Front. Comput. Sci., 2025, 19(7): 197360
https://doi.org/10.1007/s11704-024-40300-5

REVIEW ARTICLE

A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data

Hui SUN[1,2*], Huidong MA[1*], Feng LING[1], Haonan XIE[3], Yongxia SUN[1], Liping YI[1,2], Meng YAN (✉)[1], Cheng ZHONG[4], Xiaoguang LIU[1], Gang WANG (✉)[1]

1 Nankai-Baidu Joint Laboratory, Parallel and Distributed Software Technology Laboratory, TMCC, SysNet, DISSec, GTIISC, College of Computer Science, Nankai University, Tianjin 300350, China
2 College of Computing and Data Science, Nanyang Technological University, Singapore 639798, Singapore
3 Institute of Artificial Intelligence, School of Electrical Engineering, Guangxi University, Nanning 53004, China
4 Key Laboratory of Parallel, Distributed and Intelligent of Guangxi Universities and Colleges, School of Computer, Electronics and Information, Guangxi University, Nanning 53004, China

Abstract As various types of data grow explosively, large-scale data storage, backup, and transmission become challenging, which motivates many researchers to propose efficient universal compression algorithms for multi-source data. In recent years, due to the emergence of hardware acceleration devices such as GPUs, TPUs, DPUs, and FPGAs, the performance bottleneck of neural networks (NN) has been overcome, making NN-based compression algorithms increasingly practical and popular. However, the research survey for the NN-based universal lossless compressors has not been conducted yet, and there is also a lack of unified evaluation metrics. To address the above problems, in this paper, we present a holistic survey as well as benchmark evaluations. Specifically, i) we thoroughly investigate NN-based lossless universal compression algorithms toward multi-source data and classify them into 3 types: static pre-training, adaptive, and semi-adaptive. ii) We unify 19 evaluation metrics to comprehensively assess the compression effect, resource consumption, and model performance of compressors. iii) We conduct experiments more than 4600 CPU/GPU hours to evaluate 17 state-of-the-art compressors on 28 real-world datasets across data types of text, images, videos, audio, etc. iv) We also summarize the strengths and drawbacks of NN-based lossless data compressors and discuss promising research directions. We summarize the results as the NN-based Lossless Compressors Benchmark (NNLCB, See fahaihi.github.io/NNLCB website), which will be updated and maintained continuously in the future.

Received March 24, 2024; accepted December 9, 2024

E-mail: yanm@nbjl.nankai.edu.cn; wgzwp@nbjl.nankai.edu.cn

* These authors contributed equally to this work.

Keywords lossless compression, benchmark evaluation, universal compressors, neural networks, deep learning

1 Introduction

With the rapid expansion of information technologies and internet-connected computers, worldwide data growth has surpassed Moore's Law [1]. For example, according to the latest "Global Data Sphere 2023" released by IDC, the total global data volume is expected to grow from 126 ZB[1)] in 2022 to 284 ZB in 2027 [2]. Such massive amounts of data poses a challenge to data storage, transmission, and sharing. Traditional storage methods, which rely on multi-server and disk-array technologies, require high infrastructure expenses, thus is difficult to deploy and implement. Therefore, developing efficient compression methods are essential to alleviate the pressure of large-scale data storage and promote data transmission and sharing.

Compression solutions are either universal or dedicated. The former are designed to deal with a variety of data types including text, image, audio, video, etc., while the latter are usually used for specific fields such as DNA sequencing data [3–8], 3D point cloud data [9,10], and video image data [11–15]. In this paper, we focus on the lossless universal compression algorithms, which are able to recover the original data without losing any information and thus are ideal for scenarios requiring high data integrity, such as long-term backups for large-size databases.

Recently, NN-based compressors exhibit plenty of advantages and receive increasing attention. Comparing to the traditional lossless compressors (XZ [16], Zstd [17], Brotli [18], BSC [19], Szip [20], Bzip2 [21], Zpaq [22], Mcm [23], Starlit [24], WSDC [25], etc.) NN-based compressors are superior in terms of space saving. On the one side, the

1) 1 ZB = 1024^4 GB

widespread adoption of neural network technologies [26–28] fostered various kinds of Deep-learning neural networks, such as Long Short-Term Memory (LSTM) [29], Bidirectional Gate Recurrent Units (BiGRU) [30], Transformer [31], Large Language Models (LLMs) [32–34], Mamba [35], and Extended Long Short-Term Memory (xLSTM) [36]. Those techniques have promoted the development of NN-based compressors. [37–39]. On the other side, hardware acceleration devices like GPUs, TPUs, DPUs, and FPGAs have facilitated the design of parallel algorithms, helping NN-based compressors break time and throughput performance bottlenecks [39–41].

After thorough investigation, we find that the research survey for the NN-based universal lossless compressors has not been conducted yet, and two main challenges exist regarding the review and benchmark testing. Firstly, previous works primarily focused on traditional compressors and the comparison between traditional methods and NN-based methods is absent. Secondly, the evaluation metrics for NN-based compressors are disorganized, and a unified standard for performance measurement is needed. To this end, this study thoroughly reviews the latest advancements in NN-based universal lossless compressors and proposes a comprehensive benchmark evaluation. The primary contributions of this paper can be summarized as follows:

- We holistically reviewed NN-based universal compressors and unified the corresponding evaluation metrics.
- We offered a comprehensive benchmark evaluation for NN-based compressors and conducted a thorough analysis of existing state-of-the-art solutions.
- We identified the challenges of NN-based lossless compression and summarized potential research directions.

The rest of this paper is organized as follows: Section 2 provides a brief introduction to relevant state-of-the-art surveys; Section 3 presents a detailed description and analysis of NN-based universal lossless compression methods; Section 4 outlines comprehensive evaluation metrics including compression effect, resource consumption, and neural network model performance; Section 5 introduces the benchmark evaluation and displays experimental analysis; Section 6 suggests potential future research directions; and finally, Section 7 concludes our work.

2 Related work

Since Shannon brought forward his Information Theory [42], the academia and industry have endeavored to create concise representations for data streams utilizing the redundancy patterns in the data. Traditional universal lossless compression methods can be categorized into multiple types according to the algorithmic-coding schemes they use, such as Huffman Coding (HC) [43], Arithmetic Coding (AC) [44], Dictionary Coding (DC) [45], Burrows-Wheeler Transform (BWT) [46], Run-Length Encoding (RLE) [47], and Wavelet Transform (WT) [48]. Many literature [43–50] give detailed descriptions about those traditional compression technologies. The earlier surveys on data compression can be traced back to Holtz [51], Kimura [52], Chew [53], Sridevi [54], Hosseini [55], Srisooksai [56], Sharma [57], etc. We suggest readers refer to [49] for those earlier survey works.

In the following, we summarize surveys for traditional universal compression techniques over the past ten years. We also summarize some reviews of dedicated compression algorithms, because they encompass a thorough evaluation of universal methods.

Research surveys for dedicated compression methods generally concentrated on specific application domains, such as genomic, electrocardiogram (ECG), medical image, and smart grid data. Among these survey works, [6,58–62] provided comprehensive reviews of both dedicated and universal compression methods for large genomic datasets, and displayed experimental results comparing compression ratio, time, and memory usage. [63,64] compared significant techniques in compressing ECG data and evaluated their effects on compression ratio and the quality of data reconstruction. They also delved into various performance metrics and addressed open challenges in the field of ECG. [65–67] supplied a comprehensive overview of the current landscape of medical-image compression techniques, covering lossless, lossy, and hybrid compression methods. [68–70] conducted comprehensive studies on data compression techniques for big data in smart grids, aiming to enhance data analysis efficiency while reducing transmission pressure and storage costs. [71–75] presented systematic surveys of data compression methods for large volumes of data generated by the Internet of Things (IoT) sensing devices such as cameras, satellites, and seismic monitoring; they focus on exploring data compression techniques in cloud computing environments.

In the research field of universal compression technologies for multi-source data, Kaur et al. [76] systematically presented the application of data deduplication techniques in lossless compression solutions within cloud computing, and they specifically focused on data redundancy issues in storage systems. Cappello et al. [77] introduced the development of compression techniques for floating-point datasets in universal data and explored their potential applications in scientific computing. Jayasankar et al. [49] provided a comprehensive overview of the applications of compression techniques and the developmental history, and they also categorized these techniques based on the quality of recovered data, encoding schemes, data types, and applications. Chiarot et al. [50] conducted a comprehensive review of time-series compressors, sorting through the classification, evaluation metrics, and practical application cases of time series compressors.

Our study is an additional supplement to the above surveys. To the best knowledge of the authors, NNLCB is the first work providing a comprehensive review of the promising NN-based universal lossless compressors. Details are given in the next section.

3 NN-based lossless universal compressors

In this section, we first divide NN-based algorithms into three

types and compare their advantages and weaknesses at a high level. Then we introduce representative algorithms for each type, and provide pseudocode descriptions. Finally, we offer a summary of NN-based compressors.

3.1 Overview

Generally, a universal lossless compression algorithm based on neural networks contains two core components: a probabilistic prediction model $\mathcal{M}$ and an encoder $\mathcal{E}$. Here we briefly introduce their functions and workflow.

Let $x=\{x_i\}_{i=1}^{|x|}$ denotes the serialized data sequence to be compressed, where $x_i \in \mathcal{S}$ represents the token in alphabet $\mathcal{S}$ and $|x|$ is the sequence length. The model $\mathcal{M}$ is responsible for predicting the probability distribution of each target token $x_n \in x$ based on history t tokens $\{x_{n-t}, x_{n-t+1}, ..., x_{n-1}\}$, where $1 \leqslant t \leqslant |x|$ and $n=t+1, t+2, ..., |x|$. Here, the model $\mathcal{M}$ can be realized by recurrent neural networks, attention mechanisms, or large language models, etc. After probabilistic prediction, the encoder $\mathcal{E}$ encodes the target token x_n using the standard encoding algorithms such as huffman coding or arithmetic coding, etc. The more precisely the model predicts, the smaller the discrepancy between the predicted output and the actual ground truth output will be, and thus the better result the encoder outputs.

Generally, the encoder has converged to the theoretical entropy limit during the lossless compression. Therefore, the compression performance of universal NN-based lossless compression algorithms depends mainly on the ability of the probabilistic prediction model $\mathcal{M}$ to fit the input data. In the following part, we concentrate on the design and implementation of the NN-based prediction model $\mathcal{M}$. To learn more about the encoder $\mathcal{E}$, we suggest referring to [43,44].

In [78], the NN-based lossless compressors are divided into online and offline. In this paper, following the guidance of [79], we categorize NN-based lossless universal compressors into *static pre-training* (offline), *adaptive* (online), and *semi-adaptive* methods. The comparison of these methods are shown in Table 1 (in an ideal scenario). Details for representative algorithms are given in the following subsections.

