Faster and Stronger Lossless Compression with Optimized Autoregressive Framework

Yu Mao*, Jingzong Li*, Yufei Cui†, and Jason Chun Xue*
*Department of Computer Science, City University of Hong Kong
Email: {yumao7-c,jingzong.li}@my.cityu.edu.hk, jasonxue@cityu.edu.hk
†School of Computer Science, McGill University Email:yufei.cui@mail.mcgill.ca

***Abstract*—Neural AutoRegressive (AR) framework has been applied in general-purpose lossless compression recently to improve compression performance. However, this paper found that directly applying the original AR framework causes the duplicated processing problem and the in-batch distribution variation problem, which leads to deteriorated compression performance. The key to address the duplicated processing problem is to disentangle the processing of the history symbol set at the input side. Two new types of neural blocks are first proposed. An individual-block performs separate feature extraction on each history symbol while a mix-block models the correlation between extracted features and estimates the probability. A progressive AR-based compression framework (PAC) is then proposed, which only requires one history symbol from the host at a time rather than the whole history symbol set. In addition, we introduced a trainable matrix multiplication to model the ordered importance, replacing previous hardware-unfriendly Gumble-Softmax sampling. The in-batch distribution variation problem is caused by AR-based compression's structured batch construction. Based on this observation, a batch-location-aware individual block is proposed to capture the heterogeneous in-batch distributions precisely, improving the performance without efficiency losses. Experimental results show the proposed framework can achieve an average of 130% speed improvement with an average of 3% compression ratio gain across data domains compared to the state-of-the-art.**

***Index Terms*—auto-regressive, general-purpose, lossless data compression, hardware friendly, neural networks, computational efficient**

I. Introduction

With the proliferation of IoT infrastructures and the 5G techniques, data volume that needs to be transmitted and stored has expanded considerably [1], [2]. General-purpose lossless data compression techniques are potential solutions to reduce data sizes. However, standard compressors like Gzip [3] and Zstandard [4] have limited performance on multi-modal data streams due to their dictionary-based nature. The rapid development of deep learning techniques raised a number of neural-network-based compressors. Those compressors model the compression task as an AutoRegressive (AR) sequential modeling problem. AR-based compressors take a set of adjacent compressed symbols as input to estimate the probability of the following symbol (referred to as "history symbols" and "target symbol" in the rest of the paper), and can achieve substantially improvement on compression ratios. Nonetheless, the running speed of these deep learning-based compressors is far from satisfactory.

Most of the work to date has focused on slimming the model architecture. NNCP [5] has a considerable dictionary size of 16384 and a 55M transformer model, leading to a 0.48 KB/s compression speed. Byte stream compressors have been proposed to overcome this problem, as they treat each byte as a symbol and contain smaller dictionaries. Dzip [6] can reach 6.66KB/s, while TRACE [7] increases the compression speed to 15.8KB/s. Recently, the Multi-Layer Perceptron (MLP)-based compressor OREO [8] explicitly defined the importance order at the beginning of the model to replace attention, achieving a speed of 30KB/s. Although MLP is hardware friendly, OREO's ordered importance module is based on Gumble-Softmax sampling [9], thus OREO is not yet hardware compatible [8].

A blind spot of these state-of-the-arts is the adaptability of AR structures and compression processes. Section III-A summarize two problems observed in the current design:

1) *Duplicated processing problem*: Existing AR frameworks use a set of history symbols to estimate the probability of a target symbol. This causes inputs at neighboring time steps to contain significant overlap.
2) *In-batch distribution variation problem*: The specific batch construction of AR-based compression framework leads to a phenomenon: varied symbol distributions for different positions in the batch. The current design treats different in-batch positions indiscriminately.

This paper is motivated by the aforementioned observations. We propose PAC, a Progressive Ar-based Compression framework with a suitable model that can address the duplicated processing problem and in-batch distribution variation problem, while eliminating hardware-unfriendly Gumble-Softmax sampling.

