DNN-SNN Co-Learning for Sustainable Symbol Detection in 5G Systems on Loihi Chip

Shiya Liu, Yibin Liang, *Member, IEEE*, and Yang Yi, *Senior Member, IEEE*

***Abstract*—Performing symbol detection for multiple-input and multiple-output orthogonal frequency division multiplexing (MIMO-OFDM) systems is challenging and resource-consuming. In this paper, we present a liquid state machine (LSM), a type of reservoir computing based on spiking neural networks (SNNs), to achieve energy-efficient and sustainable symbol detection on the Loihi chip for MIMO-OFDM systems. SNNs are more biological-plausible and energy-efficient than conventional deep neural networks (DNN) but have lower performance in terms of accuracy. To enhance the accuracy of SNNs, we propose a knowledge distillation training algorithm called DNN-SNN co-learning, which employs a bi-directional learning path between a DNN and an SNN. Specifically, the knowledge from the output and intermediate layer of the DNN is transferred to the SNN, and we exploit a decoder to convert the spikes in the intermediate layers of an SNN into real numbers to enable communication between the DNN and the SNN. Through the bi-directional learning path, the SNN can mimic the behavior of the DNN by learning the knowledge from the DNN. Conversely, the DNN can better adapt itself to the SNN by using the knowledge from the SNN. We introduce a new loss function to enable knowledge distillation on regression tasks. Our LSM is implemented on Intel's Loihi neuromorphic chip, a specialized hardware platform for SNN models. The experimental results on symbol detection in MIMO-OFDM systems demonstrate that our LSM on the Loihi chip is more precise than conventional symbol detection algorithms. Also, the model consumes approximately 6 times less energy per sample than other quantized DNN-based models with comparable accuracy.**

***Index Terms*—Deep learning, deep neural network, knowledge distillation, machine learning, spiking neural network, sustainable MIMO symbol detection.**

I. Introduction

THE 5th generation (5G) mobile network [1], [2] interconnects everyone and everything together, such as machines and devices. The multi-Gbps data speeds, ultra-low latency, and high bandwidth brought by the 5G network enable new applications such as smart cities, smart factories, and autonomous vehicles. The driving force behind the 5G network is the utilization of orthogonal frequency division multiplexing (OFDM) in multiple-input multiple-output (MIMO) wireless channels [3], [4], [5], [6], [7]. Signal processing in OFDM-MIMO systems is challenging and resource-consuming in scenarios such as the massive MIMO architecture for millimeter-wave bands with high nonlinear distortion in RF components. With the development of deep neural networks (DNNs) [8], NN-based algorithms for signal processing in OFDM-MIMO systems have gained immense attention nowadays [5], [9], [10], [11], [12], [13], [14], [15], [16], [17].

In this work, we focus on applying machine learning algorithms to the symbol detection problem in MIMO-OFDM systems. Many research works have been proposed to solve the symbol detection task using DNNs. The authors in [11] introduce a five-layer DNN, which manages wireless OFDM channels in an end-to-end manner. The algorithm explicitly estimates channel state information (CSI) and recovers the transmitted symbols directly. A deep convolutional neural network (CNN) for symbol detection is proposed in [12]. The authors exploit CNN to capture spectral correlation in channels. In [13], the authors leverage an iterative soft-thresholding algorithm with DNNs to optimize the parameters of conventional symbol detection algorithms. These DNN-based algorithms require a large amount of training data and computation/storage resources. Symbol detection in OFDM-MIMO systems is time-critical and requires low energy consumption. Therefore, DNN-based algorithms will not have sufficient computation and storage resources to perform real-time and energy-efficient inference in practical OFDM-MIMO systems. Reducing the energy consumption of MIMO-OFDM systems is beneficial for building a sustainable environment.

Reservoir computing (RC) [18] is a framework of computation derived from recurrent neural networks (RNNs) [19]. The liquid state machine (LSM) [20] is a kind of RC that uses spiking neural networks (SNNs) [21]. An LSM has two main building blocks: the reservoir layer and the readout layer. The reservoir layer works like a recurrent block in an RNN, mapping the time-varying input to high-dimensional spaces. Then, it combines the information from both previous and current time steps. Next, the reservoir layer's output is sent to the readout layer for further processing. During the training phase, only the readout layer's weights are trained, while other weights are generated randomly. An LSM has two major advantages over DNNs. The first advantage is that LSM is easier to train and requires less training data. Also, it can capture both temporal and spatial information simultaneously, which is very helpful for restoring the corrupted symbols from distortion and interference at receivers in an OFDM-MIMO system. The second advantage is that it is more energy-efficient because of the utilization of

Manuscript received 17 November 2022; revised 30 September 2023; accepted 8 October 2023. Date of publication 13 October 2023; date of current version 3 April 2024. This work was supported by the U.S. National Science Foundation (NSF) under Grants CCF-1750450, ECCS-1731928, ECCS-2128594, ECCS-2314813, and CCF-1937487. Recommended for acceptance by D. Gizopoulos. *(Corresponding author: Yang Yi.)*

The authors are with Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: shiyal@vt.edu; yibin@vt.edu; yangyi8@vt.edu).

Digital Object Identifier 10.1109/TSUSC.2023.3324339

SNNs. SNNs are more biological-plausible and energy-efficient than DNNs because SNNs use sparse and asynchronous discrete events for communication between neurons [22]. SNNs have made significant contributions to event-based sensing and perception, odor recognition and learning, simultaneous localization and mapping, and closed-loop control for robotics [23], [24], [25], [26]. Taking advantage of event-based computation, SNNs bring advantages such as low power consumption, real-time processing, and improved perception and control to these application domains. To further improve energy efficiency, our LSM is implemented on the Loihi chip [27], a neuromorphic computing chip suitable for running SNNs.

However, SNNs generally exhibit lower accuracy performance compared to DNNs [28]. This can be attributed to the utilization of 1-bit spikes for information propagation between neurons in SNNs, which pose challenges due to their non-differentiable nature. A surrogate gradient descent algorithm [29] is required to train SNNs but there is still an accuracy gap existing between a DNN and an SNN. In order to tackle this challenge, we present a novel knowledge distillation (KD) algorithm [30] called DNN-SNN co-learning, which aims to enhance the accuracy of SNNs through the acquisition of knowledge transferred from DNNs. KD serves as a model compression technique that leverages the expertise of a larger teacher model to enhance the performance of a smaller student model. Conventional KD has three major issues. First, the efficient transfer of knowledge from DNNs to SNNs for regression tasks remains an open question, as conventional KD techniques are primarily designed for DNNs. Second, selecting an appropriate teacher model that can effectively guide the learning of a student model poses a significant challenge. Many research works [31], [32] demonstrate that high-performance teacher models cannot often produce high-performance student models because of the mismatched capacity between them. Also, the parameters of the teacher model might not be adaptable to the student model. Therefore, the student model cannot learn meaningful knowledge from the teacher model. Third, KD is built for classification tasks but symbol detection is a regression task.

To address the aforementioned issues, the DNN-SNN co-learning algorithm is proposed. We use a DNN as the teacher model and an SNN as the student model. A bi-directional learning path is built between a DNN and an SNN to train both networks in parallel. Through the bi-directional learning path, the SNN can mimic the behavior of the DNN by learning the distilled knowledge from the DNN. Also, the DNN can better adapt itself to the SNN by learning the knowledge from the SNN. We facilitate knowledge transfer by transferring knowledge from both the output and intermediate layers of the DNN to the SNN. By leveraging these two learning paths, the SNN gains the ability to acquire knowledge concerning both the output distribution and the representation captured by the intermediate layers of the DNN. Neurons in SNNs use spikes to communicate with other neurons while DNNs utilize real numbers. To establish communication between DNNs and SNNs, we employ a decoder that translates the intermediate layers of an SNN into real numbers. To enable KD on regression tasks, a new loss function is introduced. Our contributions are summarized below.