3.2 Static pre-training methods

In static pre-training methods, the compression process consists of two independent stages: (1) pre-training stage, which obtains an well-trained NN model of the to-be-compressed data, and (2) compression stage, which outputs a compact representation of the to-be-compressed data. Figure 1 presents the workflow for this category.

In the pre-training stage, the input sequences are fed into the neural network to generate a static model $\mathcal{M}_{static}$, whose parameters remain unchanged during the subsequent processes. Then, in the compression stage, the pre-trained static model is used to predict the probability distribution of the input data sequences, and the prediction results are fed into the encoder. Such methods have two advantages. Firstly, the pre-trained model can be applied to all sequences homologous with the input data, thus the pre-training stage can be omit for these homologous data. Secondly, due to the sufficient pre-training before the actual compression, the static pre-training method achieves a stable and excellent compression ratio. However, such methods also have some shortcomings. Firstly, additional computation and time overhead are entailed due to the extra pre-training stage. Furthermore, as Fig. 1 shows, the model parameters and framework information have to be saved in the compressed output file to recover the original file during the decompression. As a result, an additional amount of storage is required, and the size of model $\mathcal{M}_{static}$ should also be considered when calculating the compression ratio. Therefore, static methods are unsuitable for some small-size datasets.

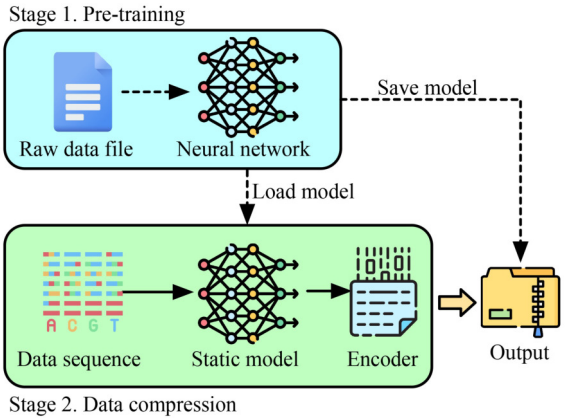


Fig. 1 The compression process of the static pre-training methods

Schmidhuber and Heil [78] were the first to propose using an NN-based prediction model combined with an encoding algorithm to achieve lossless data compression. The workflow of the compressor is as follows. First, given the to-be-compressed sequence $x=\{x_i\}_{i=1}^{|x|}$, set the sliding window (also known as context length) as t and step-size as 1, where $1 \leqslant t \leqslant |x|$. During the pre-training stage, each sequence $\{x_{n-t}, ..., x_{n-1}\}$ is embedded into the matrix $X=[X_{n-t}, ..., X_{n-1}] \in R^{t \times k}$ according to the one-hot coding, where each vector X_i has the length of $k=|\mathcal{S}|$. The NN model $\mathcal{M}_{static}$ takes the matrix X as input and predicts the probability distribution of token x_n. The network is trained by minimizing the difference between the output probability distribution and

Table 1 Overall comparison of NN-based universal lossless compression methods

Method	Model pre-training	Model saving	Compression ratio	Time cost	Representative works
Static pre-training	√	√	★★★	★	Ref [78,80,81], DeepZip [82], DecMac [83], LLMZip [84]
Adaptive	×	×	★	★★★	TRACE [37], OREA [38], PAC [39], Cmix [85]
Semi-adaptive	√	√	★★	★★	DZip (Combined Model) [79]

Notes. **Model pre-training**: Whether or not the model $\mathcal{M}$ requires pre-training. **Model saving**: Whether or not the model need to be saved and included in the compressed file. **Compression ratio**: A smaller value indicates better compression performance, detailed in the evaluation metrics section. The "√" and "×" represent "Yes" or "No", and more instances of "★" indicate a stronger advantage.

the actual conditional probability distribution. Once the model's output matches the actual conditional probability distribution, the target token and its corresponding probability distribution are input into the encoder to realize data compression. The above steps are repeated as the window slides with step size 1, and finally, the probability distribution for tokens $x_{t+1}, ..., x_{|x|}$ are all obtained. The experiments show that the proposed compressor outperforms the widely used conventional compressor Lempel-Ziv [45] regarding compression ratio. In addition, the combination of predictor and arithmetic coding can perform better than huffman coding. However, the limitation with this method is that the compression speed is slower than traditional algorithms by three orders of magnitude.

Subsequent works [80,82,83] are greatly influenced by [78]. Algorithm 1 formalizes the workflow for these methods.

Mahoney [80] proposed another improved NN-based lossless compressor. The author found that using two strategies, adaptive learning rate and learning rate decay, can effectively improve the training speed and compression effect. This work also discussed the impact of prediction accuracy on data correlations, and how to balance training and compression effects by setting a lower limit on the learning rate. The experiments prove that the proposed compressor performs excellently when compressing text data. Compared with the state-of-the-art compressors during the same period, PPM [86] and BWT-based method [87], the proposed compressor has apparent advantages in compression speed and ratio, especially when dealing with large-scale text data.

Goyal et al. [82] proposed an NN-based universal compressor, DeepZip, which integrates the Recurrent Neural Network (RNN) with arithmetic encoding. Compared with the traditional probability-distribution prediction model, the RNN model introduces recurrent connectivity, which captures the contextual relationships in the sequential data. Besides, the RNN has a specific modeling ability to establish long-term dependencies in longer sequences. This effectively solves the problem that traditional models cannot handle long sequences, and directly improves the model training effect and compression ability. For comparison, the authors investigate the compression effects of three probabilistic prediction models when they are combined with an arithmetic encoder, including i) FCNN: a Fully Connected Neural Network and two recurrent neural network variants, ii) biGRU: a multi-input single-output bidirectional Gated Recurrent Unit, and iii) LSTM-multi: a multi-input multi-output Long Short-Term Memory Network. The experimental results demonstrate that the LSTM-multi model performs best in the compression task, followed by the biGRU and FCNN models. This suggests that with the ability to capture long-term dependencies of sequences, RNNs show better performance in sequence-prediction tasks. Liu et al. [83] proposed an NN-based compressor DecMac, which employs Long Short-Term Memory Network (LSTM) and arithmetic coding. Different with RNN, the LSTM introduces three gating mechanism (input, forgetting, and output), which effectively solves the problem of gradient vanishing or gradient explosion when modeling long sequences. DecMac adopts a three-layer LSTM structure as a predictor, which better captures the dependencies in sequences and has a stronger prediction ability for target tokens. Experimental results confirm that DecMac outperforms the PAQ [88] and traditional table lookup and G-Gram based methods (such as Zip, Gzip [89], and RAR [90]) in text compression tasks.

Algorithm 1 Static pre-training compression process

Input : Sequence $\{x_1, x_2, ..., x_{|x|}\}$, and window length t.
Output : Compressed file.
Build the NN-based static model $\mathcal{M}_{static}$;
Initialize the encoder $\mathcal{E}$;
Pre-training $\mathcal{M}_{static}$ by using $\{x_1, x_2, ..., x_{|x|}\}$ and t;
$i \leftarrow 1$;
while $0 < i \leqslant t$ **do**
    Encode x_i using encoder $\mathcal{E}$;
    $i \leftarrow i + 1$;
end while
Load the trained model $\mathcal{M}_{static}$;
while $t < i \leqslant |x|$ **do**
    Obtain the probability $P(x_i|x_{i-t}, ..., x_{i-1})$ using $\mathcal{M}_{static}$;
    Encode x_i using $\mathcal{E}$ and $P(x_i|x_{i-t}, ..., x_{i-1})$;
    $i \leftarrow i + 1$;
end while

Recent advances in deep-learning technologies and accelerated hardware have prompted pre-trained Large Language Models (LLMs), which demonstrate superior performance in several natural language processing domains. Researchers on the Google team [81] advocate looking at LLMs from a compression perspective. By virtue of its outstanding predictive capability, LLMs can be combined with arithmetic encoders to achieve lossless compression. In their study, the authors evaluate the performance of three classes of compression algorithms, including i) traditional compressors Gzip and Lzma2 [91], ii) specialized compressors PNG [92] and FLAC [93] designed for images and audio, as well as iii) LLMs Chinchilla [94] and Transformers [95], both integrated with the arithmetic encoding. By ignoring model parameter size, the LLM Chinchilla (especially the version with 70 billion parameters) outperforms the other five compressors on the text dataset enwik9 [96], the image dataset ImageNet [97], and the audio dataset LibriSpeech [98]. It is interesting to note that the Chinchilla model was pre-trained on textual data only, but it demonstrated excellent compression performance on multiple types of datasets. This phenomenon suggests that the LLMs possess strong generalization ability and can adapt to various compression tasks.

Valmeekam et al. [84] proposed another NN-based compressor that integrates the large language model LLaMA [99] with three coding algorithms (zlib [100], Token-by-Token, and arithmetic coding) to perform lossless compression. Experimental results demonstrate that the algorithm exhibits the best compression efficiency with arithmetic coding and outperforms the traditional compression algorithms Zpaq [22] and Paq8h [101] on text datasets.

3.3 Adaptive methods

Different from the static pre-training methods, this methods utilize a dynamic model $\mathcal{M}_{adaptive}$ to predict the probability distribution of the to-be-compressed sequences, and the

prediction results are fed into the encoder $\mathcal{E}$. As Fig. 2 plots, each time the model predicts the probability distribution of a target token, the prediction result is transferred to the backward controller and further upgrades the model. One obvious advantage of these methods is that they do not require a pre-training process. Therefore, these methods have a lower time consumption than static pre-training methods. Besides, adaptive methods are also suitable for scenarios such as real-time data transmission. However, the shortcoming of adaptive methods is that the compression ratio is usually inferior to the static pre-training methods (under the same network architecture and scale). Algorithm 2 provides a formal description for adaptive compressors.

Cmix [85] developed by Byron is a typical adaptive compressor, which is excellent in compression ratio. The compression process of Cmix mainly consists of three parts: preprocessing, model prediction, and contextual mixing. Firstly, in the preprocessing stage, Cmix uses a specific module to recognize the data type and convert it into an easy-to-compress format. Secondly, in the model-prediction stage, The Cmix reads the data bit-by-bit and applies thousands of independent models to predict the probability of the target token. Finally, in the context-mixing phase, Cmix uses the byte-level LSTM mixer, bit-level context mixture (CM), and Secondary Symbol Estimation (SSE) algorithm to mix the probabilistic predictions of multiple models into a single probability. Experiment results show that Cmix generally outperforms existing traditional and NN-based compressors regarding compression ratio for various datasets. However, because Cmix employs thousands of prediction models, it consumes more time and memory.

