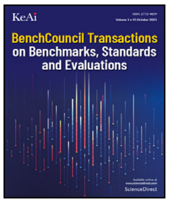
BenchCouncil Transactions on Benchmarks, Standards and Evaluations 3 (2023) 100108

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



Full length article

SNNBench: End-to-end AI-oriented spiking neural network benchmarking

Fei Tang [*], Wanling Gao

Research Center for Advanced Computer Systems, State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, China
University of Chinese Academy of Sciences, China

A R T I C L E   I N F O

Keywords:
Spiking Neural Networks
Artificial Intelligence
Deep learning
Benchmarking

A B S T R A C T

Spiking Neural Networks (SNNs) show great potential for solving Artificial Intelligence (AI) applications. At the preliminary stage of SNNs, benchmarks are essential for evaluating and optimizing SNN algorithms, software, and hardware toward AI scenarios. However, a majority of SNN benchmarks focus on evaluating SNN for brain science, which has distinct neural network architectures and targets. Even though there have several benchmarks evaluating SNN for AI, they only focus on a single stage of training and inference or a processing fragment of a whole stage without accuracy information. Thus, the existing SNN benchmarks lack an end-to-end perspective that not only covers both training and inference but also provides a whole training process to a target accuracy level.

This paper presents SNNBench—the first end-to-end AI-oriented SNN benchmark covering the processing stages of training and inference and containing the accuracy information. Focusing on two typical AI applications: image classification and speech recognition, we provide nine workloads that consider the typical characteristics of SNN, i.e., the dynamics of spiking neurons, and AI, i.e., learning paradigms including supervised and unsupervised learning, learning rules like backpropagation, connection types like fully connected, and accuracy. The evaluations of SNNBench on both CPU and GPU show its effectiveness. The specifications, source code, and results will be publicly available from https://www.benchcouncil.org/SNNBench.

1. Introduction

Spiking neural networks (SNNs) have gained considerable attention as a novel technology under development and are considered the third generation of ANNs [1]. Compared to the second-generation—DNNs, SNNs are more closely aligned with biological neural networks and use spiking neurons as computational units. Thus, SNNs support processing time-series information naturally, without requiring additional structures like Recurrent Neural Networks (RNNs), indicating a huge potential for time-series tasks like speech and natural language processing. Moreover, unlike the DNNs that perform layer-by-layer computations, SNNs are driven by sparse spiking events and can achieve high parallelism through asynchronous computations. Overall, SNNs promise to achieve higher performance, lower power consumption, and stronger expression ability [2], making them a compelling option for a wide range of AI applications. At the preliminary stages of SNNs, benchmarks lay the foundation for exploring the design space of corresponding algorithms, systems, and architectures. However, existing SNN benchmarks cannot fulfill the benchmarking requirements of the AI scenarios considering the complexities of training and inference and the tradeoff between high performance and high model accuracy.

On the one hand, most SNN benchmarks mainly focus on brain science evaluation [3–5], which is a mainstream research direction that models and simulates computational neuroscience to understand the principles of the nervous system [6]. In contrast to the evaluation of SNNs in AI, brain science evaluation focuses on more accurate simulations of neural models. This includes capturing voltage variations over time and reproducing spike statistics with high precision, often employing the highly complex Hodgkin–Huxley neuron model [7]. In contrast, SNNs used for AI applications prioritize the ability to solve specific AI problems over accurately simulating the behavior of real neural models, including voltage variance and spike activity. As a result, these SNNs often rely on simple neural models such as the leaky integrate-and-fire (LIF) model [8] which, due to its simple structure, is easy to train while still retaining important spike features [9]. Hence, the benchmarks for brain science cannot suit the evaluation of SNNs for AI [10] since they do not consider the specific AI problems and have distinct neural network architectures and targets.

On the other hand, two benchmarks have been proposed to evaluate the SNNs for AI [11,12]. One benchmark from Ostrau et al. [11] focuses on the inference stage by converting a pre-trained DNN model to an SNN model and providing accuracy information. However, it does not consider the training phase or other learning rules. Although another benchmark from Kulkarni et al. [12] includes both the training and inference stages, it employs much simpler neural network architectures

* Corresponding author.
E-mail addresses: tangfei@ict.ac.cn (F. Tang), gaowanling@ict.ac.cn (W. Gao).

https://doi.org/10.1016/j.tbench.2023.100108
Received 2 March 2023; Received in revised form 25 April 2023; Accepted 8 May 2023
Available online 12 May 2023

Table 1
Implications from SNNBench.

Method	Learning Rule	Support Unsupervised Learning	Achieve State-of-the-Art Accuracy	Accuracy Loss	Easy to Use	Stable	Scalability	Well Mapped to GPUs
Train from scratch	STDP	Yes	No	Not applicable	No	No	No	No
	Surrogate Backprop	No	Yes	Not applicable	Medium	Yes	Yes	Yes
Converted from pre-trained models	DNN-to-SNN	Not applicable[a]	Yes	Low	Yes	Medium[b]	Partial[c]	Partial[c]

[a]Depends on original DNNs.
[b]Depends on conversion quality, influenced by factors like DNN architecture and conversion methods.
[c]Depends on original DNNs and conversion quality.

Table 2
SNNBench and other relevant SNN benchmarks.

		Kulkarni et al.	Ostrau et al.	SNNBench
Domain	Vision	✓	✓	✓
	Speech			✓
Learning paradigm	Supervised			✓
	Unsupervised			✓
Connection type	One-to-one			✓
	Fully connected	✓	✓	✓
	Convolutional			✓
	Recurrent			✓
Learning rule	STDP			✓
	Backpropagation	✗[a]		✓
	DNN-to-SNN		✓	✓
	Reservoir	✓[b]		
	Evolutionary	✓[b]		
Number of different spiking neurons		1	1	2
Inference		✓	✓	✓
Training to quality				✓
Open source		×	✓	✓
Number of workloads		5	5	9

[a]This benchmark mentioned the backpropagation-based learning rule while only provided the inference stage (forward pass).
[b]Partial training process without accuracy information.

compared to realistic ones and simulates only a partial training process. Consequently, it does not offer any accuracy information, which is a crucial metric for AI. Moreover, even with a long enough training process, these neural network architectures are not verified to achieve convergent accuracy. In this condition, they fail to evaluate the training and inference performance of SNN comprehensively, and further cannot answer these questions: (1) how to design systems and architectures for SNN that achieve both high performance and high accuracy? (2) whether to train an SNN model or convert one from a pre-trained DNN model? (3) how to choose different training strategies like supervised or unsupervised, recurrent connection or fully connection?

In this paper, we propose an end-to-end AI-oriented benchmarking methodology. Here end-to-end has two-fold meanings: end-to-end evaluation for a real-world AI problem that covers both training and inference stages; end-to-end training that considers diverse strategies and achieves a target accuracy. Based on the methodology, we propose SNNBench, the first end-to-end AI-oriented SNN benchmark. Focusing on two typical AI applications: image classification and speech recognition, SNNBench provides nine workloads that cover the representative characteristics of SNN and AI. Specifically, from the perspective of SNN, we consider the dynamics of spiking neurons. In terms of AI, we consider diverse training strategies, including learning paradigms, i.e., supervised and unsupervised learning, four typical connection types, i.e., one-to-one, fully connected, convolutional, and recurrent, and three widely-used learning rules, i.e., STDP, backpropagation, and DNN-to-SNN. Table 2 provides a comparison of SNNBench with the other two relevant benchmarks.

