PAPER

A Hardware Efficient Reservoir Computing System Using Cellular Automata and Ensemble Bloom Filter

Dehua LIANG[†a)], ***Nonmember***, **Jun SHIOMI**[†], **Noriyuki MIURA**[†], **Masanori HASHIMOTO**[††], ***and*** **Hiromitsu AWANO**[††], ***Members***

SUMMARY Reservoir computing (RC) is an attractive alternative to machine learning models owing to its computationally inexpensive training process and simplicity. In this work, we propose *EnsembleBloomCA*, which utilizes cellular automata (CA) and an ensemble Bloom filter to organize an RC system. In contrast to most existing RC systems, *EnsembleBloomCA* eliminates all floating-point calculation and integer multiplication. *EnsembleBloomCA* adopts CA as the reservoir in the RC system because it can be implemented using only binary operations and is thus energy efficient. The rich pattern dynamics created by CA can map the original input into a high-dimensional space and provide more features for the classifier. Utilizing an ensemble Bloom filter as the classifier, the features provided by the reservoir can be effectively memorized. Our experiment revealed that applying the ensemble mechanism to the Bloom filter resulted in a significant reduction in memory cost during the inference phase. In comparison with *Bloom WiSARD*, one of the state-of-the-art reference work, the *EnsembleBloomCA* model achieves a 43× reduction in memory cost while maintaining the same accuracy. Our hardware implementation also demonstrated that *EnsembleBloomCA* achieved over 23× and 8.5× reductions in area and power, respectively.

key words: *EnsembleBloomCA, reservoir computing, cellular automata, Ensemble Bloom Filter*

1. Introduction

With the increasing scale of deep neural networks (DNNs), most portable and wearable devices are becoming unable to handle the large memory consumption and computing demand in both the training and inference phases. For example, AlexNet [1] requires 249MB of inference memory and performs 1.5B high precision operations to classify one image. Even applying the hardware-friendly implementation techniques to get the Binarized Neural Networks (BNNs) [2] or XNOR-Networks [3], still require expensive computation costs due to the floating-point calculation and backpropagation algorithm during the training. Most of the small edge devices do not have sufficient computing power to accomplish such sophisticated algorithms. Hence, it is crucial to meet the rising demand for more computationally efficient models.

Manuscript received September 29, 2021.
Manuscript revised February 17, 2022.
Manuscript publicized April 8, 2022.

[†]The authors are with Department of Information Systems Engineering, Graduate School of Information Science and Technology, Osaka University, 565–0871 Japan.
[††]The authors are with Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, 606–8501 Japan.
a) E-mail: d-liang@ist.osaka-u.ac.jp
DOI: 10.1587/transinf.2021EDP7203

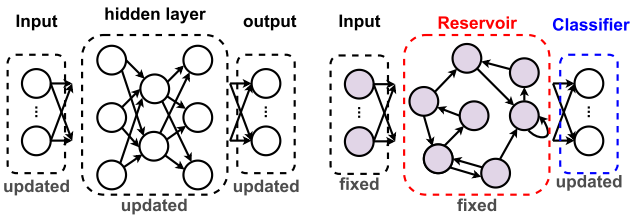


Fig. 1 Conventional DNN and reservoir computing.

Reservoir computing (RC) is a promising alternative for drastically reducing the computational burden of machine learning methods. Figure 1 shows a comparison between conventional DNN models and RC systems. The most critical advantage of RC is that only some of the parameters are trained, while the rest can be fixed. Owing to this unique feature, RC can be implemented with limited hardware resources, that is, fixed weights can be realized using hardwired logic. The standard RC architecture generally consists of a *reservoir* and *classifier*. All the input signals are given to a reservoir, which is often constructed by a recurrent neural network (RNN) whose synaptic weights are randomly initialized [4]. After being fed into the fixed and nonlinear pattern dynamic reservoir, these input signals are mapped into a higher-dimensional feature space. Finally, the output of RC is obtained using the trainable linear layer.

In comparison with the conventional CNNs with hardware implementation techniques e.g., BNNs and XOR Net, the design strategy of RC systems effectively avoid the use of complex training method in the reservoir part, and thus the learning process is simplified to a classical regression problem. Because of its simplicity and low computational cost in both the training and inference phases, RC systems have been successfully applied in many different fields, such as image recognition and robot control [5]. However, the frequent use of floating-point (FP) arithmetic found in most existing RNN models makes the implementation of RC systems on hardware challenging.

To reduce the massive usage of arithmetic units, the use of cellular automata (CA) has been proposed as a promising alternative to reservoirs [6]. A CA consists of multiple cells aligned in a one-dimensional array, where each cell takes two possible discrete states (“1” or “0”) and evolves in discrete time steps. This evolution process is guided by specific rules and interactions between the nearest neighbors. With rich pattern dynamics, CA is very well suited to the hard-

ware implementation of reservoir structures.

Hyperdimensional (HD) computing is also considered as a feasible way to implement pattern recognition systems. HD computing uses a large number of hypervectors, that is, binarized high-dimensional vectors that can be combined using well-defined vector space operations. The first step in HD computing is to map data points from the original domain to a high-dimensional space. In training, HD combines the mapped hypervectors to generate a hypervector representing each class. The utilization of these hypervectors aims to map the input data into a high-dimensional format, an approach which is similar to the idea of the reservoir structure in RC systems. The classification task at the inference phase is performed by checking the similarity of a mapped test hypervector with all trained classes. The mathematics governing the high-dimensional space enables HD models to be easily applied to many different fields, such as image processing [7]–[12], computer version [11]–[13], and language classification [14].

The Bloom filter (BF) is a space-efficient probabilistic data structure aimed at approximate member queries, that is, testing whether an element belongs to a given set [15]. This filter can be treated as a special case application of HD computing models. By adopting the BF, a weightless neural network has been proposed for image classification tasks, successfully eliminating costly FP calculations while maintaining fast single-pass training and yielding satisfactory accuracy performance [16], [17]. In contrast to conventional neural networks, the BF-based model is characterized by its simple implementation in both software and hardware. However, the large amount of memory required remains as the bottleneck for the application of BF, which makes it impractical when used for a portable device or hardware-resource-constrained system.