The authors of Cmix [85] subsequently developed another compression algorithm, named Lstm-compress [102], which

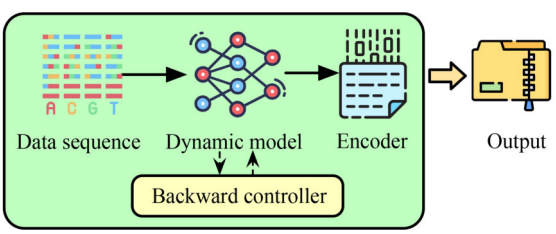


Fig. 2 Schematic of the compression process of the adaptive methods

Algorithm 2 Adaptive compression process

Input : Sequence $\{x_1, x_2, ..., x_{|x|}\}$, window length t.
Output : Compressed file.
Build the NN-based adaptive updating model $\mathcal{M}_{adaptive}$;
Initialize the encoder $\mathcal{E}$;
$i \leftarrow 1$;
while $0 < i \leqslant t$ **do**
  Encode x_i using encoder $\mathcal{E}$;
  $i \leftarrow i + 1$;
end while
while $t < i \leqslant |x|$ **do**
  Obtain the probability $P(x_i|x_{i-t}, ..., x_{i-1})$ using $\mathcal{M}_{adaptive}$;
  Encode x_i using $\mathcal{E}$ and $P(x_i|x_{i-t}, ..., x_{i-1})$;
  Backpropagation updates $\mathcal{M}_{adaptive}$ to minimize the loss;
  $i \leftarrow i + 1$;
end while

employs the LSTM model only during the prediction stage. Compared to DeepZip and DecMac, which are static pre-training compressors using the LSTM model as a predictor, Lstm-compress can compress target tokens in real-time without pre-training the model or saving model parameters to the compressed file. However, in some study cases presented in the DeepZip paper, the output file of Lstm-compress was found larger than the original file, which may be caused by the gradient vanishing or exploding during the training process or by improper setting of hyperparameters.

Proposed by Bellard, NNCP [103] uses the LSTM model as a predictor. The authors designed two structures with different network depths and parameter scales for the LSTM model. Experimental results on the datasets enwik8 and enwik9 [96] show that the NNCP algorithm with a deeper LSTM model outperforms the Gzip, XZ, and Lstm-compress algorithms regarding the compression ratio, and is second only to the Cmix compressor. In the study of NNCP, the authors also attempt to combine Transformer [31] and arithmetic coding to realize lossless compression. Transformer architecture has two advantages over traditional RNN and LSTM. On the one hand, the self-attention mechanism allows the model to process token dependencies throughout the whole sequence. On the other hand, Transformer supports parallel computing, and thus is better at processing long sequences. However, the authors found that the Transformer performed worse than the LSTM model in the compression task, which may be caused by the insufficient network depth and inadequate training.

Mao et al. [37] proposed an efficient Transformer-based universal compressor TRACE. In their work, the prediction model adopts the linear attention mechanism SLiM [104] to reduce memory and computational costs. In addition, the authors argued that the input token can be embeded into a shorter vector whose dimension is less than the hidden layer dimension. For example, on one input sequence $x = \{x_i\}_{i=n-t}^{n-1}$, a conventional Transformer would map the sequence to $X = [X_{n-t}, X_{n-t+1}, ..., X_{n-1}] \in R^{t \times D}$, where D is the hidden layer dimension. However, the method proposed by the author maps the sequence to $X = [X_{n-t}, X_{n-t+1}, ..., X_{n-1}] \in R^{t \times d}$, where d is the token dimension and $d \leqslant D$. Based on such reduction, the researchers designed the grouping strategy, in which, the neighbouring $\frac{D}{d}$ vectors in X are assigned into the same group to form a new sequence $X' = [X'_{n-\frac{t \times d}{D}}, ..., X'_{n-1}] \in R^{\frac{t \times d}{D} \times D}$. Compared with the traditional token mapping method, TRACE reduces the input sequence length without changing the parameters of Transformer, reduces the redundant information, and saves the computational cost. Therefore, the compression ratio and speed are improved. The experimental results show that TRACE outperforms Tensorflow-compress [105], NNCP, and DZip regarding peak GPU memory usage and model inference latency. With respect to the overall compression ratio, TRACE outperforms traditional algorithms Gzip, 7Z, ZSTD-19, and the NN-based algorithm DZip.

In subsequent studies from the same research team, Mao et al. [38] also designed the compression algorithm OREO, an NN-based lossless compressor that incorporates Multi-Layer Perceptron (MLP) and Ordered Masks (OM). The authors

found that during the compression process, predictors based on RNN or attentional mechanisms (e.g., DZip, NNCP, or TRACE) tend to assign higher weights to history tokens closer to the target token, thus implicitly learning the order of importance of tokens in the input sequence stream. Based on this finding, the authors apply an MLP module combined with an OM component instead of the more computationally expensive RNN and attention mechanisms. The MLP is responsible for extracting features from sequences, and the OM makes up for the shortcomings of the MLP in establishing sequence correlation. Experimental results show that OREO performs better regarding compression ratio and compression speed. In the follow-up study, the authors also developed the compression algorithm PAC [39] based on the algorithm OREO to solve the repetitive problem and the problem of intra-batch distribution variation, thus further improving the compression ratio and compression speed.

3.4 Semi-adaptive methods

The semi-adaptive methods integrate the static pre-training and adaptive methods, aiming to combine their strengths and compensate for their weaknesses. As Fig. 3 shows, the semi-adaptive compressors also include two stages: pre-training and data compression. In stage 2, it utilizes an NN-based bootstrapping model $\mathcal{M}_{bs}$, which is a pre-trained static model, as well as an NN-based supporter model $\mathcal{M}_{sup}$, which is an adaptive model coupled with a backward controller. Based on such design, semi-adaptive compression methods usually have excellent compression performance. However, their resource cost and time consumption are relatively high due to pre-training and dynamic updating. For better understanding, we also give an algorithm description of the semi-adaptive compressors, as shown in Algorithm 3.

The compression algorithm DZip proposed by Goyal et al. [79] is a representative semi-adaptive compressor. The predictor contains two structurally different models: the Bootstrap Model (BM) and the Supporter Model (SM). In the pre-training phase, DZip trains the bootstrap model $\mathcal{M}_{bs}$ by scanning the input data for multiple rounds and obtains fixed

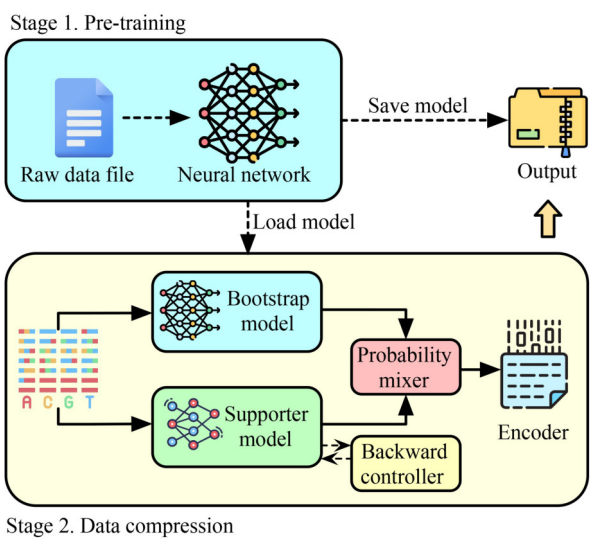


Fig. 3 The compression process of the semi-adaptive methods

Algorithm 3 Semi-adaptive compression process

Input : Sequence $\{x_1, x_2, ..., x_{|x|}\}$, window length t.
Output : Compressed file.
Build the NN-based bootstrap model $\mathcal{M}_{bs}$ and supporter model $\mathcal{M}_{sup}$;
Initialize the encoder $\mathcal{E}$;
Pre-training $\mathcal{M}_{bs}$ by using $\{x_1, x_2, ..., x_{|x|}\}$ and t;
$i \leftarrow 1$;
while $0 < i \leqslant t$ **do**
    Encode x_i using encoder $\mathcal{E}$;
    $i \leftarrow i + 1$;
end while
Combine the trained bootstrap model $\mathcal{M}_{bs}$ and supporter model $\mathcal{M}_{sup}$ into model $\mathcal{M}_{com}$;
while $t < i \leqslant |x|$ **do**
    Obtain the probability $P(x_i|x_{i-t}, ..., x_{i-1})$ using $\mathcal{M}_{com}$;
    Encode x_i using $\mathcal{E}$ and $P(x_i|x_{i-t}, ..., x_{i-1})$;
    Backpropagation updates $\mathcal{M}_{com}$ to minimize the loss;
    $i \leftarrow i + 1$;
end while

model parameters. Entering the compression stage, DZip synchronously applies the bootstrap model $\mathcal{M}_{bs}$ and supporter model $\mathcal{M}_{sup}$, integrates the outputs of the two models using the convex combination strategy, and adjusts the parameters in real time during the compression process. The experimental results show that DZip outperforms the traditional compression algorithms and other NN-based compression algorithms during the same time period in terms of compression ratio.

3.5 Brief summary of NN-based compressors

In this section, we summarize the aforementioned three types of compressors. Table 2 sketches the information of the 15 NN-based compressors, including the publication year, methodology, programming language, and primary prediction method.

i) To the best of our knowledge, static and adaptive compressors are still the mainstream of research because they correspond to a wider range of application scenarios. In contrast, semi-adaptive method has only yielded one proposal, namely Dzip (Combined), mainly because balancing model complexity and compression ratio is challenging, and is influenced by various factors such as the number of training epochs and the selection of static models. In general, static compressors are suitable for medium to long-term data storage, and adaptive compressors are suitable for real-time transmission.

ii) As can be seen from Table 2, NN-based compressors are generally implemented using Python because it encapsulates most mainstream deep-learning frameworks, such as PaddlePaddle [106], Tensorflow [107], Pytorch [108], and provides flexible and convenient interfaces.

iii) Recurrent neural networks and their variants are the primary prediction cores for NN-based compressors (such as Cmix, Lstm-compress, DeepZip, NNCP, DecMac, Tensorflow-compress, and Dzip), whether for static, adaptive, or semi-adaptive methods. This is because architectures like RNNs, biGRUs, and LSTMs can effectively capture relational features between sequences, thus possessing stronger predictive and compression capabilities.