Our experiments show the effectiveness of SNNBench. Through the evaluations on both CPU and GPU, we have several observations as follows:

(1) The workload characterization on SNNBench shows its diversity and representativeness. The workloads within SNNBench cover ten groups of diverse operators and have different dominant ones. Moreover, the experiments show the good reproducibility of SNNBench.
(2) Different from the previous work [13], we find that using STDP learning rule (88%) is hard to achieve the state-of-the-art convergence accuracy (99.91%) compared to the backpropagation (98%) and conversion-based learning rules (96.72%). Moreover, the convergence accuracies using STDP have much larger fluctuations than the other two rules, with a standard deviation higher than 2.4%, while the value is below 0.3% for the other two. In terms of the training cost, to train the same number of images, using STDP occupies 73X longer time compared to using backpropagation and 1559X compared to using conversion.
(3) GPU is not always the best for SNN. In our experiments, we found that when the number of neurons in a layer of SNN is small, like 400, the CPU performs better than the GPU. This could be due to the small size of the SNN networks, which leads to short GPU computation times that cannot offset the synchronization overhead between the CPU and GPU, or the software framework used for simulating SNNs may not be optimally designed for exploiting the full potential of GPUs. For recurrent networks, the training time on GPU using LIF neurons is 1.37 times that of on CPU. Using LSNN neurons, the gap is 1.22 times. In future work, we plan to explore larger SNN networks and further optimization of both the software framework and the mapping of SNN workloads to the GPU hardware.
(4) Even though SNNs have great potential for asynchronous parallelism, the corresponding hardware, software systems, and SNN network architectures fail to exploit this advantage and thus face poor inter-operator and intra-operator scalability currently.

Based on experiments from SNNBench, we present some insights from SNNBench in Table 1. Surrogate backpropagation and conversion-based methods are recommended, as they can achieve comparable accuracy to DNNs and require minimal modifications to existing DNNs. However, using surrogate backpropagation necessitates choosing suitable smooth functions and loss functions, which may require some professional expertise. There are existing conversion tools to convert DNNs to SNNs, so one may not even need to modify the existing DNNs, but the conversion quality can be affected by various factors. If only unlabeled data is available, STDP is the only choice, as it supports unsupervised learning, but it suffers from instability issues. We also find that the surrogate backpropagation method can utilize GPUs, while STDP is not well-mapped to GPUs.

We organize the rest of the paper as follows. Section 2 explains the related work. Section 3 illustrates the design methodology and implementation of SNNBench. Section 4 shows the experiments. Finally, we draw a conclusion in Section 5.

2. Related work

Several benchmarks have been proposed to study computational neuroscience, which employs mathematical models and computer simulations to understand how electrical and chemical signals process and represent information in the brain [6]. Brette et al. [3] simulated a network containing 4000 neurons, 80% of which were excitatory and 20% were inhibitory neurons, randomly connected with a probability of 2%. They proposed four benchmarks, each with the same network architecture but different combinations of spiking neurons and synaptic types, and provided simulation specifications that include

Hodgkin–Huxley (HH) and integrate-and-fire (IF) neuron models, as well as current-based and conductance-based synaptic types. These simulations were implemented using different simulators. Tikidji-Hamburyan et al. [4] simulated two networks, called Classical Pyramidal InterNeuron Gamma (PING) [14] and PostinhIbitory Rebound-InterNeuron Gamma (PIR-ING) [15]. These two networks are implemented using LIF and HH neurons, respectively. Van Albada et al. [5] modeled a network under one mm^2 of the surface of generic early sensory cortex, organized into multiple layers, including 77,169 neurons connected via approximately 3×10^8 synapses, which is a huge network for simulation for that time. The network architectures in these benchmarks are biologically realistic; they are not directly applicable to SNNs for AI, as spiking neuron models used for ANNs are highly abstract and only include basic features of spiking neurons, such as spike trains, thresholds, and spike firing. Thus, these computational neuroscience benchmarks are unsuitable for evaluating SNNs for AI.

There are also some benchmarks for AI tasks. Kulkarni et al. [12] selected five workloads to evaluate the performance of simulators and claimed that the benchmark could represent computer science and machine learning workloads instead of computational neuroscience. However, whether the network architecture used in the workload can achieve reasonable accuracy on real-world tasks has not been validated, which fails to reflect the state-of-the-art or state-of-the-practice works. Additionally, it only simulates the training or inference process, adopts indirect metrics such as operations per second, and ignores accuracy. Ostrau et al. [11] uses a converted SNN model from DNNs to measure the performance of neuromorphic hardware. It completely neglects the training process on neuromorphic hardware. And converting DNNs to SNNs is only one method of using SNNs, lacking the representativeness of different learning rules. These two benchmarks only cover a few aspects of SNNs oriented toward AI. This paper proposes a comprehensive and representative SNN benchmarking methodology—SNNBench. Table 2 lists these two benchmarks and SNNBench.

3. SNNBench design and implementation

In this section, we first introduce the requirements of SNN benchmarks and then illustrate the SNNBench methodology. Finally, we present the implementation of SNNBench in detail.

3.1. The requirements for SNN benchmarks

The existing SNN benchmarks either focus on brain science benchmarking instead of the ability to solve AI problems or only cover a partial evaluation of these abilities. Hence, we aim to benchmark the SNNs for real-world applications like artificial intelligence. To achieve this goal, the SNN benchmark needs to satisfy the following requirements.

(1) Covering representative real-world applications. A benchmark should have relevance to its target domain [16]. Thus, we should choose representative tasks and datasets for evaluation.
(2) Covering the typical characteristics of SNN. The SNN benchmark should consider the dynamics of spiking neurons, which contain the change of the membrane potential and firing spikes. Meanwhile, considering the benchmark is AI-oriented, suitable spiking neuron models should be selected.
(3) Covering the typical characteristics of deep learning. On the one hand, the benchmark should consider different learning approaches, like supervised, unsupervised, semi-supervised, and reinforcement learning. On the other hand, the benchmark should contain both training and inference phases. Important factors should be considered for different phases, like the diverse learning rules, spiking neurons, connection types, etc.
(4) Meeting the reproducibility and usability requirements of benchmarking. Reproducibility is of great significance for both benchmarking and deep learning communities. For benchmarking, good reproducibility guarantees the fairness and consistency of the evaluation results. For deep learning, considering its stochastic intrinsic [17], good reproducibility assures the stability of the performance and accuracy. Usability is another requirement for benchmarking. The benchmarks should be simple and have affordable evaluation costs [18].

