Accelerating General-Purpose Lossless Compression via Simple and Scalable Parameterization

Yu Mao
Department of Computer Science,
City University of Hong Kong

Yufei Cui*
School of Computer Science,
McGill University

Tei-Wei Kuo
Department of Computer Science
and Information Engineering,
National Taiwan University

Chun Jason Xue
Department of Computer Science,
City University of Hong Kong

ABSTRACT

The storage of multi-media data can benefit from the advancements in general-purpose lossless compression. The explosive growth of multi-media data volume in data centers demands a higher compression ratio and better compressors' run-time speed. However, recent deep-learning-based compressors with a high compression ratio usually build complicated dependencies on history symbols, leading to a long compression time. This paper investigates the behavior of historical symbols and finds an approximate order of importance. Namely, recent symbols have a substantially larger influence on the probability estimation of the next unknown symbol. This observation guides the designing of an interpretable structure for data compression, rather than learning implicitly from data like Recurrent Neural Network (RNN) and attention. Based on this observation, we disentangle the compression model into order learning and feature learning, which were fused in a large module in previous works. A parameterized ordered mask unit is established to learn the ordered importance of history symbols. A fast Multi-Layer Perceptron (MLP) network is designed for efficient feature learning. The proposed compressor can improve both compression performance and computational efficiency compared with transformer-based or RNN-based compressors. To further enhance computational efficiency, we propose a branch-MLP block to replace the original MLP layer. This block reduces the parameters and the FLOPs of the original MLP to a half, without sacrificing compression performance. Experiments on multi-media data demonstrate that our model improves the compression ratio by 10% on average across data domains while accelerating compression speed by 100% compared with the state-of-the-art. The source code and appendix are released at https://github.com/mynotwo/compressor_via_simple_and_scalable_parameterization.git.

*Corresponding Author.

MM '22, October 10–14, 2022, Lisboa, Portugal

ACM ISBN 978-1-4503-9203-7/22/10…$15.00
https://doi.org/10.1145/3503161.3548413

CCS CONCEPTS

• **Information systems** → *Cloud based storage*; **Data compression**; *Multimedia streaming*; *Data stream mining*.

KEYWORDS

multi-layer perceptron, general-purpose compressor, lossless data compression, neural networks, ordered importance, computational efficient

ACM Reference Format:
Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022. Accelerating General-Purpose Lossless Compression via Simple and Scalable Parameterization. In *Proceedings of the 30th ACM International Conference on Multimedia (MM '22), October 10–14, 2022, Lisboa, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3503161.3548413

1 INTRODUCTION

General-purpose compression aims at using a single compression algorithm to deal with multi-media data streams. The exponential growth of multi-media data volume in recent years has cost data centers and cloud service providers a fortune [3, 4, 16]. The German Climate Computing Center (DKRZ) has a 54 PiB storage system that costs 0.9 GWh of energy per year, resulting in a seven-figure energy bill [4]. Amazon Web Services (AWS) [29] contains more than 1.4 million hosts. These hosts transmit and store massive heterogeneous data streams every day, including text, image, video, disk backup, and MRI, to name a few. Developing a system containing a number of specialized compression approaches is a potential solution to alleviate such burdens [30]. Nevertheless, in addition to the effectiveness concern of specific data compressors, scalability is an eminent issue preventing specialized compression system from applying to the cloud. The other potential solution is general-purpose lossless data compression techniques. However, standard compressors like Gzip [12] and Zstandard [7] have limited performance when facing multi-modal data streams due to their dictionary-based nature.

To improve the compression ratio, many deep-learning-based compressors have been proposed recently. Those compressors model the compression task as a sequential modeling problem, using history symbols as input and estimating the probability of the incoming symbol (denoted as "sequence" and "target symbol" in the rest of the paper). DecMac [20], tensorflow-compress [18] and Dzip [14] use Recurrent Neural Network to capture long-term dependencies in history, while NNCP [1] and TRACE [23] introduce transformer

to achieve accurate probability estimation. Currently, those deep-learning-based compressors are too slow for practice. NNCP has a dictionary of size 16384 and a 55M transformer model, leading to a 39 kB/min compression speed. Bye-stream compressors are proposed to overcome this issue since they treat each byte as a symbol and obtain a smaller dictionary. Dzip can achieve 399.4 kB/min, while TRACE improves the compression speed to 952.3 kB/min.

This paper improves the compression speed to 1.8 MB/min, with outstripping compression ratios across data domains. The achievement is based on a crucial finding: the input sequence in compression has an ordered importance. We first analyze this problem by observing a transformer-based compressor's attention map. The attention map clearly shows an uphill trend in the dimension of the input sequence. A principle for simplifying attention for compression tasks is proposed. We then investigate the input's order on the state-of-the-art compression models by giving them a trainable mask on the input sequence. And all of the masks eventually converged into the same pattern with the attention map: the more recent symbol has a substantially higher influence on the probability estimation of the target symbol. Validation experiments on explainable logistic regression with L0-norm also demonstrate this conclusion. However, this ordered importance was implicitly captured in previous deep-learning-based compressors with massive redundancy.

To cater to the input's implicit order information, we propose an operation named Batch-wise Ordered Mask(BOM), which captures the ordered importance of the input sequence. A novel training objective for general-purpose compression tasks is provided. We design a fast MLP architecture for feature extraction to capture the dependencies between input symbols. Figure. 1 illustrates different sequential modeling strategies of NNCP, Dzip, TRACE and proposed MLP-based compressors. RNN-based compressor Dzip needs to process the input symbols one by one. NNCP requires each input symbol to calculate the attention score with the rest. TRACE groups locations to reduce computation, achieving 3x acceleration than Dzip. MLP is a simple structure known to be computationally efficient but inadequate in establishing sequential correlations. The proposed BOM makes up for this weakness. BOM with mlp can replace attention and achieve higher compression performance than previous RNN-based or transformer-based compressors.