Table 2 Brief summary of the NN-based universal compressors

Name	Year	Method	Language	Major prediction model components
Ref. [78]	1996	Static	Unknown	Easy feedforward neural network
Ref. [80]	2000	Static	Unknown	Easy feedforward neural network
Cmix [85]	2014	Adaptive	C/C++	Multi-model mixing, long short-term memory mixer, bit-level context mixture, and secondary symbol estimation (SSE) algorithm
Lstm-compress [102]	2016	Adaptive	C/C++	Long short-term memory
DeepZip [82]	2018	Static	Unknown	Recurrent neural network, fully connected neural network, bidirectional gated recurrent unit, and long short-term memory
NNCP [103]	2019	Adaptive	C/C++	Long short-term memory and transformer
DecMac [83]	2019	Static	Unknown	Long short-term memory
Tensorflow-compress [105]	2020	Adaptive	Python	Long short-term memory network
Dzip (Bootstrap) [79]	2021	Adaptive	Python	Bidirectional gated recurrent unit
Dzip (Combined) [79]	2021	Semi-adaptive	Python	Bidirectional gated recurrent unit and residual network
TRACE [37]	2022	Adaptive	Python	Transformer
OREA [38]	2022	Adaptive	Python	Multi-layer perceptron and gumbel-softmax
PAC [39]	2023	Adaptive	Python	Multi-layer perceptron and learned ordered masks
LLMZip [84]	2024	Static	Python	Large language model LLaMA
LMIC [81]	2024	Static	Python	Large language models chinchilla and transformer

iv) To design compressors based on Transformer and Large language models is the main research direction in the future, especially for the static compressors. These models are trained extensively using large-scale multi-modal data, which endows them with powerful deep learning capabilities. As a result, they exhibit better generalization in compressing multi-source data.

4 Evaluation metrics

Traditional lossless compression methods usually use metrics such as compression ratio, throughput, time cost, and peak memory consumption to evaluate the performance of compression and decompression algorithms [7,49,50,109]. However, different from traditional CPU-based compression algorithms, in NN-based compressors, the performance of the model impacts the compressing effect significantly. Therefore, we advocate that model performance should be considered when we evaluate NN-based lossless universal compression algorithms. In addition, NN-based lossless compression methods usually use specialized hardware devices to accelerate the computation, e.g., GPUs, TPUs DPUs, and FPGAs. Thus, the computational resource consumption of the hardware should also be considered. Based on the literature [37–39,79,110], we summarize the evaluation metrics for the NN-based lossless compressors from three aspects, including compression effect, resource consumption, and model performance.

4.1 Evaluation of compression effect

The compression effect metrics evaluate the compressor's ability to save space and stay robust. In this paper, we suggest using Compression Ratio (*CR*) and Storage Saving Percentage (*SSP*) to evaluate the algorithm's effectiveness in saving storage space, and using Compression Robust Performance (*CRP*) to evaluate the robustness of the compression effect. The definitions and descriptions are as follows.

Let $Size(F_b)$ and $Size(F_a)$ denote the size (in Bytes) of the file before and after compression, the *CR* and *SSP* are calculated as follows:

$$CR = \frac{Size(F_a)}{Size(F_b)} \times 8\ bits/base, \tag{1}$$

$$SSP = \left(1 - \frac{Size(F_a)}{Size(F_b)}\right) \times 100\%. \tag{2}$$

In Eq. (1), the *CR* represents the average number of bits required to store a basic unit of the original data. For image compression, a data unit is a pixel; for text compression, a data unit is a character. Here, the smaller *CR* value, the fewer bits required for the algorithm to store a basic unit of the original data, and the better the compression performance. In Eq. (2), the *SSP* denotes the percentage of storage space saved by the compression algorithm, and larger *SSP* values indicate better compression performance. For example, if $Size(F_b)$ and $Size(F_a)$ take 12587 B and 33269 B, then the $CR = \frac{12587}{33269} \times 8 = 3.027 bits/base$ and the $SSP = 1 - \frac{12587}{33269} \times 100 = 62.166\%$.

In addition, to avoid the influence of data probability distribution, the overall compression performance of the algorithm on multi-source data is evaluated using Compression Robust Performance (*CRP*) and Average or Weighted Average Compression Ratio (*Avg*/*WavgCR*), as well as Average or Weighted Space Savings Percentage (Avg/WavgSSP). Let $Size(F_b^i)$ and $Size(F_a^i)$ denote the ith file size before and after compression, where $i \leqslant N$ and N denotes the number of total files. The *CRP*, *Avg*/*WavgCR*, and *Avg*/*WavgSSP* are defined as follows:

$$CRP = \frac{\sqrt{\sum_{i=0}^{N-1}(CR_i - CR_u)^2}}{\sqrt{N} \times CR_u} \times 100\%, \tag{3}$$

$$Avg/WavgCR = \sum_{i=0}^{N-1} \alpha_i \times CR_i, \tag{4}$$

$$Avg/WavgSSP = \sum_{i=0}^{N-1} \alpha_i \times SSP_i. \tag{5}$$

In Eqs. (3)−(5), CR_i and SSP_i denote the compression ratio and storage saving percentage of the ith file obtained by

Eqs. (1) and (2), and $CR_u = \frac{\sum_{i=0}^{N-1} CR_i}{N}$ denotes the average compression ratio obtained by running the algorithm on N datasets. The α_i indicates the weighted factor. For the *AvgCR*,we have $\alpha_i = \frac{1}{N}$. For the *WavgCR*, we have $\alpha_i = \frac{Size(F_a^i)}{\sum_{j=0}^{N-1} Size(F_a^j)}$. In Eq. (3), the *CRP* is calculated in the same way as the Coefficient of Variation (CV) [110], which is calculated as the ratio of standard deviation and mean value multiplied by 100%. The smaller the *CRP* value, the less the algorithm is perturbed by the probability distribution of the tested datasets and the higher the compression robustness.

4.2 Evaluation of resource consumption

In recent years, the emergence of acceleration hardware devices such GPUs has created new opportunities for NN-based lossless compression [37,111]. Therefore, besides evaluating the usage of system memory by traditional metrics such as the Compression Peak Memory (*CPM*) and Decompression Peak memory (*DPM*) consumption, we suggest including the GPU Peak Memory (*GPM*) consumption. Here, smaller *CPM/DPM* and *GPM* values mean that the algorithm uses less computational resources, thus can run better on performance-constrained devices. Besides, the Compression Time (*CT*) and Decompression Time (*DT*) should also be included in resource consumption metrics, especially for real-time data transfer scenarios. Smaller *CT* and *DT* imply better compression and decompression performance.

To avoid the impact of extreme probability distributions of the tested dataset on the compressor's performance, we suggest using average memory consumption (*AvgCPM*, *AvgDPM*) and total time cost (*TotalCT*, *TotalDT*) to evaluate the overall resource-consuming performance of the compression algorithms across all datasets.

4.3 Evaluation of model performance

As introduced in Section 3, the NN-based compression algorithms includes two parts, a probability prediction model and an encoder, while the former plays a primary role in the compression effect. Therefore, we suggest evaluating the performance of prediction models for NN-based methods. We use the same three model performance metrics as in the works [37–39], i.e., the total number of model Parameters (*Params*), Floating Point Operations (*FLOPs*), and Model Inference Latency (*MIL*). Here, the *Params* is closely related to GPU memory consumption, representing the spatial complexity of the model. The *FLOPs* denote the overall computational quantity of completing one compression operation, representing the time complexity of the model. The *MIL* refers to the time required for the model to complete one operation, which is typically calculated by dividing the total time by the number of operations. Smaller values for all three metrics indicate that the model consumes less computational power and resources.

5 Benchmark testing and analysis

5.1 Experimental platform, datasets, and algorithms

In order to facilitate a comprehensive and fair comparison of NN-based compression algorithms, we have implemented a benchmark test. All experiments were conducted on a GPU server equipped with 4 × Intel Xeon Silver 4310 CPUs (2.10 GHz, 48 cores in total), 4 × NVIDIA GeForce RTX 4090 GPUs (16384 CUDA cores, 24 GB of GPU memory), and 128 GB of DDR4 RAM. The server runs the Linux operating system Ubuntu 20.04.6 LTS.

We used 28 open-source datasets with a total data size of 8482 MB. These datasets include various types and formats, such as text, image, and audio. Table 3 provides detailed information about the experimental datasets.

We tested the performance of 17 state-of-the-art universal lossless compressors (8 NN-based and 9 Traditional methods) on the benchmark datasets, the detailed information about these compressors is presented in Table 4. Download links of datasets and execute scripts of compressors can be found in Supplementary Material Section S1−S2, and Table S1.

5.2 Results

The overall compression effect and resource consumption of 14 compressors (except for LLMZip, Cmix, and X3) on 28 datasets are shown in Table 5. Because LLMZip, Cmix, and X3 require over 36 hours to run datasets larger than 100 MB, the results of their runs on some datasets are documented in Supplementary Material Tables S2−S4.

5.2.1 Compression effect

Among the remaining 14 algorithms (expect for LLMZip, Cmix, and X3), as Table 5 shows, the NNCP with a deeper LSTM model outperformed the other 14 algorithms on *WavgCR* and *WavgSSP*. The two adaptive compressors PAC and TRACE ranked second and third on *WavgCR* and *WavgSSP*. The semi-adaptive method DZip ranked fourth in terms of *WavgCR* and first on *AvgCR*. For static pre-training and semi-adaptive methods the size of the pre-trained static model affects the compression ratio, so some of them are more suitable for compressing relatively large datasets to eliminate the impact of the model size. For example, the semi-adaptive method DZip* lags behind all other algorithms except DeepZip* on *AvgCR* but outperforms all traditional algorithms on *WavgCR*, indicating that DZip* behaves better when compressing larger files. The DeepZip adopted the static pre-training method, which did not work well. When applies DeepZip to datasets image (D17) and enwik9 (D19), the compressed file size is twice as large as the original file, which may be caused by gradient vanishing or gradient explosion during the pre-training process. LSTM-compress has the same problem, e.g. its compression ratio on dataset dna (D27) is much worse than other NN-based and traditional algorithms (see Supplementary Material Tables S5−S7). Because DZip utilizes a deeper support model $\mathcal{M}_{sup}$ during training, the parameters are adjusted to avoid the same problem as DeepZip.