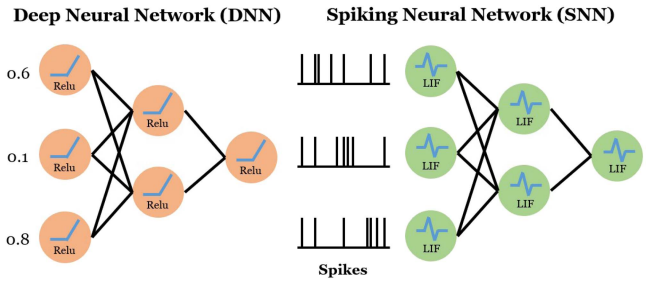


Fig. 1. Comparison of DNNs and SNNs.

- A DNN-SNN co-learning algorithm is introduced. The algorithm reduces the mismatched capacity between DNN and SNN models, and better adapts DNNs to SNN models. Also, it resolves the issue of applying KD between DNNs and SNNs. The proposed loss function enables KD on regression tasks.
- To effectively transfer the knowledge, the DNN-SNN co-learning algorithm distills knowledge from a DNN's output and intermediate layer to an SNN. To allow communication between DNNs and SNNs, a decoder is exploited to translate the output spikes of an SNN's layer into real numbers.
- On the symbol detection task, the LSM model trained by the DNN-SNN co-learning algorithm improves the average bit error rate by 5.7% compared to the LSM model trained by the surrogate gradient descent training algorithm [29].
- Our LSM model has been deployed on Loihi and achieves comparable accuracy compared to other DNN models. Moreover, Our LSM model has several times less energy consumption per sample than other DNN models on GPU.

II. Background

A. Comparison of DNNs and SNNs

Both DNNs and SNNs are brain-inspired. However, DNNs have essential differences in their neural computations compared to the brain. One of the most important differences is the way that information propagates between their neurons. In the brain, spike trains of action potentials are utilized for communications between neurons [33]. These individual spikes are sparse in time and have uniform amplitude. However, DNNs use real numbers to carry information between neurons. Due to this fundamental difference, SNNs have emerged. In SNNs, information is transmitted by event-driven firing activities, which are represented using 1-bit spikes. SNNs exploit spike latency and rates to carry information. The comparison between a DNN and an SNN is demonstrated in Fig. 1. Neurons in DNNs communicate using real numbers while neurons in SNNs communicate using spikes.

The leaky integrate-and-fire (LIF) model [34] is widely regarded as the most popular neuron model in SNNs. In this model, the membrane potential of the LIF neuron acts as a storage medium for temporal spike information. The membrane potential in the discrete-time domain can be expressed as,

$$U_{i,t} = \left(1 - \frac{1}{\tau}\right) U_{i,t-1} + \frac{1}{\tau} \sum_{j} w_{ij} o_{j,t}, \tag{1}$$

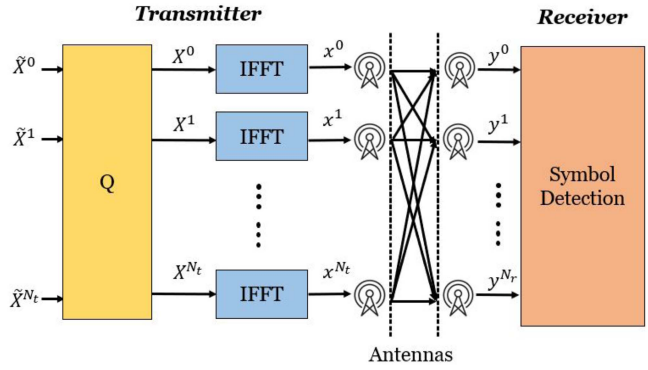


Fig. 2. MIMO-OFDM system architecture.

where $U_{i,t}$ is the membrane potential of a neuron i at timestep t and τ is the membrane time constant of the neuron. w_{ij} represents the weights connection between neuron i and neuron j. $o_{j,t}$ is the spike output of neuron j at timestep t. When potential $U_{i,t}$ is larger than a threshold, the neuron fires a spike. Then, the potential $U_{i,t}$ is reset to the resting voltage U_{rest}.

B. Knowledge Distillation

KD is a model compression technique introduced in [30]. It facilitates the transfer of knowledge from a teacher model to a compact student model, thereby enhancing the accuracy of the latter. The student model optimization process involves the utilization of two distinct loss functions. The first loss function utilizes the ground truth labels as the target for training. The second loss function leverages the class probabilities predicted by the teacher model as the target. The knowledge within the teacher model can be transferred to the student model through the second loss function. The second loss function incorporates a modified softmax function, which introduces a temperature parameter T. The authors in [30] indicate that essential knowledge is embedded in the ratios of class probabilities of the teacher model's prediction. To strengthen the influence of these ratios, the softmax function is augmented with the introduction of a parameter T. The modified softmax function is mathematically represented as,

$$P(z_i;T)=\frac{\exp\left(z_i/T\right)}{\sum_{j=1}^{n}\exp\left(z_j/T\right)} \tag{2}$$

The complete loss function for the student model is expressed as,

$$\begin{aligned}L(z_t,z_s,y)=&\,\alpha H(P(z_t;T=k),P(z_s;T=k))\\&+(1-\alpha)H(y,P(z_s;T=1)),\end{aligned} \tag{3}$$

where P represents the function defined in (2) and H is the cross-entropy loss function. α is the coefficient for the loss function. y represents the ground truth label. z_t is the output of the teacher model and z_s is the output of the student model.

III. MIMO-OFDM Systems

A MIMO-OFDM system [3], [4] is displayed in Fig. 2. At the transmitter side, there is N_t number of antennas to transmit N_t number of data streams. At the receiver side, there is N_r number of receiver antennas to recover the transmitted data. The i-th OFDM symbol of the t-th data stream in the frequency domain can be represented as,

$$\tilde{\boldsymbol{X}}_i^t \triangleq \left[\tilde{X}_i^t(0),\ldots,\tilde{X}_i^t(k),\ldots,\tilde{X}_i^t(N_{sc}-1)\right]^T, \tag{4}$$

where $\tilde{X}_i^t(k)$ donated as the modulated QAM symbol for sub-carrier k. N_{sc} represents the total data sub-carriers for each OFDM symbol.

All data streams' i-th frequency-domain QAM symbols at sub-carrier k are multiplied by a weight matrix $Q(k)$ before OFDM modulation. The weight matrix $Q(k)$ has a shape of $N_t \times N_t$. This procedure can be calculated as,

$$\boldsymbol{X}_i=\boldsymbol{Q}(k)\tilde{\boldsymbol{X}}_i, \tag{5}$$

Then, X_i^t of each transmitter is processed by an inverse fast Fourier transform (IFFT). The final N_{cp} samples of the IFFT output are placed at the beginning of the time-domain signal as cyclic prefix (CP). At the transmitter side, the i-th OFDM symbol for the t-th antenna in the time domain can be calculated as,

$$\boldsymbol{x}_i^t \triangleq \left[x_i^t(0),\ldots,x_i^t(n),\ldots,x_i^t(N_{cp}+N_{sc}-1)\right]^T, \tag{6}$$

where $x_i^t(n)$ represents the n-th sample of the i-th OFDM symbol in the time domain. Through detaching the CP of x_i^t and applying a FFT, X_i^t can be recovered. By concatenating N number of OFDM symbols, the time domain OFDM frame for antenna t at the transmitter side can be computed as,

$$\boldsymbol{x^t} \triangleq \left[\left(\boldsymbol{x}_1^t\right)^T,\ldots,\left(\boldsymbol{x}_i^t\right)^T,\ldots,\left(\boldsymbol{x}_N^t\right)^T\right]^T. \tag{7}$$

The received time-domain OFDM frame y^r at the received side's antenna r can be computed as,

$$\boldsymbol{y}^r=\sum_{t=0}^{N_t-1}u(\boldsymbol{x}^t)\circledast\boldsymbol{h}^{r,t}+\boldsymbol{z},\ \ 0\le r<N_r, \tag{8}$$

where $\circledast$ represents the convolution operator. $u(\cdot)$ is a non-linear function and it is used to model the signal distortion caused by transmitter circuits. $h^{r,t}$ is the channel impulse responses between receiver antenna r and transmitter antenna t. In this paper, the quasi-static channel assumption is used, which assumes $h^{r,t}$ is constant in one OFDM symbol and changes in different OFDM symbols. z is additive Gaussian noise.

