IEICE
TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2024EDP7074

Publicized:2024/11/08

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Lightweight Neural Data Sequence Modeling by Scale Causal Blocks

Hiroaki AKUTSU [y], Member and Ko ARAI [y], Nonmember

SUMMARY Autoregressive probability estimation of data sequences is a fundamental task in deep neural networks and has been widely used in applications such as data compression and generation. Since it is a sequential iterative process due to causality, there is a problem that its process is slow. One way to achieve high throughput is multiplexing on a GPU. To maximize the throughput of inference processing within the limited resources of the GPU, it is necessary to avoid the increase in computational complexity associated with deeper layers and to reduce the required memory consumption at higher multiplexing. In this paper, we propose Scale Causal Blocks (SCBs) which are basic components of deep neural networks that aim to significantly reduce the computational and memory cost compared to conventional techniques. Evaluation results show that the proposed method is one order of magnitude faster than a conventional computationally optimized Transformer-based method while maintaining comparable accuracy, and also shows better learning convergence.
key words: Probability Estimation, GPU, Computational Efficiency, Neural Networks

1. Introduction

One of the basic tasks in deep neural networks is the probability estimation of data sequences. Autoregressive probability estimation, which is a simple task of predicting the next data from past data sequences, is known to achieve high accuracy when implemented by deep neural networks. It has been widely applied to the generation of text data [2], audio data [3], and image data [4] by sampling data based on the estimated probability distribution of the next data. Autoregressive probability estimation can also be applied to image compression [5, 6], video compression [7, 8], and lossless compression [9] by combining it with entropy coding [10{12].

Autoregressive probability estimation generally suffers from slow processing since it is a sequential iterative process due to the causality. We therefore aim to establish efficient network components for autoregressive probability estimation. One approach to this problem is to perform multiplexing on GPUs with highly parallelized processing cores. To achieve high-throughput processing on GPUs, it is important to reduce both the computational and memory costs for limited GPU resources.

In this paper, we set the problem statement to solve the slow processing and large memory consumption of coding tasks and generating tasks with autoregressive probability estimation by deep neural networks. In particular, lossless compression, which is one of the coding tasks, is typically applied to large data such as genomics, images, and experimental data. However, lossless compression is a highly cost sensitive task and suffers from slow processing with multiple layers and large memory consumption with parallelization. In this paper, we mainly focus on the problem of lossless compression for the above reasons. In addition, we conduct an experiment on a generation task to reinforce the effectiveness of our proposal.

In this work, we propose Scale Causal Blocks (SCB) which are basic deep neural network components for autoregressive probability estimation that enables faster processing compared to conventional techniques.

Our main contributions are as follows.

- We proposed SCBs as the basic components for the autoregressive probability estimation of data sequences. The computational cost was dramatically reduced while maintaining accuracy by combining convolution, scaling, and self-attention. We introduced self-attention in a short-cut path with a scaling manner for computational efficiency and also achieved sufficient training with fewer parameters by introducing weight sharing in the deeper layers due to the structural properties of the SCBs.
- We proposed inference algorithms with different parallelization strategies during training and inference. Specifically, during training, convolution is utilized to efficiently train long sequences in the context direction, and during inference, the weights of the convolutional layer are converted into a simple linear layer for faster processing by batch parallelization. The batch multiplicity can be made to thousands or more, thus achieving high throughput by reducing the amount of memory required to cache the context for inference.
- Through our experiments, we demonstrated that the proposed algorithm can achieve faster inference throughput with comparable accuracy and better learning convergence compared to the Linear Transformer, a computationally optimized Transformer-based method.

Again, our goal is to establish efficient network components at a reasonable accuracy, not to achieve state-of-the-art accuracy. This perspective is now particularly important for bit cost reducing tasks such as compression tasks. We do not

[y]The author is with the R&D Group, Hitachi, Ltd., Yokohama-shi, 244-0817 Japan. A preliminary version of this paper was presented at ICML 2023 Workshop as "Fast Autoregressive Bit Sequence Modeling for Lossless Compression" [1] by the same author. The explanation and experimental results were expanded.

intend to claim in this paper that our method can be applied to SOTA's large scale model for any tasks such as LLMs.

2. Related Works

This section describes related research on modeling with deep neural networks for data sequencing.

2.1 Transformer-based Models

Transformer [13] is a model that has been widely utilized in data sequence modeling over the past few years. Introducing self-attention for simple linear layer networks can capture a wide range of data sequence characteristics with better prediction accuracy. Furthermore, by introducing position embedding in the input vectors, the model easily takes into account the position of the input data.

However, the conventional Transformer is computationally inefficient for processing long data sequences, as the computational cost of a self-attention is $O(N^2)$ for the length N of the data sequence. In regard to this issue, several methods have been proposed to improve the efficiency of the self-attention calculation [14–20]. Among these, the Linear Transformer [17] shows particularly promising results in reducing the computation of self-attention to $O(N)$ and has excellent computational efficiency. LinFormer [21] is computationally linear ($O(N)$), but its self-attention structure requires data in the spatial dimension to be input into the Linear layer (for BERT-style attention, as this paper is targeting). Therefore, it is difficult to apply the LinFormer to autoregressive inference (decoding), which is the target of our paper, and LinFormer paper does not disclose a method for masked attention. There is also a hardware-oriented approach to speed up self-attention research by optimizing GPU memory access [22], which has achieved a 3X speed-up in GPT-2 [23]. Also, research on very deep Transformer methods is progressing, and it is now possible to construct large-scale models [24]. However, the overall computational cost increases in proportion to the number of layers L, making the overall cost equivalent to $O(NL)$.

2.2 CNN-based Models

Since Transformer is based on a linear layer, it cannot consider any other data than the current position (except self-attention). In contrast, CNN can consider the neighboring data sequence if the kernel size k is greater than or equal to 2. An auto-regressive model with 2D CNN for image data used for image compression and image generation has been reported [4, 25].

Wavenet [3] is a CNN-based model for modeling long receptive field data sequences through dilated convolution. Caching the intermediate results of the dilated convolution can reduce the redundant computation of the feature map during inference [26]. However, an exponentially large number of caches related to the number of layers is required, and it is difficult to increase the multiplicity of inference processing beyond the order of thousands due to the limited amount of memory on GPUs. In addition, the overall computational cost increases in proportion to the number of layers L, making the overall cost equivalent to $O(NL)$ (the same as with Transformer).