To alleviate the duplicated processing problem, this work disentangles the probability estimator into two-stage: individual block and mix block. The individual block response for feature extraction on individual symbols while the mix block response for modeling correlation between extracted features. This operation guarantees separate processing of history symbols on the input side. A progressive AR compression framework is further established on proposed individual-mix block. This framework caches the extracted features for reutilization. Thereby, to compress one target symbol, only one latest history symbol needs to be transferred to GPU and performs feature

extraction. Features of the rest of the history symbol set can be retrieved from the cache, therefore omit duplicate transmits and feature extractions. On top of current design, a simple and trainable ordered mask generator is proposed to replace Gumble-Softmax sampling. Experiments show that trained ordered masks can achieve the same results as sampled masks but the generation is 20x faster. Another consequence of this modification is that PAC's probability estimation model is now a pure hardware-compatible model, containing only basic matrix multiplication and addition operations. This opens up the opportunity for future neural network-based compressors to offload to edge devices or storage hardware such as SSDs. To address the in-batch distribution variation problem, we propose a module that can extract location-specific discriminative features within a batch. The experimental results demonstrate that the proposed framework is able to achieve an average of 130% faster compression across data domains, with an average compression ratio improvement of 3%.

II. Related work

Recently, a new generation of deep-learning-based lossless compressors offered higher compression ratios. One of the categories is Variational Auto-encoder (VAE) with bits-back coding [10], [11]. The main application of these compressors is in image. Another category considers the input as an 1D sequence and use auto-regressive (AR) methods to estimate the probability of occurrence of the current symbol directly using its context [6]–[8], [12], [13], and can deal with any data without specialized design.

AR-based compressors have been improved over the past few years. Table. I lists current AR-based compressors' compression speed on the same NVIDIA Geforce RTX 2080 GPU platform. Cmix [5] was proposed in 2015, starting applying Long Short-Term Memory in AR-based compression. NNCP [13] utilizes transformers to estimate the probability, while Dzip [6] uses Recurrent Neural Networks. TRACE [7] designed a slimmed transformer to accelerate the compression procedure. OREO [8] further builds an order model to replace attention to achieve faster compression speed. Nonetheless, even the fastest OREO can only compress ∼1.77 MB per minute, which means ∼30KB per second. Proposed method boosts the compression speed to 71.42KB/s.

TABLE I: Speed of current AR-based compressors.

Compressor	Cmix	NNCP	Dzip	TRACE	OREO	PAC
Speed(KB/s)	0.67	0.48∼2.04	6.66	15.87	30.3	71.42

III. PAC Compression Framework

This section presents the motivation and the detailed design of PAC. The rest of the section starts with analyzing the current AR-based compression architecture, pointing out the duplicated processing problem and in-batch distribution variation problem. Then a progressive compression framework is proposed with "individual-mix" architecture, which fundamentally resolves the duplicated processing problem. Furthermore, the original Gumble-Softmax sampling module is replaced with a simple trainable matrix, substantially reducing the computational complexity and providing the opportunity to run on resource-constrained edge devices. Finally, we propose a batch-location-aware structure to address the in-batch distribution variation problem, which can improve the compression ratio without computational loss.

A. Background and Motivation

We start by motivating PAC with an analysis of conventional AR-based compression process. Some basic concepts are emphasized:

a) Symbol: The basic process unit in compression. Typical categories are bits, bytes, or tokens (a combination of bytes).

b) Target symbol: The symbol to be compressed, denoted as x_i. The neural model estimates x_i's probability and feeds it into an arithmetic coder.

c) History symbols: A set of symbols adjacent to the target symbol, denoted as $x_{i-k}, ..., x_{i-1}$, where k is the length of history symbol set. These symbols are the actual input to the model when x_i is compressed. In other words, the feature of x_i is extracted from its history symbols, but x_i itself is not visible to the model. The estimated probability of x_i can be expressed as

$$P_e\left(x_i\right) = P\left(x_i \mid x_{i-1}, x_{i-2}, \ldots, x_{i-k}\right) \tag{1}$$

This concept is also referred to as "sequence" or "input sequence" in the rest of the paper.