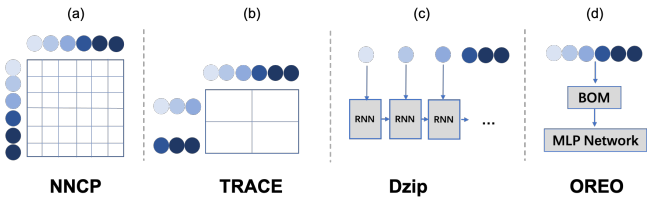

Figure 1: The illustration of NNCP, TRACE, Dzip and our's sequence modeling approaches. Each box or sub-box represents a parameterized computation unit.

We further propose a branch-MLP block to reduce MLP's computational complexity. This branch-MLP block consists of a branch-block and a mix-block. Branch-block contains several branches which perform local feature extraction. The following mix-block fuses the local feature extracted by the branch-block and extracts a global feature. The architecture gradually fuses local features on the block level by stacking branch-MLP blocks with progressive branch numbers. The overall architecture, ORdered mlp comprEssOr with branch (OREO), has fewer number of FLOPs while retaining advanced compression performances across data domains.

In summary, the contributions of this paper are:

- This paper discovers an approximate one-dimensional ordered importance phenomenon that widely exists in deep-learning-based compressors.
- This paper disentangles the compression model into order learning and feature learning, which were fused in complex modules like self-attention or RNN layers in previous works. Ordering learning is accomplished by the exponentially ordered L0-regularized learning model, which generates a batch-wise flexible ordered importance mask on the input sequence. An MLP probability estimator is established for efficient feature learning.
- This paper proposes a compressor OREO based on BOM and a lightweight branch-MLP block. This block is more computationally efficient than the original MLP structure and can maintain a comparable high compression ratio across data domains.

2 A REVIEW OF BYTE-STREAM DATA COMPRESSION

Deep-learning-based lossless data compression consists of two parts: probability estimation and coding. Any data can be represented as a binary data stream. Different compressors choose different units. LPAQ chooses bit as a unit. Thus, its dictionary size is two. Byte-stream data compressors like Dzip and TRACE choose byte (8 bits) as a unit, and the dictionary size is 256. NNCP has pre-processing progress that projects the most frequent long byte combinations into a shorter representation with dictionary size 16384. In this paper we choose byte as basic compression units, which is denoted as "symbol" in the rest of the paper.

After dictionary building, the compressor usually segments the data stream into B equal-length pieces. When compressing, each piece takes a position in batch, so the batch size is B, too. In other words, a position in the input batch is always obtained sequentially from the same piece. This operation is designed to guarantee the speed of decoding. We emphasize that the following process is described as batch size=1 for clarity. But batch operation exists in practice and following algorithm design.

Probability Estimation. After segmentation, the compressor encodes the first S symbols with uniform distribution. These symbols are aggregated as input batch to estimate the $S + 1_{th}$ symbol's probability, which is referred to as the "target symbol".

The estimated probability is then fed into an entropy coder with $S + 1_{th}$ symbol to encode. Decoding follows the same procedure instead of the entropy coder doing decoding rather than encoding. After encode $S + 1_{th}$ symbol, the compressor then use 2_{nd} symbol to $S + 1_{th}$ symbol to estimate $S + 2_{th}$ symbol's probability.

Most deep-learning-based data compression works focus on this part, using history bytes as input and building models to extract

correlations between history bytes and the target byte to make accurate probability estimation.

Coding. The entropy theory introduced by Shannon gives out the lower bound of entropy coding, and many coders can get close to this boundary like Huffman Coding, Arithmetic Coding, etc. We use arithmetic coding in this work following the settings in [14, 18, 20].

Whole compression procedure. This paper focuses on dynamic compression settings, which means the probability estimator starts from a random state and dynamically adapts its parameters during the compression/decompression procedure. The whole compression/decompression procedure is expressed in Algorithm 1 and Algorithm 2. To make it clear, we assume batch size is 1.

3 ANALYSIS ON ORDERED IMPORTANCE

In this section, we analyze the sequential importance and redundancy in the previous transformer-based compressor, then conclude the design principle of the general-purpose compressor. We visualize the transformer-based compressor's attention map in Section 3 and investigate the basic importance pattern of compression's input sequence. We concluded that there's an implicit importance order in input sequences, which is also demonstrated by experiments on other state-of-the-art deep-learning-based compressors, including NNCP, Dzip, and TRACE. Furthermore, a fully white-box logistic regression model is presented for validation of ordered importance, providing insight for our compressor design in Sec. 4.

3.1 1-D ordered importance in 2-D attention

We first analyze the attention map of a small transformer compressor trained on Enwik8 [22], a 95MB compression benchmark extracted from Wikipedia. Following settings in TRACE, the hidden dim and FFN dim are 256 and 4096, with 8 heads. After training, we randomly selected 512 samples in the test set and calculated their attention maps. The attention map is calculated by:

$$AttentionMap(x) = softmax(xW^Q * xW^{K^T}/\sqrt{h_k}) \quad (1)$$

$$Attention(x) = AttentionMap(x) * xW^V \quad (2)$$

where W_Q, W_K and W_V are parameter matrices to be learned, and project input x to query, key and value matrix respectively.

The average of 512 sample attention maps is shown in Figure. 2. Although each head behaves slightly differently, they all focus more on the positions with larger indices, which are closer to the target symbol. The average attention map of 8 heads shows a much clearer trend. That is, the model tends to give more attention to symbols that are adjacent to the target symbol.