At the receiver side, the i-th OFDM symbol of antenna r in the time domain is,

$$\boldsymbol{y}_i^r \triangleq [y_i^r(0),\ldots,y_i^r(n),\ldots,y_i^r(N_{cp}+N_{sc}-1)]^T, \tag{9}$$

the i-th OFDM symbol of antenna r in frequency domain is,

$$\boldsymbol{Y}_i^r \triangleq [Y_i^r(0),\ldots,Y_i^r(k),\ldots,Y_i^r(N_{sc}-1)]^T. \tag{10}$$

The purpose of symbol detection is to retrieve all data streams $\tilde{\boldsymbol{X}}_i^t$. It is fulfilled by processing received signals $\boldsymbol{y}_i^r$ from receiver antennas. To facilitate symbol detection, known information such as reference signals are inserted into the OFDM symbols $\tilde{\boldsymbol{X}}_i^t$ by MIMO-OFDM systems [35]. In this paper, the first N_{TS} OFDM symbols in a frame are used as a training sequence. The remaining OFDM symbols carry normal data.

The performance of symbol detection is strongly influenced by the signal-to-noise ratio (SNR), distortion, and interference. Linear minimum mean square error (LMMSE) [36], [37] is the conventional symbol detection algorithm and it consists of two stages. In the first stage, LMMSE estimates channels using the training sequence. Then, the estimated channels are exploited to retrieve the transmitted symbols. The disadvantage of the LMMSE algorithm is that it requires knowledge of the noise variance and channel statistics, which is difficult to acquire accurately at low SNR scenarios.

This paper proposes a resource-efficient symbol detection algorithm using LSM on Loihi. LSM is energy-efficient and easy to train. Meanwhile, LSM can process both spatial and temporal information in the received signals. To improve the performance in terms of accuracy, we introduce a DNN-SNN co-learning algorithm to enhance the accuracy of an LSM through the acquisition of knowledge transferred from a DNN such as ESN. Our experimental results show that LSM trained by the DNN-SNN co-learning algorithm has better results compared to conventional symbol detection algorithms and DNN-based algorithms such as multilayer perceptron (MLP) [38]. Also, our LSM model has comparable accuracy as RNN [19] and CNN [39] models with much lower energy consumption per sample.

IV. Reservoir Computing

RC is a variant of RNN and it consists of two layers. The first layer is called the reservoir layer and the second layer is called the readout layer. The readout layer's weights are trained during training while the reservoir layer's weights are not trained. A reservoir layer can be computed as,

$$x(t) = (1-\alpha)x(t-1) + \alpha \tanh(W_{in}u(t) + \theta + W_p x(t-1)), \tag{11}$$

where $u(t)$ is the input of a reservoir layer at time t. $x(t)$ is the output of a reservoir layer at time t. W_{in} is the weight matrix between the input and reservoir layer. W_p is the recurrent weight matrix for the reservoir layer. The readout layer can be expressed as,

$$y(t) = W_{out}x(t) + \theta_{out}, \tag{12}$$

where W_{out} is the weight matrix of the readout layer and θ_{out} is the bias term. Both W_{out} and θ_{out} are trained during training.

A. Advantages of Reservoir Computing

1) Training of RC: Backpropagation through time algorithm [40] is used to train RNNs. Training of RNNs often suffers gradient explosion or vanishing issues. RCs do not have the gradient explosion and vanishing issue since the weights in the reservoir layers are built randomly and are not trained during the training phase.

2) Efficient Hardware Implementation: An RC has a less complicated architecture compared to an RNN. Therefore, RCs consume fewer computation resources and achieve faster inference speed. These characteristics are suitable for hardware implementations [41], [42].

3) Memory Capacity: To improve the performance, the concept of DNNs has been applied to RCs. Several deep RC networks have been introduced in [20], [43]. The authors in [20], [43] empirically show that deep RC networks have higher timescale differentiation and richer memory capacity compared to shallow RC networks.

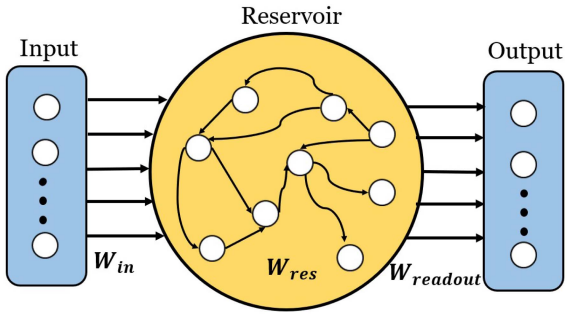


Fig. 3. Overview of ESN.

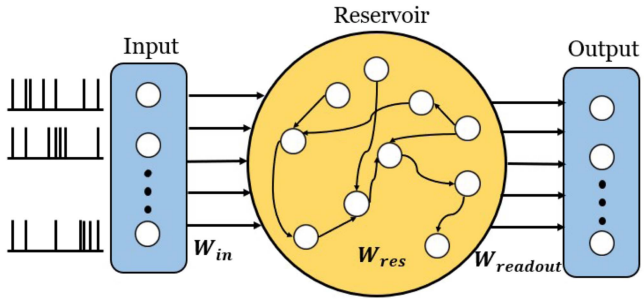


Fig. 4. Overview of LSM.

B. Different Types of Reservoir Computing

Echo state network (ESN) and LSM are the two major types of RC.

1) Echo State Network: In ESN, The weights W_{in} is the weights between the input and the reservoir layer. The weights W_{res} are the recurrent weights for the reservoir layer. The weights $W_{readout}$ are the weights in the readout layer. The weights W_{in} and W_{res} are fixed while only the weights $W_{readout}$ are trained during the training phase. The architecture of an ESN is demonstrated in Fig. 3.

2) Liquid State Machine: The overview of LSM is shown in Fig. 4 and it is very similar to the architecture of ESN. In LSM, input is a sequence of spikes. Input spikes are transformed by the weights W_{in} and then sent to the reservoir layer. the weights W_{res} are the recurrent weights in the reservoir layer. The output of the reservoir layer is further processed by the readout layer to generate the final output. Only the weights of the readout layer are trained and other weights are generated randomly. An essential distinction between ESN and LSM is that LSM uses SNN for data processing.

An LSM uses a spiking neural network and has better energy-efficient than an ESN. Due to this advantage, an LSM is more

suitable for communication systems applications such as symbol detection.

V. Liquid State Machine With DNN-SNN Co-Learning Algorithm

In recent years, DNNs have achieved remarkable success [8]. DNN-based algorithms for signal processing in OFDM-MIMO systems have gained immense attention nowadays [5], [9]. The performance of a DNN model highly depends on the number of neurons and layers in the network. Therefore, the computational and space complexity of a high-performance DNN model is very high. Many applications in OFDM-MIMO systems are time-critical and require low energy consumption. However, typical OFDM-MIMO systems do not have sufficient computation and storage resources to perform real-time inference with high energy efficiency.

SNNs are a group of promising models that mimic the neuronal dynamics of the brain. Efficient hardware implementation of SNNs has been successfully demonstrated in previous studies [27]. In contrast to DNNs, SNNs offer a higher level of biological plausibility. By leveraging sparse and asynchronous discrete events for neuron communication, SNNs are more energy-efficient than DNNs [44], [45]. LSM is a type of RC that uses an SNN. Similar to RNN, LSM can capture both temporal and spatial information from the input data. This characteristic is very useful for restoring the corrupted symbols from the distortion and noise at the receiver in an OFDM-MIMO system.