Suppose there is an input sequence $X = \{x_0, x_1, x_2, ..., x_{B*N}\}$ on host for compression. The compression process comprises of the following:

1) *Structured batch construction*. As illustrated in Figure 2, X is chunked into B sub-sequences of length N to parallize compression procedure. The compressor collects symbols at the same location in each sub-sequence as a batch. For instance, at timestep i, there are B symbols compressed in timestep i, namely x_i, x_{i+N}, x_{i+2N}, $\ldots x_{i+(B-1)N}$. And these symbols form the batch. At timestep $i+1$, the compressor compresses x_{i+1}, x_{i+N+1}, x_{i+2N+1}, $\ldots$, $x_{i+(B-1)N+1}$, which constitutes the current batch.
2) *Probability estimation*. To compress x_i, the neural network first estimates $P_e(x_i)$ using x_{i-1}, $\ldots$,x_{i-k}. At this stage, x_{i-1}, $\ldots$,x_{i-k} are transferred from host to GPU and fed into the model. No matter which model is used, the final output of the model is $P_e(x_i)$. NNCP and TRACE use a transformer to build correlations between history symbols, while OREO provides a more efficient way using ordered masks and MLP. Our approach is also MLP-based but more efficient than OREO by individual-mix block, trainable ordered mask, and batch-location-aware design.
3) *Coding*: $P_e(x_i)$ and x_i are passed to arithmetic coder for entropy coding. Arithmetic coding is applied in this work following the settings in [6]–[8], [13].

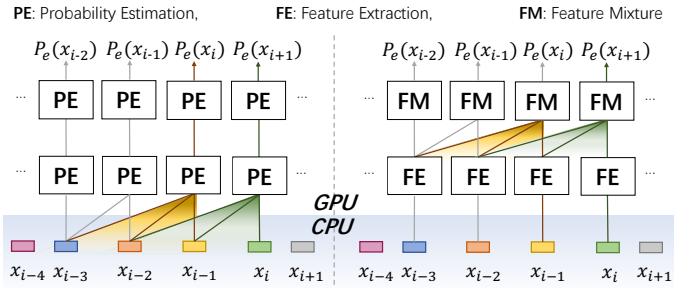


Fig. 1: Illustration of duplicated processing problem and it is solved. The left figure shows the probability estimator under the traditional auto-regressive compression framework, which requires repeated transmission and processing of a symbol, and the right figure shows the proposed framework, in which a symbol is transmitted and extracted only once.

Discussion. Standard AR-based compression frameworks have overlaps between inputs on neighboring timesteps. As shown in Figure 1, to compress x_i, $x_{i-1}, x_{i-2}, x_{i-3}$ are sent to the GPU for probability estimation. At next timestep, to compress x_{i+1}, x_i, x_{i-1}, x_{i-2} are sent to GPU for probability estimation. Here x_{i-1}, x_{i-2} are sent and processed repeatedly. Following this setting, if the size of history symbol set is set to k, a history symbol would be sent and processed for k times in total. This phenomenon is noted as "duplicated processing problem". In Dzip, TRACE and OREO, k is maximum set as 16, which means 15 symbols are duplicated between input on neighboring timesteps, resulting in 93.75% overlap ratio. For NNCP, which takes 64 as k, the overlap ratio raises to 98.44%.

The duplicated processing problem arises from the idea of treating all history symbols as a whole. For example, the attention-based compressor computes the history symbols' query, key and value in pairs at every iteration. In this way, queries, keys, and values are recalculated every time a new history symbol is added. Another example is the MLP-based compressor OREO. OREO first projects history symbols to vectors, then concatenates these vectors to one large feature vector to proceed. This structure dictates that OREO must start over for all history symbols when a history symbol changes.