3.2. SNNBench methodology

The methodology underpinning SNNBench is illustrated in Fig. 1. SNNBench represents an end-to-end benchmarking methodology in two distinct aspects. Firstly, it encompasses both training and inference, providing a comprehensive solution for AI tasks. Secondly, it trains the model to the target accuracy (to a convergent state) instead of simply mimicking a limited number of training iterations, which would be insufficient as accuracy can only be determined upon reaching the convergent state.

SNNBench selects representative AI tasks and datasets from vision, speech, and natural language processing (NLP) domains. Two representative tasks – image classification and speech recognition – and their corresponding datasets are chosen. Image data serves as an exemplar of static data, while speech data exemplifies temporal data.

In addition, SNNBench takes into account the characteristics of both SNNs and deep learning, including training paradigms, connection types, spiking neurons, and learning rules. Training paradigms encompass supervised and unsupervised learning. While supervised learning is widely employed in deep learning, unsupervised learning is less common in recognition tasks. However, due to learning rules such as STDP in SNNs, unsupervised learning can be applied to recognition tasks. Consequently, it is crucial to consider training paradigms.

SNNs exhibit various neuron organization types, and we select representative connection types from [13], as well as other commonly used connection types in deep learning. We exclude connection types that employ reservoir computing and evolutionary optimization learning rules (c and d in Fig. 2 in [19]) because these rules are infrequently used, and there is a promising trend toward combining deep learning and SNNs [20–23].

Moreover, SNNBench should account for different spiking neurons. Ultimately, SNNBench primarily includes learning rules such as STDP learning, surrogate backpropagation, and DNN-to-SNN conversion, while excluding reservoir computing and evolutionary optimization for the same reasons stated earlier regarding the choice of connection types.

SNNBench also addresses benchmarking requirements of usability and reproducibility. Further details on the methodology will be provided in this section.

3.2.1. Considering an end-to-end solution for AI tasks

The basic paradigm of AI is to train a model using the training dataset first and then make inferences on the test dataset. It is well-known that training and inference are different. The most significant difference is that training involves weight updating and even the evolution of network architectures, such as neural architecture search [24] and evolutionary optimization [25]. On the other hand, weights and architecture remain unchanged during inference. Hence, they have different workload characteristics, as verified in Section 4.2. As a direct result, hardware architectures designed for training and inference differ, like various DNN accelerators. If one only considers one of the phases, it may mislead the hardware design. Therefore, it is not sufficient to only consider training or inference.

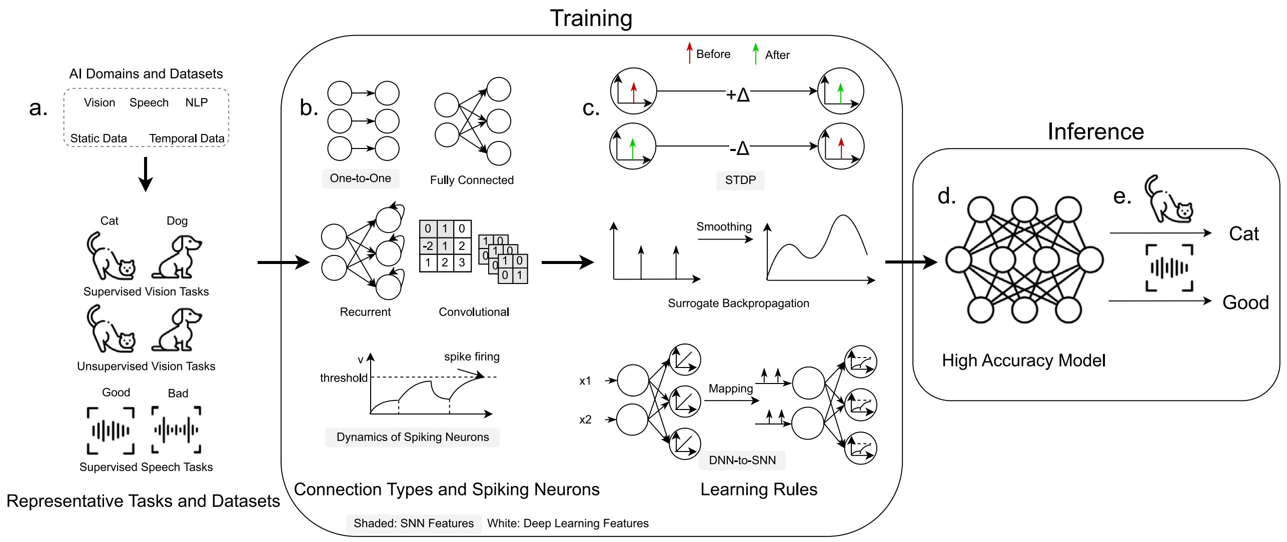


Fig. 1. SNNBench methodology. SNNBench is an end-to-end benchmarking methodology designed to cover both training and inference phases for SNNs while training the model to the target accuracy. (a) SNNBench selects representative tasks and datasets from vision, speech, and NLP domains, as well as static (image) and temporal (speech) data (upper part of (a)), while considering different training paradigms—supervised and unsupervised learning (bottom part of (a)). (b) A variety of network architectures can be built by combining different connection types, including one-to-one, fully connected, recurrent, and convolutional (upper and middle parts of (b)), and spiking neurons (bottom part of (b), where the potential dynamics of LIF neurons are used to represent spiking neurons). (c) SNNBench also takes into account different learning rules, such as STDP (upper part of (c), where connections are strengthened if pre-synaptic spikes occur before post-synaptic spikes, and vice versa), surrogate backpropagation (middle part of (c)), and DNN-to-SNN conversion (bottom part of (c)). (d) Once trained to the target accuracy (and converted to an SNN model if necessary), a high-accuracy SNN model is obtained. (e) The SNN model can then be employed to perform inference tasks on test data.

3.2.2. *Training the model to the target accuracy*

Many benchmarks use indirect metrics like operations per second because they are easy to measure. However, these metrics may not reflect whether a hardware system can solve real AI tasks. Different design strategies like float point precision can lead to non-objective assessments. For example, a hardware system that achieves high operations per second may not be able to train to the target accuracy if it adopts a low float point precision implementation. To address this issue, SNNBench trains the model to the target accuracy and uses accuracy as an important metric.

3.2.3. *Covering representative real-world applications*

SNNBench is designed for AI applications and focuses on image classification and speech recognition as benchmarking tasks. These two tasks are widely used in deep learning and serve as representative benchmarks. The benchmark suite includes state-of-the-art, state-of-the-practice, classical spiking neural architectures that are widely accepted and highly cited in SNN research. For the image classification task, SNNBench uses the Modified National Institute of Standards and Technology (MNIST) handwritten digits database [26]. For the speech recognition task, SNNBench uses the Speech Commands v2 dataset [27], which contains 105,829 audio files of spoken words and is widely used for simple speech recognition tasks.

3.2.4. *Covering the typical characteristics of SNN*

The SNNBench benchmark suite is designed to represent the typical characteristics of SNNs. SNNs use spike timing as the input and encode it as a series of 0 s and 1 s that indicate whether a post-synaptic neuron has received a spike from the pre-synaptic neuron at a given time. Upon receiving a spike, the membrane potential of the post-synaptic neuron increases. If it reaches a threshold, the neuron fires a spike, resetting the potential to the resting potential. The post-synaptic neuron is unresponsive to incoming spikes during the subsequent refractory period.