To further reduce the memory footprint, we propose a novel RC model: *EnsembleBloomCA*. Similar to the *ReCA* proposed in [18], *EnsembleBloomCA* adopts CA as the reservoir. The uniqueness of *EnsembleBloomCA* lies in the utilization of the ensemble Bloom filter as a classifier, which can alleviate the pollution of Bloom filters, even when memory capacity is limited, and thus contribute to the significant reduction of memory cost. The main contributions of our model are the following:

- Eliminating all floating-point calculation and integer multiplication, which makes *EnsembleBloomCA* suitable for hardware implementation.
- Achieving 43× memory reduction during inference in comparison with [16] without hurting the accuracy.
- Achieving reductions of over 23× and 8.5× in area and power consumption, respectively.

The remainder of this paper is organized as follows. Section 2 summarizes related research. Then, in Sect. 3, the preliminaries required to introduce *EnsembleBloomCA* are provided. Section 4 describes the proposed method, *EnsembleBloomCA*, and ensemble BF, followed by the details of our experiment and the evaluation results in terms of software and hardware in Sect. 5). Finally, concluding remarks are provided in Sect. 6.

2. Reservoir Computing in Digital Circuit

2.1 Basics of Reservoir Computing

Reservoir computing (RC) circumvents the difficulty of learning a number of RNN parameters, which has been successfully applied in numerous fields such as robot control [19] and image/video processing [20]. More recently, echo state networks [21] and liquid state machines [22] have been proposed for the use in different research domains. These technologies are collectively referred to as RC because both have a component called a *reservoir* [23]. The fixed and nonlinear hidden layers are generally referred to as the *reservoir* part, which can map the original input data into a higher-dimensional feature space. Meanwhile, the output layer is considered as a *classifier*.

Compared with conventional neural networks, RC systems use fixed weights in the input and hidden layers and only modify the weights of the output layer in the training process [4]. Because most of the weights can be fixed during training and inference, they can be implemented using hardwired logic and thus do not require memory circuits. Hence, RC is considered to be a promising alternative to replace DNNs, the model size of which continues to increase exponentially [5]. The state of each node is updated over M steps according to a nonlinear mapping F, which aims to map the low-dimensional input signal into the high-dimensional feature space as follows:

$$x_i(k) = F[\boldsymbol{x}(k-1), \boldsymbol{u}(k-1)]. \tag{1}$$

Here, $x_i(k)$ is the state of the i-th node at time step k, where $i \in \{1, 2, \cdots, N\}$ and $k \in \{1, 2, \cdots, M\}$. $\boldsymbol{x}(k) = \{x_1(k), x_2(k), \cdots, x_N(k)\}$ is the N-dimensional node state vector. Further, $\boldsymbol{u}(k) = \{u_1(k), u_2(k), \cdots, u_N(k)\}$ is the N-dimensional input vector, where $u_i(k)$ is the input to the i-th node at step k. As Eq. (1) shows, the state of node $x_i(k)$ depends on the previous state $x_i(k-1)$ and input $u_i(k-1)$.

Note again that the weights in the reservoir, that is, the nonlinear mapping F in Eq. (1), can be fixed during training and inference. Hence, the reservoir can be implemented using hardwired logic, which contributes to a reduction in hardware resources. However, the nonlinear mapping F still requires floating-point computations, which may lead to constraints when considering the implementation of RC in portable devices. To eliminate floating-point computations in the reservoir, several studies [6], [18], [24] have proposed exploiting cellular automata (CA) as an alternative to traditional reservoirs, which are summarized in the following Sect. 2.2.1.

2.2 Suitable Components for Hardware Implementation

The key idea of RC is to may the input to a higher-dimensional space to facilitate the classification. As the

basic components of the RC system, various methods have been proposed to perform the *reservoir* and *classifier*. The following techniques are commonly used because they are considered suitable for hardware implementation.

2.2.1 Cellular Automata

A cellular automaton is a discrete computational model consisting of a regular grid of cells, each in one of a finite number of states. The state of an individual cell evolves in time according to a fixed rule, depending on the current state and the states of its neighbors. CA governed by certain rules have been proven to be computationally universal, that is, capable of simulating a Turing machine [25].

Both [18], [24] perform exhaustive studies of the performance of different CA rules when applied to pattern recognition of time-independent input signals using an RC scheme. For [24], the authors evaluate the model on a 5-bit task, which is insufficient for the complicated application field. For [18], they select the most accurate CA rules to represent the reservoir structure, which can easily be reproduced using a set of XOR gates and shift registers. They achieve a high-performance alternative for RC hardware implementation in terms of circuit area, power, and system accuracy, which can be considered as a low-cost method to implement fast pattern recognition digital circuits. However, this model exploits the softmax function as its classifier, which still requires FP calculations; hence, it is not suitable for implementation on resource-constrained devices. For [6], authors utilize a random forest algorithm as the classifier, which avoids the costly FP calculation during the inference but also brings higher consumption for the training.

2.2.2 Hyper-Dimensional Computing

HD computing is another popular computing paradigm that emulates the activity of neurons in a high-dimensional space, an approach which is similar to the non-linear mapping idea of reservoir structure in RC systems. The main difference is that most reservoir structures accomplish nonlinear mapping via specific computations or operations, whereas HD computing requires reading hypervectors from pre-stored memory. However, HD models also incur a huge memory cost, which also limits the application of HD models. For further applications, there are three mainstream methods for solving these problems in most existing HD computing models.

The first method is to adopt emerging nanoscale resistive memory or memristive devices for storage. Compared with SRAM/DRAM, HD computing has recently been implemented at scale for multi-class classification tasks using emerging phase-change memory devices. In [26], [27], 3D vertical resistive random access memory (ReRAM) devices were used to perform individual operations for HD computing. In another study, a carbon nanotube field-effect transistor-based logic layer was integrated into ReRAMs for more efficient memory implementation [28]. In [29], the authors proposed a complete integrated in-memory HD computing system in which all HD computing operations are implemented on two planar memristive crossbar engines together with peripheral digital CMOS circuits. For the same scale of HD computing models (memory cost), the hardware efficiency was shown to improve when using the ReRAM in these studies. While emerging devices have the potential to open up new possibilities in HD computing, their incompatibility with CMOS circuits still makes it difficult to realize large-scale systems.

The second general idea is replacing binary operations with costly FP calculations. The systems proposed in [7]–[12] need to map the data points to hypervectors with non-binary elements. This means that to perform even a single addition between hypervectors, HD needs to compute thousands (e.g., 10,000) of FP operations. Moreover, in HD computing, training is achieved by the accumulation of several hypervectors, which makes the training operations very expensive.