Fig. 1 Overview of receptive fields (shown as orange circles) in Transformers and SCBs.

3. Scale Causal Blocks

In light of the above background, we propose SCBs as the basic components of deep neural network for the autoregressive probability estimation of data sequences. In this section, we describe the unique features of SCB and estimate the effects of its computational and memory costs. By combining convolution, scaling, and self-attention, we achieve significant reductions in computational cost while maintaining high accuracy. We also introduce weight sharing and parallelization strategies to optimize training efficiency and speed up inference. The reason we focused on autoregressive probability estimation is that it is generally a sequential iterative process based on causality, and so it has the problem of being slow with the inference phase, and this is a particularly big issue. The strategy in this study is to parallelize in the context dimension during training and then change to parallelization in the batch dimension during the inference phase, and this will contribute to speeding up the inference phase. The proposed method not only improves computational efficiency, but also improves memory efficiency, so that even GPUs with memory constraints can process with a high degree of parallelism, and improve processing speed.

3.1 Scaling Causal Convolution with Self-attention

SCB has a unique feature that combines convolution, scaling, and self-attention for autoregressive probability modeling of data sequences at low computational cost. Figure 1 shows an overview of the receptive fields of the Transformer and the SCBs. In Transformers, past contexts are considered by self-attention, whereas in the proposed SCB, past contexts are considered by both convolution and self-attention to improve efficiency. The structures of the two basic building blocks that make up the SCB, which we call Downscale Block (DB) and Upscale Block (UB). As shown in Figure 1, the input data to the DB is reduced by the Down operation, which reduces the size of the context dimension. In Figure 1, only one DB and one UB are shown for simplicity, but in the actual configuration, multiple DBs and UBs are nested. This reduces the size of the context dimension with each layer, it reduces the computational cost. In terms of autoregressive probability estimation, reducing the size of the context dimension has the effect of reducing the number of layers processed on average in each inference process in an iteration, which in turn results in faster processing speeds. In addition, the memory cost can be reduced by reducing the number of attention and the number of attention channels. Further, the input and output to the DB and UB are expressed in tensor format. Specifically, tensor x is composed of a $\times \times N$ three-dimensional array, and the number of elements in the context dimension in the explanation of Figure 1 corresponds to N. Although it is omitted in Figure 1, is expressed as the number of channels and is expressed as the number of batches.

3.1.1 Building blocks

The detailed structures of the DB and UB are shown in Fig. 2. DBs/UBs handle *scaling*, the reduction/expansion of a feature map to the context dimension, and it is key function for reducing computational cost as in U-net [27]. Each block inputs a tensor $x^{(l)}$ and a list of short-cut path tensors $s^{(l)} = [a^{(1)}, \ldots, a^{(l)}]$ and outputs $x^{(l+1)}$ and $s^{(l+1)}$ at the lth block layer. The DB halves the size of the tensor x in the context dimension, and the UB doubles its size. A typical configuration using these blocks is to connect multiple layers of DBs followed by multiple layers of an equal number UBs such that the sequence length of the first input N and the last output match.

DBs process an input tensor $x^{(l)} \in \mathbb{R}^{C_{in} \times N}$ by a 1D CNN and then produce two types of outputs: half-downscaled tensor $x^{(l+1)} \in \mathbb{R}^{C_{out} \times N/2}$ and short-cut path tensor $a^{(l+1)} \in \mathbb{R}^{C_{out}/2 \times N}$ appended on $s^{(l+1)}$, as shown in Fig. 2. The short-cut path tensor is processed with a masked linear attention [17]. UBs first concatenate an input tensor $x^{(l)} \in \mathbb{R}^{C_{in} \times N}$ and short-cut path tensor $a^{(l)}$ poped from $s^{(l)}$ using the padding and deleting tensor operations shown in Fig. 2. Pad$(1:0)$ represents one zero padding on the left side of the context dimension, and Delete$(0:1)$ represents one deletion from the right side. The UBs then process the concatenated tensor by a 1D CNN and finally produce a twice-upscaled tensor $x^{(l+1)} \in \mathbb{R}^{C_{out} \times 2N}$. In each block, the kernel size of the 1D CNN is 2 and the stride size is 1. The exponential linear unit (ELU) [28] is used as the activation function for 1D CNNs. In the DBs, the results of convolution is processed in the order of the Split and the Down. By executing the Split before the Down, it is possible to pass a finer-grained context to the UBs. From the perspective of reducing the context dimension, convolution with a stride greater than 1 might be considered as an alternative to the combination of Down (of previous layer) and the 1D CNN ($= 2$). However, when we employ this structure, the

Fig. 2 Architecture of SCB.

context passed to the UBs will be reduced and fine-grained context will disapper. That is why we employed the structure described above.

3.1.2 Scaling with Down / Up operations

In DBs and UBs, the operations that reduce and expand the context dimension are Down and Up. Our method aims to achieve these operations quickly and without any arithmetic operations by replacing elements of the context dimension with the channel dimension (where is the size), similar to PixelShuffle [29] in the image processing field. Assume a tensor $x \in \mathbb{R}^{C \times N}$ (batch dimension is omitted because it is simply multiplexed), where each element of x is denoted as $x_{1,1} \ldots x_{C,N}$. The Down operation is then defined as

$$\mathrm{Down}(x) = \begin{bmatrix} x_{1,1} & x_{1,3} & \cdots & x_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{C,1} & x_{C,3} & & x_{C,N-1} \\ x_{1,2} & x_{1,4} & & x_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{C,2} & x_{C,4} & & x_{C,N} \end{bmatrix}_{2C \times N/2} . \tag{1}$$

And the Up operation is defined as

$$\mathrm{Up}(x) = \begin{bmatrix} x_{1,1} & x_{C/2+1,1} & & x_{1,N} & x_{C/2+1,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{C/2,1} & x_{C,1} & & x_{C/2,N} & x_{C,N} \end{bmatrix}_{C/2 \times 2N} . \tag{2}$$

The goal of scaling is to reduce the computational cost while exponentially expanding the receptive field in the context dimension. This goal is precisely the theme of our paper, to reduce computational cost without sacrificing accuracy.