Structured batch construction is AR-based compression's another remarkable characteristic. As mentioned in *Structured batch construction*, the input sequence is broken into several sub-sequences for parallel compression and decompression. The compressor compresses one symbol from each sub-sequence within one timestep. These symbols are usually located at the same position in the sub-sequence, as illustrated in Figure 2. At the next timestep, the compressor will compress the symbols at the next adjacent position in each sub-sequence. This structured batch construction leads to batch properties distinct from other machine-learning tasks:

1) Distribution between sub-sequences is varied because the semantic meanings of sub-sequences are much different. Since each sub-sequence corresponds to a specific position in the batch, we can say there is a bias in the distribution of symbols at different positions in the batch.

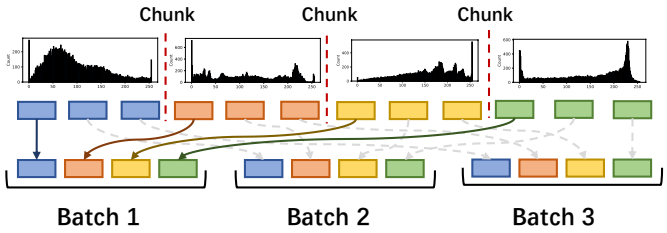


Fig. 2: Illustration of structured batch construction and in-batch distribution variation problem. The x-axis of the distribution figure is the byte value, and the y-axis is the number of byte value.

2) Distribution of symbols that appear consecutively at the same batch position are usually close because they belong to the same sub-sequence, not to mention adjacent symbols also have overlaps in their history symbol sets.

Figure 2 briefly illustrates structured batch construction of a 10^7 bytes ImageNet data, with the batch size set to 4096. We uniformly sample the data distribution of $1000th$, $2000th$, $3000th$, and $4000th$ sub-sequences, count the frequency of occurrences of bytes, and generate distributions. Summarizing from Figure 2, the distribution of symbols in the four selected sub-sequences is highly varied. We believe each batch position should have its own parameters when designing a general lossless compression model. These particular parameters can help the model distinguish different sub-sequences, resulting in better compression performance.

B. Progressive Compression Architecture

The key to addressing the duplicated processing problem is disentangling history symbol set's process. Therefore when the history symbol set changes, the probability estimator does not have to recompute and re-transmit the full history symbol set. Nonetheless, this solution is inconsistent with current AR probability estimation designs that aim to entangle the inputs and extract correlations.

We reformulate the AR probability estimation in general-purpose lossless compression as a two-stage task: feature extraction and correlation modeling. These two parts were implicitly mixed in early compressors. We decouple these two parts to strip out the feature extraction part, which can be executed regardless of the relation between symbols. Thus, feature extraction of the historical symbols can be conduct separately at the beginning. After separate feature extraction, the correlation modeling part entangles features and performs probability estimation as usual.

Following the above definition, the basic module of the proposed model, individual-mix block, is two-fold: The first part independently extracts features for each history symbol, and the second part fuses features and performs probability estimation, named as the individual block and mix block. The individual-mix block can be stacked for further performance improvement. Also, the parameter of proposed block can be dynamically adjusted based on the current compression requirements, which is left as our future work. Figure 3

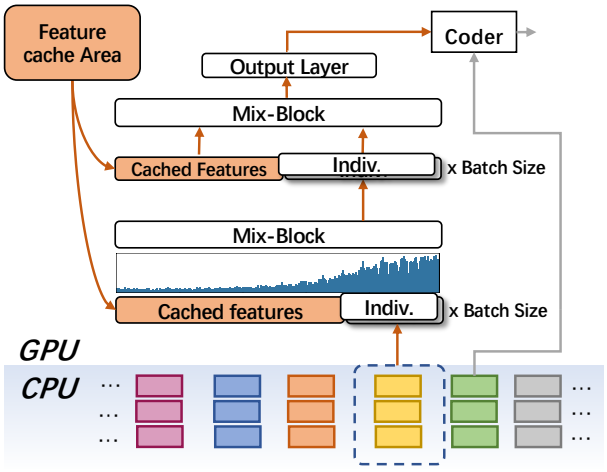


Fig. 3: Illustration of the progressive AR-based compression framework.

shows an example that stacks two individual-mix blocks with different output sizes.