The third idea is to adopt iterative training for model optimization, which does not require FP calculations [7]–[9], [11]–[13]. After mapping all the training data points into high-dimensional binary vectors, these hypervectors are stored in associative memory. Utilizing the labeled training data, the similarity between the encoded hypervectors and stored hypervectors of each class can be measured. The HD models are then adjusted and optimized iteratively according to the correctness of the classification results. By employing such a gradient descent, the error rate of the HD model can be significantly reduced. The disadvantage of this training strategy is that it requires tens of iterations to adjust the model, which leads to a long training time and expensive computational cost in comparison with single-pass training.

2.2.3 Bloom Filter

Bloom filters (BFs) are probabilistic data structures that represent a set as a small bit array allowing the occurrences of false positives, i.e., in a Bloom filter, an element can be incorrectly classified as a member of a set when it is not. Such memory-oriented classifiers for pattern recognition are typically very simple and can be easily implemented in hardware and software.

In [17], *Bloom WiSARD* was presented as an optimized framework that utilizes the BFs in a memory-segment way. Compared with the standard BFs, this model significantly reduces the memory requirement to some extent at the cost of allowing false positives and shows practically useful performance in image recognition tasks. The elimination of FP calculation and fast single-pass training are important advantages of *Bloom WiSARD* in terms of hardware efficiency. However, a large amount of memory required remains a key bottleneck. Take the MNIST classification task [30] as an example, when the false positive rate is 10%, the memory requirement is over 800MB for the inference implementation. It makes the adoption of *Bloom WiSARD* impractical for use in portable devices or memory-constrained systems.

3. Preliminary

3.1 Cellular Automata (CA)

Because CA provides a simple method to map the input into a high-dimensional space, it is useful as a hardware reservoir. Therefore, we adopt CA as a reservoir in the proposed method, similar to the existing methods [18]. Elementary cellular automata (ECA) is the simplest class of 1-dimensional CA [31], where each cell takes binary states, i.e., either "1" or "0." The updated state of a cell is determined by three cells, i.e., the cell and two neighboring cells, and hence, the time evolution of cell states can be written as

$$x_i(k) = F[x_{i-1}(k-1), x_i(k-1), x_{i+1}(k-1)]. \tag{2}$$

There are $2^{2^3} = 256$ possible evolution rules in total which can be labeled from Rule 0 to Rule 255. Figure 2 is an example of ECA Rule 90.

3.2 Cellular Automata Applied to Reservoir Computing

For the image classification tasks, assume the image u as the input data of the RC system. To apply CA to RC, the internal nodes and inputs should be converted into a binary format. [18] proposed the use of thermometer encoding, where an n-bit integer value is converted into a 2^n-bit binary string as follows:

$$u^{(l)} = \begin{cases} 1, & \frac{R \cdot l}{d} < u, \\ 0, & \frac{R \cdot l}{d} \geq u, \end{cases} \tag{3}$$

where d is the length of the binary data, R is the range of intensity for each pixel, u is the original input image with decimal data, and $u^{(l)}$ represents the l-th channel of the binary input data. For each pixel, the vector is initialized with d bits of 0s. The bits ranging from the most significant bit (MSB) to the first bit with a threshold higher than the pixel value are changed to 1. All the integer values $u \in [0, R]$. Thus, the thresholds of thermometer encoding can be obtained by dividing the pixel space R into d parts.

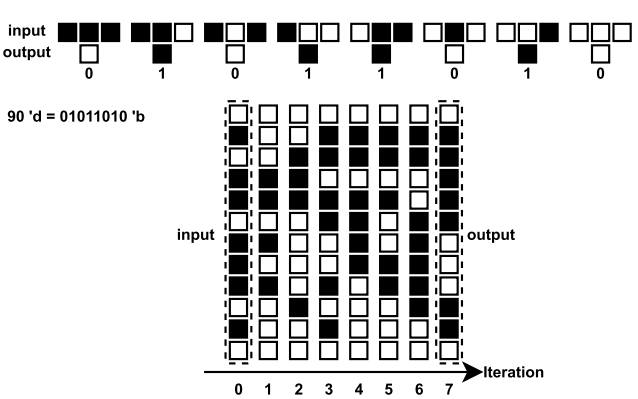


Fig. 2 Cellular automata evolution *Rule*90.

This encoding mechanism has been proven to significantly increase the error tolerance of neural networks, especially in terms of constructing adversarial samples [32]. Meanwhile, we note that there is another advantage of our model for hardware implementation, which benefits from thermometer encoding and is rarely noticed. This binary encoding mechanism can implement the max-pooling function utilizing only bitwise OR gates. This hardware friendliness decreases the energy consumption for both the training and inference phases.

After obtaining the binarized input signal, the ECA rule is applied to rows and columns independently with a fixed boundary condition. These two image results are combined with a bitwise XOR operation. This process is repeated for M iterations; for all images, rows and columns are independently iterated over with the same ECA rule, and the resulting vectors are combined with a bitwise XOR operation.

After decomposing the original images into d binary channels, we obtain the l-th channel of the input signal $u^{(l)}$, where $l \in [1, d]$. There is no communication between binary channels. The same ECA rule is repeated M times in total.

Let g^k be the function g applied k times, and let $x^{(l)}(k)$ be a Boolean time-dependent image, which can be expressed as

$$x^{(l)}(k) = g^k(u^{(l)}) = \begin{cases} u^{(l)}, & k = 0, \\ g^1(x^{(l)}(k-1)), & k > 0. \end{cases} \tag{4}$$

We can obtain the state of the reservoir in the i-th position, k-th iteration, and l-th binary channel as $x_i^{(l)}(k)$, where $i \in \{1, 2, \ldots, N\}$, $k \in \{1, 2, \ldots, M\}$ and $l \in \{1, 2, \ldots, d\}$. Thus,

$$x^{(l)}(k) = [x_1^{(l)}(k), x_2^{(l)}(k), \ldots, x_N^{(l)}(k)]. \tag{5}$$

The images are iterated over independently by rows and columns. We define $x_{row}^{(l)}(k)$ as the result of iterating images by rows, that is, the state of each updated cell is determined by two horizontal neighboring cells and the cell itself. Similarly, $x_{col}^{(l)}(k)$ represents the results of iterating over images by columns. The vectors $x_{feature}^{(l)}(k)$ are obtained by combining $x_{row}^{(l)}(k)$ and $x_{col}^{(l)}(k)$ with an XOR operation:

$$x_{feature}^{(l)}(k) = x_{row}^{(l)}(k) \oplus x_{col}^{(l)}(k). \tag{6}$$

The $x_{feature}(k)$ is defined as

$$x_{feature}(k) = \sum_{l=0}^{d-1} x_{feature}^{(l)}(k), \tag{7}$$

where $k \in [0, M]$. Subsequently, we apply a max-pooling layer to improve the generalization of the network and reduce the weights in the classifier. Because the internal states are binarized, the CA is suitable for digital hardware implementation. However, the classifier still requires a softmax operation in [18], which should be eliminated for ease of implementation on resource-constrained devices.