3.1.3 Self-attention on short-cut path

Since SCB has a short-cut path similar to U-net [27], we utilize it in the outputs s of each DB as a direct input to the corresponding UB, without processing of deeper blocks that have been reduced in the context dimension. In this way, it can avoid missing granularity information in the context dimension. We apply masked linear attention [17] to the feature maps of the short-cut path a (split tensor) to further improve the prediction accuracy at a low computational cost. The masked linear attention process [17] is expressed by

$$a'_i = \frac{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j)} + a_i, \tag{3}$$

where i represents the position of a context dimension, $Q_i = a_i W_Q$, $K_i = a_i W_K$, and $V_i = a_i W_V$ represent queries, keys, and values, respectively, and $\phi(x) = \mathrm{elu}(x) + 1$. For more detail, see the supplementary material.

Again, we introduced to combining masked linear attention with the scaling, which is computationally less expensive than typical self-attention. This aims to ensure that our SCB can consider the probability of data sequence in a uniform way across the wide context view.

Algorithm 1 Inference of down-scale block.

Require: $x \in \mathbb{R}^{d_{in} \times s}$
(Initial: $c_x \leftarrow 0^{d_{in}}$, $c_s \leftarrow$ None)
Ensure: $x \in \mathbb{R}^{d_{out} \times s}$

```
if x is None then
    append(s, None)
    return None, s
end if
t ← cat(c_x, x)                    ▷ t ∈ R^{2 d_in}
c_x ← x
x ← t
x ← linearfromconv(x)              ▷ x ∈ R^{d_out}
x ← elu(x)
x, a ← split(x)                    ▷ x, a ∈ R^{d_out/2}
a' ← attention(a) + a
append(s, a')
if c_s is None then
    c_s ← x
    x ← None
else
    x ← cat(c_s, x)
    c_s ← None
end if
return x, s
```

Algorithm 2 Inference of up-scale block.

Require: $x \in \mathbb{R}^{d_{in} \times s}$
(Initial: $c_{x1}, c_{x2}, c_s \leftarrow 0^{d_{in}/2}$)
Ensure: $x \in \mathbb{R}^{d_{out} \times s}$

```
a' ← pop(s)
if a' is None then
    return x, s
end if
if x is None then
    x ← cat(c_x1, c_s, c_x2, a')   ▷ x ∈ R^{2 d_in}
else
    t ← c_x2
    c_x1, c_x2 ← split(x)
    x ← cat(t, c_s, c_x1, a')      ▷ x ∈ R^{2 d_in}
end if
c_s ← a'
x ← linearfromconv(x)              ▷ x ∈ R^{d_out}
x ← elu(x)
return x, s
```

3.1.4 Weight sharing of deep layers

Due to the characteristics of SCBs, the data size of the feature map in the context dimension halves each time it goes to deeper layer, which makes it difficult to achieve stable training on these blocks. Therefore, we propose a method for sufficient training with fewer parameters that shares the weights of deep layers of the block by taking advantage of the characteristics of CNNs that can process even if the input size is changed in multi-scale. Specifically, assume an SCB network consisting of L layers of both DBs and UBs, where the weights of the 1D CNNs of DBs after the l-th DB layer are shared.

3.2 Fast Inference Algorithm

This section describes the processing of SCB during inference, where the SCB has different parallelization strategies during learning and inference. The two main features are explained below.

- Convolution into Linear: During learning, the SCB uses convolution to efficiently learn long sequences in the context direction (i.e., the batch multiplicity is relatively small). In contrast, during inference, the batch multiplicity increases to achieve high throughput since it is a iterative sequential process. During inference, the weights are converted to the linear layer format, which enables efficient procesing with a high batch multiplicity.
- Minimal caching: SCB is more memory efficient because it does not maintain a large amount of cache even in deep layers. For the dilated convolutions, exponential large cache size of context in deeper layer, whereas for SCBs, each layer requires only constant cache size $x \in \mathbb{R}^{d_{in}}$ since the context is extended by scaling. In addition, since attention is applied only to features in the shortcut paths, less memory is required for the iterative attention process in SCBs than Linear Transformers.

Algorithms 1 and 2 are the inference algorithms of SCB. The architecture in Fig. 2 is essentially a process during learning, while these algorithms are a process during inference. Although they have different parallelization policies, they are equivalent in terms of results and computational complexity. The details are omitted since the operation has already been outlined in Fig. 2, but we briefly go over the notable parts of the inference process in the following.

Since this is an autoregressive inference, the tensor $x \in \mathbb{R}^{d_{in}}$ at a certain location in the context dimension is used as the input of DBs and UBs. The initialize process sets initial values for the variables (c_x, c_s, c_{x1}, and c_{x2}) used as the caches of intermediate data in DBs and UBs before data sequence processing (corresponding to the Pad operations in Fig. 2). These variables are internal static variables in each block and are maintained during data sequence processing. By using these caches and concatenating the calculation results, equivalent processing to convolution can be done in the sequential processing of inference while increasing the multiplicity in the batch dimension (e.g., to several thousand or more). The cat and split are operated for the context dimension. The linearfromconv is a process that replaces the convolution layer with a linear layer. Speci cally, the convolution process can be viewed as a linear layer with $_{in}$ channels as input and $_{out}$ channels as output, so the parameters of the convolution kernel are converted into those of the linear layer. This eliminates the process for dimensional conversion of tensors, and allows for faster processing. The e ect of reducing the computational complexity of the deeper layers by scale with decreasing the size of the feature map of the context dimension corresponds to the fact that, in the inference, the linearfromconv process and the attention process are executed less frequently as the layers of the block become deeper (due to the conditional branching of the algorithm), so the frequency of execution decreases.

3.3 Analysis

In this section, we investigate the potential of the SCB by estimating its computational and memory costs and comparing it with conventional methods.

3.3.1 Computational cost

The relationship between the number of layers and oating operations (FLOPs) for SCB and other methods is shown in Fig. 3. The number of channels is assumed to be $_{in}$– $_{out}$ = 256 and is the same for all network types. This assumption also holds for the experimental results that follow, which show that the networks have approximately the same prediction accuracy. Increasing the number of layers not only increases the nonlinearity of the processing and improves the expressiveness of the network but also increases the receptive eld in the convolution, which is advantageous because it means that longer contexts can be considered. The computational complexity increases with the number of layers in general as indicated by orange and grey curves, but as we can see in Fig. 3, the computational cost of the SCBs saturates with respect to the increase in the number of layers.