Many different neural models range from simple LIF models to complex Hodgkin–Huxley models. While more complex models are more accurate in simulating the neurons in the brain, they also have more parameters, which can make them difficult to train in a neural network. As a result, LIF-based models are widely used in the AI field due to their simplicity. These models have been implemented in various neuromorphic architectures, such as IBM's TrueNorth [28] and Intel's Loihi [29]. Complex models like the Hodgkin–Huxley model contain many components suitable for studying neural dynamics but are too complex to implement in cognitive neuromorphic architectures.

Therefore, SNNBench focuses on LIF-based spiking neurons and provides different LIF-based models for comparison in the speech recognition task.

3.2.5. *Covering the typical characteristics of deep learning*

SNNBench is a benchmark suite designed specifically for AI applications and, therefore, must capture the essence of deep learning. It covers multiple facets of AI, including various phases of learning, learning types, learning rules, and connection types. Unlike previous works that only consider the inference phase, SNNBench includes both the training and inference phases. The four primary types of AI learning are supervised, unsupervised, semi-supervised, and reinforcement learning. SNNBench supports supervised and unsupervised learning and plans to incorporate reinforcement learning in a future release. Supervised learning is the dominant paradigm in artificial intelligence, but its reliance on manual labeling of data presents a challenge in terms of cost and scalability. On the other hand, unsupervised learning does not require labeled data and more closely resembles how the brain learns.

Three main learning rules are used to train SNNs: spike-timing–dependent plasticity (STDP), surrogate backpropagation, and DNN-to-SNN conversion. The STDP learning rule is based on the biological principle of spike-timing–dependent plasticity [13] and is most similar to the way the brain learns. Backpropagation, which has been widely used in traditional ANNs and has produced remarkable results in fields such as computer vision, natural language processing, and robotics, cannot be directly applied to SNNs due to the discrete and non-differentiable nature of spikes. However, some variants of backpropagation, such as surrogate backpropagation, have been proposed and have achieved performance close to the state-of-the-art on some datasets [19]. The DNN-to-SNN conversion method involves mapping the weights of an DNN to spike firing probabilities. Common techniques to mitigate accuracy loss include using the ReLU [30] activation function, fixing the bias to zero during training, weight normalization, and

Table 3
Workloads from SNNBench. Except for Image-Conversion, which only includes inference workloads, all other workloads consist of both training and inference workloads, resulting in a total of 9 workloads.

Benchmark	Task	Dataset	Network layout	Learning paradigm	Spiking neuron	Connection type	Learning rule	Spike encoding	Accuracy	Note
Image-STDP	Image classification	MNIST	One input layer, one excitatory layer and one inhibitory layer	Unsupervised	LIF	Fully connected, One-to-one	STDP	Rate encoding	95%	STDP learning strategy is more similar as the brain works, and this task is the representative of the unsupervised task
Image-Backprop	Image classification	MNIST	Two convolutional layers and one fully connected layer	Supervised	LIF	Convolutional	Surrogate Backpropagation	Rate encoding	95%	Using surrogate backpropagation is another common way to use SNN
Image-Conversion	Image classification	MNIST	Three fully connected layers	Supervised	LIF	Convolutional	Backpropagation	Rate encoding	95%	One commonly method to use SNN
Speech-LIF	Speech recognition	Google Speech Commands v2	One input layer, one recurrent layer and one readout layer	Supervised	LIF	Recurrent	Surrogate Backpropagation	Rate encoding	90%	Recurrent SNN with the LIF neuron
Speech-LSNN	Speech recognition	Google Speech Commands v2	One input layer, one recurrent layer and one readout layer	Supervised	LSNN	Recurrent	Surrogate Backpropagation	Rate encoding	90%	Recurrent SNN with the LSNN neuron

threshold tuning [31,32]. This conversion-based approach has achieved competitive performance compared to traditional DNNs and is widely used. SNNBench includes workloads with all three learning rules.

In the brain, neurons with similar functions are organized into a neural population group. Similarly, in deep learning, neurons are organized layer-by-layer into distinct connection types, such as fully connected, convolutional, and recurrent layers. SNNBench provides workloads with different connection types to take advantage of the mature and efficient connection types developed in deep learning.

3.2.6. *Meeting the reproducibility and usability requirements of benchmarking*

To ensure the reproducibility and usability of our benchmarks, we have taken several steps to address the intrinsic stochastic nature of AI. Firstly, AI algorithms typically rely on randomness, such as choosing a random initial state for training and shuffling the input data, to enhance their robustness. However, these methods can also make it challenging to reproduce benchmark results. Secondly, the operations used in deep learning algorithms, such as convolution and matrix multiplication, have various implementations, which can further increase the volatility of benchmark results [33]. To mitigate this, we have set the same random seed for all benchmarks, including PyTorch, Python, and NumPy random libraries, ensuring that the initial states and input order remain consistent with each run. We have also disabled the cuDNN benchmarking feature that selects the most efficient convolution implementation in time. Instead, we have made PyTorch choose a deterministic algorithm for all operations, ensuring that the same algorithms and implementations are used for each run. Additionally, to improve usability, we have used Docker to set up a consistent experiment environment for each run of the benchmark.

3.3. *SNNBench implementation*

Table 3 outlines the workloads from SNNBench, which includes image classification and speech recognition tasks. The MNIST dataset is utilized for the image classification task, while the Google Speech Commands v2 dataset is used for the speech recognition task, both of which are described in detail in Section 3.2.3. Except for Image-Conversion, which only contains inference workload, all the benchmarks contain both training and inference workloads. In this subsection, we delve into the implementation of SNNBench.

3.3.1. *Image-STDP*

This benchmark involves an image classification task that trains an SNN network using the STDP learning rule. We have selected the most classic and widely cited STDP learning rule [13], which is representative of unsupervised learning tasks and has significantly impacted subsequent research. Our implementation is based on the BindsNet framework [34].

The network architecture consists of input, excitatory, and inhibitory layers, and the excitatory and inhibitory layer contain the same number of neurons. The input layer contains 28×28 neurons, corresponding to the MNIST dataset's image size. The excitatory and inhibitory layers simulate excitatory and inhibitory neurons, respectively.

The connection between the input layer and the excitatory layer is a fully connected one, while the connection between the excitatory and inhibitory layers follows a one-to-one map-style pattern, with each inhibitory neuron connecting to all excitatory neurons except the one it is connected to. The input layer accepts 28×28 spike trains generated using a Poisson distribution, where the parameter α is proportional to the corresponding pixel's intensity.