3.3 Application of Bloom Filter

To completely eliminate the FP calculations, we propose the exploitation of the Bloom filter (BF) to construct the classifier in the RC system. The Bloom filter is considered as a space-efficient probabilistic data structure, which aims to test whether an unknown item is a member of the given set [15]. As the term "probabilistic" suggests, the query result may contain errors, i.e., the Bloom filter returns either "possibly in the set" or "definitely not in the set." From a neural processing point of view, BFs are a special case of an artificial neural network with two layers (input and output), where each position in a filter is implemented as a binary neuron. Such a network does not have interneuronal connections; that is, output neurons (positions of the filter) have only individual connections with themselves and the corresponding input neurons. The most significant advantage of the Bloom filter is its memory space efficiency over other data structures, which is suitable for error-resilient applications such as machine learning [33].

The standard BF allows the addition of new elements to the filter and is characterized by a perfect true positive rate (i.e., 1), but a nonzero false positive rate. The false positive rate depends on the number of elements to be stored in the filter, as well as the filter parameters, including the number of hash functions and the size of the filter. In the BF model, the element is considered as an L-bit binarized vector, which represents the position within the Bloom filter. Thus, each Bloom filter contains 2^L bits. We define $B(index)$ as the $index$-th bit in the Bloom filter, where $index \in [0, 2^L - 1]$.

In the insertion phase, all the values in the Bloom filters are initially set to zero. Each training sample is inserted into the corresponding Bloom filter based on their labels. The value of the accessed bit $B(index)$ is set to one. In the query phase, the testing sample is sent to all the Bloom filters and returns the value of the accessed bit $B(index)$ in every Bloom filter. When it returns positive (logic "1"), the testing sample is considered as "possibly in this category." When it returns negative (logic "0"), the sample is judged as "definitely not in this category."

Figure 3 shows an example of Bloom filter operations with a 16-bit array and three hash functions. In the insertion operation, each element is mapped into three positions according to the three different hash functions (e.g., MurmurHash [34]). Then these corresponding hashed bits are set to 1. The query operation looks up the positions mapped from the input element, indicating whether it is a member of the set. As Fig. 3 shows, d is a false positive, as it was returned as a member of the set (only a, b and c were inserted).

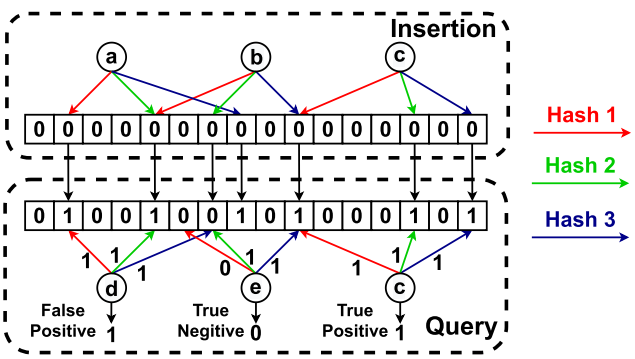


Fig. 3 Example of Bloom filter operations with a 16-bit array and three hash functions.

4. Proposed Method

4.1 *EnsembleBloomCA*

EnsembleBloomCA is a novel RC architecture comprising an ingenious combination of CA and an ensemble Bloom filter. Figure 4 shows the overview of *EnsembleBloomCA*, which consists of CA for extracting a high-dimensional binary feature vector from an input figure, and Bloom filters, each of which corresponds to a class label. Once an input image is provided, *EnsembleBloomCA* first extracts the binarized feature vector in the same manner as in Sect. 3.1. Then, the similarity between the extracted feature vector and each Bloom filter is computed and the class whose corresponding Bloom filter exhibits the maximum similarity is output as the model prediction. In the following, we detail the algorithm of *EnsembleBloomCA*, which exploits the benefits of CA and Bloom filters.

Training Phase: Initially, the Bloom filter classifier is blank, that is, all the values are set to zero (logic "0"). Then, the input image is fed into the CA part for feature extraction. Here, the evolution rule is applied M times to elevate the pattern dynamics that are ready for classification. In the next step, we extract patches of the CA output by applying a $W_{rec} \times H_{rec}$ size receptive field with a stride of L_{rec} and again apply a simple hash function to each extracted patch to obtain a binary feature vector. Details regarding the hash function are provided in Sect. 4.2. Finally, the extracted feature vector is fed into the Bloom filter, which is specified by the corresponding training label, with bitwise OR gates, and the training for this image is completed. For example, a training image labeled "5" is inserted into Bloom filter #5.

Inference Phase: Similar to the training phase, the input images are fed into the CA, followed by image patch extraction and application of the hash function to obtain the feature vector for the input image. Then, using counters and bitwise AND gates to calculate the similarity between the feature vector and Bloom filters, the Bloom filter with the highest response is chosen as a representative category for the testing image.