3.3.2 Memory cost

CNNs and attentions other than simple linear layers require cache memory to hold intermediate data (context) during inference. The capacity of this cache memory is proportional to the multiplicity (number of batches) during inference, so it must be smaller to achieve high multiplicity. The relationship between the number of layers and intermediate cache memory cost for SCB and other methods is shown in Fig. 4. In the case of dilated convolution, the amount of cache memory used increases exponentially with the number of layers, but the cost of the SCBs is only proportional to the number of layers. Furthermore, since attention is applied only to the features in short-cut paths, it reduces both the dimensionality of the channels of the attention networks and the number of attention mechanisms compared to conventional linear attention networks, and thus requires less memory for the iterative attention process in SCBs.

Fig. 3 Computational cost analysis

Fig. 4 Memory cost analysis.

4. Experimental Results

This section presents the results of experimental studies on the e ectiveness of SCB. We performed experiments on two tasks: lossless block compression and image generation.

4.1 Experiment 1: Lossless Block Compression

Lossless block compression is a simple task that divides data into blocks of a xed length #, treats each block simply as a bit sequence G1– •••– #G2 f0–1g, and autoregressively estimates probabilities by a model for entropy coding. The model \ is trained by an average of the Kullback-Leibler divergence of the ground truth probability distribution ?¹G⁰ and the estimated probability distribution @¹G⁰, as

$$L = E_8» ! ¹?¹G_8ºjj@¹G_8jG_1– •••– _8G_1ºº¼ \quad (4)$$

Entropy coding using the estimated probability can compress with bitrate approximately equal to the negative log-likelihood [30]. This loss L is equal to the negative log-likelihood (and also it is equal to cross-entropy) when ?¹G⁰ is equal to one-hot encodings of the ground truth bits. So L can be treated as the theoretical average compression ratio.

The processing throughput of the compression task is important from a practical point of view because one of the main objectives of compression is to reduce storage costs. If the throughput of the compression process is slow, more time spent to occupy computing resources such as GPUs, which results in the e ect of reducing storage costs by compressing data will be o set by the computational costs.

We experimented with SCB on this task to determine whether SCB can handle probability prediction at high speed. We utilized three di erent types of open datasets (Genomics

Table 1 Details of evaluation datasets for lossless block compression.

Dataset	Items	Description
Genomics	Name	Illumina HiSeq 2000 paired end sequencing GSM1080195: mouse oocyte 1 Mus musculus RNA-Seq [31]
	URL	https://www.ebi.ac.uk/ena/
	File (train)	SRR689233_1.fastq (3.87 GB) (md5: 56cb883e8b42344384b9e4ccc90ec9db)
	File (test)	SRR689233_2.fastq (3.87 GB) (md5: 92439bb6745f4abbf46b99efcbf20a02)
MRI	Name	In vivo High Angular Resolution Diffusion -weighted Imaging of Mouse Brain at 16.4 Tesla [32]
	URL	https://dataverse.harvard.edu/
	File (train)	in-vivo-DWI-EPI.tar (0.94 GB) (md5: 4b247a403110dceb9631b365cee42813)
	File (test)	invivo-insitu-experiment.tar (0.76 GB) (md5: 5eb5203b0fca67411f39c2377336605b)
Physics	Name	HEPMASS Dataset [33]
	URL	http://archive.ics.uci.edu/
	File (train)	all.train.csv (5.18 GB) (md5: 5b1fc2dafe14aa2f661cc3de5ccf3984)
	File (test)	all.test.csv (2.59 GB) (md5: 414f886d007f18b1eb97257a36120389)

[31], MRI [32], and Physics [33]) to evaluate the theoretical average compression ratio and the processing speed of the probability estimation model. We compared the results to the Linear Transformer [17] as a baseline. Lossless block compression divides chunks of data into blocks of a fixed size for faster loading by partial decoding and parallel processing. In our experiment, the size was set to 1,024 bytes (# = 8192 bits). In the SCB experiment, the DB and UB were configured to be coupled with ten layers each, and the channel sizes $_{in}$ and $_{out}$ were set to 256. In the Linear Transformer experiment, we set the embedding size to 256, the number of heads to 8, and the number of layers to 16, as in the experimental configuration described in [17]. In both experiments, as with the general Transformers [13], the input bits were embedded to a 256-dimensional value and positional encoding was added. As a final layer, a linear layer with one output channel and a Sigmoid function were applied. The model was trained to output a probability that the bit is 1. We used the ADAM optimizer [34] with a learning rate of $1 \cdot 10^{-4}$ for training and a batch size of 8. The training iteration was 200,000. We implemented the experimental code on Pytorch and used the library for fast transformer implementations [17] for the masked linear attention part. The experiments were performed with 32-bit floating-point arithmetic simply for the sake of pure method comparison. We used a single NVIDIA® V100 for each experiments. The details of the datasets used in the lossless block compression evaluation experiments are shown in Table 1.

Table 2 lists the results. We measured throughput with a batch multiplicity of 8,192. As we can see, the SCB achieves a speedup of more than one order of magnitude over the conventional Linear Transformer with an equivalent theoretical average compression ratio. These results are supported by the fact that the estimated computational cost (MFLOPs/bit) of SCBs is smaller than that of Linear Transformers and Performer.

Note that the Performer throughput is due to the encoding process and not the result of the autoregressive decoding process. This is because the current implementation of peformer is not optimized for the decoding process and is a reference value representing the ideal state for comparison. Therefore, it is expected that the performers' throughput will be slightly slower than this value for the autoregressive decoding process, which is the subject of this paper.

The Physics dataset achieves a lower theoretical average compression ratio with SCB. For more details about this, please see the supplementary material.

A comparison of the compression ratios with gzip [35], a common conventional compression, is shown in Table 3. Note that the compression ratio represented in Table 3 is a few percentage points higher than the bpd represented in Table 2, due to the impact of entropy coding. SCB allows partial encode/decode in 8192 bits units due to block compression. The SCB compression ratio includes the coding overhead. SCB has an advantage in the compression ratio even when compared to gzip without block compression (full-file compression) in the highest compression mode (option -9).