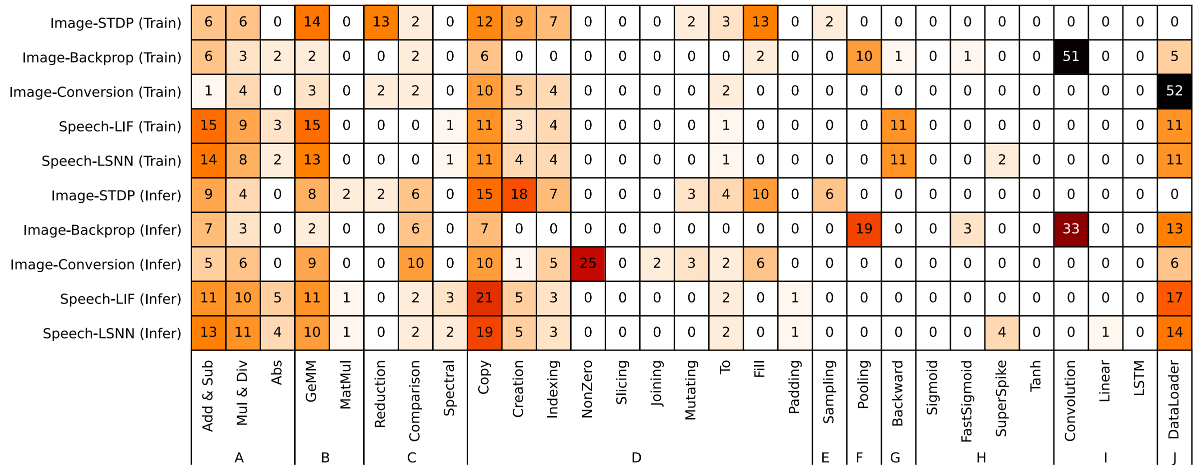
Workload #1: Image-STDP Training workload uses the STDP learning rule to update the synaptic weights, where stronger connections are formed if pre-synaptic neurons consistently lead to post-synaptic neurons firing, and weaker connections are formed if the opposite is true. This reflects the impulsiveness of the brain, where some neurons enhance the reaction while others prevent it. The excitatory and inhibitory neurons both have LIF behavior but opposite weight-updating strategies. We check the accuracy of every iteration and train the model to the convergent state, and check if it reaches the target accuracy.

Workload #2: Image-STDP Inference workload uses the trained model to infer on the test dataset. After training, the label of the image that elicits the most firing in a neuron is assigned to that neuron. During inference, the label of the image is assigned based on the most fired neurons. Counting the spiking distributions can get the inference result.

We study the STDP learning rule in different scales by implementing SNNs with 100, 400, 1600, and 6400 neurons in the excitatory and inhibitory layers.

3.3.2. *Image-Backprop*

This benchmark is also an image classification task but uses surrogate backpropagation. Backpropagation is the basis of the success of deep learning, and large models and big-volume data can be trained using this learning rule. However, backpropagation cannot be directly applied to SNNs because of the non-differentiable spikes. Even though the STDP learning rule can be used to train SNNs, the training has many problems that need to be solved; for example, the neurons close to the output layer rarely fire in deep SNNs so that SNNs' networks cannot be constructed as deep as current deep neural networks. Since the lack of efficient learning rules, many researchers have focused on training SNNs through backpropagation using a workaround approach. Surrogate gradient descent is one of the popular methods. It replaces the non-differentiable part with a differentiable function, for example, using an approximate function to replace the derivative of the spike to the membrane potential. SuperSpkie [35] is one of the representatives of this method. It uses the fast sigmoid function to approximate the Dirichlet function of firing spikes so that the gradient can be calculated in the backward pass. And it does not modify the forward pass, so the



	A			B		C			D										E	F	G	H				I			J
	Add & Sub	Mul & Div	Abs	GeMM	MatMul	Reduction	Comparison	Spectral	Copy	Creation	Indexing	NonZero	Slicing	Joining	Mutating	To	Fill	Padding	Sampling	Pooling	Backward	Sigmoid	FastSigmoid	SuperSpike	Tanh	Convolution	Linear	LSTM	DataLoader
Image-STDP (Train)	6	6	0	14	0	13	2	0	12	9	7	0	0	0	2	3	13	0	2	0	0	0	0	0	0	0	0	0	0
Image-Backprop (Train)	6	3	2	2	0	0	2	0	6	0	0	0	0	0	0	0	2	0	0	10	1	0	1	0	0	51	0	0	5
Image-Conversion (Train)	1	4	0	3	0	2	2	0	10	5	4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	52
Speech-LIF (Train)	15	9	3	15	0	0	0	1	11	3	4	0	0	0	0	1	0	0	0	0	11	0	0	0	0	0	0	0	11
Speech-LSNN (Train)	14	8	2	13	0	0	0	1	11	4	4	0	0	0	0	1	0	0	0	0	11	0	0	2	0	0	0	0	11
Image-STDP (Infer)	9	4	0	8	2	2	6	0	15	18	7	0	0	0	3	4	10	0	6	0	0	0	0	0	0	0	0	0	0
Image-Backprop (Infer)	7	3	0	2	0	0	6	0	7	0	0	0	0	0	0	0	0	0	0	19	0	0	3	0	0	33	0	0	13
Image-Conversion (Infer)	5	6	0	9	0	0	10	0	10	1	5	25	0	2	3	2	6	0	0	0	0	0	0	0	0	0	0	0	6
Speech-LIF (Infer)	11	10	5	11	1	0	2	3	21	5	3	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	17
Speech-LSNN (Infer)	13	11	4	10	1	0	2	2	19	5	3	0	0	0	0	2	0	1	0	0	0	0	0	4	0	0	1	0	14

Fig. 2. Operator heatmap of SNNBench. All operators are divided into ten groups: Pointwise (A), BLAS (B), Math (C), Tensor-Related (including Creation, Indexing, Slicing, Joining, and Mutating) (D), Sampling (E), Pooling (F), Backward (G), Activation (H), Layer-Computing (I), and Data-Loading (J), as depicted in the bottom part of the figure. The left part displays individual workloads, with "(Train)" indicating a training workload and "(Infer)" indicating an inference workload. The time cost of each operator is counted, and the numbers in the grid represent the percentage of time consumed by each operator for each workload. To enhance visibility, a deeper color is used to indicate a larger percentage.

network architecture is similar to DNN, except it has different computing units. We adopt an implementation that is a simple convolution network containing two convolution and max pooling layers and a fully connected layer as the output layer. All the computing units are LIF neurons.

Workload #3: Image-Backprop Training workload uses the fast sigmoid to smooth the spiking train and trains the neural work using backpropagation.

Workload #4: Image-Backprop Inference workload does not involve the smoothing progress used in the training workload, and it directly computes the result through the forward pass.