A feature cache is allocate *on GPU* as the compression begins. Given the history symbol set length k, feature dimension v, batch size b, and data type double, the feature cache's storage overhead is $4(k \cdot v \cdot b)$ bytes. For a compression process where $k = 16, v = 16, b = 8192$, such storage overhead is 8MB, which can be negligible comparing with model memory usage, which is usually gigabytes level.

The progressive compression framework can be described as follows: For every iteration, the individual block extracts feature independently for one distinct history symbol. That is, only one symbol needs to be transmitted and processed per compression iteration. The newly extracted feature is stored in the feature cache. If the number of features in feature cache exceeds k, the oldest feature is discarded. Mix block merges the new feature with other history symbol features fetched from the cache and finally performs probability estimation. Detailed progressive compression process for one individual-mix block is described in Algorithm 1.

MLP is chosen as the basic structure. The individual block consists of one fully connected layer, while the mix block contains two. Both blocks contain a layernorm at the end. MLP-based architecture lacks the ability to estimate attention on input sequences, therefore, needs an explicit module to build the ordered importance. Previous approaches utilize Gumble-Softmax sampling, which is generally difficult to apply to edge devices like FPGAs. We replace it with a simple matrix multiplication in the next section.

C. Learned ordered importance

The ordered mask is a crucial component of an MLP-based compressor. It highlights the prominent history symbols and is computationally simpler than the transformer's attention module. The general form of an ordered mask is a one-dimensional vector containing k importance scores for k history symbols.

Algorithm 1: Progressing Compression Process for one individual-mix block.

Input: Byte Stream$\{x_0, x_1, ..., x_{end}\}$, History symbol set length k.
Output: Compressed file.

$i = 0$;
Model.initialize();
Cache.initialize();
while $0 <= i < k$ **do**
  $F(x_i)$ = Model.FeatureExtract(x_i);
  Cache.push($F(x_i)$);
  Encode(x_i, $\frac{1}{256}$);
  $i+ = 1$;
end
while $k <= i <= end$ **do**
  $F(x_{i-1})$ = Model.FeatureExtract(x_{i-1});
  Cache.pop($F(x_{i-k-1})$);
  Cache.push($F(x_{i-1})$);
  Mask = Model.OrderMask($F(x_{i-1}), \ldots, F(x_{i-k})$);
  $F(x_{i-1}), \ldots, F(x_{i-k})$ *= Mask;
  $P_e(x_i)$ = Model.Mix($F(x_{i-1}), \ldots, F(x_{i-k})$);
  Encode(x_i, $P_e(x_i)$);
  Model.backward(x_i, $P_e(x_i)$, loss function);
  $i+ = 1$;
end

Values in ordered mask should be an approximate uphill trend as shown in Figure 4, which indicates closer symbols usually have a substantially larger influence on the target symbol.

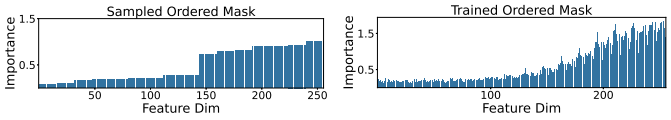
Previous ordered masks are generated by the Gumble-Softmax sampling, which requires complex operations such as log and exponential function. This makes the previous general-purpose compressor incompatible with hardware that can only perform matrix addition and multiplication.