The processing speeds with different numbers of batches are shown in Fig. 5, where we can see that the performance improves as the number of batches increases. This is because the parallel processing on the GPU is working effectively.

Table 4 also shows a comparison of the experimental results of theoretical compression rate when SCB scale is disabled/enabled and when self-attention is disabled/enabled. As we can see, scaling and self-attention were both effective for theoretical average compression ratio reduction. This is because those function have the effect of expanding the receptive field. Table 5 shows a comparison of the experimental results of theoretical compression rate when weight sharing is disabled/enabled. Higher accuracy was achieved when weight sharing was enabled, and fewer parameters were required when compared with the same number of layers ! = 10. Note that the Linear Transformer has the same level of accuracy with 12.6M parameters, and the SCB can be achieved with very few parameters.

4.2 Experiment 2: Image Generation

For the image generation experiment, we use an autoregressive model to generate images by sampling the predicted probability distribution of pixels. Here, the image data is converted into a one-dimensional byte data sequence with raster scan order of 3-color information (3 bytes), and then processed by the model to evaluate the data sequence modeling performance. This autoregressive model can be learned by Kullback-Leibler divergence, as in the compression case in Eq. 4. It differs from the bit-sequence case above in that it targets pixels, which are generally 8-bit non-negative integers $G_1, \ldots, G_\# \in \{0, \ldots, 255\}$, and the probability distribution $@(G)$ is represented by ten mixed logistic distributions

Table 2 Experimental results of probability estimation of lossless block compression task (values in parentheses are standard deviations).

Method	Theoretical compression rate			Throughput	Cost
	Genomics	MRI	Physics	[Mbit/sec]	[MFLOPs/bit]
Linear Transformer [17]	0.218 (0.004)	0.418 (0.007)	0.213 (0.002)	0.143 (0.0002)	12.8
Performer [16]	0.261 (0.003)	0.482 (0.017)	0.262 (0.001)	0.162 (0.0007) *	14.7
SCB (proposed)	0.217(0.006)	0.419(0.004)	0.137(0.004)	2.613(0.0039)	0.7

Table 3 Experimental results of compression ratio. (values in parentheses are standard deviations)

Method	Compression ratio		
	Genomics	MRI	Physics
gzip -9 with 8192 bits block	0.464 (0.000)	0.749 (0.000)	0.481 (0.000)
gzip -9 with no block	0.332 (0.000)	0.635 (0.000)	0.334 (0.000)
SCB (Proposal)	0.222(0.006)	0.425(0.004)	0.143(0.004)

Table 4 Preliminary experimental results of scale and self-attention using Genomics dataset (values in parentheses are standard deviations).

Method	Rate	Cost [MFLOPs/bit]
No scale or attention	0.291 (0.002)	1.4
No scale	0.259 (0.005)	2.1
No attention	0.241 (0.009)	0.5
Full (proposed)	0.217(0.006)	0.7

Table 5 Preliminary experimental results of weight sharing using Genomics dataset (values in parentheses are standard deviations).

Method	Rate	Parameters
No sharing (= 6)	0.226 (0.003)	2.0M
No sharing (= 10)	0.220 (0.007)	3.3M
Weight sharing (! = 10– , = 6)	0.217(0.006)	2.8M

Fig. 5 Throughput of SCB with di erent batch sizes.

rather than a simple bit probability output, as introduced by [25]. Our experimental conditions are essentially the same as in [17], and we evaluated their implementation as a base. The SCB and Linear Transformer channel con guration is the same as the block compression task. We used the RAdam optimizer [36] with a learning rate of 10^{-4} and gradually reduced the learning rate to 10^{-5} for stable training. We used a dropout rate of 0.1 and the training iteration was 500,000 with a batch size of 10.

Experimental results of the image generation for SCB and Linear Transformer [17] when trained on the CIFAR10 dataset [37] are shown in Fig. 7. As we can see, the same level of plausible images can be generated by both. Table 6 shows the processing speed of image generation and the quality of image generation in Bits/dimension (bpd), where we measured the throughput with a batch multiplicity of 10,000. It is clear that under conditions of similar accuracy in bpd, image generation by the proposed method is extremely fast: speci cally, by more than one order of magnitude with the same image generation quality.

Fig. 6 Comparison of training convergence of Transformer and SCB.

Fig. 6 compares the training convergence of SCB and Linear Transformer. For each 3,000 iterations, bpd was evaluated using a test set. We can see that SCB takes less time to train and converges faster. This is due to its reduction in the computational cost of learning.

5. Conclusion

In this work, we proposed SCBs as the basic components for autoregressive probability estimation of data sequences. The computational cost was dramatically reduced while maintaining accuracy by combining the convolution, scaling, and self-attention. We also proposed algorithms with di erent parallelization strategies during training and inference for faster processing. Experimental evaluations demonstrated that the proposed algorithms can achieve faster inference throughput and comparable accuracy to the Linear Transformer, a computationally optimized Transformer-based method.

We evaluated the proposed method under a relatively small number of parameters for tasks such as cost-sensitive data compression. We believe it could reduce the environmental impact on society by reducing the consumption of storage and network bandwidth in the future. Comparative evaluation with a larger number of parameters against

Fig. 7 Conditional generated images using CIFAR10 dataset (left: Linear Transformer and right: SCB (proposed)). Upper half of each image was conditioned.

Table 6 Experimental results of image generation task using CIFAR10 dataset (values in parentheses are standard deviations)

Method	Bpd	Throughput [pixel/sec]	Time (10K images) [sec]
Linear Transformer	3.433 (0.010)	48.4K (0.054)	211.5 (0.237)
SCB (proposed)	3.407(0.004)	805.0K (5.744)	12.7(0.092)

state-of-the-art methods was not considered here, nor was the e ectiveness of our technique for large-scale models; we leave this to future work.

References

[1] H. Akutsu and K. Arai, \Fast autoregressive bit sequence modeling for lossless compression," ICML 2023 Workshop Neural Compression: From Information Theory to Applications, 2023.

[2] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., \Google's neural machine translation system: Bridging the gap between human and machine translation," arXiv preprint arXiv:1609.08144, 2016.