However, it is commonly observed that the precision of an SNN tends to be inferior when compared to that of a DNN. The main challenge is that SNNs are difficult to train since spikes in SNNs are not differentiable. The STDP learning algorithm [46] has gained popularity in training SNNs due to its simplicity and effectiveness. This algorithm updates synaptic weights by considering the relative timing of pre- and post-synaptic action potentials within a defined learning window. However, relying solely on the STDP learning algorithm is insufficient for developing high-performance SNN models. Unlike DNNs, which can be trained using mature algorithms like gradient descent [47], SNNs face challenges due to the non-differentiable nature of spike activities. To address this training obstacle, several surrogate gradient descent training algorithms have been proposed [29], [48]. These approaches involve approximating the spiking activities using an approximation function. Subsequently, the back-propagation through time algorithm [40] is employed to train an SNN model.

Even though the performance of SNN models can be improved by the aforementioned surrogate gradient descent training algorithms, there is still a gap between the performance of DNNs and SNNs. To reduce the performance gap, we propose a KD algorithm called DNN-SNN co-learning to improve SNN performance by learning the behavior of DNNs. KD serves as a model compression technique that leverages the expertise of a larger teacher model to enhance the performance of a smaller student model. Conventional KD suffers three major drawbacks. First, conventional KD has only been applied to DNNs. The efficient transfer of knowledge from DNNs to SNNs for regression tasks remains an open question. Second, picking a suitable teacher model is challenging. Many research works [31], [32] found that high-performance teacher models cannot often generate high-performance student models because of the mismatched capacity between them. Also, the parameters of the teacher model might not be adaptable to the student model. Therefore, the student model cannot learn helpful knowledge from the teacher model. Third, KD is designed for classification tasks while symbol detection is a regression task.

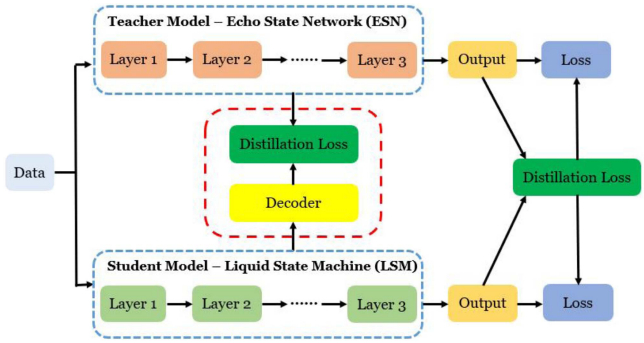


Fig. 5. Overall training process of DNN-SNN co-learning.

To address these issues, our DNN-SNN co-learning algorithm builds a bi-directional learning path between a DNN and an SNN model. A DNN is the teacher model and an SNN is the student model. Through the bi-directional learning path, the SNN can mimic the behavior of the DNN by learning the distilled knowledge from the DNN. Also, the DNN can better adapt itself to the SNN by learning the transferred knowledge from the SNN. To effectively transfer knowledge, the knowledge from the output and intermediate layer of the DNN is transferred to the SNN. Using these two knowledge transfer paths, the SNN model can learn the DNN's output distribution and intermediate layer representation. SNNs utilize spikes to perform communication between neurons while DNNs utilize real numbers. To establish communication between DNNs and SNNs, we employ a decoder that translates the intermediate layers of an SNN into real numbers. In our DNN-SNN co-learning algorithm, a new loss function is proposed to allow KD on regression tasks.

A. DNN-SNN Co-Learning Algorithm

The overall training process of the DNN-SNN co-learning algorithm is described in Fig. 5. The DNN-SNN co-learning algorithm incorporates an LSM as the student model, while an ESN serves as the teacher model. This configuration effectively minimizes the disparity in model capacity between the teacher-student model pair. During the training process, three distinct loss functions are employed to train the student model. The first loss function, known as the output loss function, employs the ground truth label as the target value for training. The second loss function, termed the output distillation loss function, leverages the model predictions generated by the teacher model as the target values. The third loss function, referred to as the layer distillation loss function, utilizes the output of the intermediate layers in the teacher model as the target values. In contrast to

conventional KD, our algorithm transfers knowledge from the output and intermediate layer of the teacher model to the student model. This setup enables the student model not only to learn the output distribution of the teacher model but also to mimic the intermediate representation of the teacher model. LSMs use 1-bit spikes to propagate information between neurons while ESNs use real numbers. In the following subsections, we discuss how to enable the communication between ESNs and LSMs through decoders and loss functions.

Two loss functions optimize the teacher model during training. The first loss function is the output loss function, which uses the ground truth label as the target value. The second loss function is the output distillation loss function, where the target value is the final output generated by the student model. By learning the knowledge from the student model, the teacher model becomes more adaptable to the student model. This adaptability enables the teacher model to provide more beneficial expertise to the student model.

B. DNN-SNN Communication Via Decoders

To better optimize an SNN using the knowledge from a DNN, our DNN-SNN co-learning algorithm transfers the knowledge from a DNN's intermediate layers and output layer to an SNN. However, DNNs cannot communicate with SNNs because of the way that information propagates between neurons. Both DNNs and SNNs are brain-inspired neural networks. However, there exist fundamental disparities between DNNs and SNNs. The most important difference is that DNNs use real numbers to propagate information between neurons while SNNs utilize a 1-bit spike train to communicate. These individual spikes are sparse in time and have uniform amplitude. SNNs are more biological-plausible than DNNs because brains exploit spike trains of action potentials for propagating information.

To establish effective communication, we introduce a decoder that is designed to translate the output of intermediate layers in an SNN into real numbers. Assume the l-th intermediate spiking layer I^l of an SNN has N_s neurons with T time steps and the corresponding intermediate layer of a DNN has N_d neurons. The decoder D^l is a dense layer and the shape of the layer is $N_s \times N_d$. The output of the intermediate spiking layer I^l is sent to the decoder D^l. The decoder takes spikes as input and generates membrane potentials for each time step. The output of the k-th neuron of the decoder at t-th time step can be expressed as,

$$D^l_{k,t} = \alpha^l_k U^l_{k,t-1} + W^l_k I^l_t, \tag{13}$$

where $U^l_{k,t-1}$ is the membrane potential for the k-th neuron at $(t-1)$-th time step. I^l_t is the spike output of the l-th intermediate spiking layer of the SNN at t-th time step. α^l_k is the decay rate of the membrane potential. W^l_k is the weights for the k-th neuron. For each output neuron in the decoder, we take the average of the output at each time step to generate the final output. The procedure is illustrated as,

$$O_k = \frac{1}{T}\sum_{t=1}^{T} D^l_{k,t}. \tag{14}$$

The decoded output O_k is sent to the loss function. The target value of the loss function is the output of the DNN's intermediate layer.

C. Input Spike Encoding

One of the key questions for using SNN is how to convert real-valued inputs into binary spikes. Several coding schemes have been proposed to accomplish this conversion, encompassing temporal coding, phase coding, Burst coding, and rate coding [49], [50], [51], [52], [53].

Temporal coding operates by producing a single spike for each neuron, with the latency of the spike being inversely related to the magnitude of the real-valued input [51]. When the input surpasses a predefined threshold, a spike is generated, and any further spikes from that particular input are inhibited. This coding scheme maps inputs to the precise timing of the initial spikes. The noteworthy benefits of temporal coding include its swiftness and energy efficiency.

Phase coding, as described by Montemurro et al. [52], introduces temporal information into spike patterns through the incorporation of phase information. This is achieved by assigning distinct weights to different time steps within the representation, with the spike weight undergoing periodic changes over time. The number of phases utilized in this coding scheme is determined by the highest magnitude among the inputs. Notably, phase coding exhibits resilience against input noise, making it a robust coding method for handling perturbations in the input.

