of the compression performance across different algorithms, Figure 7 presents the compression ratio results for each algorithm on the telecommunications dataset in the form of histograms. As clearly shown in Figure 7, our algorithm exhibits more stable and comprehensive performance improvements overall.

**Figure 7.** Column chart comparing multiple algorithms on telecommunication dataset.

5.4. Computational Complexity

Assuming that the time-series data are divided into m data segments, each containing n data points, the computational complexity of our method is derived as follows:

The computational complexity of the first stage is determined by three core components: stationarity preprocessing, ARIMA model fitting, and adaptive quantization. The overhead for stationarity testing and differencing is $O(n)$. During model identification and parameter estimation, although the process involves AIC-based order selection and BFGS-based nonlinear iterative optimization, the associated costs scale linearly with the

data length n , i.e., $O(n)$, given that the model orders (p, q) and the maximum number of iterations are constrained to constants. Finally, the adaptive quantization process entails a single-pass linear scan. Consequently, the cumulative computational overhead of the entire first stage can be quantitatively expressed as $O(n * m)$.

In Section 4.3.2, we have pointed out that the first segment is needed to initialize the Q-value by calculating the reward of each action. Therefore, the computational complexity of the first segment is $O(n^2)$. For the next $m - 1$ segments, we simply select the action and update the Q-value based on the Q-value calculated earlier. Since the Q-value is updated using an incremental formula based on the average, both the update and the selection of action with the maximum Q-value have a computational complexity of $O(1)$. After action, the segments need to be divided and the base sub-block is obtained through comparison, so the computational complexity is $O(n)$. Therefore, for the latter $m - 1$ segments, the overall computational complexity is $O(n * m)$. The computational complexity of the whole process is $O(n^2) + O(n * m)$.

Since time-series data are continuous, m increases as processing time extends. If m is directly used as the variable for total complexity, the total complexity would grow infinitely over time, making it difficult to fairly compare different methods. Therefore, we adopt “unit complexity” as the benchmark for comparison. Let k segments be generated within a fixed time interval, with each segment having the same length n . Since the compression ratio calculation and update complexity for each segment in this paper’s algorithm is $O(n)$, the overall complexity within a fixed time interval is $O(n * k)$. Based on this, Table 3 presents the comparison results of different algorithms under unit complexity.

Table 3. Computational complexity of different algorithms.

Algorithm	Unit Complexity
Exhaustive Search	$O((n \cdot k)^2)$
LZ4, ZSTD	$O(n \cdot k)$
Chimp, ALP, BUFF	$O(n \cdot k)$
QL-based	$O(n \cdot k)$
ARIMA-RQ	$O(n \cdot k)$
ARIMA-RQ+LZ4, ARIMA-RQ+ZSTD	$O(n \cdot k)$
ARIMA-RQ+Chimp, ARIMA-RQ+ALP, ARIMA-RQ+BUFF	$O(n \cdot k)$
Proposed	$O(n \cdot k)$

The unit complexity of the exhaustive search exhibits quadratic growth, reflecting relatively high computational costs. In contrast, the unit complexities of LZ4, ZSTD, Chimp, ALP, BUFF, and quantization are solely related to the total amount of data within a unit time period, specifically $O(n \cdot k)$, which aligns with our proposed algorithm. Therefore, from the perspective of unit time intervals, our proposed algorithm incurs no additional computational overhead.

To further validate the effectiveness of the theoretical unit complexity in practical operations, we measured the compression time of each algorithm on the telecommunications dataset to more objectively reflect their computational overhead. The results are presented in Table 4. It can be observed that the lightweight traditional compression algorithm LZ4 exhibits an extremely high compression speed with a latency of approximately 0.02 ms. Mature algorithms such as ZSTD, Chimp, and ALP also demonstrate high compression speeds, with latencies around 0.5 ms. The BUFF algorithm requires slightly more time, yet remains efficient at approximately 2.2 ms. The Arima-RQ, pre-compression algorithm proposed in this paper also shows superior computational efficiency, with an average execution time of only about 7 ms, proving its feasibility as a lightweight preprocessing layer.