[3] A.v.d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, \Wavenet: A generative model for raw audio," arXiv preprint arXiv:1609.03499, 2016.

[4] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al., \Conditional image generation with pixelcnn decoders," Advances in neural information processing systems, vol.29, 2016.

[5] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, \Conditional probability models for deep image compression," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp.4394{4402, 2018.

[6] D. Minnen, J. Balle, and G.D. Toderici, \Joint autoregressive and hierarchical priors for learned image compression," Advances in neural information processing systems, vol.31, 2018.

[7] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, \Dvc: An end-to-end deep video compression framework," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp.11006{11015, 2019.

[8] F. Mentzer, G. Toderici, D. Minnen, S. Caelles, S.J. Hwang, M. Lucic, and E. Agustsson, \Vct: A video compression transformer," Advances in Neural Information Processing Systems, 2022.

[9] F. Bellard, \Nncp v2: Lossless data compression with transformer." https://bellard.org/nncp/nncp_v2.1.pdf , Feb. 2021.

[10] G.N.N. Martin, \Range encoding: an algorithm for removing redundancy from a digitised message," Proc. Institution of Electronic and Radio Engineers International Conference on Video and Data Recording, 1979.

[11] D. Marpe, H. Schwarz, and T. Wiegand, \Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard," IEEE Transactions on circuits and systems for video technology, vol.13, no.7, pp.620{636, 2003.

[12] J. Duda, \Asymmetric numeral systems," arXiv preprint arXiv:0902.0271, 2009.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin, \Attention is all you need," Advances in neural information processing systems, vol.30, 2017.

[14] R. Child, S. Gray, A. Radford, and I. Sutskever, \Generating long sequences with sparse transformers," arXiv preprint arXiv:1904.10509, 2019.

[15] N. Kitaev, L. Kaiser, and A. Levskaya, \Reformer: The e cient transformer," arXiv preprint arXiv:2001.04451, 2020.

[16] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarbs, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L.J. Colwell, and A. Weller, \Rethinking attention with performers," CoRR, vol.abs/2009.14794, 2020.

[17] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, \Transformers are rnns: Fast autoregressive transformers with linear attention," International Conference on Machine Learning, pp.5156{5165, PMLR, 2020.

[18] X. Ma, X. Kong, S. Wang, C. Zhou, J. May, H. Ma, and L. Zettlemoyer, \Luna: Linear uni ed nested attention," Advances in Neural Information Processing Systems, vol.34, pp.2441{2453, 2021.

[19] K. Irie, I. Schlag, R. Csordas, and J. Schmidhuber, \Going beyond linear transformers with recurrent fast weight programmers," Advances in Neural Information Processing Systems, vol.34, pp.7703{7717, 2021.

[20] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, \E cient transformers: A survey," ACM Computing Surveys, vol.55, no.6, pp.1{28, 2022.

[21] S. Wang, B.Z. Li, M. Khabsa, H. Fang, and H. Ma, \Linformer: Self-attention with linear complexity," CoRR, vol.abs/2006.04768, 2020.

[22] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Re, \Flashattention: Fast and memory-e cient exact attention with io-awareness," Advances in Neural Information Processing Systems, vol.35, pp.16344{16359, 2022.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, \Language models are unsupervised multitask learners," 2019.

[24] H. Wang, S. Ma, L. Dong, S. Huang, D. Zhang, and F. Wei, \Deepnet: Scaling transformers to 1,000 layers," arXiv preprint arXiv:2203.00555, 2022.

[25] T. Salimans, A. Karpathy, X. Chen, and D.P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," arXiv preprint arXiv:1701.05517, 2017.
[26] P. Ramachandran, T.L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M.A. Hasegawa-Johnson, R.H. Campbell, and T.S. Huang, "Fast generation for convolutional autoregressive models," arXiv preprint arXiv:1704.06001, 2017.
[27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pp.234–241, Springer, 2015.
[28] D.A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," arXiv preprint arXiv:1511.07289, 2015.
[29] W. Shi, J. Caballero, F. Huszár, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," Proceedings of the IEEE conference on computer vision and pattern recognition, pp.1874–1883, 2016.
[30] J. Ho, E. Lohn, and P. Abbeel, "Compression with flows via local bits-back coding," Advances in Neural Information Processing Systems, vol.32, 2019.
[31] EMBL, "European nucleotide archive: Illumina hiseq 2000 paired end sequencing; gsm1080195: mouse oocyte 1; mus musculus; rna-seq."https://www.ebi.ac.uk/ena , 6 2016.
[32] O.I. Alomair, I.M. Brereton, M.T. Smith, G.J. Galloway, and N.D. Kurniawan, "In vivo high angular resolution diffusion-weighted imaging of mouse brain at 16.4 tesla," PloS one, vol.10, no.6, p.e0130133, 2015.
[33] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, "Parameterized machine learning for high-energy physics," arXiv preprint arXiv:1601.07913, 2016.
[34] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
[35] P. Deutsch, "Gzip file format specification version 4.3," tech. rep., 1996.
[36] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," arXiv preprint arXiv:1908.03265, 2019.
[37] A. Krizhevsky, G. Hinton, et al., "Learning multiple layers of features from tiny images," 2009.

Appendix A: Details of Masked Linear Attention

In this section, Masked Linear Attention reported in [17] is explained for the estimation of the computational and memory costs in the next section. The batch dimension and layer notations are omitted for simplicity. C_{att}, H, D and $M = C_{att}/H$ denote the number of the channel dimensions, heads, query dimensions, and value dimensions, respectively. Note that C_{att} is equivalent to C_{in} for Linear Transformer, while for SCBs, C_{att} is equivalent to $C_{out}/2$.