Burst coding Burst coding, as explored in the study by Park et al. [53], operates by transmitting a cluster or "burst" of spikes within a short temporal window, thereby enhancing the reliability of synaptic communication between neurons. In this coding scheme, information is encoded within both the number of spikes present in the burst and the intervals between consecutive spikes within the burst. It has been demonstrated that burst coding exhibits compatibility with network compression techniques, including quantization and pruning.

Rate coding is a coding scheme that transforms real-valued inputs into spike trains, where the quantity of spikes is directly proportional to the magnitude of the input [49]. In this method, each input is treated as a firing rate, and subsequently, the input is converted into a Poisson spike train with a firing rate equivalent to the input value. By utilizing rate coding, information is encoded in the frequency of spikes, enabling the representation of varying input intensities through the rate of neuronal firing.

For this study, we have opted to employ the rate coding method as the encoding technique for our inputs. Rate coding is extensively utilized due to its robustness, particularly in the context of deeper SNN models. While temporal coding, phase coding, and burst coding have been employed in shallow networks, their application becomes challenging when scaling up both the network size and dataset size [54]. In contrast, rate coding is well-suited for large-scale settings. Many recent state-of-the-art SNN models leverage this coding scheme [55], [56], [57]. In our encoding method, we first normalize the input data using

the equation,

$$x = \frac{x - a}{b - a} \tag{15}$$

where x is the real-valued input. a and b are the lower and upper bound of the input. Then, the normalized input is converted into a Poisson spike train using a Poisson encoder.

D. Loss Functions for Regression

As shown in Fig. 5, three loss functions are used to train the student model, which are output loss, output distillation loss, and layer distillation loss functions, respectively. The output loss function is an L1 loss function, which can be expressed as,

$$L_{SO} = \|y - O_s\|_1 , \tag{16}$$

where y is the ground truth label and O_s is the output of the student model.

The output distillation loss function uses the final output of the teacher model as the target. For some samples, the teacher model performs worse than the student model so the knowledge from the teacher model is not helpful for the student. Therefore, we use a conditional loss function. When the student model has better performance on some samples than the teacher model, we set this loss function to 0. Otherwise, we use an L1 loss function to train the student model. The output distillation loss function is shown as,

$$L_{SOD}(y, O_t, O_s) = \begin{cases} 0 & if \|y - O_s\|_1 \leq \|y - O_t\|_1 \\ \|O_t - O_s\|_1 & otherwise, \end{cases} \tag{17}$$

where O_t and O_s are the teacher and student output, respectively. y is the target label from the dataset.

The layer distillation loss function uses the output of intermediate layers of the teacher model as the target value. The loss function is expressed as,

$$L_{SLD}(I_t, I_s) = \|I_t - I_s\|_1 , \tag{18}$$

where I_t is the intermediate layer of the teacher model and I_s is the decoded intermediate layer of the student model.

The complete loss function for the student model is expressed as,

$$L_s = L_{SO} + \alpha_s L_{SOD} + \beta_s L_{SLD} \tag{19}$$

The coefficients α_s and β_s represent the weighting factors associated with the loss functions L_{SOD} and L_{SLD}, respectively. These two parameters determine the contribution of each distillation loss function to the overall loss function.

The teacher model is optimized by the output loss function and output distillation loss function. The output loss function is expressed as,

$$L_{TO} = \|y - O_t\|_1 , \tag{20}$$

where O_t is the output of the teacher model.

The output distillation loss function of the teacher model uses the final output of the student model as the target. By learning the knowledge from the student model, the teacher model can better adapt itself to the student model. Then, more beneficial guidance can be provided for the student model. We also use a conditional loss function here. When the teacher model has better performance on some samples than the student model, we set this loss function to 0. Otherwise, we use an L1 loss function to train the teacher model. The output distillation loss function for the teacher model is expressed as,

$$L_{TOD}(y, O_t, O_s) = \begin{cases} 0 & if \|y - O_t\|_1 \leq \|y - O_s\|_1 \\ \|O_t - O_s\|_1 & otherwise, \end{cases} \tag{21}$$

The complete loss function for the teacher model is,

$$L_t = L_{TO} + \alpha_t L_{TOD} \tag{22}$$

where α_t is the coefficient to determine the contribution of the output distillation loss function to the overall loss function.

E. Comparison of the Bootstrap SNN Training and the DNN-SNN Co-Learning

The bootstrap SNN training [58] is a popular DNN-SNN conversion algorithm. This algorithm is characterized by a structured two-phase approach, namely the sample phase and the fit phase. During the sample phase, the algorithm initially gathers input/output data points from the SNN model. Subsequently, these data points are utilized to approximate the corresponding DNN activation, constituting a piecewise linear layer. Moving on to the fit phase, an equivalent DNN network is trained over a designated number of epochs. Following this training, the resultant network is then transformed into an SNN. The sample and fit phases are iteratively alternated throughout the training process.

There are three major differences between the bootstrap SNN training algorithm and our proposed DNN-SNN co-learning algorithm. First, the bootstrap method employs an indirect approach to SNN model training, while our DNN-SNN co-learning method directly trains the SNN model. In the bootstrap method, an initial DNN equivalent network is trained, followed by its conversion into an SNN model. In contrast, our method facilitates direct SNN model training through the utilization of DNN knowledge and ground-truth labels, eliminating the need for a conversion step between the DNN and SNN models. This obviates potential issues arising from the adaptability of DNN weight parameters to the SNN model, ensuring that the SNN model can enhance its performance by assimilating knowledge from the DNN model.

The second point of divergence pertains to the utilization of feedback mechanisms: the bootstrap method relies on delayed feedback, whereas our DNN-SNN transfer learning approach leverages instant feedback. In the bootstrap method, the sample and fit phases alternate during the training process, causing a delay in the exchange of feedback and knowledge between the DNN and SNN models. In contrast, our method concurrently trains both the DNN and SNN models, facilitating real-time exchange of knowledge and feedback. This immediate feedback mechanism enhances the tuning of parameters in both the DNN and SNN models, rendering them more mutually adaptable.

The third point of difference between the two approaches lies in how they handle the weight parameters of the SNN model. In

the bootstrap method, SNN model weights are derived directly from the trained weights of the DNN equivalent network. In contrast, our approach involves the direct training of SNN model weights, incorporating feedback and knowledge from both the DNN model and the ground-truth labels. Notably, our method employs a bi-directional learning algorithm, allowing the SNN to emulate the DNN's behavior by assimilating distilled knowledge from the DNN model. Simultaneously, the DNN model enhances its adaptability to the SNN by incorporating insights from the SNN's knowledge. This bi-directional learning mechanism facilitates a mutually beneficial exchange of information, optimizing the performance of both the DNN and SNN models.

VI. Experiments

A. Experimental Setup

In the symbol detection experiments, our LSM has 1 reservoir layer and two readout layers. The input sequence length is 6 and the output size is 2. The reservoir layer has 32 neurons. The input and output size of the first readout layer is 32 and 16, respectively. The input and output size for the second readout layer is 16 and 2, respectively. Our LSM runs on a single Loihi 2 chip. The configuration on Loihi 2 is as follows. We use the current-based LIF neuron in our SNN models. The neuron's voltage decay time constant is 0.03 and the current decay time constant is 0.25. The threshold is set at 1.0. We use an ESN as the teacher model for our DNN-SNN co-learning algorithm and the ESN has exactly the same architecture as the LSM. The reservoir layer has 32 neurons. The first and second readout layer has 16 and 2 neurons, respectively. The only difference is that the LSM is a spiking neural network that uses spiking neurons. We compare our LSM trained by the DNN-SNN co-learning algorithm with other state-of-the-art algorithms such as LMMSE, DetNet [10], MMNet [14], and OAMPNet [15]. Deep learning algorithms such as MLP, CNN, and RNN are also included. In our experiments, all the models not using Loihi are implemented using Pytorch [59] and run on an NVIDIA RTX A2000. All Loihi experiments are implemented using Intel's Lava framework.