To elaborate on this interesting problem, we write the calculation related to the attention map within one head i in Fig. 2. The head index is omitted for simplicity.

$$\mathbf{E} = \mathbf{A}\mathcal{V}$$

$$\mathbf{e}_r = \mathbf{a}_r\mathcal{V} = \sum_s a_{r,s}\mathbf{v}_s \quad (3)$$

where $\mathbf{E} \in \mathbb{R}^{S\times D}$ is the output weighted by attention, $\mathbf{A} \in \mathbb{R}^{S\times S}$ is the attention matrix in Fig. 2 and $\mathcal{V} = xW^V \in \mathbb{R}^{S\times D}$. S and D denote each symbol's sequence length and embedding dimension, respectively. $r \in [1, S]$ is the index of rows in $\mathbf{E}$ and $\mathbf{A}$, while

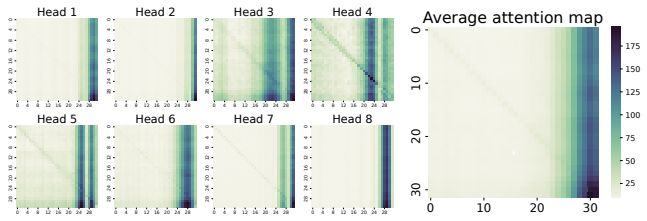

Figure 2: Transformer-based compressor's attention map with sequence length 32.

$s \in [1, S]$ is columns in $\mathbf{A}$ and rows in $\mathcal{V}$. $\mathbf{e}_r$ is the r-th row of the output matrix. $\mathbf{a}_r$ is the categorical probabilities over the r-th row. Note that $\mathbf{A}$ and $\mathcal{V}$ are conditioned on the input x.

The pattern shown in Fig. 2 denotes that for different r, $\mathbf{a}_r$'s are similar: elements in $\mathbf{a}_r$ with larger indices (close to the target symbol) tend to have greater attention value. This concludes that *there exists 1-dimensional sequential importance instead of 2-D.*

Remark 1. *A simplification is replacing the attention matrix* $\mathbf{A} \in \mathbb{R}^{S\times S}$ *with a vector* $\mathbf{a} \in \mathbb{R}^S$ *for compression tasks, thus*

$$\mathbf{E} = \mathbf{a}^T \circ \mathcal{V}$$

$$\mathbf{e}_r = a_r\mathcal{V}_r \quad (4)$$

a_r is a r-th scalar element in $\mathbf{a}$.

In this way, the computation and space complexity is reduced from $O(S^2)$ to $O(S)$. As the 1-D importance pattern is much simpler, there is a chance to aggressively use weights $\mathbf{a}$ that is not conditioned on x. Thus, a shared ordered importance is learned across the input symbols. In the rest of the paper, the model design follows this philosophy.

3.2 1-D Order in previous compressors

The observation in Sec. 3.1 reveals the ordered importance in the attention map. Due to the attention map's attributes, this order information would be applied on xW_i^V, which is the "value" linear projection of the input sequence. To further generalize this conclusion, a natural assumption is *the ordered importance directly exists on input sequence.* We validate this assumption with three recent deep-learning compressors: NNCP, Dzip, and TRACE, by placing a learnable mask over the input.

The input of compressors is $[B, S, D]$, where b denotes the batch size. To investigate the importance of sequence dimension, we give a sequence mask on S-dim. This sequence mask's weight is initialized as one and can be trained during the compression procedure via backpropagation.

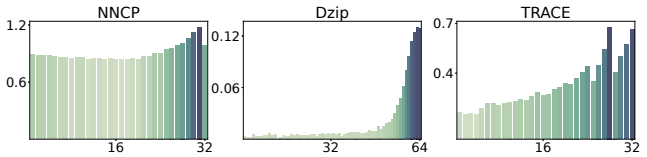

Figure 3: Sequence Mask of (a) NNCP, (b) Dzip and (c) TRACE on Enwik8.

As shown in Figure. 3, NNCP, Dzip, and TRACE shows surprisingly unified selection pattern on the sequence mask, even though they use entirely different sequence modeling structure. The nearest position to the target symbol tends to be assigned with more considerable importance weight. The importance rate decreased with the growth of positional distance.

The conclusion in Section. 3 further explains TRACE's design philosophy. TRACE argues that one attention with multiple feed-forward layers can achieve satisfactory compression performance. In other words, more attention layer is redundant in the compression area. They demonstrated this conclusion through empirical studies. The discovery of ordered importance further reveals the underneath logic: the attention map captures the ordered importance information on the input. Repeating this order information apparently cannot improve the performance in the rest of the model.

Although placing a learnable mask over input could reveal the ordered importance in data for previous compressors, these models are with massive redundancy as analyzed in Sec. 3.1. The OREO model we designed design follows the spirit in Remark. 1.

3.3 Explainable 1-D ordered sparsity

We design an L0-regularized logistic regression-based compressor as a toy example following Remark. 1, for a white box interpretability of the ordered importance in data, where the probability estimator is a simple multi-class logistic regression function. The reason that we use L0 regularization is to prune the unnecessary weights which has clear correspondence with the input sequence:

$$p(x_{S+1}) = \text{softmax}([x_1, \ldots, x_S]^T \mathbf{W}) \tag{5}$$

where $[x_1, \ldots, x_S]$ are the historical symbols with squeezed feature dimension $D = 1$ and $\mathbf{W} \in \mathbb{R}^{S \times 256}$ is the weight matrix. *In this way, the s-th row in* $\mathbf{W}$ *conducts a regression that models the way s-th historical symbol* x_s *contributes to the likelihood of* x_{S+1}*. The number of rows or the count of non-zero elements of* $||W_s||_0$ *(density) indicates the importance of s-th historical symbol for predicting* x_{S+1}.

This compressor is also trained on Enwik8. As shown in Figure. 4(a) (c), with the L0-norm, a logistic regression tends to give the recent symbol a larger weight, which is consistent with the attention map's behavior. The number of weights larger than 0.001 is counted to examine the model density and shown in Figure. 4(b) (d). We can see that the density also increased on more recent positions.