As can be seen, the similarity should reach the maximum value if the input pattern belongs to the corresponding category, which enables us to determine which class the unseen input pattern belongs to without using computationally expensive floating-point arithmetic. Although the pattern dynamics are extremely elevated by the CA reservoir, this also increases the number of Bloom filters in every category, which results in untenable memory usage during the

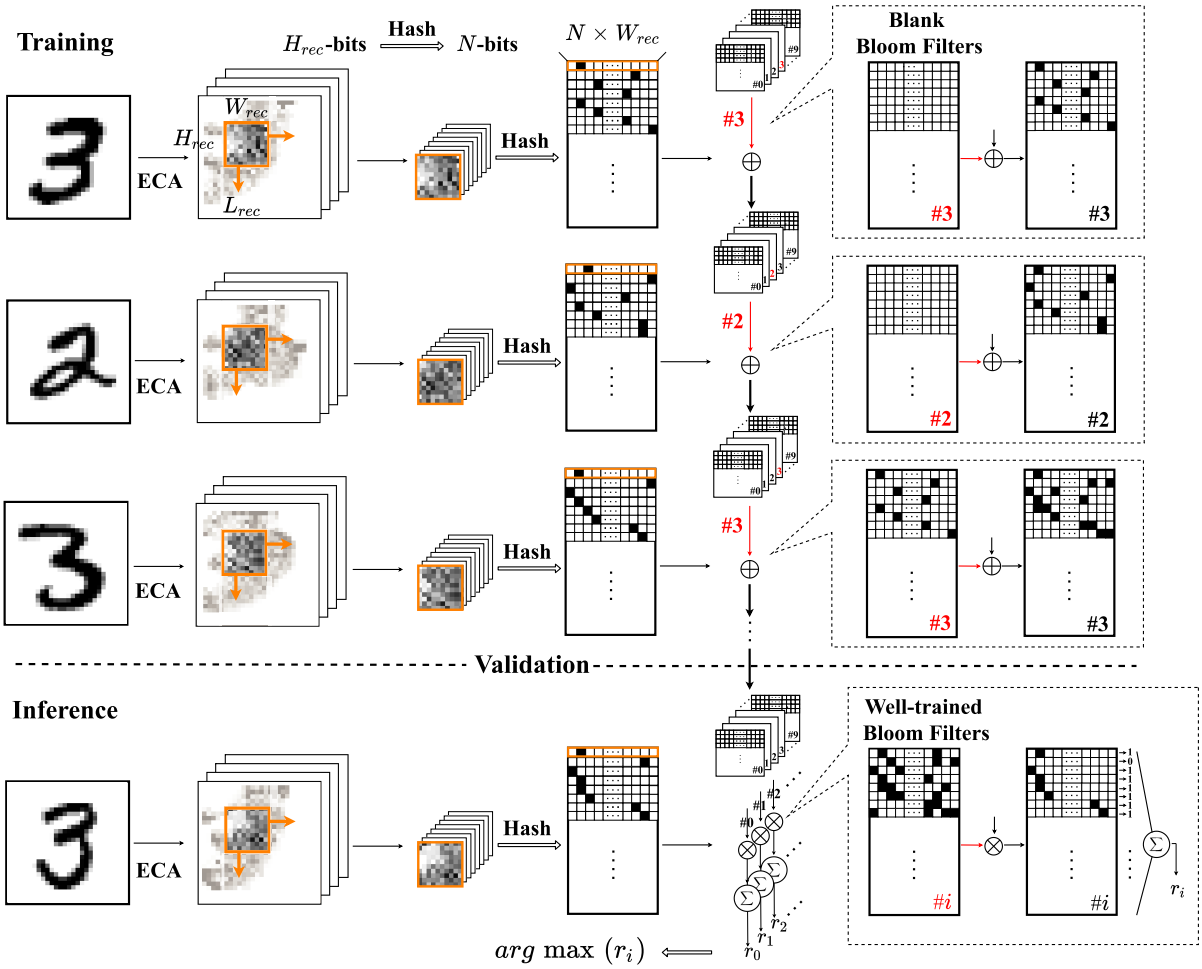


Fig. 4 Overview of *EnsembleBloomCA*.

inference phase. N_{sub}, which represents the number of samples inserted into each Bloom filter, has a significant correlation with the performance of the Bloom filter. The more elements are added to a single Bloom filter, the higher the probability of false positives (FPR) [35]. To optimize this data structure, we propose utilizing an ensemble Bloom filter as the classifier in our RC system.

4.2 Ensemble Bloom Filter

Ensemble learning is a machine learning paradigm in which multiple base learners are trained to solve the same problem. The generalization ability of an ensemble is usually much stronger than that of the base learners. The base learners are usually generated from training data using a base learning algorithm that can be a decision tree, neural network, or other type of machine learning algorithm [36].

Several algorithms are commonly used for ensemble learning, including bagging. In this method, training data subsets are randomly drawn from the entire training dataset. Each training data subset was used to train a classifier of the same type. Individual classifiers are then combined by taking a simple majority voting in the classifier or using relatively weak classifiers (such as decision stumps, an approach which constitutes a random forest classifier). Another popular method, boosting, also creates an ensemble classifier by resampling the data and then combining it through majority voting. However, in boosting, resampling is strategically geared to provide the most informative training data for each consecutive classifier [37].

In our case, we apply the bagging algorithm. The training dataset is divided into N_{sub} subsets and inserted into different Bloom filters as base learners. Then, the results of these base learners are summed up together, a process which can be considered as an ensemble Bloom filter classifier. The basic operations of the ensemble Bloom filter model involve adding elements to the corresponding set (insertion phase) and querying for element membership in the probabilistic set representation (query phase).

For the ensemble Bloom filter, a binarized pattern with $N \cdot L$ bits is split into N vectors of L bits. We define $B_n(i)$ as the i-th bit in the n-th mini Bloom filter, where $n \in \{1, 2, \ldots, N\}$. In our case, the patterns from the $W_{rec} \times H_{rec}$ size receptive field need to be memorized by a different ensemble Bloom filter. Similar to [16], each binarized pattern is split into W_{rec} and H_{rec} vectors in rows and columns.

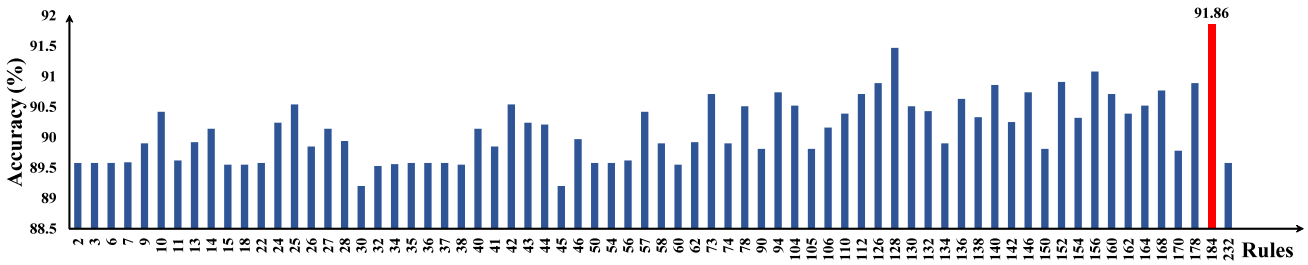


Fig. 5 Accuracy-ECA rules.