However, the second-stage compression algorithm based on reinforcement learning exhibits a total execution latency of approximately 674.6 ms (IP2). Notably, approximately 197.7 ms of this initial overhead is concentrated in the phase of initializing the Q-table by exhaustively traversing the first data segment. Once the algorithm enters the stable operational phase, it only needs to perform lightweight action selection and real-time Q-table updates, which significantly reduces the average processing latency per segment to approximately 8.1 ms. This implies that as the scale of the processed data continues to increase, the one-time initial setup time will be effectively amortized across the massive subsequent data segments, causing the overall average latency of the system rapidly converge to around 8.1 ms. Therefore, from the perspective of long-term operation, the unit complexity of each algorithm is effectively of the same order of magnitude. The two-stage compression algorithm proposed in this paper achieves significant compression gains without introducing unacceptable computational overhead.

Table 4. Compression times for multiple algorithms (milliseconds).

Dataset	Lz4	Zstd	Chimp	Alp	Buff	QL-Based	Arima-RQ	Proposed
IP1	0.024	0.262	0.386	0.667	2.212	622.121	7.074	629.195
IP2	0.021	0.142	0.335	0.605	2.167	674.551	7.054	681.605
IP3	0.019	0.189	0.364	0.513	2.142	537.812	7.218	545.020
IP4	0.024	0.131	0.408	0.639	2.454	618.995	7.213	626.208
IP5	0.021	0.128	0.451	0.678	2.598	656.224	7.656	663.879
IP6	0.659	0.510	0.387	0.622	2.654	642.499	7.458	649.949

5.5. Reconstruction Performance

To evaluate the reconstruction performance, Figure 8 presents a comparison between the original telecommunication data and the reconstructed data. The results demonstrate that the reconstructed curve (orange dashed line) maintains a high degree of consistency with the original data (blue solid line) in terms of overall trends.

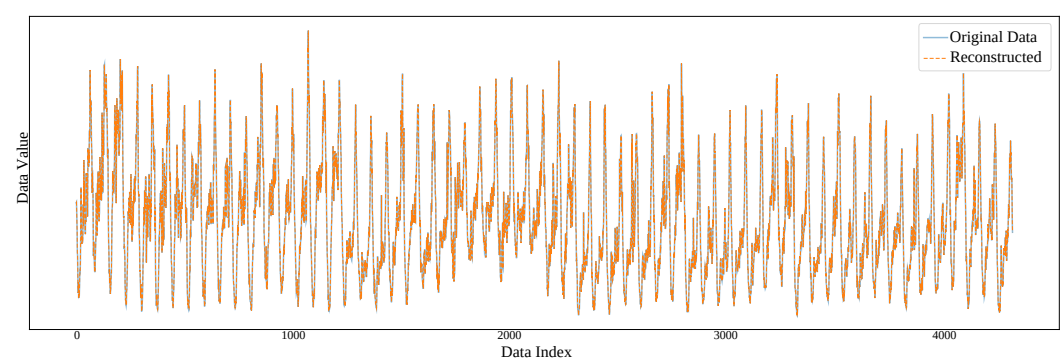


Figure 8. Global view of reconstruction performance.

To further verify the local precision, Figure 9 extracts and magnifies a sequence of 100 consecutive data points from index 2130 to 2230. Observations indicate that the reconstruction algorithm is capable of accurately capturing the data fluctuations.

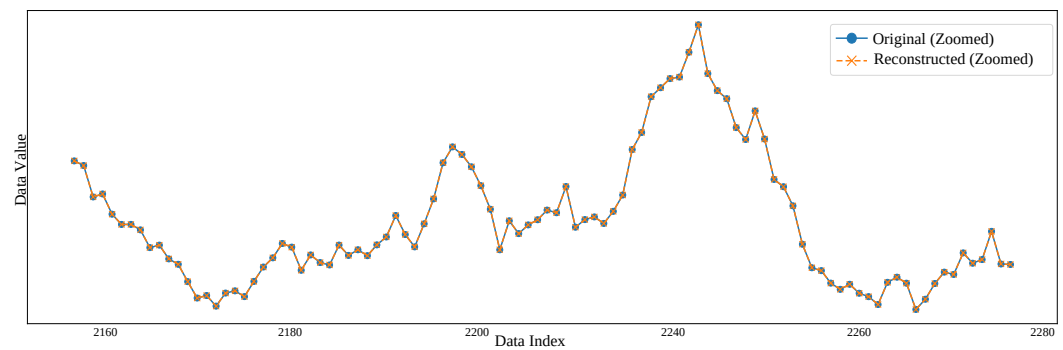


Figure 9. Local detail view of reconstruction performance.