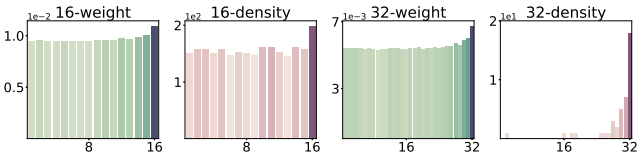

Figure 4: Logistic function's weight and density with L0-norm on different positions. (a)(b)for sequence length=16, and (c)(d) for sequence length=32.

4 ARCHITECTURES OF OREO

This section introduces OREO's architecture and explains the design philosophy. Based on our conclusion in last section, we propose a simple parameterization to model the ordered importance on the input sequence. A simple MLP compressor can achieve state-of-the-art compression performance with a much faster compression speed with establishing ordered importance. A branch-separate block is introduced to further reduce the model size and the number of FLOPs in MLP.

We re-emphasize some basic concepts introduced in Section 2.

Symbol. The basic unit a compressor works on. Some typical categories are bits, bytes, or tokens (combinations of bytes).

Target symbol. The symbol to be compressed is denoted as x_{S+i}, where S is the context (history) length.

History symbols. A sequence of symbols adjacent to the target symbol which is already compressed, denoted as $x_i, ..., x_{S+i-1}$. The probability estimator uses these symbols to estimate x_{S+i}'s probability distribution during compression. This concept is also referred as "sequence".

4.1 Batch-aware ordered importance parmeterization

As ordered importance is explicitly observed in compression tasks, we present a novel objective function for compression tasks. Assume we have a vector of Bernoulli variables $p(\mathbf{z}|\beta)$ and $p(z_s|\beta_s) = \text{Bern}(z_s|\beta_s)$. $\beta \in \mathbb{R}^S$ is the parameter vector of the Bernoulli variables, with each entry β_s being parameter for a binary gate applied to the input x_{i+s-1}.

$$h_{i+s-1} = z_s x_{i+s-1}, z_s \sim \text{Bern}(z_s|\beta_s), \tag{6}$$

where h_{i+s-1} is the output of the binary gate. The vector of binary masks slides over the sequence as i increments to $i + 1$.

The training objective of the ordered mask could be written as,

$$\min_{\theta,\beta} \mathbb{E}_{\mathbf{x},\mathbf{y}} \mathcal{L}(y, P(x_{S+i}|x_i, \ldots, x_{S+i-1}, \theta)) + \lambda R(\beta) \tag{7}$$

where $R(\beta)$ is a regularization term for the parameters of the mask with a constant factor λ. θ is the compressor's parameter for training. Specifically, we could write

$$R(\beta) = \sum_{s=1}^{S} e^{-s} \beta_s \tag{8}$$

Intuitively, e^{-s} is a monotonically decreasing function with s. Thus, a recent symbol with a larger s value is given a smaller regularization value. The corresponding β_s tends to have greater value, thus $\text{Bern}(z_s|\beta_s)$ is more likely to generate 1 instead of 0. The regularization term e^{-s} is called *order enhancer*.

The Bernoulli variable $\text{Bern}(z_s|\beta_s)$ only generates hard masks, which prunes part of the historical inputs. We propose to use a soft relaxation [13], for generating the soft masks with learnable parameters β. The generated ordered masks are visualized in Figure. 5.

To maximize the advantageousness of the ordered mask, we design a batch-wise ordered mask utilizing the unique batch generation process of the compression task. As introduced in Setion 2, deep-learning-based compressor segment input data into B equal-length pieces, and each piece takes a position in the compression batch. This special batch generation process indicates two attributes for compression batches:

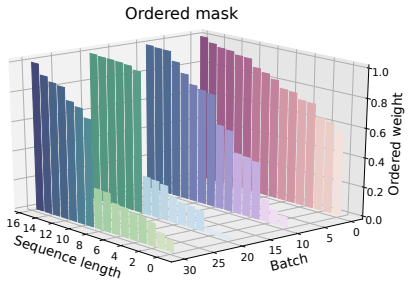

Figure 5: Samples of ordered mask, trained on Enwik9.

(1) Over different batches, the distribution change of each item would be slight.
(2) For a single batch, the distribution change between items can be relatively large, since the item's locations could be far from each other.

Based on the two attributes, we take a batch-wise treatment for Eq. 8, namely *batch-wise ordered mask (BOM)*. The final training objective could be written as,

$$\min_{\theta,\beta} \mathbb{E}_{\mathbf{x},\mathbf{y}} \mathcal{L}(y, P(x_{S+i}|x_i, \ldots, x_{S+i-1}, \theta)) + \lambda R([\beta_{\mathbf{b}}]_{\mathbf{b=1}}^{\mathbf{B}})$$

$$R([\beta_{\mathbf{b}}]_{\mathbf{b=1}}^{\mathbf{B}}) = \frac{1}{B} \sum_{b=1}^{B} \sum_{s=1}^{S} e^{-s} \beta_{b,s} \tag{9}$$

The objective in Eq. 9 is proven to be a *batch-wise exponentially ordered L0-regularized objective* in the appendix, which matches our design philosophy in Sec.3.

Implementation: For implementing the BOM, there are three ways to explore, around the binary masks:

- Independent Bernoulli variable:

$$p(\mathbf{z}|\beta) = \prod_{s=1}^{S} \text{Bern}(z_s|\beta_s) \tag{10}$$

- Bernoulli Markov chain:

$$p(z_s|\beta_s, z_{s-1}) = z_{s-1}\text{Bern}(z_s|\beta_s), \;\; z_{s-1} \sim p(z_{s-1}|\beta_{s-1}) \tag{11}$$

- Variational nested dropout [8]: a categorical approximation to Bernoulli Markov chain.

Note that the first independent Bernoulli variable is the most flexible that generates non-strict ordered masks where the z_s might be greater z_{s-1}, but provides an overall approximation to the ordered importance in data. The second and third choices could generate strictly ordered binary masks, while the third is more computationally efficient. In practice, the implementation could be adjusted according to the actual demand. The implementation details of the proposed batch-wise ordered mask are shown in the appendix.