3.3.3. *Image-Conversion*

This benchmark is an image classification task that uses the DNN-to-SNN conversion method, which is one of the strategies to overcome the challenge of training SNNs. The conversion-based method trains a DNN model first and then maps the trained model to an SNN model. The common approach to mapping deep neural networks (DNNs) to spiking neural networks (SNNs) is to replace real values in the DNNs with spiking frequencies and to replace activation functions with spiking neurons. In the training phase, Diehl et al. [32] use ReLUs as activation functions for all network units and eliminate biases. The ReLU function ensures that the values are non-negative, which SNNs cannot represent, while the biases are fixed at zero. To further improve the performance of the SNN, Diehl et al. use two weight normalization methods: model-based normalization and data-based normalization. Model-based normalization normalizes weights based on the trained model's weights, while data-based normalization normalizes weights based on the training data. In this workload, a multi-layer perception network with three layers is used, and each layer is followed by a ReLU activation function except for the output layer. The DNN model is trained to convergence, then converted to an SNN model and normalized using data-based normalization. The converted SNN model is then used for inference on the test dataset. This conversion-based method is a popular approach to overcoming the challenges of training SNNs and has attracted much attention from researchers.

Workload #5: Image-Conversion Inference workload is performed after converting the trained DNN model to the SNN model. We also check the accuracy drop after converting.

3.3.4. *Speech-LIF and Speech-LSNN*

We present two benchmarks that use the same network architecture and learning rule but with different spiking neurons to assess the impact on performance. Both workloads utilize the surrogate backpropagation method and LIF-based neurons. One workload uses traditional LIF neurons, while the other uses the LSNN neuron, a LIF variant with adaptive thresholds that are dynamic during training but fixed during inference. Additionally, we evaluate a workload that uses standard activation functions as computing units, serving as the baseline for deep neural network (DNN) performance.

Workload #6: Speech-LIF Training workload uses the recurrent neural network, is built with LIF neurons, and uses the surrogate backpropagation learning rule.

Workload #7: Speech-LIF Inference workload uses the same neural architecture as the training workload and infers on the test dataset.

Workload #8: Speech-LSNN Training workload is the same as Speech-LIF Training workload, but is built with LSNN neurons.

Workload #9: Speech-LSNN Inference workload is the same as Speech-LIF Inference workload, but is built with LSNN neurons.

4. Experiment

In this section, we conduct a series of experiments to show the effectiveness of SNNBench. First, we perform workload characterization on SNNBench in Section 4.2 and show the reproducibility of SNNBench in Section 4.3. Then, we compare the impact of learning rules and computing units on the SNN training and inference performance in Section 4.4 and Section 4.5, respectively. Finally, we evaluate the scalability of SNNBench in Section 4.6.

4.1. *Experiment setup*

We deploy SNNBench on a server node equipped with two Intel Xeon E5-2620 v3 @ 2.40 GHz CPUs and four Nvidia GeForce RTX 2080-Ti GPU cards. Each CPU contains six physical cores and enables hyper-threading. The software versions are CUDA toolkit 10.2, Python 3.10 and Pytorch 1.12. We have mainly investigated three AI-oriented SNN frameworks: BindsNet, snnTorch, and Norse. Due to the distinct APIs and functions provided by these frameworks, we have adopted the following strategy for simplicity and convenience: We utilized BindsNet 0.3.1 for both STDP and DNN-to-SNN methods in the image classification task, employed snnTorch 0.5.3 for surrogate backpropagation in the image classification task, and used Norse 0.0.7 for all speech recognition workloads. To make the experimental environment consistent for each run and avoid the performance drop due to default security settings, we use Docker to build the environment and disable Docker's seccomp security option.

Table 4
Variations in loss and accuracy after each epoch during the training process for speech recognition tasks. This table focuses exclusively on speech recognition tasks, as these metrics remain consistent across each run for image classification tasks. The differences among three runs after each epoch are highlighted in bold, illustrating that the discrepancies are minimal.

Epoch	Speech-LIF						Speech-LSNN					
	CPU			GPU			CPU			GPU		
0	3.5458 (4%)	3.5477 (3%)	3.5945 (4%)	3.5963 (2%)	3.5207 (3%)	3.5659 (4%)	2.7021 (4%)	2.7195 (3%)	2.7290 (4%)	2.7219 (4%)	2.7114 (3%)	2.7210 (4%)
1	1.3690 (63%)	1.3698 (63%)	1.3693 (63%)	1.3663 (63%)	1.3587 (63%)	1.3614 (63%)	1.9193 (62%)	1.9206 (62%)	1.9176 (62%)	1.9189 (62%)	1.9182 (62%)	1.9168 (62%)
2	1.3185 (63%)	1.3076 (63%)	1.3036 (63%)	1.3072 (63%)	1.3003 (63%)	1.3061 (63%)	1.8503 (62%)	1.8540 (62%)	1.8523 (62%)	1.8504 (62%)	1.8504 (62%)	1.8497 (62%)
3	1.2794 (63%)	1.2728 (63%)	1.2709 (63%)	1.2679 (63%)	1.2682 (63%)	1.2711 (63%)	1.8077 (62%)	1.8100 (62%)	1.8095 (62%)	1.8063 (63%)	1.8081 (63%)	1.8118 (62%)
4	1.2500 (63%)	1.2466 (63%)	1.2486 (63%)	1.2460 (63%)	1.2459 (63%)	1.2466 (63%)	1.7831 (63%)	1.7898 (62%)	1.7911 (63%)	1.7847 (63%)	1.7890 (63%)	1.7931 (63%)
5	1.2298 (63%)	1.2204 (63%)	1.2198 (63%)	1.2182 (64%)	1.2228 (63%)	1.2238 (63%)	1.7625 (63%)	1.7696 (62%)	1.7708 (63%)	1.7637 (63%)	1.7713 (63%)	1.7715 (63%)

We use PyTorch Profiler [36] to collect the runtime information for all the experiments, including the involved operators, the input size of each operator, and the time consumption. We report the profile data on the CPU for simplicity and veracity since a mass of operations like memory synchronization on GPUs would interfere with the analysis. We run each benchmark ten times and report the average values, each containing a two-batch warm-up stage (see Fig. 2).

4.2. Workload characterization

In this experiment, we conduct a top-down analysis for each workload. Considering that training and inference are the two most consuming parts, we only profile these two phases and exclude other phases like model initialization.