Beyond visual inspection, we employ Signal-to-Noise Ratio (SNR) and Root Mean Square Error (RMSE) to quantitatively assess the reconstruction accuracy. The SNR is defined as follows:

$$SNR = 10 \cdot \log_{10} \left(\frac{\sum_{i=1}^N x_i^2}{\sum_{i=1}^N (x_i - \hat{x}_i)^2} \right), \quad (32)$$

where x_i denotes the original value, $\hat{x}_i$ represents the reconstructed value, and N signifies the total number of data points. The RMSE is defined as follows:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2}. \quad (33)$$

Since the value of RMSE is inherently sensitive to the magnitude of the original data, it is challenging to perform a fair comparison across datasets with different scales. Consequently, we employ the Normalized Root Mean Square Error (NRMSE) by normalizing the RMSE to yield a dimensionless metric. To better reflect the algorithm's capability in reconstructing data with significant fluctuations, we adopt a range-based normalization formula:

$$NRMSE = \frac{RMSE}{x_{max} - x_{min}}, \quad (34)$$

where x_{max} and x_{min} denote the maximum and minimum values in the original sequence, respectively.

Table 5 summarizes the reconstruction performance of the proposed algorithm across multiple datasets. Experimental results demonstrate that our algorithm achieves an average SNR of up to 89 dB, with the NRMSE consistently remaining at the order of 10^{-5} . These findings verify that our algorithm maintains significant reconstruction precision and numerical stability while achieving substantial data compression.

Table 5. Reconstruction performance of different datasets.

Dataset	SNR	NRMSE
IP1	88.32	0.002733%
IP2	89.4	0.002167%
IP3	84.89	0.003849%
IP4	86.4	0.002418%
IP5	92.62	0.001730%
IP6	80.53	0.002589%

Mirroring the compression stage, we adopt unit complexity as the benchmark to prevent the total computational overhead from growing infinitely over time. We assume that k data segments, each of length n , are generated within a fixed time interval. During

the first stage of reconstruction, the system re-allocates the bitstream to its original positions based on recorded masks. Although the input consists of a binary bitstream, the process essentially involves sequential parsing and constant-time memory addressing for each target data point. Given that the bit-length per compressed sample is finite and bounded, the total number of bit-level operations remains linearly proportional to the original data volume. Thus, the unit complexity for this stage is $O(k * n)$.

The second stage of reconstruction comprises inverse quantization and ARIMA restoration. During inverse quantization, a point-wise multiplication is performed on the parsed values to restore the data scale, yielding a complexity of $O(k * n)$. In the ARIMA reconstruction, the transmitted coefficients are used to restore residuals through point-to-point predictive calculations, which also maintains a complexity of $O(k * n)$. Consequently, the total unit complexity of reconstruction is the summation of these linear stages, denoted as $O(k * n)$. The reconstruction complexity per unit time is consistent with the other algorithms.

When evaluating the comprehensive performance of compression algorithms, decompression efficiency is a key metric for assessing their practical value in real-time analytics scenarios. From Table 6, compared with the compression stage, our proposed algorithm demonstrates a significant performance advantage during decompression. Experimental results indicate that the second-stage compression algorithm based on reinforcement learning, as it no longer involves complex processes such as policy updates during decompression, substantially reduces its time consumption to approximately 1.5 ms. The first-stage pre-compression algorithm also maintains high stability during decompression, with an average time consumption of around 3.7 ms. The overall average decompression time of the proposed two-stage algorithm remains stable at about 5.5 ms. These results suggest that while achieving high compression gains, the proposed algorithm controls the computational overhead of data reconstruction within the millisecond-level range, demonstrating strong practical feasibility.

Table 6. Decompression times for multiple algorithms (milliseconds).