Performance: With our batch-wise ordered importance to regularize the input, it is no longer required to use the computationally expensive attention module to learn the feature importance. The new parameterization could fully capture the order of importance per sequence through the batch-wise ordered mask. We display the experiment result for a pure MLP compressor with or without the proposed ordered mask on Enwik9 (text), Sound, and Image data. As shown in Table. 1, with batch-wise ordered importance, a simple MLP compressor's performance can be considerably improved with only 8k additional parameters. Therefore, the efficiency could be significantly improved.

Table 1: Comparison of Parameter Number, FLOPs and Compression Ratio of pure MLP compressor w/o ordered mask.

Order	Model	Param	FLOPs	Enwik9	Sound	Image
✗	Dzip (RNN)	1M	5.84×10^8	4.47	2.04	1.72
✗	TRACE (transformer)	2.4M	2.65×10^7	5.29	2.16	1.81
✗	F-block (256-4096)x8	16M	1.6861×10^7	5.20	2.14	1.80
✓	F-block (256-4096)x8	8k+16M	1.6862×10^7	5.77	2.24	1.86

We further exploit a compression-aware branch structure to slim the MLP structure in the next section.

4.2 Branch-MLP Block

In this section, we design a branch-MLP block that can does local feature extraction, local feature fusion and global feature extraction. This block requires fewer computations resources but can obtain a comparable high compression ratio with standard MLP architecture.

The structure of a branch-MLP block is shown in the Figure. 6. We first define a basic MLP feature extraction block consisting of two Fully-Connected (FC) and a layernorm layer. We denote this basic structure as Fully-connected-block (F-block).

Define input vector as H_{in}. Then F-block is:

$$F\text{-}block(H_{in}) = Layernorm(Gelu(W_1 \cdot (Gelu(W_2^T \cdot H_{in}))))$$

Based on F-block, a branch-MLP block is established. Branch-MLP block consists of two child blocks: the branch-block and the mix-block. Branch-block divides the input into N groups, each with an individual small F-block to extract local features. When features of each group are extracted, these features are concatenated together and fed into the mixed block. The mixed block is an F-block with a larger dimension, and its purpose is to communicate the information between groups. Branch-block can be defined as follows:

$$h_1, h2, \ldots, h_N = Split(H_{in})$$
$$\overline{h}_1, \overline{h}_2, \ldots, \overline{h}_N = F\text{-}block_1(h_1), F\text{-}block_2(h_2), \ldots, F\text{-}block_N(h_N)$$
$$H_{out} = Concatenate(\overline{h}_1, \overline{h}_2, \ldots, \overline{h}_N)$$

Mix-block is defined similarly as F-block.

BOM generates global sequence ordering at the input sequence. With the proposed branch-MLP block, the probability estimator can do local feature extraction with the branch-block and exchange information with the following mix-block.

The branch block can significantly reduce model parameters and the number of FLOPs. Assume batch size is B, the number of symbols is S, and each symbol's embedding dimension is D. Then the BOM's parameter number is BSs, and the number of FLOPs is $2BS$. Each block's input and output dimension is SD, and the middle dimension of the F-block is defined as M. The input and output dimension of N-branch-block is SD/N since we divided input into N groups and need to concatenate them for output. The middle

layer dimension can be flexible. In our implementation, we always set it as $M/16$.

The parameters and the FLOPs of so far proposed structure in this paper are listed in Table. 2, and compared with the transformer's block under the same dimensions. We can see that the N-branch-MLP block contains significantly fewer parameters and FLOPs than F-block. We use branch-block+mix-block to replace every two F-blocks, where mix-block contains the same amount of parameters as an F-block and branch-block contains 1/16 of mix-block. Thus, the proposed structure contains 17/32, nearly half the original parameter amounts. The comparison of the proposed block and transformer block also demonstrates the proposed block's computational efficiency.

Table 2: Parameters and FLOPs comparison of different type of blocks.

Block type	Parameters	FLOPs
BOM	BS	$B \times 4BS$
mix-block	$2SDM$	$B \times 4SDM$
branch-block	$SDM/8$	$B \times SDM/4$
branch-block+mix-block	$SDM/8 + 2SDM$	$2B \times (SDM/8 + 2SDM)$
Attention + FFN	$3SD + 2SDM$	$2B \times (3SD + S^2D + 2SDM)$

4.3 Ordered Branch Compressor

Based on the proposed branch-MLP block, we further build an MLP-based general-purpose lossless compressor. A simplified model structure is shown in Figure. 6. Suppose the number of history symbols is four, and the fifth symbol is the target symbol. The compressor first generates an ordered importance mask containing each history symbol's corresponding importance score and multiplies on history symbol's embeddings. The weighted embeddings are then sent to the corresponding branch-block and aggregated by mix-block. The same process would be repeated several times. After those branch-MLP blocks, a linear output layer would project the feature vector to final output logits, then pass through a softmax function to get the estimated probability of the target symbol. The estimated probability is sent to the arithmetic coder together with the target symbol to do encoding/decoding.

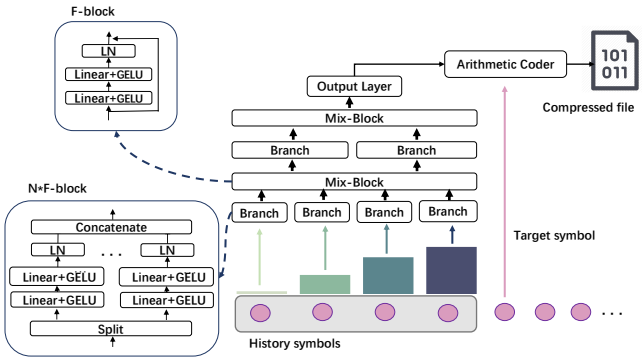

Figure 6: Compressor architectures.