During the insertion phase, the vectors are inserted into the corresponding Bloom filters, which we call "mini Bloom filters." These vectors are considered as addresses within the mini Bloom filters, as well as the results of the hash function. In the query phase, if and only if all the values of accessed bit $B_n(i)$ in the mini Bloom filters are positive (logic "1"), this Bloom filter returns one. Otherwise, this Bloom filter returns zero. All the returned results of Bloom filters in each category are summed together and considered as the discriminator response r, which also represents the similarity between the testing sample and the corresponding category. The discriminator with the highest response r is chosen as the representative category.

After training each Bloom filter, we select those exhibiting good classification accuracy using validation samples. Similar to the training phase, the class labels of the validation images are predicted to evaluate the classification accuracy of each Bloom filter. Then, we select N_{inf} Bloom filters exhibiting the top N_{inf} performance rankings; these filters construct the classifier used in the inference phase.

The key idea behind using the Bloom filter as a classifier is to store the pattern information within the given set, which means that an excessive difference between patterns leads to the pollution of Bloom filters. Instead of using a single standard Bloom filter with high memory cost, utilizing Bloom filters in an ensemble can effectively improve the performance of image recognition tasks. Meanwhile, the number of well-trained Bloom filters in each Bloom filter pool decreases, which leads to a significant reduction in memory cost during the inference phase.

5. Experiment

5.1 Experiment Setup

In our experiment, we focused on the handwritten digit number classification task based on the MNIST dataset, which is a collection of 70k handwritten digits in grayscale format. This task is extensively used to compare the performance of many classification models by evaluating the performance of a machine learning algorithm [30]. Among 60k images, we randomly selected 55k images for training and 5k images for validation. The training images were used to populate Bloom filters, while the validation images were used to optimize the hyperparameters, such as the CA evolution rules or evolution times. The remaining 10k images were used for testing. We will make a comparison between the following three methods:

Bloom WiSARD is the baseline algorithm in [16]. It is an optimized application of standard BFs, which utilizes BFs in a memory-segment way. The input images belonging to the same category with the size of 28×28, are split into 28 rows, and individually stored into the 28 BFs during the training. When it comes to the inference phase, each BFs returns the query results as logic "0" or "1". The sum of these query results represents the response of the corresponding category. Hence, the class with the highest response is considered as the output of the classification.

Ensemble Bloom filter is a special case of our proposed method. For the comprehensive exploration, we also evaluate the performance of our model without utilizing CA. In this case, no reservoir architecture is applied and we directly use the proposed ensemble Bloom filter in Sect. 4.2 as the classifier, which can help us to evaluate the contribution of CA and ensemble Bloom filter individually.

EnsembleBloomCA is our proposed method. 55k training images were inserted into the ensemble Bloom filter as the training set. We then divided 55k training images into N_{sub} subsets for every category in order. The other 5k training images were used as the validation set for adjusting the ensemble Bloom filter classifier. Then, we utilized 10k testing images as the inference set to evaluate the performance of our approach.

In our experiment, $w = 28$, $h = 28$, $d = 16$, and $R = 256$. For the reservoir, CA maps the original input into a higher-dimensional space and obtains high-dimensional patterns. The max-pooling layer was selected to have a stride of two, a squared window of size two, and zero padding. A 5×5 receptive field was applied to every binary channel and iteration of the reservoir output, with a stride of 3. The software implementation was emulated in C++ to evaluate the performance of *EnsembleBloomCA*.

5.2 Optimization of ReCA

Although the reservoir using CA recreates a rich pattern for the *EnsembleBloomCA* model, extra features also require more memory resources and data transfers. In [18], an approach using only the 8th iteration to train the classifier was proposed. Every time the iterative patterns are obtained with a fixed CA evolution rule, the pattern is changed to some extent.

Considering the difference in contribution from every iterative pattern, we only choose the first iterative pattern which represents the original input data and the latest iterative pattern as features. This strategy can effectively refine the input feature for the ensemble Bloom filter and reduce the memory cost in both the training and inference phases.

5.3 Experiment Results

Using the *EnsembleBloomCA* model described in Sect. 4.1, we examined the performance of all existing CA evolution rules. The iteration count M is in the range from 1 to 24. As shown in Fig. 5, the RC system achieves different accuracy performances under different ECA rules. According to the hyperparameter optimization from the validation set, we adopt the ECA evolution Rule 184 and iteration count $M = 8$.

Figure 6 shows the classification accuracy as a function of the number of samples inserted into a single Bloom filter and memory cost required during training. The dashed horizontal line shows the classification accuracy of [16], which is the baseline of our work. When the training memory cost is lower than 400 KB, the accuracy drops sharply. In these cases, an excessive number of feature vectors are inserted into the same Bloom filter, which pollutes the Bloom filter and affects the accuracy of the ensemble Bloom filter. When the training memory cost is up to 825 KB, the accuracy trend appears to be saturated.

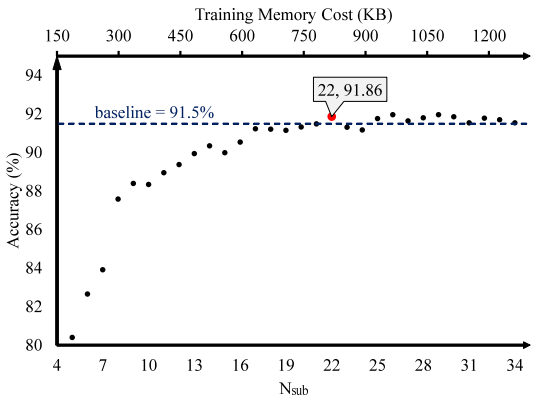


Fig. 6 Accuracy - N_{sub} / Training Memory Cost.

Table 1 Performance Comparison.

	Bloom WiSARD (baseline)	**Ensemble Bloom filter (proposed)**	***Ensemble--BloomCA* (proposed)**
Arithmetic	multiplication	addition	addition
Number of hash	3	1	1
Accuracy(%)	91.50	89.58	91.86
Inference Memory (KB)	819.05	18.75	18.75

Table 1 shows the results of several different models in the MNIST handwritten number classification task. The *EnsembleBloomCA* model achieved over $819.05/18.75 \approx 43\times$ memory reduction compared with the baseline while maintaining the same accuracy.