In order to standardize power consumption for comparison purposes, we establish a baseline measurement for both the GPU and Loihi platforms. This is accomplished by measuring the power consumption of each platform during idle periods. Subsequently, the power and energy consumption of the application can be measured and compared against this established baseline. Meanwhile, we ensure that the number of parameters of the model and the model's multiply-accumulate operations on both the GPU and Loihi platforms are comparable. To achieve statistical significance, we execute each application multiple times and calculate the average power and time for each application on both the GPU and Loihi platforms. To measure power and energy consumption on the Loihi platform, we follow the tutorial outlined in the notebook provided by Intel [60]. For power and energy consumption measurements on the GPU platform, we utilize Nvidia's Performance Analysis Tools, such as NVIDIA Nsight Systems and NVIDIA Visual Profiler. Finally, we can compare the absolute energy consumption per sample of different models on GPU and Loihi 2.

TABLE I
Comparison of MAC Operations for Different Models

Network	Number of MACs
QLSM	7648
FPESN	7648
QMLP	2944
QCNN	7552
QRNN	7648

The evaluation metric we used for the symbol detection task is bit error rate (BER) [61]. We use the following abbreviations for different algorithms in symbol detection experiments.

- *"QLSM"* represents the LSM model trained without the DNN-SNN co-learning algorithm. The LSM model has one reservoir layer and two readout layers. The reservoir layer has 32 neurons. The first and second readout layer has 16 and 2 neurons, respectively.
- *"QLSM+DSCL"* represents the LSM model trained with the DNN-SNN co-learning algorithm. The QLSM+DSCL model has one reservoir layer and two readout layers. The reservoir layer has 32 neurons. The first and second readout layer has 16 and 2 neurons, respectively.
- *"FPESN"* represents the floating-point ESN model. The ESN model has the same architecture as the QLSM. It has 32, 16, and 2 neurons in the reservoir, the first readout layer, and the second readout layer, respectively.
- *"LMMSE"* represents the algorithm proposed in [36].
- *"DetNet"* represents the model proposed in [10].
- *"MMNet"* represents the model proposed in [14].
- *"OAMPNet"* represents the model proposed in [15].
- *"QMLP"* is a quantized MLP model with 3 dense layers. The model is quantized to 8 bits. The first and the second dense layer has 32 and 64 neurons, respectively. The last dense layer has 2 neurons.
- *"QCNN"* is a quantized CNN model with two convolution layers and one dense layer. The model is quantized to 8 bits. Both convolution layers perform 1D convolution with a kernel size of 3. The first convolution layer has 6 input channels and 16 output channels. The second convolution has 16 input channels and 32 output channels. The last dense layer has an output size of 2.
- *"QRNN"* is a quantized RNN model. The model is quantized to 8 bits. It has one recurrent layer followed by two dense layers. The recurrent layer has 32 neurons. The two dense layers have 16 and 2 neurons, respectively.

The number of Multiply–accumulate (MAC) operations for each model are summarized in Table I.

B. Symbol Detection Datasets

The performance of our LSM model is evaluated on the symbol detection task in MIMO-OFDM systems. The received MIMO-OFDM signals are sent to our LSM directly to predict the transmitted signals. The MIMO-OFDM system has $N_t = 4$

transmit antennas and $N_r = 4$ receive antennas. An LTE/5G-compatible frame structure is used. The frame length is $N = 100$ OFDM symbols, in which the first $N_{TS} = 8$ OFDM symbols are used as the training sequence, and the remaining OFDM symbols carry user data. The total sub-carriers for each user is $N_{sc} = 128$, and 16-QAM modulation is adopted to produce information symbols $\tilde{X}_i^t(k)$. We focus on single-user scenarios in our analysis to simplify the simulation setup. The wireless channel coefficients are generated in accordance with the 3GPP propagation channel model [62]. Specifically, the delay profile of the channels follows the Extended Pedestrian A model (EPA), considering the evolving nature of the channel across OFDM symbols with a Doppler frequency of 20 Hz.

C. Training Setup

A mini-batch of 128 is used to train the models, and the optimization algorithm is the Adam learning algorithm [63]. The learning rate for the Adam learning algorithm is 1e-3. The maximum training epoch is 100. The loss functions are described in Section V. The loss functions Hyperparameters such as α_s, α_t, β_s in (19) and (22) are set to 0.1, 0.05, and 0.05. The hyper-parameters are determined by the hyperparameter optimization toolkit provided in [64].

D. Accuracy Comparison

The BER of our QLSM+DSCL on the symbol detection task with different signal-to-noise ratios (SNRs) is compared with the state-of-the-art model-based and DNN-based symbol detection algorithms such as LMMSE, DetNet [10], MMNet [14], OAMP-Net [15], QMLP, QCNN, and QRNN. The LMMSE is a classic model-based algorithm for symbol detection. LMMSE requires accurate channel information but the channel information is difficult to acquire at low SNR. DetNet, a DNN model, has been developed by unfolding the iterations of the projected gradient descent algorithm. It has exhibited commendable performance in scenarios characterized by independent and identically distributed (iid) Gaussian channels and low-order modulation schemes. MMNet is also a DNN model. It is constructed based on the theory of iterative soft-thresholding algorithms. Notably, MMNet surpasses DetNet when operating in more realistic channels and high-order modulation schemes. OAMPNet has been specifically designed to acquire the optimal parameters of the orthogonal AMP algorithm. OAMPNet demonstrates impressive performance in Kronecker model-based correlated MIMO channels. The QMLP is a DNN-based algorithm, which uses multilayer perceptron neural networks. The QCNN and the QRNN are another two DNN-based algorithms. The QCNN uses convolution kernels to extract information at nearby input signals. The QRNN uses recurrent blocks to process a sequence of input signals for extracting the temporal information in the input signals. The BER for different models at various SNRs is shown in Fig. 6. As demonstrated in Fig. 6, our LSM+DSCL outperforms symbol detection models such as LMMSE, DetNet, MMNet, OAMPNet, and QMLP in terms of BER at SNR from 0 dB to 15 dB. Compared to QCNN and QRNN models, our LSM+DSCL achieves comparable BER at SNR from 0 dB to 15 dB. Later, we demonstrate that the LSM+DSCL model is more energy-efficient than the QCNN and QRNN models. The performance of DetNet, MMNet, and OAMPNet is observed to be inadequate, as these models were designed to operate optimally with significantly larger training datasets. Notably, DetNet exhibits limited functionality when faced with the 3 GPP channel model, as it was specifically designed for iid Gaussian channels.

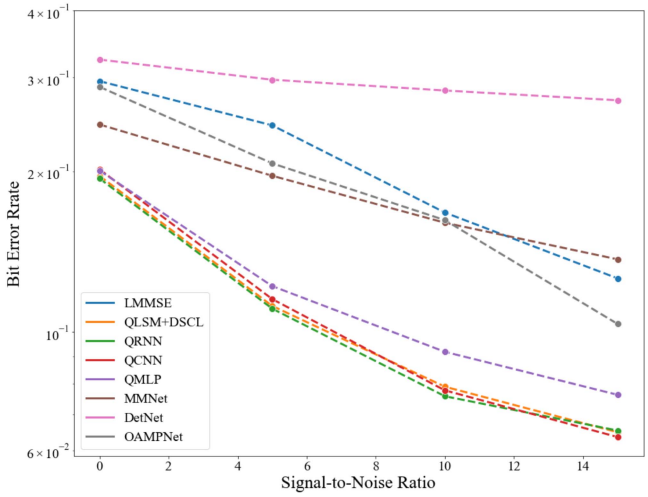


Fig. 6. BER at various SNR ratios for different models.