The structure we finally choose for the experiment consists of four branch-MLP blocks, with each block containing 16, 8, 4, 2 branches. Sixteen branches on the first layer corresponds to sixteen history symbols. After each branch processes a symbol's information separately in the first layer, local feature fusion is performed on each layer by reducing branch numbers.

Compared to the transformer-based compressor, we only do the importance learning initially. The rest layer performs local feature fusion, local feature extraction, and global feature fusion in a predefined way. The reason behind our design is the one-dimensional orderness in compression tasks found in Section. 3. Empirically, our network is scalable to multi-layer and has significant efficiency and performance improvement.

5 RELATED WORK

General-purpose Lossless Compression. General-purpose lossless compression make a huge progress[5, 6, 15, 19, 27, 31] since Shannon proposed the entropy coding method [25]. Prediction by Partial Matching (PPM) [5] is a popular statistical-based method which based on a probability estimator and entropy coding that often takes arithmetic coding [31]. Another popular variant of PPM is PPM*C [6], which unbounded length contexts for PPM. PAQ [19] uses a context mixer same as PPM, and a mixed predictor, which combines the probability of different prediction models. Recently, the new generation of CPU-based compressors, such as Microsoft's Zipline [26], and Facebook's Zstd [7] offer compression ratios higher than standard deflate-based algorithms. CMIX [17] uses 2,122 mixed models, including some neural networks, and achieves the highest compression ratio almost in all domains. However, those compression methods suffer from the trade-off between compression ratio and compression speed.

Purely neural network-based architecture can be categorized into three classes: static-pretrained-method, dynamic-pretrained-method, dynamic-Random-method. Dzip-bootstrap [14] and DecMac [20] belongs to static-pretrained-method, since they pre-trained on file F for several epochs and freeze parameters during compression. Dynamic-Pretrained-method allows updating its parameter during compression. Thus, the probability estimator can fit the local context dynamically, and Dzip-combined [14] falls into this category. Nevertheless, pretraining could be computationally expensive. There are some approaches that start compression using a randomly initialized model and update its parameters during compression, including Tensorflow-compress [18], NNCP [1] and TRACE [23]. These methods omit pretraining but may suffer from a cold start problem.

L0 regularization. The normal L0 regularization over the weights has been studied by Louizos et al. [21]. The Bernoulli dropout with equivalent importance is adopted as the vanilla L0 norm over the network weights. FN^3 [9] proposes using *nested dropout* to train a fully nested neural network, which adjusts size during inference. VND [8] proposes to use a variational nested dropout (VND) to train the fully nested neural network. The Downhill distribution generating VND could be one optional method for implementing our binary gates. The details are presented in the appendix.

6 EXPERIMENTS

Experiments are benchmarked on several datasets from multi-media domains. A wide variety of real-world datasets with different data types are considered, including text (bookcorpus [28], enwik9 [22]),

Table 3: Effectiveness validation of proposed techniques. "16-b" denotes branch-block with 16 branch and "mix" denotes mix-block.

Components									FLOPs (batch=1)	Homogeneous Data					Heterogeneous Data	
Order	L1	L2	L3	L4	L5	L6	L7	L8		Enwik9	BookCorpus	Sound	Image	Floating Point	Silesia	Backup
✗	F-block (256-4096) x 8								1.6861×10^7	5.20	4.54	2.14	1.80	1.28	4.67	1.78
✓	F-block (256-4096) x 8								1.6862×10^7	5.77	4.99	2.24	1.86	1.28	4.92	1.87
✓	16-b	mix	16-b	mix	16-b	mix	16-b	mix	0.8990×10^7	5.65	4.91	2.24	1.86	1.27	4.8	1.85
✓	2-b	mix	2-b	mix	2-b	mix	2-b	mix	0.8990×10^7	5.68	4.94	2.25	1.86	1.27	4.86	1.87
✓	2-b	mix	4-b	mix	8-b	mix	16-b	mix	0.8990×10^7	5.64	4.92	1.27	1.86	1.27	4.84	1.86
✗	16-b	mix	8-b	mix	4-b	mix	2-b	mix	0.8989×10^7	5.43	4.74	2.17	1.82	1.26	4.64	1.81
✓	**16-b**	**mix**	**8-b**	**mix**	**4-b**	**mix**	**2-b**	**mix**	$\mathbf{0.8990}\times10^7$	**5.68**	**4.94**	**2.25**	**1.86**	**1.28**	**4.86**	**1.87**

floating point data (obs-spitzer [2]), audio (environmental sound [24]), image data (ImageNet [10]), disk backup data and Silesia [11] (contains 12 data types from windows dll to x-ray image). The detailed description of those datasets can be found in supplementary materials.

We use Compression Ratio, Peak GPU Memory Usage, Compression Speed, Model Latency, and the FLOPs (Floating point operations) as metrics. All experimental result is an average of ten repetitive experiments. All results are reported on an 11GB NVIDIA Geforce RTX 2080 GPU.

We compare OREO with both standard compressors and deep-learning-based compressors. To ensure fairness, all deep-learning method follows the dynamic compression procedure, which means no pretraining exists. The history symbol's number and dimension are 16, which means the hidden dimension of the model is 256. The branch-block and mix-block's first layer output dimensions are settled as 256 and 4096, respectively. The compression batch size for OREO is set as 512. Adam is applied to optimize the model with a learning rate=0.001.

6.1 Compression Efficiency

Table 4: Peak GPU memory usage and Inference Latency of GPU-based method with batch=128.