This reduction in memory cost mainly comes from the use of Bloom filters in an ensemble. Although this method also leads to a slight decrease in accuracy, this disadvantage can be overcome by using CA as the reservoir. The rich pattern dynamics recreated by CA can effectively provide more candidates for Bloom filter pools and improve accuracy from 89.58% to 91.86%, which also illustrates the impact of CA in our model.

Overall, in comparison with another state-of-the-art model, *Bloom WiSARD* [16], *EnsembleBloomCA* significantly reduced the memory cost for the inference phase. Meanwhile, the simplicity of the hash function avoids high computational costs, allowing for practical hardware implementation.

5.4 Hardware Implementation

The hardware architecture of the baseline *Bloom WiSARD* and our proposed *EnsembleBloomCA* were designed using SystemVerilog. We used *Synopsys Design Compiler* to synthesize and report the area and power consumption of our approach in a 65-nm ASIC flow. The hardware costs of the memory part were individually simulated using *CACTI*, which is an integrated memory access time, area, leakage, and power model. In addition, the memory type was chosen to be the main memory in the 65-nm ASIC flow, which does not contain any tag array, and every access occurs at a page granularity. Table 2 shows the comparison between *EnsembleBloomCA* and [16] in terms of ASIC area and energy consumption. The maximum propagation delay of our model is 3.78 ns.

The memory takes $0.253/0.348 \approx 72.7\%$ of the area of *EnsembleBloomCA*, while the other circuits take 37.3%. In terms of power consumption, the memory portion only uses $34.5/155.9 \approx 22.13\%$, and the power consumption percentage of other circuits is increased to 77.87%. Although the

Table 2 Hardware Performance Comparison.

	Area (mm^2)			**Power consumption** [mW]			**Maximum propagation delay**
	Reservoir	**Memory**	**Total**	**Reservoir**	**Memory**	**Total**	[$10^{-9}s$]
Bloom WiSARD (baseline)	0.206	7.805	8.012	258.9	1059.8	1318.7	33.77
***EnsembleBloomCA* (proposed)**	0.095	0.253	0.348	121.4	34.5	155.9	3.78
Reduction	2.2×	30.8×	23.0×	2.13×	30.7×	8.5×	8.9×

memory occupies almost half of the entire circuit area, the energy is mainly consumed by the non-memory part, that is, CA, owing to the high switching activity of the non-memory part. Overall, *EnsembleBloomCA* achieved over 23× and 8.5× reductions in area and power, respectively.

6. Conclusion

In this work, we propose a novel RC architecture, the *EnsembleBloomCA* model, which is a combination of a reservoir using CA and an ensemble Bloom filter classifier. By utilizing *EnsembleBloomCA*, we achieved a 43× reduction in memory cost for the inference phase while maintaining accuracy. Our hardware implementation also demonstrated that *EnsembleBloomCA* achieves over 23× and 8.5× reductions in area and power, respectively. The experimental results also illustrate the efficacy of using CA as a reservoir. Mapping the input signal into a higher-dimensional feature space, the rich dynamics recreated by CA can effectively improve the performance of the ensemble Bloom filter classifier. This model can completely eliminate expensive computational operations, such as floating-point calculation and integer multiplication. Owing to the simplicity of *EnsembleBloomCA*, our model shows promising potential for hardware implementation.

Acknowledgments

This work was partially supported by JST, PRESTO Grant No. JP-MJPR18M1, JSPS KAKENHI Grant No. 21H03409.

References

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," Proc. 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, Red Hook, NY, USA, p.1097–1105, Curran Associates Inc., 2012.

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," arXiv:1602:02830, 2016.

[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," Computer Vision – ECCV 2016, ed. B. Leibe, J. Matas, N. Sebe, and M. Welling, Cham, pp.525–542, Springer International Publishing, 2016.

[4] B. Schrauwen, D. Verstraeten, and J. Campenhout, "An overview of reservoir computing: theory, applications and implementations," European Symposium on Artificial Intelligence, pp.471–482, 2007.

[5] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," Comput. Sci. Rev., vol.3, no.3, pp.127–149, 2009.

[6] Á. López, J. Yu, and M. Hashimoto, "Low-cost reservoir computing using cellular automata and random forests," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–5, 2020.

[7] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," Proc. 56th Annual Design Automation Conference 2019, DAC'19, New York, NY, USA, Association for Computing Machinery, 2019.

[8] M. Imani, Y. Kim, M.S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," IEEE Cloud Computing (CLOUD), IEEE, IEEE, 08/2019 2019.

[9] K. Behnam, X. Hanyang, M. Justin, and R. Tajana, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," IEEE/ACM Design Automation and Test in Europe Conference (DATE), IEEE, IEEE, 2021.

[10] A. Hernandez-Cane, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," IEEE/ACM Design Automation and Test in Europe Conference (DATE), IEEE, IEEE, 2021.

[11] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," IEEE International Symposium on Field-Programmable Cusstom Computing Machines (FCCM), 2019.

[12] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp.1–6, 2018.

[13] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp.126–131, 2019.

[14] A. Rahimi, T.F. Wu, H. Li, J.M. Rabaey, H.S.P. Wong, M.M. Shulaker, and S. Mitra, "Hyperdimensional computing nanosystem," ArXiv, vol.abs/1811.09557, 2018.

[15] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol.13, no.7, pp.422–426, 1970.

[16] L. Santiago, L. Verona, F. Rangel, F. Firmino, D. Menasché, W. Caarls, M. Breternitz, S. Kundu, P. Lima, and F.M.G. França, "Weightless neural networks as memory segmented bloom filters," Neurocomputing, 2020.

[17] L.S. de Araújo, L. Verona, F. Rangel, F.F. de Faria, D. Menasché, W. Caarls, M. Breternitz, S. Kundu, P. Lima, and F. França, "Memory efficient weightless neural network using bloom filter," European Symposium on Artificial Neural Networks, pp.307–312, 2019.

[18] A. Morán, C.F. Frasser, and J.L. Rosselló, "Reservoir computing hardware with cellular automata," ArXiv, vol.abs/1806.04932, 2018.

[19] E.A. Antonelo and B. Schrauwen, "On learning navigation behaviors for small mobile robots with reservoir computing architectures," IEEE Trans. Neural Netw. Learn. Syst., vol.26, no.4, pp.763–780, 2015.

[20] A. Jalalvand, G.V. Wallendael, and R. Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," 2015 7th International Conference on Computational Intelligence, Communication Systems and Networks, pp.146–151, 2015.