The metric *SSP* reflects the space-saving ability of the tested algorithms. Table 5 also showed that the *AvgSSP* and *WavgSSP* of NN-based compression algorithms are also better than that of traditional algorithms in general. For example, the *AvgSSPs* of DZip and NNCP exceeded 68%, which means that they save an average of more than 68% of storage space

Table 3 Detailed information of used datasets

ID	Name	Type	File size (Byte)	Alp-Size ($\|\mathcal{S}\|$)	Entropy ($\|\log_2^{\|\mathcal{S}\|}\|$)	Description
D1	xml [111]	Text	5345280	104	6.700	Files in xml format
D2	ooffice [111]	Heterogeneous	6152192	256	8.000	Files consisting of Office programs
D3	reymont [111]	Text	6627202	256	8.000	A pdf file with the contents of Reymont's book
D4	sao [111]	Homogeneous	7251944	256	8.000	Files containing information of 258, 996 stars
D5	x-ray [111]	Image	8474240	256	8.000	12-bit grayscale scaled x-ray medical image of a child's hand
D6	mr [111]	Image	9970564	256	8.000	A magnetic resonance medical image of the head
D7	osdb [111]	Heterogeneous	10085684	256	8.000	Open source database files for testing
D8	dickens [111]	Text	10192446	100	6.644	Text file consisting of multiple novels by Dickens
D9	samba [111]	Heterogeneous	21606400	256	8.000	An collected open source project
D10	nci [111]	Homogeneous	33553445	62	5.954	Files in SDF format
D11	webster [111]	Heterogeneous	41458703	98	6.615	An English dictionary stored in HTML format
D12	mozilla [111]	Homogeneous	51220480	256	8.000	The executable file Mozilla
D13	enwik8 [96]	Text	100000000	205	7.679	First 10^8 bytes of the English Wikipedia dump on 2006
D14	text8 [96]	Text	100000000	27	4.755	First 10^8 bytes of the English Wikipedia (only text) dump on 2006
D15	MNIST [112]	Image	54880032	256	8.000	A widely studied dataset containing handwritten digital images
D16	CIFAR-10 [113]	Image	186213868	256	8.000	A standard dataset of images with multiple categories
D17	ImageNet [97]	Image	745823247	256	8.000	Training datasets in task3 from ISLVRC on 2012
D18	ImageTest [114]	Image	470611702	256	8.000	A new 8-bit benchmark dataset for image compression evaluation
D19	Silesia [111]	Heterogeneous	211938580	256	8.000	A heterogeneous corpus of 12 documents with various data types
D20	Backup [37]	Heterogeneous	1000000000	256	8.000	10^9 bytes random extract from the disk backup of TRACE
D21	enwik9 [96]	Text	1000000000	206	7.687	First 10^9 bytes of the English Wikipedia dump on 2006
D22	Book [31]	Text	1000000000	202	7.658	First 10^9 bytes of BookCorpus
D23	ESC [115]	Audio	220522000	256	8.000	First 500 audio files of the ESC
D24	Command [116]	Audio	327759206	256	8.000	First 10000 audio files of the Google Speech Commands Dataset
D25	LibriSpeech [117]	Audio	359034309	256	8.000	Development set ("clean" speech) of LibriSpeech ASR corpus
D26	LJSpeech [117]	Audio	293847664	256	8.000	First 10000 audio files of the LJ Speech Dataset
D27	DNACorpus [118]	Genome	685597124	4	2.000	A corpus of DNA sequences from 15 different species
D28	ERR7091247 [119]	Genome	1926041160	32	5.000	A collection of genomics sequencing dataset with FastQ format

Table 4 Detailed information of tested universal lossless compression algorithms

Algorithm	Year	Language	Parallelism	Methods	Hyperparameters
NN-based methods:					
Cmix [85]	2014	C/C++	Unknown	NN, CM, AC	
Lstm-compress [102]	2016	C/C++	Unknown	NN, LSTM, AC	
NNCP [103]	2019	C/C++	GPU-parallel	NN, LSTM, AC	-t 16, -cuda
DeepZip [82]	2019	Python	GPU-parallel	NN, GRU/LSTM, AC	-bs 64, -ts 64
DZip [79]	2021	Python	GPU-parallel	NN, GRU/LSTM, AC	-bs 64, -ts 64
TRACE [37]	2022	Python	GPU-parallel	NN, Transformer, AC	-bs 512, -ts 8
PAC [39]	2023	Python	GPU-parallel	NN, MLP, AC	-bs 512, -ts 1
LLMZip [84]	2024	Python	GPU-parallel	NN, LLM, AC	-ts 511, -node 1
Traditional methods:					
Gzip [89]	1996	C/C++	Multi-Cores	LZ77	
PBzip2 [120]	2015	C/C++	Multi-Cores	BWT, HC	-p = 16, -9, -m = 2000
XZ [16]	2015	C/C++	Multi-Cores	LZ77	-t = 16, -9, -e
BSC [19]	2016	C/C++	Multi-Cores	BWT	-e2
SnZip [121]	2016	C/C++	Unknown	LZ77	-t = snzip
Lzma2 [91]	2022	C/C++	Multi-Cores	LZ77, AC	-mmt = 16, -m0 = lzma2, -mx9
PPMD [122]	2022	C/C++	Multi-Cores	PPM, AC	-mmt = 16, -m0 = ppmd
X3 [123,124]	2023	C/C++	Unknown	DC	
LZ4-multi [125]	2024	C/C++	Multi-Cores	LZ77	-t 16 -best

Notes. **Year**: the date of publication of the literature or toolkit used; **Language**: the main programming language for compressors; **Parallelism**: the parallel mechanism of the compressor, where "**Multi – Cores**" indicates that the algorithm supports parallel computation using multiple CPU cores (threads), "**GPU – Parallel**" indicates that the algorithm supports GPU parallel computation and "**Multi – Nodes**" denotes the compressor support multiple GPU nodes for parallel; **Methods**: the main techniques used; **CM** (Context Mixing): bits, modeled by combining predictions of independent models; **BWT** (Burrows-Wheeler Transform): bytes are sorted by context, then modeled by order-0 symbol ranking; **AC** (Arithmetic Coding): commonly used with dynamic modeling methods; **LZ77**: repeated strings are coded by offset and length of previous occurrence; **DC** (Dictionary-based compressor); **PPM** (Prediction by Partial Match); **HC** (Huffman Coding); **MLP** (Multi-Layer Perceptron); **LSTM** (Long Short-Term Memory); **GRU** (Gated Recurrent Units); **NN** (Neural Networks); **Hyperparameters**: default or document-recommended parameters.

for all datasets. SnZip had excellent running speed and memory usage performance, but the *AvgSSP* was only 36.244%, significantly lower than other traditional compressors.

Table 5 Compression effect of different universal lossless compression algorithms on the benchmark datasets

Algorithm	*WavgCR* (bits/base)	*AvgCR* (bits/base)	*WavgSSP*/%	*AvgSSP*/%	*CRP*/%	*TotalCT* (Hour)	*TotalDT* (Hour)	*AvgCPM* (GB)	*AvgDPM* (GB)
NN-based methods:									
NNCP [103]	**4.183** #1	**2.521** #2	**47.713** #1	**68.476** #2	13.084	942.928	926.049	0.111	0.111
PAC [39]	**4.327** #2	**2.638** #3	**45.912** #2	**67.019** #3	12.720	74.398	116.868	6.102	6.295
TRACE [37]	**4.411** #3	2.718	**44.867** #3	66.032	12.486	69.128	131.110	6.106	6.449
DZip [79]	4.494	**2.516** #1	43.819	**68.545** #1	14.272	332.787	148.374	10.113	4.790
DZip* [79]	4.562	3.802	42.971	52.476	11.158	332.787	148.374	10.113	4.790
Lstm-compress [102]	5.340	3.023	33.252	62.208	13.498	493.762	482.245	**0.009** #3	**0.009** #3
DeepZip* [82]	16.835	7.045	−110.434	11.933	18.504	250.714	52.449	13.708	4.292
DeepZip [82]	16.865	5.760	−110.811	28.003	24.092	250.714	52.449	13.708	4.292
Traditional methods:									
BSC [19]	4.826	2.928	39.677	63.394	13.045	0.353	0.300	0.121	0.116
Lzma2 [91]	4.912	3.122	38.590	60.967	12.289	0.584	0.030	1.264	0.427
XZ [16]	4.923	3.118	38.463	61.021	12.365	0.879	0.040	1.612	0.504
PPMD [122]	4.960	3.025	38.001	62.181	12.934	0.893	0.953	0.226	0.225
PBzip2 [120]	5.052	3.275	36.845	59.062	11.798	**0.024** #1	**0.016** #2	0.115	0.084
Gzip [89]	5.351	3.862	33.113	51.728	**10.342** #3	0.451	0.026	**0.002** #1	**0.002** #1
LZ4-multi [125]	5.618	4.280	29.770	46.501	**9.656** #2	**0.064** #3	**0.009** #1	0.116	0.025
SnZip [121]	5.981	5.100	25.235	36.244	**7.473** #1	**0.031** #2	**0.021** #3	**0.003** #2	**0.003** #2

Notes. For the compressors DeepZip and DZip, the results for considering the model are marked with asterisks "∗". The top-3 performance results are shown in boldface, with "#" indicating the performance rank. We ensure data integrity by comparing the hash values of the uncompressed and compressed files. The Supplementary Material Section S3 Tables S5∼S20 provide *CR*, *CT*, *DT*, *CPM*, and *DPM* results on all 28 datasets for each tested compressor.

To some extent, the *CRP* reflects the compressor's sensitivity to datasets, but algorithms with high sensitivity may perform badly in compression ratio. For example, the NNCP, whose compression ratio was the highest, behaved poorly in robustness (13.084%), while the SnZip, whose compression ratio was the lowest, had the best robustness (7.473%).

5.2.2 Resource consumption

For resource consumption, the *AvgCPM*, *AvgDPM*, *TotalCT*, and *TotalDT* values of NN-based compressors are much higher than that of traditional algorithms.

In our benchmark results, the RNN-based compression algorithm NNCP had the best compression ratio but the worst compression and decompression time cost (1869.022 hours), five orders of magnitude slower than the fastest traditional compression algorithm PBzip2 (0.040 hours). The advanced NN-based compressors TRACE (200.238 hours) and PAC (191.266 hours) were faster than the previous NNCP, Lstm-compress (967.008 hours), DeepZip (303.163 hours), and DZip (481.161 hours). This is because the TRACE employs a single-layer Performer structure, byte grouping, and shared feed-forward network to improve the compression speed. The PAC designs a progressive compression framework and employs a more lightweight MLP as a probability distribution predictor. The semi-adaptive algorithm DZip with an additional Support Model had a significantly slower speed than the static method DeepZip, especially at the decompression process. Regarding peak memory consumption, except for NNCP (0.111 GB) and Lstm-compress (0.009 GB), other NN-based compression algorithms performed worse than traditional compressors.

It is worth mentioning that it is hard to assess the GPU memory usage of compressors Cmix, NNCP, and Lstm-compress, therefore we only tested the *GPM* of DeepZip, Dzip, TRACE, and PAC. The detailed explanations are provided in the next subsection.

5.2.3 Model performance

We tested the model performance of 4 NN-based compressors, which are all implemented in Python language and also support GPU parallel acceleration, including DeepZip, DZip, TRACE, and PAC. We set batchsize and timestep for all algorithms to 512 and 16, respectively. The results are shown in Table 6.