Compressor	Peak GPU Memory Usage (GB)	Model Latency (ms)	FLOPs
2080Ti	10.75G		
NNCP	7.75G	95.67ms	15.83×10^{10}
Dzip	6.39G	5.82ms	7.48×10^{10}
TRACE	2.02G	2.08ms	0.34×10^{10}
OREO	1.18G	1.54ms	0.12×10^{10}

We first represent the efficiency metrics of GPU-based compressors in Table. 4, including Peak GPU Memory Usage, Model Latency, and the FLOPs. MLP architecture makes the proposed compressor more efficient over all metrics. Under batch size 128, the proposed OREO only takes 1.18G GPU memory, which is half of TRACE and 1/6 of Dzip. This means it can achieve a significantly larger batch size with the same computational resources. On a standard 2080Ti with 10.75G memory capacity, our compressor can achieve maximum batch size=27576, which is five times of TRACE (maximum 5120).

OREO's model latency and the number of FLOPs are also in advance. The proposed compressor achieves nearly 2x compression speed of TRACE and 4.5x of Dzip.

The compression approach utilizing statistic dictionary collection usually face unstable compression speed problem. As shown in Figure. 7, traditional compression approaches Gzip, 7z, and zstd's compression speed all have considerable variance across data types. As for deep-learning approaches, dictionary-based NNCP also has unstable compression speeds. Byte-stream-based compressors Dzip, TRACE, and propsed OREO can obtain a fast and steady compression speed across data domains. And OREO can achieve significantly faster compression speed across data domains. Since NNCP's compression speed is far slower from other compressors, we plot Figure. 7 (b) in log_{10} scale.

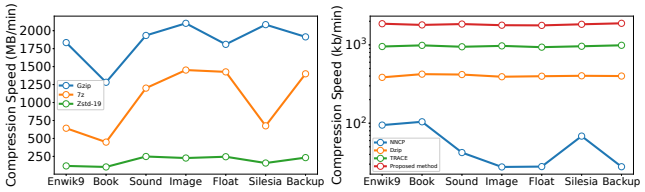

Figure 7: Compression speed of compressors across data domains.

The overall compression speed is shown in Table. 6. The deep-learning compressors are significantly slower than traditional compressors. A significant reason is that current deep-learning-based compressors use a single thread with python as program language (including model inference, backpropagation, and arithmetic coding), which means they have much to optimize on the software development level. For fairness, we also use PyTorch to build our model with a python version arithmetic coder and use a single thread when compressing. The proposed OREO can achieve a 1818.55 kB/min compression speed, which is faster than other existing deep-learning compressors. Our compressor's speed is 4.5x of Dzip and 2x of TRACE.

Table 5: Compression Ratios on Large Datasets.

Methods		Homogeneous Data					Heterogeneous Data	
		Enwik9	BookCorpus	Sound	Image	Floating Point	Silesia	Backup
Non-Deep learning	Gzip	3.09	2.77	1.37	1.14	1.06	3.10	1.28
	7z	4.35	3.80	1.59	1.38	1.14	4.25	1.56
	zstd-19	4.24	3.73	1.40	1.16	1.10	3.97	1.36
Deep Learning	Dzip-combined	4.47	3.95	2.04	1.72	1.26	4.78	1.78
	TRACE	5.29	4.58	2.16	1.81	1.28	4.63	1.78
	OREO	**5.68**	**4.94**	**2.25**	**1.86**	**1.28**	**4.86**	**1.87**

Table 6: Compression speed of Compressors.

Method	Compression Speed
Traditional	
Gzip	1851MB/min
7z	1032MB/min
Zstd-19	191MB/min
Deep Learning-based	
Cmix	39.9 kB/min
NNCP	28.6~122.9 kB/min
Dzip	399.4 kB/min
TRACE	952.3 kB/min
OREO	**1818.55** kB/min

6.2 Ablation study

In this section, we display multiple choices of OREO's architecture and their impact on computational efficiency and compression performance in Table. 3. We first validate the proposed BOM's effectiveness on a pure MLP probability estimator consisting of 8 F-blocks. The experiment result shows that BOM can raise the compression ratio by 10%, with barely any extra computation. We then investigate the impact of different block-stacking choices. The experiment is designed with four stacking strategies: 16-16-16-16, 2-2-2-2, 2-4-8-16, and 16-8-4-2. The number denotes branch numbers in the corresponding block. For instance, 16-b indicates there are 16 branches in this block. The 16-8-4-2 strategy could achieve the best performance with fewer parameters. In other words, the structure that captures each symbol's feature separately at first and gradually aggregates features with the growth of network depth can achieve better performance.

6.3 Overall performance

Table. 5 shows the compression ratio comparison between the proposed compressor and other state-of-the-art compressors. Gzip lags far behind other compressors in compression ratios across all data domains. zstd-19 and 7z are more advance, and each has its strong point. 7z can achieve 3% to 5% better compression ratio than zstd-19 and are much faster, while zstd offered offers a wide range of compression levels to choose from, which is more suitable for industry.

Nevertheless, deep-learning-based compressors can beat these non-deep-learning compressors easily. Dzip-combined improves text data by about 5%, but improves multimedia data even more. Compared to the best-performing 7z, Dzip has a 25% compression improvement on sound data, 24.6% improvement on image data, and 10.5% improvement on floating-point data. Dzip improved 12.5% and 14.1% over 7z on heterogeneous dataset silesia and Backup data, respectively.

TRACE leverages a slim transformer structure to achieve better compression ratios than Dzip with faster speeds. TRACE improved 21.6% and 20.5% over 7z on text data Enwik9 and Book. For the multimedia data, including Sound, Image and Floating point data, the compression ratio of TRACE is 31.2%, 24%, and 12.3% higher than 7z, respectively. However, TRACE's performance is less than satisfactory on heterogeneous dataset Silesia and Backup data. It maintains the same compression ratio as Dzip on Backup, but is lower than Dzip on Silesia, only exceeding 7Z by 8.9%.