Dataset	Lz4	Zstd	Chimp	Alp	Buff	QL-Based	Arima-RQ	Proposed
IP1	0.015	0.789	0.136	0.354	0.07	1.554	3.810	5.364
IP2	0.014	0.044	0.118	0.315	0.067	1.495	3.830	5.325
IP3	0.014	0.032	0.160	0.276	0.067	1.404	3.718	5.114
IP4	0.024	0.035	0.156	0.408	0.075	1.728	3.843	5.568
IP5	0.011	0.038	0.169	0.363	0.096	1.328	3.659	5.295
IP6	0.009	0.039	0.145	0.318	0.081	1.527	3.525	5.042

6. Conclusions

To achieve efficient compression of time-series data while dynamically tracking pattern changes, we propose a unified two-stage time-series compression algorithm. In the first stage, residual modeling and adaptive quantization via ARIMA models enhance data compression suitability. The second stage employs a reinforcement learning-based chunk optimization strategy, leveraging continuously updated experience to determine the optimal number of blocks per segment. This algorithm achieves a high compression ratio while significantly reducing computational overhead. By employing the Upper Confidence Bound (UCB) algorithm to balance exploration and exploitation, we ensure more precise reward estimation while effectively tracking evolving data patterns. Experimental results demonstrate that this algorithm achieves superior compression rates compared to mainstream compression algorithms (LZ4, ZSTD, Chimp, ALP, BUFF) across multiple highly volatile telecom datasets, while also showing clear advantages over using reinforcement

learning or quantization alone. Complexity analysis and compression efficiency analysis further confirm that this algorithm achieves high compression ratios and high reconstruction accuracy without significantly increasing computational overhead. In summary, this algorithm strikes a good balance between high compression efficiency and low computational cost, demonstrating application potential for processing large-scale time-series data.

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

Funding: This research received no external funding.

Data Availability Statement: The telecommunication dataset used in this study is owned by China Telecom and was provided under a specific collaboration agreement. Due to confidentiality restrictions, the data are not publicly available. Access to the data may be possible upon reasonable request to the corresponding author and with the explicit permission of China Telecom.

Acknowledgments: The authors have reviewed and edited the output and take full responsibility for the content of this publication.

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

References

1. Fu, T.C. A review on time series data mining. *Eng. Appl. Artif. Intell.* **2011**, *24*, 164–181. <https://doi.org/10.1016/j.engappai.2010.09.007>.
2. Namiot, D. Time Series Databases. In Proceedings of the the XVII International Conference on Data Analytics and Management in Data Intensive Domains, Obninsk, Russia, 13–16 October 2015; Volume 1536, pp. 132–137.
3. Jensen, S.K.; Pedersen, T.B.; Thomsen, C. Time series management systems: A survey. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 2581–2600. <https://doi.org/10.1109/TKDE.2017.2720168>.
4. InfluxData Inc. InfluxDB: Open Source Time Series, Metrics, and Analytics Database. 2015. Available online: <https://www.influxdata.com/> (accessed on 20 October 2025).
5. Sigoure, B. OpenTSDB: A Distributed, Scalable Monitoring System. 2013. Available online: <https://opentsdb.net/> (accessed on 20 October 2025).
6. Dunning, T.; Friedman, E. *Time Series Databases: New Ways to Store and Access Data*; O'Reilly Media: Newton, MA, USA, 2014.
7. Pelkonen, T.; Franklin, S.; Teller, J.; Cavallaro, P.; Huang, Q.; Meza, J.; Veeraraghavan, K. Gorilla: A fast, scalable, in-memory time series database. *Proc. VLDB Endow.* **2015**, *8*, 1816–1827. <https://doi.org/10.14778/2824032.2824078>.
8. Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343. <https://doi.org/10.1109/TIT.1977.1055714>.
9. Ziv, J.; Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory* **1978**, *24*, 530–536. <https://doi.org/10.1109/TIT.1978.1055934>.
10. Xu, Z.; Zhang, W. QuickCDC: A quick content defined chunking algorithm based on jumping and dynamically adjusting mask bits. In Proceedings of the Proc IEEE Intl Conf Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York, NY, USA, 30 September–3 October 2021; pp. 288–299. <https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00049>.
11. Rabin, M.O. *Fingerprinting by Random Polynomials*; Technical Report; Harvard University: Cambridge, MA, USA, 1981.
12. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **2002**, *47*, 235–256. <https://doi.org/10.1023/A:101368970435>.
13. Box, G.E.; Jenkins, G.M.; Reinsel, G.C.; Ljung, G.M. *Time Series Analysis: Forecasting and Control*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
14. Al Tamimi, A.K.; Jain, R.; So-In, C. SAM: A simplified seasonal ARIMA model for mobile video over wireless broadband networks. In Proceedings of the 2008 Tenth IEEE International Symposium on Multimedia; IEEE: Piscataway, NJ, USA, 2008; pp. 178–183. <https://doi.org/10.1109/ISM.2008.112>.