From Table 6, it can be seen that the prediction model of DeepZip is more lightweight and outperforms the other three algorithms regarding *GPM*, *FLOPs*, *Params*, and *MIL*. Conversely, DZip applies a deeper model than DeepZip to improve the compression ratio, thus its prediction model is the heaviest among all models, and it performs worse than the other three algorithms in all metrics, especially *Params* and *FLOPs*. TRACE adopts a single-layer Performer and a shared feed-forward neural network, resulting in only a slightly higher parameter count compared to DeepZip. PAC fuses MLP and ordered masking as a probability prediction model, and although its parameter count is nearly double that of TRACE, its *FLOPs* are less than half of TRACE, and its *GPM* and *MIL* are also significantly lower than TRACE. This also indicates that the computational cost and resource consumption of the MLP-based compressor are considerably lower than that of the Transformer-based compressor.

Table 6 Model performance and *GPM* results of NN-based compressors

Algorithm	*GPM* (GB)	*FLOPs* (Giga)	*Params* (Million)	*MIL* (Millisecond)
DeepZip [82]	0.364	3.477	0.606	0.811
DZip [79]	0.496	16.563	26.180	1.635
TRACE [37]	0.431	9.133	2.366	1.614
PAC [39]	0.175	4.336	8.476	1.345

Notes. Since the GPU consumption for model compression and decompression is nearly identical, we only evaluated the algorithm's compression GPU peak memory usage.

6 Limitations and future directions

After reviewing the literature and conducting benchmark tests, we found that the NN-based lossless universal compressors are more effective than traditional methods in terms of compression ratio. This is because deep neural units can learn the probability distribution of compressed data well. However, software, hardware, and time consumption currently limit the widespread adoption of NN-based compressors. To further help develop NN-based lossless universal compressors, this section outlines four limitations and potential research directions, as follows.

6.1 Deep model compression

In the benchmark testing, we found that the size of model parameters is an essential factor affecting the compression ratio of NN-based compressors. However, it seems that not every neuron in the network is necessary. For example, the PAC [39] compressor has a parameter size of 8.476 million, while the TRACE [37] compressor has a size of 2.366 million. While the parameter size of the former is 3.584 times higher than that of the latter, the increase of *AvgCR* and *WavgCR* values are only 2.943% and 1.904%, respectively. In fact, Qin et al. [126] investigated the relationship between model size and compressed data size and achieved a uniform structural reduction of the model. They proposed model compression as an effective data-compression method and established a comprehensive pre-training approach under the lasso framework [126,127]. Unlike other domains in deep learning science (such as disease prediction, drug target prediction, protein spatial structure, and financial default prediction), NN-based compression prefers to overfit the model according to the dataset and maximizes the learning for the original probability distribution. Therefore, to prune and compress the model specifically for NN-based compressors using this characteristic will help improve the compression ratio and also help reduce model training cost and inference latency.

6.2 Automated parameter tuning

Our research has revealed that the perturbations of input data probability distributions greatly influence the effectiveness of the NN-based compressors. For instance, the NNCP [103] algorithm, which achieves the best compression ratio among all compressors, exhibits a *CRP* value of 3.084% only, demonstrating the poor robustness to different datasets. The TRACE [37] algorithm achieves the best compression ratio on the dataset nci (D10), while it behaves the worst on the dataset image (D17), resulting in a performance difference of 19.956 times (see Supplementary Material Section S3 Table 12). The poor robustness issue may be due to the fixed model architectures. For specific, existing compressors employ fixed model architectures to process all kinds of data streams without exploring the probability distribution of the input dataset or adjusting the model size flexibly based on the data probability distribution. Therefore, adapting the model architectures (i.e., model size and model parameters) automatically to fit the tested dataset's data probability distribution and the alphabet size might improve the compression ratio and reduce time cost.

6.3 System-aware parallel compression

According to the testing results, we noticed that the tested algorithms failed to fully utilize the CPU and GPU resources. For example, our computer system had 4*12 CPU cores and 4*16384 GPU computing units, but these cores or units were not fully utilized during the compressing process. Furthermore, our compression system had memory of 128 GB on the CPU side and 24 GB on the GPU side, but the resource utilization was low during the compression and decompression. In our tests, even the high *CPM*-cost compressor Cmix [85] consumed a maximum of 19.174 GB memory merely, while the high *GPM*-cost compressor DZip [79] consumed a maximum of 0.496 GB memory only. Besides, NN-based compressors require longer compression and decompression times compared to traditional compressors, which restricts the use of compressors in real-time compression application scenarios. For example, in our testing, the total time spent on compression and decompression for TRACE was 200.238 hours, whereas for PPMD, it was only 1.846 hours. The time difference was as high as 108.471 times.

Therefore, designing system-aware parallel compressors that effectively utilize system resources, maximize compression ratios, and save time cost holds broad application prospects, particularly for long-term data backup. In addition, designing system-aware parallel compressors can improve the robustness of compressors, particularly when the compression servers offer limited computing resources, such as memory.

6.4 Incremental and searchable compression

Through our benchmark tests, the NN-based lossless universal compression algorithms have demonstrated overwhelming advantages over traditional methods in terms of the compression ratio, although the resource consumption is relatively high. With the rapid improvement of hardware acceleration device, these promising compression algorithms are expected to further break through performance bottlenecks in the coming years. Therefore, we believe that NN-based compressors will be widely applied in the future. Considering practical application demands, incremental and searchable compression algorithms have a wide prospect for multiple application and research scenarios. For instance, when we compress time series or genetic sequencing data for data backup, newly generated data often strongly correlate with the historical data. To design and implement incremental compression techniques, the redundancy characteristics between compressed and uncompressed data can be fully explored. Therefore, the compression ratio and the processing time is optimized, and the storage and sharing costs are lowered. Additionally, retrieving data from massive compressed datasets is computationally intensive. Thus, designing efficient data retrieval algorithms to access the compressed data on demand is a meaningful research direction.

7 Conclusion

In this paper, we have conducted a comprehensive and essential literature survey and algorithm evaluation work.

Specifically, i) we thoroughly reviewed the universal lossless compression algorithms designed for multi-source data based on neural networks technologies and provided a systematic classification guide for NN-based compressors. ii) We unified the NN-based compression metrics and benchmarked the state-of-the-art NN-based and traditional compression tools on 28 multi-source real-wolrd datasets, testing the compression effect, resource consumption, and model performance. iii) We also summarized the limitations and future directions for the NN-based lossless universal compressors.

Based on the current work, future studies will involve testing more NN-based and traditional compression tools and provide a more detailed classification method for NN-based compressors. Additionally, designing and implementing an automated universal lossless compressors evaluation framework would be valuable foundational work.

Acknowledgements This work was partly supported by the National Natural Science Foundation of China (Grant Nos. 62272253 and 62272252) and the Fundamental Research Funds for the Central Universities. It was also supported in part by the China Scholarship Council (CSC202406200085) and the Innovation Project of Guangxi Graduate Education (YCBZ2024005). The High-performance Computing Center of Guangxi University partly supported the experimental work. The authors thank the editor and anonymous reviewers for their constructive comments and suggestions for improving our manuscript.

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

Electronic supplementary material Supplementary material is available in the online version of this article at journal.hep.com.cn and link.springer.com.

References

1. Schaller R R. Moore's law: past, present and future. IEEE Spectrum, 1997, 34(6): 52−59
2. Rydning J. Global DataSphere, Data Marketplaces, and Data as a Service. See idc. com/getdoc.jsp?containerId=IDC_P38353 website, 2023
3. Sun H, Ma H, Zheng Y, Xie H, Wang X, Liu X. SR2C: a structurally redundant short reads collapser for optimizing DNA data compression. In: Proceedings of the 29th IEEE International Conference on Parallel and Distributed Systems. 2023, 60−67
4. Ji Z, Zhou J R, Jiang L, Wu Q H. Overview of DNA sequence data compression techniques. Acta Electronica Sinica, 2010, 38(5): 1113−1121
5. Numanagić I, Bonfield J K, Hach F, Voges J, Ostermann J, Alberti C, Mattavelli M, Sahinalp S C. Comparison of high-throughput sequencing data compression tools. Nature Methods, 2016, 13(12): 1005−1008
6. Kredens K V, Martins J V, Dordal O B, Ferrandin M, Herai R H, Scalabrin E E, Ávila B C. Vertical lossless genomic data compression tools for assembled genomes: a systematic literature review. PLoS One, 2020, 15(5): e0232942
7. Sun H, Zheng Y, Xie H, Ma H, Liu X, Wang G. PMFFRC: a large-scale genomic short reads compression optimizer via memory modeling and redundant clustering. BMC Bioinformatics, 2023, 24(1): 454
8. Sun H, Zheng Y, Xie H, Ma H, Zhong C, Yan M, Liu X, Wang G. PQSDC: a parallel lossless compressor for quality scores data via sequences partition and run-length prediction mapping. Bioinformatics, 2024, 40(5): btae323
9. Ai D, Lu H Y, Yang Y R, Liu Y H, Lu J, Liu Y. A brief overview 3D point cloud data compression technology. Journal of Xi'an University of Posts and Telecommunications, 2021, 26(1): 90−96
10. Chen X, Tian J, Beaver I, Freeman C, Yan Y, Wang J, Tao D. FCBench: cross-domain benchmarking of lossless compression for floating-point data. Proceedings of the VLDB Endowment, 2024, 17(6): 1418−1431
11. Mishra D, Singh S K, Singh R K. Deep architectures for image compression: a critical review. Signal Processing, 2022, 191: 108346
12. Jamil S, Piran M J, Rahman M U, Kwon O J. Learning-driven lossy image compression: a comprehensive survey. Engineering Applications of Artificial Intelligence, 2023, 123: 106361
13. Bourai N E H, Merouani H F, Djebbar A. Deep learning-assisted medical image compression challenges and opportunities: systematic review. Neural Computing and Applications, 2024, 36(17): 10067−10108
14. Tian T, Wang H. Large-scale video compression: recent advances and challenges. Frontiers of Computer Science, 2018, 12(5): 825−839
15. Im S K, Ghandi M M. Improved rate-distortion optimized video coding using non-integer bit estimation and multiple Lambda search. Frontiers of Computer Science, 2016, 10(1): 157−166
16. Lasse C. The official website of the XZ compressor. See tukaani.org/xz/ website, 2015
17. Meta. Zstandard-Fast real-time compression algorithm. See facebook/zstd: Zstandard - Fast real-time compression algorithm website, 2024
18. Google. Brotli compression format. See github.com/google/brotli website, 2024
19. IlyaGrebnov. High performance block-sorting data compression library. See github.com/IlyaGrebnov/libbsc website, 2024
20. Michael. szip homepage. See compressconsult.com/szip/ website, 2002
21. Julian S. The official website of the Bzip2 compressor. See sourceware.org/bzip2/ website, 2019
22. Mahoney M. Incremental journaling backup utility and archiver. See mattmahoney.net/dc/zpaq website, 2016
23. mathieuchartier. MCMfile compressor. See github.com/mathieuchartier/mcm website, 2016
24. Margaritov. Hutter prize submission 2021a: STARLIT + cmix. See github.com/amargaritov/starlit website, 2021
25. Aslanyurek M, Mesut A. A static dictionary-based approach to compressing short texts. In: Proceedings of the 6th International Conference on Computer Science and Engineering. 2021, 342−347
26. Kasneci E, Sessler K, Kuchemann S, Bannert M, Dementieva D, Fischer F, Gasser U, Groh G, Günnemann S, Hüllermeier E, Krusche S, Kutyniok G, Michaeli T, Nerdel C, Pfeffer J, Poquet O, Sailer M, Schmidt A, Seidel T, Stadler M, Weller J, Kuhn J, Kasneci G.

ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 2023, 103: 102274

27. Wei C, Wang Y C, Wang B, Kuo C C J. An overview on language models: recent developments and outlook. 2023, arXiv preprint arXiv: 2303.05759
28. Thirunavukarasu A J, Ting D S J, Elangovan K, Gutierrez L, Tan T F, Ting D S W. Large language models in medicine. Nature Medicine, 2023, 29(8): 1930−1940
29. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735−1780
30. Yu Y, Si X, Hu C, Zhang J. A review of recurrent neural networks: LSTM cells and network architectures. Neural Computation, 2019, 31(7): 1235−1270
31. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 6000−6010
32. Huang Y, Xu J, Lai J, Jiang Z, Chen T, Li Z, Yao Y, Ma X, Yang L, Chen H, Li S, Zhao P. Advancing transformer architecture in long-context large language models: a comprehensive survey. 2024, arXiv preprint arXiv: 2311.12351
33. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. OpenAI Blog, 2019, 1(8): 9
34. Devlin J, Chang M W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2018, 4171−4186
35. Gu A, Dao T. Mamba: linear-time sequence modeling with selective state spaces. 2024, arXiv preprint arXiv: 2312.00752
36. Beck M, Pöppel K, Spanring M, Auer A, Prudnikova O, Kopp M, Klambauer G, Brandstetter J, Hochreiter S. xLSTM: extended long short-term memory. 2024, arXiv preprint arXiv: 2405.04517
37. Mao Y, Cui Y, Kuo T W, Xue C J. TRACE: a fast transformer-based general-purpose lossless compressor. In: Proceedings of ACM Web Conference 2022. 2022, 1829−1838
38. Mao Y, Cui Y, Kuo T W, Xue C J. Accelerating general-purpose lossless compression via simple and scalable parameterization. In: Proceedings of the 30th ACM International Conference on Multimedia. 2022, 3205−3213
39. Mao Y, Li J, Cui Y, Xue J C. Faster and stronger lossless compression with optimized autoregressive framework. In: Proceedings of the 60th ACM/IEEE Design Automation Conference. 2023, 1−6
40. Zhong C, Sun H. Parallel algorithm for sensitive sequence recognition from long-read genome data with high error rate. Journal on Communications, 2023, 44(2): 160−171
41. Sayood K. Introduction to Data Compression. 5th ed. Sydney: Morgan Kaufmann, 2017
42. Shannon C E. A mathematical theory of communication. The Bell System Technical Journal, 1948, 27(3): 379−423
43. Moffat A. Huffman coding. ACM Computing Surveys (CSUR), 2019, 52(4): 85
44. Langdon G G. An introduction to arithmetic coding. IBM Journal of Research and Development, 1984, 28(2): 135−149
45. Ziv J, Lempel A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 1977, 23(3): 337−343
46. Schindler M. A fast block-sorting algorithm for lossless data compression. In: Proceedings of 1997 Data Compression Conference. 1997, 469
47. Capon J. A probabilistic model for run-length coding of pictures. IRE Transactions on Information Theory, 1959, 5(4): 157−163
48. Smith C A. A survey of various data compression techniques. International Journal of Recent Technology Engineering, 2010, 2(1): 1−20
49. Jayasankar U, Thirumal V, Ponnurangam D. A survey on data compression techniques: from the perspective of data quality, coding schemes, data type and applications. Journal of King Saud University-Computer and Information Sciences, 2021, 33(2): 119−140
50. Chiarot G, Silvestri C. Time series compression survey. ACM Computing Surveys, 2023, 55(10): 1−32
51. Holtz K. The evolution of lossless data compression techniques. In: Proceedings of WESCON ’93. 1993, 140−145
52. Kimura N, Latifi S. A survey on data compression in wireless sensor networks. In: Proceedings of International Conference on Information Technology: Coding and Computing. 2005, 8−13
53. Chew L W, Ang L M, Seng K P. Survey of image compression algorithms in wireless sensor networks. In: Proceedings of 2008 International Symposium on Information Technology. 2008, 1−9
54. Me S S, Vijayakuymar V R, Anuja R. A survey on various compression methods for medical images. International Journal of Intelligent Systems and Applications (IJISA), 2012, 4(3): 13−19
55. Hosseini M. Data compression algorithms and their applications. See scribd.com/document/77511910/Data-Compression-Algorithms-and-Their-Applications website, 2012
56. Srisooksai T, Keamarungsi K, Lamsrichan P, Araki K. Practical data compression in wireless sensor networks: a survey. Journal of Network and Computer Applications, 2012, 35(1): 37−59
57. Sharma N, Kaur J, Kaur N. A review on various Lossless text data compression techniques. Research Cell: An International Journal of Engineering Sciences, 2014, 2: 58−63
58. Zhu Z, Zhang Y, Ji Z, He S, Yang X. High-throughput DNA sequence data compression. Briefings in Bioinformatics, 2015, 16(1): 1−15
59. Hernaez M, Pavlichin D, Weissman T, Ochoa I. Genomic data compression. Annual Review of Biomedical Data Science, 2019, 2: 19−37
60. Kryukov K, Ueda M T, Nakagawa S, Imanishi T. Sequence compression benchmark (SCB) database—A comprehensive evaluation of reference-free compressors for FASTA-formatted sequences. GigaScience, 2020, 9(7): giaa072
61. Gilmary R, Venkatesan A, Vaiyapuri G. Compression techniques for DNA sequences: a thematic review. Journal of Computing Science and Engineering, 2021, 15(2): 59−71
62. Sun H, Ma H, Zheng Y, Xie H, Yan M, Zhong C. LRCB: a comprehensive benchmark evaluation of reference-free lossless compression tools for genomics sequencing long reads data. In: Proceedings of 2024 Data Compression Conference. 2024, 584
63. Singh B, Kaur A, Singh J. A review of ECG data compression techniques. International Journal of Computer Applications, 2015, 116(11): 39−44
64. Rajankar S O, Talbar S N. An electrocardiogram signal compression techniques: a comprehensive review. Analog Integrated Circuits and Signal Processing, 2019, 98(1): 59−74
65. Kumar P, Parmar A. Versatile approaches for medical image compression: a review. Procedia Computer Science, 2020, 167: 1380−1389
66. Patidar G, Kumar S, Kumar D. A review on medical image data compression techniques. In: Proceedings of the 2nd International

Conference on Data, Engineering and Applications. 2020, 1−6
67. Seeli D J J, Thanammal K K. A comparative review and analysis of medical image encryption and compression techniques. Multimedia Tools and Applications, 2024
68. Wen L, Zhou K, Yang S, Li L. Compression of smart meter big data: a survey. Renewable and Sustainable Energy Reviews, 2018, 91: 59−69
69. Prokop K, Bień A, Barczentewicz S. Compression techniques for real-time control and non-time-critical big data in smart grids: a review. Energies, 2023, 16(24): 8077
70. Tcheou M P, Lovisolo L, Ribeiro M V, da Silva E A B, Rodrigues M A M, Romano J M T, Diniz P S R. The compression of electric signal waveforms for smart grids: state of the art and future trends. IEEE Transactions on Smart Grid, 2014, 5(1): 291−302
71. Sheltami T, Musaddiq M, Shakshuki E. Data compression techniques in wireless sensor networks. Future Generation Computer Systems, 2016, 64: 151−162
72. Sandhya Rani I, Venkateswarlu B. A systematic review of different data compression technique of cloud big sensing data. In: Proceedings of the 2nd International Conference on Computer Networks and Communication Technologies. 2020, 222−228
73. Ketshabetswe K L, Zungeru A M, Mtengi B, Lebekwe C K, Prabaharan S R S. Data compression algorithms for wireless sensor networks: a review and comparison. IEEE Access, 2021, 9: 136872−136891
74. Correa J D A, Pinto A S R, Montez C. Lossy data compression for IoT sensors: a review. Internet of Things, 2022, 19: 100516
75. De Romarategui D G F. Compressing network data with deep learning. Universitat Politècnica de Catalunya, Dissertation, 2024
76. Kaur R, Chana I, Bhattacharya J. Data deduplication techniques for efficient cloud storage management: a systematic review. The Journal of Supercomputing, 2018, 74(5): 2035−2085
77. Cappello F, Di S, Li S, Liang X, Gok A M, Tao D, Yoon C H, Wu X C, Alexeev Y, Chong F T. Use cases of lossy compression for floating-point data in scientific data sets. The International Journal of High Performance Computing Applications, 2019, 33(6): 1201−1220
78. Schmidhuber J, Heil S. Sequential neural text compression. IEEE Transactions on Neural Networks, 1996, 7(1): 142−146
79. Goyal M, Tatwawadi K, Chandak S, Ochoa I. DZip: improved general-purpose lossless compression based on novel neural network modeling. In: Proceedings of 2020 Data Compression Conference. 2020, 372−372
80. Mahoney M V. Fast text compression with neural networks. In: Proceedings of the 13th International Florida Artificial Intelligence Research Society Conference. 2000, 230−234
81. Delétang G, Ruoss A, Duquenne P A, Catt E, Genewein T, Mattern C, Grau-Moya J, Wenliang L K, Aitchison M, Orseau L, Hutter M, Veness J. Language modeling is compression. In: Proceedings of the 12th International Conference on Learning Representations. 2024
82. Goyal M, Tatwawadi K, Chandak S, Ochoa I. DeepZip: lossless data compression using recurrent neural networks. In: Proceedings of 2019 Data Compression Conference. 2019, 575
83. Liu Q, Xu Y, Li Z. DecMac: a deep context model for high efficiency arithmetic coding. In: Proceedings of 2019 International Conference on Artificial Intelligence in Information and Communication. 2019, 438−443
84. Valmeekam C S K, Narayanan K, Kalathil D, Chamberland J F, Shakkottai S. LLMZip: lossless text compression using large language models. 2023, arXiv preprint arXiv: 2306.04050
85. Byronknoll. Cmix. See github.com/byronknoll/cmix website, 2024
86. Bell T, Witten I H, Cleary J G. Modeling for text compression. ACM Computing Surveys (CSUR), 1989, 21(4): 557−591
87. Burrows M, Wheeler D J. A block-sorting lossless data compression algorithm. Palo Alto: Systems Research Center, 1994: 124
88. Mahoney M V. Adaptive weighing of context models for lossless data compression. See mattmahoney.net/dc/cs200516.pdf, 2005
89. Gailly J L, Adler M, GZip offical website. See gnu.org/software/gzip/manual/ website, 2023
90. Roshal E. RAR offical website. See rarlab.com/ website, 2024
91. LZMA2 Official Website. LZMA2. See 7-zip website, 2024
92. Boutell T. RFC2083: Png (portable network graphics) specification version 1.0. RFC Editor. See dl.acm.org/doi/pdf/10.17487/RFC2083, 1997
93. Coalson J. Free lossless audio codec. See xiph.org/flac website, 2023
94. Hoffmann J, Borgeaud S, Mensch A, Buchatskaya E, Cai T, Rutherford E, de Las Casas D, Hendricks L A, Welbl J, Clark A, Hennigan T, Noland E, Millican K, van den Driessche G, Damoc B, Guy A, Osindero S, Simonyan K, Elsen E, Rae J W, Vinyals O, Sifre L. Training compute-optimal large language models. 2022, arXiv preprint arXiv: 2203.15556
95. Zaheer M, Guruganesh G, Dubey A, Ainslie J, Alberti C, Ontanon S, Pham P, Ravula A, Wang Q, Yang L, Ahmed A. Big bird: Transformers for longer sequences. In: Proceedings of the 34th Conference on Neural Information Processing Systems. 2020, 17283−17297
96. Mahoney M. Large text compression benchmark. See mattmahoney.net/dc/text website, 2006
97. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg A C, Fei-Fei L. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 2015, 115(3): 211−252
98. Panayotov V, Chen G, Povey D, Khudanpur S. Librispeech: an ASR corpus based on public domain audio books. In: Proceedings of 2015 IEEE International Conference on Acoustics, Speech and Signal processing. 2015, 5206−5210
99. Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LlaMA: open and efficient foundation language models. 2023, arXiv preprint arXiv: 2302.13971
100. Gailly J L, Adler M. zlib. See zlib.net/ website, 2024
101. Rhatushnyak A. PAQ8H. See mattmahoney.net/dc/paq website, 2006
102. byronknoll. Lstm-compress. See github.com/byronknoll/lstm-compress website, 2017
103. Bellard F. NNCP. See bellard.org/nncp/nncp website, 2019
104. Likhosherstov V, Choromanski K, Davis J, Song X, Weller A. Sub-linear memory: How to make performers slim. In: Proceedings of the 35th Conference on Neural Information Processing Systems. 2021, 6707−6719
105. Knoll B. TensorFlow-compress. See github.com/byronknoll/tensorflow-compress website, 2020
106. Ma Y, Yu D, Wu T, Wang H. PaddlePaddle: an open-source deep learning platform from industrial practice. Frontiers of Data & Computing, 2019, 1(1): 105−115
107. Pang B, Nijkamp E, Wu Y N. Deep learning with TensorFlow: a review. Journal of Educational and Behavioral Statistics, 2020, 45(2): 227−248
108. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of the 33rd Conference on