To address this issue, a trainable 1D vector is introduced to replace the Gumble-Softmax sampling. Instead of assigning importance to each history symbol, we assign learnable importance scores on the dimension of the extracted features, after the first individual layer. The ordered importance is modeled as:

$$F(x_{i-1}), \ldots, F(x_{i-k}) = W \cdot \{F(x_{i-1}), \ldots, F(x_{i-k})\} \quad (2)$$

where $F(x_{i-1})$ is the extracted feature of x_{i-1} and W is learned ordered importance. W is joint optimized with other parameters. We put the ordered mask after the first individual layer since it will be applied to the whole history symbol set. Thus the operation has to wait until the first individual layer finished to obtain the features of other history symbols from the cache, as in Figure 3.

As shown in Figure 4, the trained ordered mask can also generate an uphill ordered importance mask. Moreover, the speed of the proposed trained ordered mask is much faster. For samples with a batch size of 512, the proposed method can achieve 0.02 ms per batch, while the Gumble-Softmax sample method takes 0.3 ms.



(a) Gumble-Softmax sampled ordered mask.

(b) Proposed learned ordered mask.

Fig. 4: Examples of sampled ordered masks and proposed trained ordered masks.

D. Batch-location-aware Individual Layer

To address the in-batch distribution variation problem analyzed in Section III-A, a batch-location-aware compression model is proposed to capture the in-batch distribution variation. Based on the proposed model architecture, the individual layer is modified to assign distinct parameters for different positions in the batch. We only modify the individual block because it extracts the features of symbols directly, thus being more sensitive to the symbol distribution. The mix block is mainly responsible for feature fusion and remains as before.

The batch-location-aware modification does not increase FLOPs during inference, despite the increased parameters. This is because the amount of computation remains the same. Assuming batch size is N, if only one set of parameters exists, then this set of parameters must be executed N times for N positions in the batch. Modified batch-location-aware individual layer has N sets of parameters, but each parameter set only needs to compute once on one element in the batch, so the overall computation remains the same. Also, the final model size which increases with the growth of parameters should be considered. Nonetheless, the dynamic compression process randomly initializes the probability estimation model on both the transmitter and receiver, therefore eliminating the need to transmit the model [14]. In other words, the model size will not affect compression ratios.

IV. Experimental Evaluations

A. Experimental Setup

Experiments are benchmarked on seven datasets from various domains. Descriptions of datasets are provided in Table. II. We use Compression Ratio, Peak GPU Memory Usage, Compression Speed, Latency, and the FLOPs (Floating point operations) as metrics. All experimental result is an average of ten repetitive experiments and conducted on an 11GB NVIDIA Geforce RTX 2080 GPU. The proposed method is implemented using Libtorch [15] in C++.

To ensure fairness, all deep-learning methods follow the dynamic compression procedure, which means no pretraining exists. If not mentioned, the history symbol's number and feature dimension are both 16, the mix-block's first layer output dimensions are settled as 4096. The experimental probability estimator contains four individual-mix blocks with individual block in/output size=[16, 32, 64, 128]. This is the same as OREO's experimental model settings. Adam is applied with a learning rate=0.001.

TABLE II: Description of Compression Datasets.

Name	Size	Description
Book	1000MB	First 1000M byte of BookCorpus [16].
Enwik9	1000MB	First 1000M byte of the English Wikipedia [17].
Float	1.1GB	Spitzer Space Telescope data. [18].
Sound	842MB	ESC [19] Dataset for environmental sound.
Image	1.2GB	100000 pictures from ImageNet.
Backup	1000MB	1000M byte random extract from a disk backup.
Silesia	206MB	A compression benchmark [20].

TABLE III: Peak GPU memory usage, Inference Latency and FLOPs of GPU-based method with batch=128. The total 2080 Ti memory size is 10.75G.