[21] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," Neural Information Processing Systems, pp.609–616, 2002.

[22] T. Natschläger, W. Maass, and H. Markram, "The "liquid computer": A novel strategy for real-time computing on time series," Special issue on Foundations of Information Processing of TELEMATIK, vol.8, no.LNMC-ARTICLE-2002-005, pp.39–43, 2002.

[23] Y. Kume, S. Bian, and T. Sato, "A tuning-free hardware reservoir based on mosfet crossbar array for practical echo state network implementation," 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), pp.458–463, 2020.

[24] S. Nichele and M.S. Gundersen, "Reservoir computing using non-uniform binary cellular automata," Complex Systems, vol.26, no.3, pp.225–246, 2017.

[25] M. Cook, "Universality in elementary cellular automata," Complex Systems, vol.15, no.01, pp.1–40, 2004.

[26] H. Li, T.F. Wu, A. Rahimi, K.S. Li, M. Rusch, C.H. Lin, J.L. Hsu, M.M. Sabry, S.B. Eryilmaz, J. Sohn, W.C. Chiu, M.C. Chen,

T.T. Wu, J.M. Shieh, W.K. Yeh, J.M. Rabaey, S. Mitra, and H.S.P. Wong, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," 2016 IEEE International Electron Devices Meeting (IEDM), pp.16.1.1–16.1.4, 2016.

[27] H. Li, T.F. Wu, S. Mitra, and H.S.P. Wong, "Device-architecture co-design for hyperdimensional computing with 3d vertical resistive switching random access memory (3d vrram)," VLSI Technology, Systems and Application (VLSI-TSA), 2017 International Symposium on, IEEE, IEEE, 2017.

[28] T.F. Wu, H. Li, P.C. Huang, A. Rahimi, J.M. Rabaey, H.S.P. Wong, M.M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," 2018 IEEE International Solid - State Circuits Conference - (ISSCC), pp.492–494, 2018.

[29] G. Karunaratne, M.L. Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," Nature Electronics, vol.3, no.6, pp.327–337, June 2020.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol.86, no.11, pp.2278–2324, 1998.

[31] S. Wolfram, "New Kind of Science, Wolfram Media," 2002.

[32] J. Buckman, A. Roy, C. Raffel, and I.J. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," International Conference on Learning Representation, 1-22, 2018.

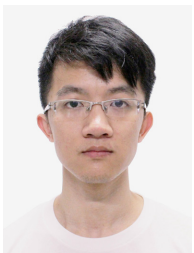
[33] S. Tarkoma, C.E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," IEEE Communications Surveys & Tutorials, vol.14, no.1, pp.131–155, 2012.

[34] Wikipedia contributors, "Murmurhash — Wikipedia, the free encyclopedia," 2021. [Online; accessed 13-January-2022].

[35] I. Aleksander, M. Gregorio, F. França, P. Lima, and H. Morton, "A brief introduction to weightless neural systems," European Symposium on Artificial Neural Networks, pp.299–305, 2009.

[36] Z.H. Zhou, Ensemble Learning, pp.270–273, Springer US, Boston, MA, 2009.

[37] R. Polikar, "Ensemble learning," Scholarpedia, vol.4, no.1, p.2776, 2009. revision #186077.



Dehua Liang received the B.E. degree in Microelectronic Science and Engineering from Xian Jiaotong University, China in 2018. Presently, he is a master course student at Department of Information Systems Engineering, Osaka University, Japan.

Jun Shiomi received the B.E. degree in electrical and electronics engineering, the M.E. degree in Informatics, and the Ph.D. degree in Informatics from Kyoto University, Kyoto, Japan, in 2014, 2016 and 2017, respectively. From 2016 to 2017, he was a Research Fellow of the Japan Society for the Promotion of Science. From December 2017 to March 2021, he was an Assistant Professor in the Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Japan. In April 2021, he joined Osaka University where he is currently an Associate Professor in the Graduate School of Information Science and Technology. His research interests include modeling and computer-aided design for low power and low voltage system-on-chips.

Noriyuki Miura received the B.S., M.S., and Ph.D. degrees in electrical engineering all from Keio University, Yokohama, Japan. From 2005 to 2008, he was a JSPS Research Fellow and since 2007 an Assistant Professor with Keio University, where he developed wireless interconnect technology for 3D integration. In 2012, he moved to Kobe University, Kobe, Japan, and became a Professor at Osaka University, Suita, Japan in 2020. Also, he was concurrently appointed as a JST PRESTO researcher, and now working on hardware security/safety and next-generation heterogeneous computing systems. Dr. Miura is currently serving as a Technical Program Committee (TPC) Member for A-SSCC and Symposium on VLSI Circuits. He served as the TPC Vice Chair of 2015 A-SSCC. He was a recipient of the Top ISSCC Paper Contributors 2004-2013, the IACR CHES Best Paper Award in 2014, the IEICE Suematsu Yasuharu Award in 2017, and the Marubun Research Encouragement Award in 2019.

Masanori Hashimoto received the B.E., M.E., and Ph.D. degrees in communications and computer engineering from Kyoto University, Kyoto, Japan, in 1997, 1999, and 2001, respectively. He is currently a Professor at the Graduate School of Informatics, Kyoto University. His current research interests include design for reliability, timing and power integrity analysis, reconfigurable computing, soft error characterization, and low-power circuit design. Dr. Hashimoto was on the technical program committees of international conferences including DAC, ICCAD, ITC, IRPS, Symposium on VLSI Circuits, ASP-DAC, and DATE. He serves/served as the Editor-in-Chief for Elsevier Microelectronics Reliability and an Associate Editor for IEEE TVLSI, TCAS-I, and ACM TODAES.

Hiromitsu Awano received the B.S., M.S. and Ph.D. degrees in Informatics from Kyoto University in 2010, 2012 and 2016 respectively. He was with Hitachi, Ltd., Tokyo, Japan in 2016, with the VLSI Design and Education Center, the University of Tokyo, Japan, from 2017 to 2018, and with the Graduate School of Information Science and Technology, Osaka University, Osaka, Japan, from 2019 to 2020. In 2020, he joined the Graduate School of Informatics, Kyoto University, Kyoto, Japan, where he is currently an associate professor. His research interests include CAD for VLSI design and hardware accelerator for machine learning. He was a research fellow of japan society for the promotion of science and a member of IEEE, IEICE, and IPSJ.