Neural Information Processing Systems. 2019, 32
109. Wang D, Cui W. An efficient graph data compression model based on the germ quotient set structure. Frontiers of Computer Science, 2022, 16(6): 166617
110. Xing Y, Li G, Wang Z, Feng B, Song Z, Wu C. GTZ: a fast compression and cloud transmission tool optimized for FASTQ files. BMC Bioinformatics, 2017, 18(16): 549
111. Deorowicz S. Silesia corpus. See github.com/MiloszKrajewski/Silesia Corpus website, 2018
112. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11): 2278−2324
113. Krizhevsky A. Learning multiple layers of features from tiny images. See cs.toronto.edu/~kriz/learning-features-2009-TR.pdf website, 2009
114. Image Compression Benchmark official website. See imagecompression.info/ website, 2015
115. Piczak K J. ESC: dataset for environmental sound classification. In: Proceedings of the 23rd ACM international conference on Multimedia. 2015, 1015−1018
116. Warden P. Speech commands: a dataset for limited-vocabulary speech recognition. 2018, arXiv preprint arXiv: 1804.03209
117. Ito K, Johnson L. The LJ speech dataset. See keithito.com/LJ-Speech-Dataset website, 2017
118. Pratas D, Pinho A J. A DNA sequence corpus for compression benchmark. In: Proceedings of the 12th International Conference on Practical Applications of Computational Biology & Bioinformatics. 2019, 208−215
119. Geer L Y, Marchler-Bauer A, Geer R C, Han L, He J, He S, Liu C, Shi W, Bryant S H. The NCBI biosystems database. Nucleic Acids Research, 2010, 38(suppl_1): D492−D496
120. PBzip2. PBzip2. See launchpad.net/pbzip2 website, 2009
121. Takehiro K. SnZip Official Website. See github.com/kubo/snzip website, 2021
122. PPMD Official Website. PPMD. See 7-zip website, 2010
123. Barina D, Klima O. X3: lossless data compressor. In: Proceedings of 2022 Data Compression Conference. 2022, 441
124. Barina D. Experimental lossless data compressor. Microprocessors and Microsystems, 2023, 98: 104803
125. LZ4. LZ4 official website. See github.com/lz4/lz4 website, 2024
126. Qin L, Sun J. Model compression for data compression: neural network based lossless compressor made practical. In: Proceedings of 2023 Data Compression Conference. 2023, 52−61
127. Tibshirani R. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 1996, 58(1): 267−288

Hui SUN obtained his BSc and MSc degrees in information security and computer science from China University of Mining and Technology and Guangxi University, China in 2019 and 2022, respectively. He is currently pursuing the PhD degree at the College of Computer Science, Nankai University, China, and is also a visiting student at the College of Computing and Data Science (CCDS), Nanyang Technological University (NTU), Singapore. His research interests include AI for data compression, deep learning, and parallel computing. He has authored technical conferences and journal papers in DCC, ICPADS, ISBRA, Journal on Communications, Journal of Chinese Computer Systems, Bioinformatics, BMC Bioinformatics, etc.

Huidong MA obtained his BSc and MSc degrees in computer science from Hainan University, China and Guangxi University, China in 2020 and 2023, respectively. He is currently pursuing a PhD degree in computer science at Nankai University, China. His main research interests include data storage systems, machine learning, AI for data compression, and large language models. He has authored technical papers in conferences and journals, such as DCC, ICPADS, ISBRA, Bioinformatics, and BMC Bioinformatics.

Feng LING obtained a BSc degree in information security from Northeast University, China in 2022. He is currently pursuing a MSc degree at Nankai University, China. He is a member of the Nankai-Baidu Joint Laboratory and the Parallel and Distributed Software Technology Laboratory. His research interests include AI for data compression, high-performance computing, parallel algorithm design, and neural network inference frameworks.

Haonan XIE is currently pursuing a PhD degree in automation at the School of Electrical Engineering, Guangxi University, China. He is a CAAI member, IEEE member, and IET member. His main research interests include artificial intelligence-engaged energy conversion, systems engineering modeling, and compression & management of big data in the power industry. He has authored technical papers in RSER, AE, DCC, ICPADS, Bioinformatics, and BMC Bioinformatics journals and conferences.

Yongxia SUN received her BSc and MSc degrees in information management & system and computer technology from Tianjin Agricultural University, China and Hunan University of Technology and Business, China. She is currently pursing the PhD degree at college of computer science, Nankai University, China. She is a member of the Nankai-Baidu Joint Laboratory and the Parallel and Distributed Software Technology Laboratory. Her research interests include data compression and storage, data auditing, machine learning, and blockchain application. She has authored technical papers in Computer Networks, CMC, etc.

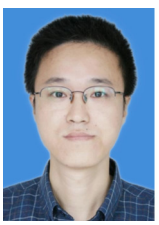
Liping YI is currently pursuing the PhD degree at the College of Computer Science, Nankai University, China and is also a visiting student at the College of Computing and Data Science (CCDS), Nanyang Technological University (NTU), Singapore. Her research interests include federated learning, she has authored technical papers in NeurIPS, ICML, MM, IJCAI, ICASSP, ICWS, DASFAA, etc. conferences and TSC, TMC, KBS journals. She served as the reviewer of NeurIPS, ICML, ICLR, KDD, AAAI, IJCAI, CVPR, MM, ICASSP, ICME, FL-IJCAI'23 workshop, FL@FM-NeurIPS'23

workshop, FL@FM-TheWebConf'24 workshop, FL@FM-ICME'24 Workshop conferences, and TMC, TNNLS, TGCN, Neurocomputing journals.

Meng YAN obtained her BSc and PhD degrees in physics and computer science from Tianjin University and Nankai University, China in 2014 and 2022, respectively. She is an assistant professor at Nankai University, conducting postdoctoral research at the Nankai-Baidu Joint Laboratory. Her main research areas include blockchain, machine learning, and AI for data compression. She has published technical papers on SRDS, Chinese Journal of Electronics, Bioinformatics, etc. She is currently a member of editor board for Journal Blockchain.

Cheng ZHONG received the PhD degree in computer science and technology from the University of Science and Technology of China, China. He is currently a professor with the School of Computer, Electronics and Information at Guangxi University, China. He has been hosted several national and provincial research projects. He has published more than 150 journal/conference papers and edited 5 books. His research interests include parallel computing, bioinformatics, distributed computing, and information security. He is an outstanding member of Chinese Computer Federation.

Xiaoguang LIU received his BSc, MSc, and PhD degrees in computer science from Nankai University, China in 1996, 1999, and 2002, respectively. He is currently a professor at the Department of Computer Science, Nankai University, China. His research interests include search engines, storage systems, GPU computing, and federated learning. He has authored technical papers in DCC, ICML, AAAI, IJCAI, WWW, SIGIR, VLDB conferences and TC, TPDS, TOS, TKDE, TDSC, TMM, TNNLS, TCSVT journals, etc.



Gang WANG received his BSc, MSc, and PhD degrees in computer science from Nankai University, China in 1996, 1999 and 2002, respectively. He is currently a professor at the Department of Computer Science, Nankai University, China. His research interests include parallel computing, storage systems, data mining, machine learning, and federated learning. He has authored technical papers in ICML, AAAI, IJCAI, WWW, SIGIR, DCC, VLDB, ACM MM conferences and TC, TPDS, TOS, TKDE, TDSC, TNNLS, TCSVT journals, etc.