Compressor	Peak GPU Memory Usage (GB)	Inference (ms)	FLOPs
NNCP	7.75	95.67	15.83×10^{10}
Dzip	6.39	5.82	7.48×10^{10}
TRACE	2.02	2.08	0.34×10^{10}
OREO	1.18	1.54	0.12×10^{10}
PAC	1.07	1.54	0.1×10^{10}

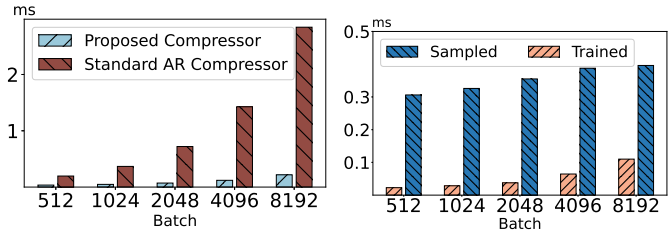
B. Efficiency Evaluations

We first represent the efficiency metrics of GPU-based compressors in Table. III, including Peak GPU Memory Usage, Model Latency, and the FLOPs.

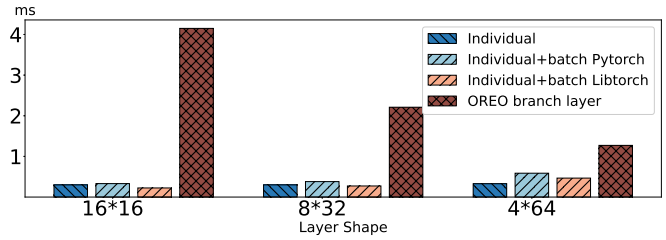
Under batch size 128, PAC only takes 1.07G GPU memory, which is half of TRACE and 1/6 of Dzip. PAC's latency and the number of FLOPs also shows comparable results with the state-of-the-art AR-based methods. In Table. III, PAC has the same inference time as OREO. This is because the batch size=128 is too small for these two compressors, so the GPU's processing units are not fully utilized. When batch size=8192, the inference time of PAC is 18.2ms while OREO takes 23.5ms, which is notably faster than OREO.

Next, we investigate the time breakdown of each key component. Figure 5a shows the host-GPU transfer time for standard AR-based compression and PAC. Transfer time has been reduced by a large margin in all batch settings, which demonstrates proposed progressive compression framework's effectiveness on data tranmission aspect. Figure 5b shows the proposed trainable ordered mask can reduce ordered mask generation time from $\sim$0.4 to $\sim$0.1 for large batch size 8192. When the batch size is 512, the ordered mask generation time can be reduced from 0.3ms to 0.01ms.

We test the inference speed of individual blocks under different configurations in Figure 5c. The definition of $m * n$ on the x-axis is m history symbols with each feature vector dimension is n. A detailed explanation can be found in [8]. Compared with OREO's branch layer, the proposed individual block is significantly faster across all configurations. The speed of batch-location-aware individual layers in Pytorch and libtorch is also reported. We can see that batch-location-aware individual block in python is slightly slower than the original individual block, but libtorch version can obtain the same speed. This demonstrates proposed batch-location-aware solution does not increase computational loads.



(a) Host-GPU data transmit time. (b) Ordered mask generation time.



(c) Individual layer inference time.

Fig. 5: Efficiency validation on proposed techniques.

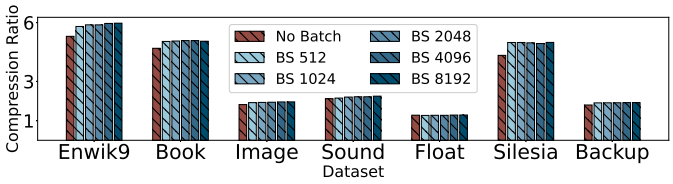


Fig. 6: Compression Ratio for proposed model with different batch settings.