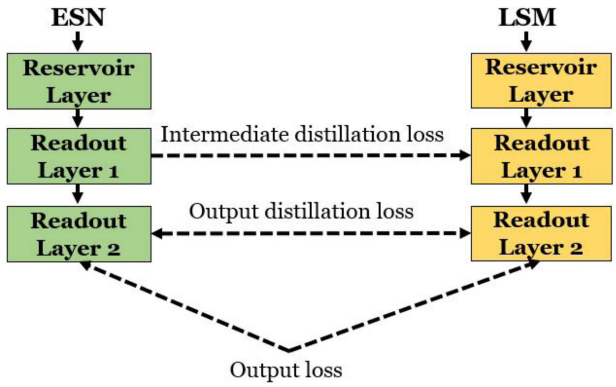


Fig. 7. Architecture of knowledge transfer between an ESN and an LSM.

E. Accuracy Improvement Using DNN-SNN Co-Learning Algorithm

To demonstrate the effectiveness of the DNN-SNN co-learning algorithm, we compare the BER of two LSM models trained by different training algorithms. The first model QLSM uses the surrogate gradient descent algorithm in [29]. The second model QLSM+DSCL is trained by the DNN-SNN co-learning algorithm. The overall architecture of the DNN-SNN co-learning algorithm is shown in Fig. 7.

The BER of these two models at different SNR ratios is shown in Fig. 8. On average, QLSM+DSCL reduces the BER by 5.7% compared to the QLSM model. The experimental results demonstrate that our DNN-SNN co-learning algorithm can significantly improve the BER.

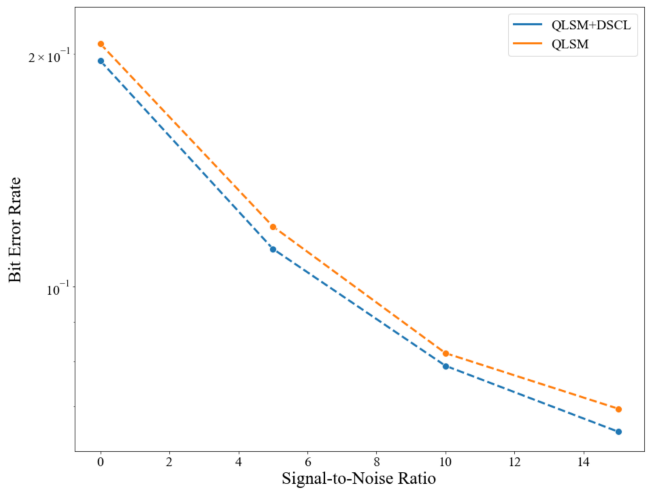


Fig. 8. LSM and LSM+DSCL models at different SNR ratios.

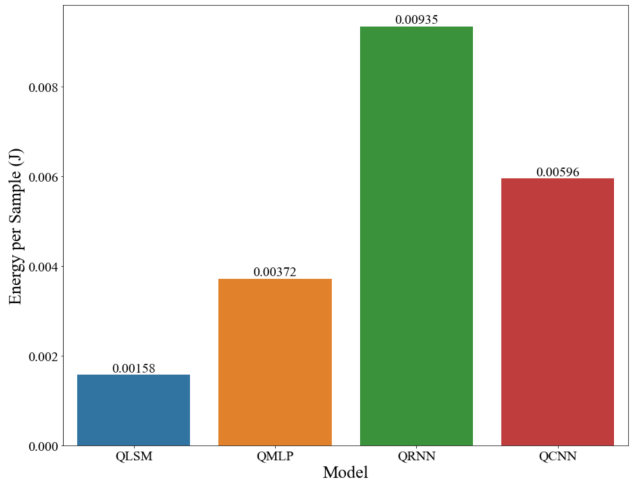


Fig. 9. Energy consumption per sample of different symbol detection models.

F. Resource Consumption

To illustrate the energy efficiency of our LSM models, we compare energy consumption per sample of the QLSM model on Loihi with other DNN-based networks on GPU. The energy per sample (J) with a batch size of 1 is shown in Fig. 9. We use a batch size of 1 on both GPU and Loihi since Loihi only supports a batch size of 1. As shown in Fig. 9, the energy per sample of the QLSM model on Loihi is 2.35X, 5.92X, and 3.77X less than the QMLP, QRNN, and QCNN models on GPU. The results demonstrate that energy-efficient symbol detection can be fulfilled using LSMs with Loihi.

VII. Conclusion

In this paper, we introduce an energy-efficient and sustainable symbol detection algorithm for MIMO-OFDM systems using LSM on the Loihi chip. LSM can recover corrupted symbols from distortion and noise at the receiver by capturing spatial and temporal information from input signals. Moreover, LSM is more energy-efficient than other DNN-based models because of the use of an SNN. To improve the performance of our LSM in accuracy, we propose a DNN-SNN co-learning algorithm to train our LSM. Through the bi-directional learning path between the DNN and our LSM, our LSM can mimic the behavior of the DNN by learning knowledge from it. Also, the DNN can better adapt to the LSM by learning the transferred knowledge from our LSM. To empower better knowledge transfer, the DNN-SNN co-learning algorithm transfers knowledge from the DNN's output and intermediate layers to our LSM. A decoder is used to convert the intermediate layer outputs of our LSM into real numbers to enable communication between the DNN and our LSM. Compared to symbol detection algorithms such as LMMSE, DetNet, MMNet, and OAMPNet, our LSM model trained by the DNN-SNN co-learning achieves a significant reduction in BER. Compared to other DNN-based models, such as RNN and CNN, our LSM model has approximately 6 times less energy consumption per sample with comparable accuracy. Our LSM model trained by the DNN-SNN co-learning algorithm achieves an average 5.7% improvement on BER compared to the LSM model trained by the surrogate gradient descent training algorithm. These results demonstrate that sustainable symbol detection in MIMO-OFDM systems can be realized using SNNs on Loihi.

VIII. Availability

All data and code used for running experiments are available on a GitHub repository at https://github.com/lsy105/symbol_detection.

References

[1] J. Thompson et al., "5G wireless communication systems: Prospects and challenges [guest editorial]," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 62–64, Feb. 2014.
[2] M. A. Habibi, M. Nasimi, B. Han, and H. D. Schotten, "A comprehensive survey of RAN architectures toward 5G mobile communication system," *IEEE Access*, vol. 7, pp. 70 371–70 421, 2019.
[3] A. Van Zelst and T. C. Schenk, "Implementation of a MIMO OFDM-based wireless LAN system," *IEEE Trans. Signal Process.*, vol. 52, no. 2, pp. 483–494, Feb. 2004.
[4] H.-G. Yeh, "Architectures for MIMO-OFDM systems in frequency-selective mobile fading channels," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 62, no. 12, pp. 1189–1193, Dec. 2015.
[5] G. L. Santos, P. T. Endo, D. Sadok, and J. Kelner, "When 5G meets deep learning: A systematic review," *Algorithms*, vol. 13, no. 9, 2020, Art. no. 208.
[6] L. Liu, R. Chen, S. Geirhofer, K. Sayana, Z. Shi, and Y. Zhou, "Downlink MIMO in LTE-advanced: SU-MIMO versus MU-MIMO," *IEEE Commun. Mag.*, vol. 50, no. 2, pp. 140–147, Feb. 2012.
[7] L. Liu, J. Zhang, and Z. Pi, "Inter-cell interference avoidance for downlink transmission," US Patent 8,238,954, Aug. 07 2012.
[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[9] H. Huang et al., "Deep learning for physical-layer 5G wireless techniques: Opportunities, challenges and solutions," *IEEE Wirel. Commun.*, vol. 27, no. 1, pp. 214–222, Feb. 2020.
[10] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2554–2564, May 2019.
[11] H. Ye, G. Y. Li, and B.-H. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 1, pp. 114–117, Feb. 2018.
[12] Z. Zhao, M. C. Vuran, F. Guo, and S. D. Scott, "Deep-waveform: A learned OFDM receiver based on deep complex-valued convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2407–2420, Aug. 2021.