OREO solved the problem of TRACE and significantly improved the compression ratio compared with Dzip in all data. In Enwik9 and Book, OREO is 30.5% and 30% better than 7z. For multimedia Sound, Image, and floating-point data, OREO is 41.5%, 34.8%, and 12.3% better than 7z. As for heterogeneous data, OREO is 14.4% and 31% better than 7z on Silesia and Backup, respectively.

7 CONCLUSION

This paper proposes a fast and efficient compression method, OREO, which is entirely based on fully connected layers. By analyzing the attention map of the transformer-based compressor, this paper finds that current compressors establish a one-dimensional ordered importance on the input symbols. This relationship was implicitly established in previous models. Based on this observation, this paper uses a simple parameterized model to build batch-wise ordered importance explicitly. This model has very few parameters and a fast sampling speed and can be used to replace complex 2-D sequence modeling processes like attention. Based on the ordered mask, a pure MLP probability estimator can exceed the compression ratio of the previous RNN-based or transformer-based compressor. With much better compression speed. In addition, this paper further uses parameterization to reduce the parameters and computation of MLP, which can make compression more efficient. Experimental results show that the proposed compressor can achieve a faster and more stable compression speed than SOTA on various data types. We expect more work in the future to take note of this ordered importance modeled implicitly in deep-learning-based compressors and propose more advanced parameterization methods to improve compression performance.

8 ACKNOWLEDGEMENT

This work was supported by a grant from the grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11219319).

REFERENCES

[1] Fabrice Bellard. 2019. NNCP: Lossless Data Compression with Neural Networks. (2019). https://bellard.org/nncp/

[2] M. Burtscher and P. Ratanaworabhan. 2009. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Comput.* 58, 1 (2009), 18–31.

[3] Worldwide Quarterly Enterprise Infrastructure Tracker: Buyer and Cloud Deployment. 2021. (2021).

[4] DKRZ – German Climate Computing Centre. 2021. (2021). https://www.research-in-germany.org/en/research-landscape/why-germany/research-infrastructure/dkrz.html

[5] J. Cleary and I. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transacitons on Communications* 32, 4 (1984), 396–402.

[6] John G Cleary and William J Teahan. 1997. Unbounded length contexts for PPM. *Comput. J.* 40, 2_and_3 (1997), 67–75.

[7] Y. Collet. 2016. Zstd github repository from facebook. https://github.com/facebook/zstd

[8] Yufei Cui, Ziquan Liu, Qiao Li, Antoni B Chan, and Chun Jason Xue. 2021. Bayesian Nested Neural Networks for Uncertainty Calibration and Adaptive Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2392–2401.

[9] Yufei Cui, Ziquan Liu, Wuguannan Yao, Qiao Li, Antoni B. Chan, Tei-wei Kuo, and Chun Jason Xue. 2020. Fully Nested Neural Network for Adaptive Compression and Quantization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2080–2087.

[10] J. Deng, W. Dong, R. Socher, et al. 2009. Imagenet: A large-scale hierarchical image database[C]//2009 IEEE conference on computer vision and pattern recognition. *Ieee* (2009), 248–255.

[11] Sebastian Deorowicz. 1985. Silesia Dataset. (1985). http://sun.aei.polsl.pl/sdeor/index.php?page=silesia

[12] Peter Deutsch. 1996. GZIP file format specification version 4.3. *RFC* 1952 (1996), 1–12. https://doi.org/10.17487/RFC1952

[13] Yarin Gal, Jiri Hron, and Alex Kendall. 2017. Concrete dropout. *Advances in neural information processing systems* 30 (2017).

[14] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2021. DZip: Improved general-purpose loss less compression based on novel neural network modeling. In *2021 Data Compression Conference (DCC)*. IEEE, 153–162.

[15] S. Idreos, R. Kaushik, V. Narasayya, and R. Ramamurthy. 2010. Estimating the compression fraction of an index using sampling. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. 441–444.

[16] David Reinsel John Rydning, John F.Gantz. 2021. 2021–2025: The World Keeps Creating More Data — Now, What Do We Do with It All? (2021). https://www.idc.com/getdoc.jsp?containerId=US46410421

[17] B. Knoll. 2014. CMIX. (2014). http://www.byronknoll.com/cmix.html

[18] B. Knoll. 2016. Tensorflow-compress. https://github.com/byronknoll/tensorflow-compress

[19] Byron Knoll and Nando de Freitas. 2012. A machine learning perspective on predictive coding with PAQ8. In *2012 Data Compression Conference*. IEEE, 377–386.

[20] Qian Liu, Yiling Xu, and Zhu Li. 2019. DecMac: A Deep Context Model for High Efficiency Arithmetic Coding. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 438–443.

[21] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning sparse neural networks through L_0 regularization. *arXiv preprint arXiv:1712.01312* (2017).

[22] Matt Mahoney. 2006. Large Text Compression Benchmark. (2006). http://mattmahoney.net/dc/text.html

[23] Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022. A Fast Transformer-based General-Purpose Lossless Compressor. *arXiv preprint arXiv:2203.16114* (2022).

[24] Karol J. Piczak. 2015. ESC: Dataset for Environmental Sound Classification. (2015). https://doi.org/10.7910/DVN/YDEPUT

[25] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

[26] Rajeev Sharma. 2016. Zipline. (2016). https://github.com/opencomputeproject/Project-Zipline

[27] James Townsend, Tom Bird, and David Barber. 2019. Practical lossless compression with latent variables using bits back coding. *arXiv preprint arXiv:1901.04866* (2019).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[29] Wikipedia. 2020. Amazon Web Serivce. (2020). https://en.wikipedia.org/wiki/Amazon_Web_Services

[30] Accelerating Lossless GPU Compression with New Flexible Interfaces in NVIDIA nvCOMP. 2021. (2021). https://developer.nvidia.com/blog/accelerating-lossless-gpu-compression-with-new-flexible-interfaces-in-nvidia-nvcomp/

[31] Ian H Witten, Radford M Neal, and John G Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM* 30, 6 (1987), 520–540.