C. Compression Ratio Evaluation

We first report the evaluation of the compression ratio for the proposed model with different batch settings in Figure 6. After adding batch-location-aware, we can see that although the impact of the batch size varies from data to data, the compression ratio improvement brought by batch-location-aware components is general on all data types.

Table. IV presents the compression ratio of the proposed compressor and other state-of-the-art compressors. Gzip lags far behind other compressors in compression ratios for all data domains. Zstd-19 and 7z are more advanced. Nevertheless, deep-learning compressors can easily outperform these non-deep-learning compressors. TRACE leverages a slim transformer structure to achieve better compression ratios with faster speeds. OREO solved the problem of TRACE and improved the compression ratio in all data.

PAC significantly improved over OREO for data compression in all domains. If we use OREO as the baseline, PAC has a 5.1% and 2.22% compression improvement on Enwik9 and Book. On Image and Float, the improvements are 5.38%, and 0.78%, respectively. For heterogeneous data Silesia and Backup, PAC is 2.68% better than OREO on both dataset, respectively.

V. Conclusion

This paper proposes PAC, a fast and efficient compression framework by addressing the incompatibility between the AutoRegressive framework and general-purpose lossless compression. This paper addresses two problems: duplicated processing problem and in-batch distribution variation problem. This paper first proposes a progressive compression framework with an individual-mix block to alleviate the duplicated processing problem, and a trainable matrix multiplication operation to replace hardware-unfriendly Gumble-Softmax sampling. The in-batch distribution variation problem is mitigated by capturing the local distribution of different sub-sequences, which can improve compression performance without computational loss. Experimental results show that PAC can achieve average to 130% faster compression speed and 3% higher compression ratio than SOTA for various data types.

TABLE IV: Compression Ratios on Large Datasets.

Methods	Homogeneous Data					Heterogeneous Data	
	Enwik9	*Book*	*Sound*	*Image*	*Float*	*Silesia*	*Backup*
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
Dzip	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
PAC	5.97	5.05	2.25	1.96	1.29	4.99	1.92

References

[1] DKRZ, "Unique status among german high-performance computing centres." 2021.
[2] Wikipedia, "Amazon web serivce." 2020.
[3] P. Deutsch, "GZIP file format specification version 4.3," *RFC*, vol. 1952.
[4] Y. Collet, "Zstd github repository from facebook.," 2016.
[5] B. Knoll, "Cmix." 2014.
[6] G. Mohit *et al.*, "Dzip: Improved general-purpose loss less compression based on novel neural network modeling," in *DCC'2021*.
[7] Y. Mao *et al.*, "Trace: A fast transformer-based general-purpose lossless compressor," in *WWW*, pp. 1829–1838, 2022.
[8] Y. Mao *et al.*, "Accelerating general-purpose lossless compression via simple and scalable parameterization," in *MM'22*, pp. 3205–3213, 2022.
[9] Cui *et al.*, "Fully nested neural network for adaptive compression and quantization," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 2080–2087, International Joint Conferences on Artificial Intelligence Organization, 7 2020.
[10] Kingma *et al.*, "Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables," in *ICML*, pp. 3408–3417, PMLR, 2019.
[11] van den Berg *et al.*, "Idf++: Analyzing and improving integer discrete flows for lossless compression," in *ICLR*, 2020.
[12] B. Knoll, "Tensorflow-compress," 2016.
[13] F. Bellard, "Nncp: Lossless data compression with neural networks.." 2019.
[14] C. Zhang *et al.*, "Osoa: One-shot online adaptation of deep generative models for lossless compression," *NIPS*, 2021.
[15] Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NIPS*, 2019.
[16] Vaswani *et al.*, "Attention is all you need," *NIPS*, vol. 30, 2017.
[17] M. Mahoney, "Large text compression benchmark." 2006.
[18] M. Burtscher *et al.*, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, 2009.
[19] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," 2015.
[20] S. Deorowicz, "Silesia dataset." 1985.