Fig. 2 shows the operator heat map of SNNBench workloads. We classify these operators into ten groups, and they are Pointwise (A), BLAS (B), Math (C), Tensor-Related (Creation, Indexing, Slicing, Joining, and Mutating) (D), Sampling (E), Pooling (F), Backward (G), Activation (H), Layer-Computing (I), and Data-Loading (J). From the result, we find that for most workloads, the Pointwise, BLAS, Tensor-Related (especially copy, creation, and indexing), and Data Loading operators consume a lot of time. The framework uses tensor as the basic data structure and implements many operators based on BLAS libraries so that Pointwise and BLAS operations consume most reasonably. However, Image-Backprop (Train), Image-Backprop (Infer), and Image-Conversion (Train) consume little time on BLAS operations. This is because the Image-Backprop use convolution connections, so the convolution operator occupies a lot of time while the BLAS operation takes up very little. As for Image-Conversion (Train), it uses only three fully-connected layers, and the input sizes are small, thus, the computing process is fast, and the general matrix multiplication (GeMM) operator occupies relatively much less compared to data loading and copy operators. When the data loading time ratio decreases, the GeMM time ratio correspondingly increases a lot (from 2% to 9% in Image-Conversion (Infer)). From the perspective of each operator category, for Pointwise operators, add, sub, mul, and div operators consume the most time. And as for BLAS operators, GeMM takes up almost all the time while matmul barely does. For Tensor Related operators, tensor copy, creation, and indexing occupy the most time. From the perspective of workloads, Image-STDP does not spend much time in data loading and uses sampling operators to construct the inputs; these characteristics are different from other workloads. Image-Backprop is more like a traditional convolution neural network, except that it uses the LIF neuron as the computing unit, so it spends much time in convolution and pooling operators. Image-Conversion trains an DNN model and converts the well-trained DNN model to an SNN model, and uses the SNN model to infer, so its training process is exactly the same as the traditional fully connected networks. In the inference phase, it spends much more time in the nonzero operator than other directly trained SNN models. The cause is that the converted SNN model has high spike firing rate than other directly trained SNN models. Thus the converted model has more non-zero values in a tensor; in other words, data have higher density. And the more dense the data, the more time the nonzero counting operator spends. So the nonzero operator occupies high. We did a small validation experiment for a size of (10000, 64) tensor using random initialization and zeros initialization, performing the nonzero operator on these two tensors costs 603.57 us and 56.32 us, respectively. The performance gap is one order of magnitude. This also implies that the impact of different input characteristics on performance is enormous. The workload characteristics are almost the same for the two speech recognition tasks because they use similar learning rules and are only different in computing units. They differ from image classification tasks in that they need to process speech data so that they contain the spectral operator (FFT transformation in this case). The backward operator also occupies a high total time that image classification tasks do not.

In this experiment, we perform a top-down analysis for each workload. Our findings reveal that the majority of workloads allocate considerable time to Pointwise, BLAS, Tensor-Related, and Data Loading operators. Image-Backprop and Image-Conversion workloads display distinct characteristics compared to others, such as reduced BLAS operation time and elevated spike firing rate in the converted SNN model, impacting performance. The workload characteristics remain consistent between the two speech recognition tasks, as they employ similar learning rules and vary only in computing units. Additionally, they incorporate the spectral operator, which is absent in image classification tasks.

4.3. Reproducibility

SNNBench applies several strategies to eliminate the randomness mentioned in Section 3.2.6. In this subsection, we evaluate the reproducibility of SNNBench by running each benchmark multiple times on the same hardware and software systems.

Our experiments show that the image classification task obtains exactly the same results for different runs. The deviations for the speech recognition task are very slight. Table 4 shows the changes in loss and accuracy after each epoch during the training process for the speech recognition task. Note that we do not list the results of the three image classification workloads because their changes in loss and accuracy are exactly the same. The results show that the stochasticity of SNNBench is small enough to meet the benchmarking requirement for reproducibility.

4.4. Comparison of different learning rules

In this subsection, we use the three image classification workloads as an example to compare different learning rules.

4.4.1. Accuracy comparison

Fig. 3(a) demonstrates the training progress of the three image classification workloads with different learning rules and the upper part of Table 5 lists the convergent accuracy and time to achieve that accuracy. We trained MNIST for 10 epochs on each workload to ensure that all of them reached a convergent state and achieved the respective accuracy range for each learning rule, thereby ensuring convergence for all workloads. For Image-Backprop and Image-Conversion workloads, the accuracy reached approximately 98%, with no further improvement. For the Image-STDP workloads, we checked the accuracy and updated the weights every 250 samples during 10 epochs, resulting in a total of 2400 accuracy checks and weight updates (only 50 of which are presented in Fig. 3(a) for clarity). This training process took almost

Table 5
Accuracy, epochs to convergent accuracy, and time to achieve accuracy on testsets.

	Accuracy on testset	Epochs to accuracy	Time to accuracy on CPU (s)	Time to accuracy on GPU (s)
Image-STDP (100)	75%	1	25858.14	34812.06
Image-STDP (400)	83%	1	32379.48	31739.34
Image-STDP (1600)	88%	4	363866.4	137905.2
Image-STDP (6400)	84%	6	12248568	355468.32
Image-Backprop	**98.59%**	3	1060.43	221.36
Image-Conversion	97.4%	5	**82.91**	**58.23**
Speech-LIF	67%	40	22773.35	31191.47
Speech-LSNN	63%	1	**679.20**	**828.51**
Speech-LSTM	**90%**	43	19908.03	8201.03

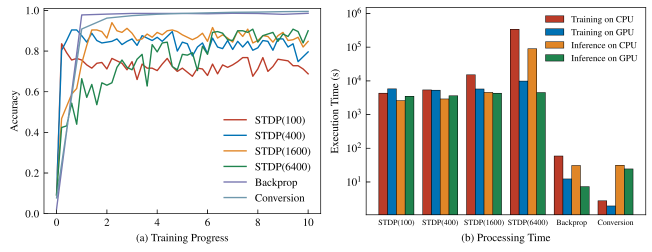


Fig. 3. Changes of accuracy and execution time using three different learning rules on the image classification task. For the Image-STDP workload, the accuracy is presented five times per epoch, and for other workloads, once per epoch. The numbers inside the brackets after STDP indicate the STDP workloads for different network sizes, which correspond to the number of spiking neurons in the excitatory and inhibitory layers. The time in the figure is to process the train (10000 images) and test (10000 images) dataset and has been processed using logarithms.

a week to complete on a 2080 Ti GPU for the network of 6400 neurons (6.86 days). In conclusion, the accuracy of all workloads remained stable, with no significant improvement observed over time, indicating that they reached a convergent state.

For the Image-STDP workload, the origin paper [13] achieved 82.9%, 87.0%, 91.9%, 95.0% accuracy when the number of excitatory and inhibitory neurons are set to 100, 400, 1600, and 6400, respectively. We achieve 75%, 83%, 88%, and 84% accuracy after convergence and get 84.8%, 93.2%, 96.0%, and 94.8% best accuracy during the training using 100, 400, 1600, and 6400 neurons, respectively. The number in the bracket is the number of spiking neurons in the excitatory and inhibitory layers. We can see that the higher the number of spiking neurons, the more epochs required to the convergent state, and the higher accuracy relatively, which is intuitive. When the number of neurons used is below 400, the accuracy reaches a relatively stable state after only one epoch. When the number of neurons is more than 400, there are more epochs needed to train to the convergence state. When the number is 6400, there even need 6 epochs to the convergence state. However, the training process is far less stable than other learning rules. After training to the convergence state, the standard deviation of accuracy is bigger than 2.4%, while other learning rules are less than 0.3%.

For the Image-Backprop workload, after training one epoch, the accuracy of the model reaches 97.73%, which can be compared with similar architecture DNNs. And the training and inference time are only 1/73 and 1/84 of the STDP learning rule. We think this is because the surrogate backpropagation can train the data batch by batch, but the STDP learning rule can only train the data in one sample once. The surrogate backpropagation can fully use the underlying parallel computing hardware.

For the Image-Conversion workload, after training the network to the convergent state, we get 97.76% accuracy. We export the trained model, transform the model to the SNN model, and apply the data-based weight normalization, we test that the accuracy of the converted SNN model is 96.72%, which is a 1.03% accuracy drop. This is acceptable for applications that are not sensitive to accuracy.