First, an input tensor $a \in \mathbb{R}^{C_{att} \times L}$ is projected to the queries $Q^{(h)} \in \mathbb{R}^{L \times D}$, the keys $K^{(h)} \in \mathbb{R}^{L \times D}$, and the values $V^{(h)} \in \mathbb{R}^{L \times M}$ by weight matrices $W_q^{(h)}, W_k^{(h)} \in \mathbb{R}^{C_{att} \times D}$ and $W_v^{(h)} \in \mathbb{R}^{C_{att} \times M}$ for each head $h = 0, \ldots, H-1$ as follows:

$$Q^{(h)} = a^T W_q^{(h)}$$
$$K^{(h)} = a^T W_k^{(h)}$$
$$V^{(h)} = a^T W_v^{(h)}. \quad \text{(A 1)}$$

From here, a subscript i is introduced to represent the i-th position in the context dimension (e.g., $Q_i^{(h)}$ is a vector whose shape is $\mathbb{R}^{D}$). Next, the attention memory $S_i^{(h)} \in \mathbb{R}^{D \times M}$ and the normalizer memory $Z_i^{(h)} \in \mathbb{R}^{D}$ is calculated as

$$S_i^{(h)} = \sum_{j=1}^{i} \phi(K_j^{(h)}) V_j^{(h)T} \quad \text{(A 2)}$$

$$Z_i^{(h)} = \sum_{j=1}^{i} \phi(K_j^{(h)}), \quad \text{(A 3)}$$

where $\phi(\cdot)$ is a function defined by $\phi(x) = \mathrm{elu}(x) + 1$. Note that $\phi(K_j^{(h)}) V_j^{(h)T}$ in Eq. (A 2) is an outer product, not a matrix multiplication.

After that, the scaled dot-product attentions $A_i^{(h)} \in \mathbb{R}^{M}$ are calculated for each head and the self attention $A_i \in \mathbb{R}^{C_{att}}$ is projected by a weight matrix $W \in \mathbb{R}^{C_{att} \times C_{att}}$ as

$$A_i^{(h)} = V_i'^{(h)} = \frac{\phi(Q_i^{(h)})^T S_i^{(h)}}{\phi(Q_i^{(h)})^T Z_i^{(h)}} \quad \text{(A 4)}$$
$$A_{cat} = \mathrm{cat}(A_i^{(0)}, \ldots, A_i^{(H-1)})$$
$$A_i = W A_{cat}, \quad \text{(A 5)}$$

where $V_i'^{(h)}$ represents the updated values.

In the case of SCBs, the short-cut path tensor $\hat{a}$ is calculated as

$$\hat{a}_i = A_i + a_i. \quad \text{(A 6)}$$

Appendix B: Details of Computational and Memory Cost Estimations

B.1 Computational costs

Operations in the Masked Linear Attention and LinearFromConv are provided in Table A 1 and Table A2, respectively.

Table A 1 Computational costs of Masked Linear Attention.

Operation	Details	Comp. costs [FLOPs/bit/layer]	Note
$a^T W_q^{(h)}$	$\mathbb{R}^{C_{att}} \to \mathbb{R}^{D}$ (H times)	$C_{att} D H$	Eq. (A 1)
$a^T W_k^{(h)}$	$\mathbb{R}^{C_{att}} \to \mathbb{R}^{D}$ (H times)	$C_{att} D H$	Eq. (A 1)
$a^T W_v^{(h)}$	$\mathbb{R}^{C_{att}} \to \mathbb{R}^{M}$ (H times)	$C_{att} M H$	Eq. (A 1)
$\phi(K_j^{(h)}) V_j^{(h)T}$	$\mathbb{R}^{D} \to \mathbb{R}^{D \times M}$ (H times)	$D M H$	Eq. (A 2)
$\phi(Q_i^{(h)})^T S_i^{(h)}$	$\mathbb{R}^{D} \to \mathbb{R}^{M}$ (H times)	$D M H$	Eq. (A 4)
$W A_{cat}$	$\mathbb{R}^{C_{att}} \to \mathbb{R}^{C_{att}}$	C_{att}^2	Eq. (A 5)

Under the conditions used in the main text, the parameters shown in Table A 1 and Table A2 can be described by

Table A 2 Computational costs of LineaFromConv.

Operation	Details	Comp. costs [FLOPs/bit/layer]
LinearFromConv	R¹ in° R¹ in° out	in out

only using in and as follows:

$$
\begin{aligned}
{}^{1LT^o} &= {}_{in}\bullet \\
" \, {}^{1LT^o} &= {}_{in}\bullet \\
{}^{1LT^o}_{att} &= {}_{in} \\
{}^{1SCB^o} &= {}_{in}\bullet 2 \\
" \, {}^{1SCB^o} &= {}_{in}\bullet 2 \\
{}^{1SCB^o}_{att} &= {}_{in}\bullet 2 \\
{}_{out} &= {}_{in}
\end{aligned} \tag{A 7}
$$

First, we explain the computational costs of the Linear Transformer[¹LT°. In addition to the operations provided in Table A1, there is a FeedForward operation in each layer. The FeedForward operation consists of R ¹LT° att R ¹LT° att 4 ¹LT° att and R⁴ ¹LT° att R⁴ ¹LT° att ¹LT° att, which is equivalent to 2 4¹ ¹LT° att o2 FLOPs per bit per layer. From the above[¹LT° is calculated as

$$
\begin{aligned}
[\,{}^{1LT^o} = \,&»2 \;{}^{1LT^o}_{att} \;{}^{1LT^o} \;¸\; {}^{1LT^o}_{att}" \;{}^{1LT^o} \\
&¸\, 2 \;{}^{1LT^o}" \;{}^{1LT^o} \;¸\, 1 \;{}^{1LT^o}_{att}o2 \\
&¸\, 2 \;\; 4^1 \;{}^{1LT^o}_{att}o2¼ \;»FLOPs\, bit¼
\end{aligned} \tag{A 8}
$$

By applying Eq. (A7), [¹LT° is summarized as

$$
[\,{}^{1LT^o} = 12 \,¸\, \frac{2}{} \;{}^{2}_{in}! \;»FLOPs\, bit¼ \tag{A 9}
$$

Next, we explain the computational costs of SCBs [¹SCB°. The UB process consists of just a LinearFromConv while the DB process consists of a LinearFromConv and a Masked Linear Attention. Additionally, th layer is computed 1 2 times per bit in SCBs. From the above, [¹SCB° is calculated as

$$
\begin{aligned}
[\,{}^{1SCB^o} = \,&»2 \;{}^{1LT^o}_{att} \;{}^{1LT^o} \;¸\; {}^{1LT^o}_{att}" \;{}^{1LT^o} \\
&¸\, 2 \;{}^{1LT^o}" \;{}^{1LT^o} \;¸\, 1 \;{}^{1LT^o}_{att}o2¼ \\
&1 \,¸\, \frac{1}{2} \,¸\, \frac{1}{2! \bullet 2 \; 1} \\
&¸\, 1 \;{}_{in} \;{}_{out}{}^{o} \;\; 2 \;\; 1 \,¸\, \frac{1}{2} \,¸\, \frac{1}{2! \bullet 2 \; 1} \\
&»FLOPs\, bit¼
\end{aligned} \tag{A 10}
$$