[13] J. Liu, K. Mei, X. Zhang, D. Ma, and J. Wei, "Online extreme learning machine-based channel estimation and equalization for OFDM systems," *IEEE Commun. Lett.*, vol. 23, no. 7, pp. 1276–1279, Jul. 2019.

[14] M. Khani, M. Alizadeh, J. Hoydis, and P. Fleming, "Adaptive neural signal detection for massive MIMO," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5635–5648, Aug. 2020.

[15] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "A model-driven deep learning network for MIMO detection," in *Proc. IEEE Glob. Conf. Signal Inf. Process.*, 2018, pp. 584–588.

[16] H.-H. Chang, H. Song, Y. Yi, J. Zhang, H. He, and L. Liu, "Distributive dynamic spectrum access through deep reinforcement learning: A reservoir computing-based approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1938–1948, Apr. 2019.

[17] R. Shafin, L. Liu, V. Chandrasekhar, H. Chen, J. Reed, and J. C. Zhang, "Artificial intelligence-enabled cellular networks: A critical path to beyond-5G and 6G," *IEEE Wirel. Commun.*, vol. 27, no. 2, pp. 212–217, Apr. 2020.

[18] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: Theory, applications and implementations," in *Proc. 15th Eur. Symp. Artif. Neural Netw.*, 2007, pp. 471–482.

[19] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Conf. Interspeech*, 2010, pp. 1045–1048.

[20] N. Soures and D. Kudithipudi, "Deep liquid state machines with neural plasticity for video activity recognition," *Front. Neurosci.*, vol. 13, 2019, Art. no. 686.

[21] F. Ponulak and A. Kasinski, "Introduction to spiking neural networks: Information processing, learning and applications," *Acta Neurobiologiae Experimentalis*, vol. 71, no. 4, pp. 409–433, 2011.

[22] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," 2020, *arXiv: 2005.01807*.

[23] M. Davies et al., "Advancing neuromorphic computing with Loihi: A survey of results and outlook," *Proc. IEEE*, vol. 109, no. 5, pp. 911–934, May 2021.

[24] S. F. Muller-Cleve et al., "Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware," 2022, *arXiv:2205.15864*.

[25] G. Haessig et al., "Event-based computation for touch localization based on precise spike timing," *Front. Neurosci.*, vol. 14, 2020, Art. no. 420.

[26] G. Tang, A. Shah, and K. P. Michmizos, "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 4176–4181.

[27] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[28] L. Deng et al., "Rethinking the performance comparison between snns and anns," *Neural Netw.*, vol. 121, pp. 294–307, 2020.

[29] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[30] G. Hinton et al., "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[31] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 4794–4802.

[32] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5191–5198.

[33] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *Int. J. Neural Syst.*, vol. 19, no. 04, pp. 295–308, 2009.

[34] M. A. Nahmias, B. J. Shastri, A. N. Tait, and P. R. Prucnal, "A leaky integrate-and-fire laser neuron for ultrafast cognitive computing," *IEEE J. Sel. Top. Quantum Electron.*, vol. 19, no. 5, pp. 1–12, Sep./Oct. 2013.

[35] M. Gast, *802.11 Wireless Networks: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005.

[36] M. Latva-Aho and M. J. Juntti, "LMMSE detection for DS-CDMA systems in fading channels," *IEEE Trans. Commun.*, vol. 48, no. 2, pp. 194–199, Feb. 2000.

[37] Q. Guo and D. D. Huang, "A concise representation for the soft-in soft-out LMMSE detector," *IEEE Commun. Lett.*, vol. 15, no. 5, pp. 566–568, May 2011.

[38] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—A review of applications in the atmospheric sciences," *Atmospheric Environ.*, vol. 32, no. 14/15, pp. 2627–2636, 1998.

[39] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015, *arXiv:1511.08458*.

[40] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[41] G. Tanaka et al., "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, 2019.

[42] M. C. Soriano et al., "Delay-based reservoir computing: Noise effects in a combined analog and digital implementation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 388–393, Feb. 2015.

[43] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, 2017.

[44] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021.

[45] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 388–404.

[46] N. Caporale et al., "Spike timing-dependent plasticity: A Hebbian learning rule," *Annu. Rev. Neurosci.*, vol. 31, no. 1, pp. 25–46, 2008.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[48] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Front. Neurosci.*, vol. 12, 2018, Art. no. 331.

[49] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems," *Front. Neurosci.*, vol. 15, 2021, Art. no. 638474.

[50] E. Forno, V. Fra, R. Pignari, E. Macii, and G. Urgese, "Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task," *Front. Neurosci.*, vol. 16, 2022, Art. no. 999029.

[51] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2020, pp. 8529–8533.

[52] M. A. Montemurro, M. J. Rasch, Y. Murayama, N. K. Logothetis, and S. Panzeri, "Phase-of-firing coding of natural visual stimuli in primary visual cortex," *Curr. Biol.*, vol. 18, no. 5, pp. 375–380, 2008.

[53] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.

[54] Y. Kim, H. Park, A. Moitra, A. Bhattacharjee, Y. Venkatesha, and P. Panda, "Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks?," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2022, pp. 71–75.

[55] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11 062–11 070.

[56] Y. Li, S. Deng, X. Dong, and S. Gu, "Converting artificial neural networks to spiking neural networks via parameter calibration," 2022, *arXiv:2205.10121*.

[57] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda, "Neural architecture search for spiking neural networks," in *Proc. 17th Eur. Conf. Comput. Vis.*, Springer, 2022, pp. 36–56.

[58] I. Corporation, "BootStrap SNN training," 2021. [Online]. Available: https://lava-nc.org/lava-lib-dl/bootstrap/notebooks/mnist/train.html

[59] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 8026–8037, 2019.

[60] I. Corporation, "PilotNet lif benchmarking example," 2022. [Online]. Available: https://github.com/lava-nc/lava-dl/blob/main/tutorials/lava/lib/dl/netx/pilotnet_snn/benchmark.ipynb

[61] G. Breed, "Bit error rate: Fundamental concepts and measurement issues," *High Freq. Electron.*, vol. 2, no. 1, pp. 46–47, 2003.

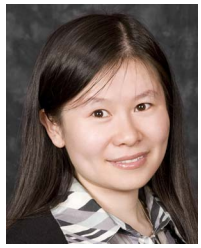
[62] 3rd Generation Partnership Project (3GPP), LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception, (TS 36.101v16.0.0), 2019.

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[64] M. Research, "Neural network intelligence," 2020. [Online]. Available: https://github.com/microsoft/nni

Shiya Liu received the BS and MS degrees in electrical engineering from Iowa State University, Ames, USA in 2013 and 2015, respectively. He is currently working toward the PhD degree with Bradley Department of Electrical and Computer Engineering at Virginia Tech, Blacksburg, USA. His research interests include machine learning, very large-scale integrated (VLSI) circuits, and neuromorphic computing.

Yibin Liang (Member, IEEE) received the MS degree in electrical engineering from Virginia Tech in 2004, where he is currently working toward the PhD degree in electrical engineering. His research interests include neuromorphic computing, machine learning, very large-scale integrated (VLSI) circuits, information theory, and wireless communication systems.



Yang Yi (Senior Member, IEEE) is an associate professor with Bradley Department of Electrical Engineering and Computer Engineering at Virginia Tech. Her research interests include very large-scale scale integrated (VLSI) circuits and systems, computer-aided design (CAD), neuromorphic architecture for brain-inspired computing systems, and low-power circuits design with advanced nano-technologies for high-speed wireless systems.