15. Tang, L.; Yi, Y.; Peng, Y. An ensemble deep learning model for short-term load forecasting based on ARIMA and LSTM. In *Proceedings of the 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6. <https://doi.org/10.1109/SmartGridComm.2019.8909756>.
16. Satrio, C.B.A.; Darmawan, W.; Nadia, B.U.; Hanafiah, N. Time series analysis and forecasting of coronavirus disease in Indonesia using ARIMA model and PROPHET. *Procedia Comput. Sci.* **2021**, *179*, 524–532. <https://doi.org/10.1016/j.procs.2021.01.036>.
17. Pai, P.F.; Lin, C.S. A hybrid ARIMA and support vector machines model in stock price forecasting. *Omega* **2005**, *33*, 497–505. <https://doi.org/10.1016/j.omega.2004.07.024>.
18. Chen, P.; Yuan, H.; Shu, X. Forecasting crime using the arima model. In *Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*; IEEE: Piscataway, NJ, USA, 2008; Volume 5, pp. 627–630. <https://doi.org/10.1109/FSKD.2008.222>.
19. Banihabib, M.E.; Ahmadian, A.; Jamali, F.S. Hybrid DARIMA-NARX model for forecasting long-term daily inflow to Dez reservoir using the North Atlantic Oscillation (NAO) and rainfall data. *GeoResJ* **2017**, *13*, 9–16. <https://doi.org/10.1016/j.grj.2016.12.002>.
20. Zhang, G.P. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* **2003**, *50*, 159–175. [https://doi.org/10.1016/s0925-2312\(01\)00702-0](https://doi.org/10.1016/s0925-2312(01)00702-0).
21. Zahid, H.; Mahmood, T.; Morshed, A.; Sellis, T. Big data analytics in telecommunications: Literature review and architecture recommendations. *IEEE/CAA J. Autom. Sin.* **2019**, *7*, 18–38.
22. Costa, C.; Chatzimilioudis, G.; Zeinalipour-Yazti, D.; Mokbel, M.F. Efficient exploration of telco big data with compression and decaying. In *Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE)*; IEEE: Piscataway, NJ, USA, 2017; pp. 1332–1343.
23. Rippel, O.; Bourdev, L. Real-time adaptive image compression. In *Proceedings of the International Conference on Machine Learning*, PMLR, Sydney, Australia, 6–11 August 2017; pp. 2922–2930.
24. Jiang, N.; Xiang, Q.; Wang, H.; Zheng, B. Time series compression based on reinforcement learning. *Inf. Sci.* **2023**, *648*, 119490. <https://doi.org/10.1016/j.ins.2023.119490>.
25. YULE, G. On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer’s Sunspot Numbers (1927). In *The Foundations of Econometric Analysis*; Cambridge University Press: Cambridge, UK, 1995; p. 159.
26. Dudley, R.M. *Uniform Central Limit Theorems*; Cambridge Univ. Press: Cambridge, UK, 2014; Volume 142.
27. Kwak, S.G.; Kim, J.H. Central limit theorem: The cornerstone of modern statistics. *Korean J. Anesthesiol.* **2017**, *70*, 144–156. <https://doi.org/10.4097/kjae.2017.70.2.144>.
28. Collet, Y. RFC 8878: Zstandard Compression and the application/zstd’Media Type. *Standard*, RFC Editor: Fremont, CA, USA, 2021.
29. Liakos, P.; Papakonstantinou, K.; Kotidis, Y. Chimp: Efficient lossless floating point compression for time series databases. *Proc. VLDB Endow.* **2022**, *15*, 3058–3070.
30. Afroozeh, A.; Kuffo, L.X.; Boncz, P. ALP: Adaptive lossless floating-point compression. *Proc. ACM Manag. Data* **2023**, *1*, 1–26. <https://doi.org/10.1145/3626717>.
31. Liu, C.; Jiang, H.; Paparrizos, J.; Elmore, A.J. Decomposed bounded floats for fast compression and queries. *Proc. VLDB Endow.* **2021**, *14*, 2586–2598. <https://doi.org/10.14778/3476249.3476305>.

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