By applying Eq. (A7), [¹SCB° is summarized as

$$
\begin{aligned}
[\,{}^{1SCB^o} = \,&1 \,¸\, \frac{1}{2} \,¸\, 2 \quad 2 \;\frac{1}{2! \bullet 2 \; 1} \;{}^{2}_{in} \\
&»FLOPs\, bit¼
\end{aligned} \tag{A 11}
$$

Lastly, we explain the computational costs of dilated convolutions[¹DC°. Since each layer consists of just a LinearFromConv, [¹DC° is calculated as

$$
\begin{aligned}
[\,{}^{1DC^o} &= \;{}_{in} \;{}_{out} \;! \\
&= \;{}^{2}_{in}! \;»FLOPs\, bit¼
\end{aligned} \tag{A 12}
$$

B.2 Memory costs

The required memory for Masked Linear Attention, SCBs, and dilated convolutions is listed in Table A3, Table A4, and Table A5, respectively.

Table A 3 Memory costs of Masked Linear Attention.

Description	Shape	Requried memory [dim/batch/layer]
Attention memory $S_8^{1\;o}$	R " (pcs.)	"
Normalizer memory $Z_8^{1\;o}$	R (pcs.)	

Table A 4 Memory costs of the cached results in SCBs.

Block	Description	Shape	Requried memory [dim/batch/layer]
DB	c_X	R in	in
	c_S	R out•2	out•2
	$s^{1;o}$	R out•2	out•2
UB	c_{X1}	R in•2	in•2
	c_{X2}	R in•2	in•2
	c_S	R in•2	in•2

Table A 5 Memory costs of dilated convolutions.

Description	Shape	Requried memory [dim/batch/layer]
Caluculated results of each layer	in (2 pcs.)	in 2

Similar to the previous section, the memory costs (elements per dim) of the Linear Transformer V^{1LT^o}, SCBs V^{1SCB^o}, and the dilated convolutions V^{1DC^o} is calculated as follows:

$$
\begin{aligned}
V^{1LT^o} &= 1 \;{}^{1LT^o}" \;{}^{1LT^o} \;! \;¸\; {}^{1LT^o} \;o \;! \\
&= \frac{{}^{2}_{in}}{} \,¸\, {}_{in} \;! \;»dim\bullet batch¼
\end{aligned} \tag{A 13}
$$

$$
\begin{aligned}
V^{1SCB^o} &= 1 \;{}^{1SCB^o}" \;{}^{1SCB^o} \;¸\; {}^{1SCB^o} \;o \;\frac{!}{2} \\
&¸\; {}_{in} \,¸\, 2\frac{out}{2} \;\frac{!}{2} \\
&¸\; 3\frac{in}{2} \;\frac{!}{2} \;! \\
&= \frac{{}^{2}_{in}}{8} \,¸\, 2 \;{}_{in} \;! \;»dim\bullet batch¼
\end{aligned} \tag{A 14}
$$

Fig. A 1 Physics (HEPMASS) dataset analysis.

$$\begin{aligned} V^{(DC)} &= C_{out}(2^0 + 2^1 + \cdots + 2^{L-1}) \\ &= C_{in}(2^L - 1) \times \mathrm{dim} \cdot \mathrm{batch}, \end{aligned} \quad \text{(A 15)}$$

Note that in $V^{(SCB)}$, the memory costs per layer in DBs ($L/2$ layers) and in UBs ($L/2$ layers) differ since only DBs have the linear attention process. And note that in $V^{(DC)}$, the reason this model requires a large amount of memory is that it requires 2^{l-1} cached values in lth layer and the kernel size $k = 2$. In this model, the dilation size of the lth layer is 2^{l-1} and therefore refers to 2^{l-1} previous values. As shown above, the memory cost increases exponentially with the number of layers.

Appendix C: Analysis of Prediction Accuracy

In the Physics dataset, SCBs showed singularly higher prediction accuracy than Linear Transformers. We therefore analyzed the prediction accuracy in the Physics dataset, which is csv files consist of floating-point data. Figure 8 shows the input data sequence in text (horizontal axis) and theoretical compression ratio (vertical axis) in the Physics dataset. For visibility, bitwise compression ratios are averaged into bytes. The latter half of the floating point data beyond the number of significant digits of the floating point formed a pattern that appeared in common with other data points, indicating that SCB was able to learn longer patterns than Linear Transformers, resulting in improved prediction accuracy, i.e., a higher theoretical compression ratio.

Appendix D: Channel size dependency of accuracy, computational cost, and memory cost

The channel size C_{in}, C_{out} affect the accuracy, the computational cost, and the memory cost. We conducted experiments by varying the channel size and summarized the results in Table A 6. By choosing an appropriate channel size, it is possible to adjust model size to achieve a desired accuracy or costs.

Table A 6 Channel size dependency of accuracy, computational cost, and memory cost (values in parentheses are standard deviations).

Channel size	128	256	512
Bpd (Genomics)	0.225 (0.004)	0.217 (0.006)	0.207 (0.002)
Computational cost [MFLOPs/dim]	0.2	0.7	2.7
Memory cost [dim/batch]	1.0E+04	3.1E+04	1.0E+05

Hiroaki Akutsu received his M.Eng. and Dr.Eng. degrees from Waseda University in 2005 and 2017, respectively. He now with Hitachi, Ltd. Research & Development Group, Data Storage Research Dept. working as a Principal Researcher. He received the Best Paper Award at IEEE VCIP 2021. His interest is on data compression, image and video coding, neural network, computing architecture and data storage systems. He is a member of the IEICE and IPSJ.

Ko Arai received his M.Eng. degree from the University of Tokyo in 2021. He is now with Hitachi, Ltd. Research & Development Group, Data Storage Dept. .