4.4.2. *Performance comparison*

Fig. 3(b) shows the processing time for training and inferring 10000 images on CPU and GPU, respectively. For training, when the number of neurons is 100, processing images on the CPU is faster than on GPU, but when the number of neurons exceeds 100, training on GPU is faster. And When the number of neurons changes from 400 to 1600 to 6400, the time cost on the CPU increases significantly. However, the time cost on the GPU is always at the same level for 100, 400, 1600, and 6400 neurons. For inference, when the number of neurons is 1600, processing on the CPU costs more time than on the GPU. This phenomenon is related to how GPUs work. When the number of neurons is small, the GPU's large number of parallel computing units may not be fully utilized, leading to less efficient performance compared to the CPU. Additionally, the lower clock frequency of the GPU could contribute to the observed performance difference. While memory synchronization latency may also play a role, it is likely not the primary reason for the performance discrepancy in this case. As a result, the approach of using GPU acceleration may not be applicable or beneficial for all cases, particularly when working with smaller neural networks.

4.4.3. *Overall comparison*

Table 1 shows the comparison of different learning rules. We cannot achieve the same level of accuracy as the state-of-the-art work using the STDP learning rule as described in [13], and the accuracy the STDP learning rule (88%) achieved is below the surrogate backpropagation (98%) and conversion (96.72%). Meanwhile, the training progress of the STDP learning rule fluctuates greatly even after reaching the convergence state. The standard deviation of accuracy is high as 2.4%, while other learning rules do not exceed 0.3%. This shows that the STDP learning rule is hard to train. STDP is unsupervised, making it considerably more complex to achieve good results compared to supervised backpropagation. This complexity might be the reason for the observed instability. On the other hand, surrogate backpropagation and conversion-based learning rules are easy to train, and there is a huge advantage that they have less modification to existing work than the STDP learning. Thus, they can fully use the current deep learning work, including mature architectures like ResNet [37], EfficientNet [38], and Transformer [39] and high-performance hardware and software systems, like GPUs, TensorFlow, and PyTorch. Despite the disadvantages compared to those two learning rules, the most significant advantage of the STDP learning rule is that unsupervised learning can be used and can avoid the tedious manual labeling of the training dataset. And it has the potential to construct more robust networks that can also perform well for unseen data.

Based on the experimental results presented above, we can summarize the following recommendations for using SNNs:

- Employing Surrogate-Backpropagation and DNN-to-SNN methods is highly recommended, as these two approaches can achieve accuracy comparable to that of DNNs.
- When using the Surrogate-Backpropagation method, it is crucial to find an appropriate activation function that smooths spike peaks well. In the case of DNN-to-SNN, an effective conversion

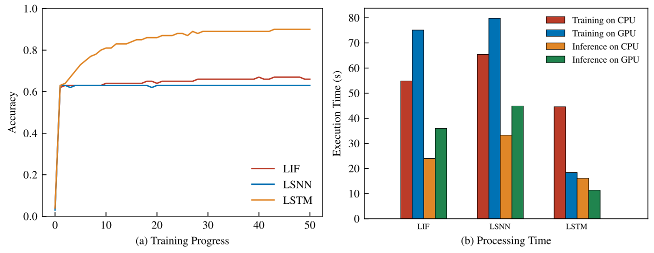


Fig. 4. Changes of accuracy and execution time using different spiking neurons compared to the LSTM unit on the speech recognition task. The time in the figure is to process the train (10000 spoken voices) and test (10000 spoken voices) dataset.

method is required. Otherwise, a significant decrease in accuracy may occur. However, both methods can fully exploit existing DNN works, including algorithms, hardware, and software. The specific choice between these two can be determined based on the actual situation.

- The STDP training rule is relatively challenging to use. Firstly, it is difficult to train and may not necessarily achieve accuracy comparable to the previous two methods. Secondly, the training speed is too slow, and current SNN frameworks cannot efficiently map the STDP algorithm to GPUs. However, if unsupervised learning is desired, the STDP learning rule is the only option.

4.5. Comparison of different spiking neurons

In addition to the two SNN workloads, we add a DNN workload with the same architectures but use the LSTM unit for comparison. We train the three different neural networks on the Google Speech Commands v2 dataset to the convergence state, and Fig. 4 shows the result. And the bottom part of Fig. 5 lists the convergent accuracy and the time taken to achieve that accuracy. The accuracy reaches 62%∼63% after the first epoch for all these three neural networks. After that, the accuracy of neural networks using LIF and LSNN remains at the same level as the first epoch, but the accuracy of the neural network with the LSTM unit reaches 90% after reaching the convergence state. Using the same network architecture, two SNN networks achieve lower accuracy than DNN networks, this may be because the parameters of the SNN network have not been sufficiently tuned, such as the threshold value of firing spikes. [40] indicated that LSNN has better performance than LIF, but the accuracy of LIF and LSNN are similar, which may prove that the parameters of the SNN networks are not tuned well. However, this paper primarily focuses on benchmarking rather then the algorithm, we did not spend much time on parameter optimization, as it is beyond the scope of this work. Even though we can fine-tune the parameters the final accuracy gap is large (67% vs. 90%). Hence we can conclude that for the same architecture, SNN can hardly achieve the accuracy that DNN achieved. In terms of ease of use, SNNs are currently not comparable to DNNs because even though the user does not carefully adjust the LSTM parameters, good results can be achieved.

In terms of processing speed, the LSTM neural network is the fastest in both training and inference. For training one epoch, the time spent on the LIF, LSNN, and LSTM is 779.787 s, 828.515 s, and 190.721 s. The time of the LIF is 4.09 times that of the LSTM. For inferring the test dataset, the time spent on the LIF, LSNN, and LSTM is 39.5661 s, 49.4007 s, and 12.4881 s. The time of the LIF is 3.17 times that of the LSTM. For the same network architecture, using spiking neurons speeds more times than that using non-spiking neurons. This may be due to the framework's inability to effectively map the SNN operators to GPUs.

4.6. Scalability

There are many parallelism methods for accelerating deep learning, such as data parallelism and model parallelism. We can generalize all parallelism patterns into inter-operator parallelism and intra-operator parallelism. Inter-operator parallelism means that operators are mapped to different computing units to execute, and intra-operator parallelism means that one operator is sliced to multiple parts mapped to different computing units. Thus data parallelism and model parallelism can be regarded as one type of inter-operator parallelism. The basic computing unit is a hardware thread within a node or within a distributed system. Reasonably splitting operators and slicing operators according to workloads' characteristics and mapping them to different threads are the key to fully utilizing the system. In this subsection, we investigate the potential of these workloads.

To improve inter-operator parallelism, a common method is to schedule non-dependent operators to execute in parallel. However, this is dependent on the degree of parallelizability of the network architecture, and the execution plan of operators needs to be carefully orchestrated to ensure that dependencies are respected and synchronization points are properly managed. To improve intra-operator parallelism, it is important to consider the differences between stateful networks like SNNs and RNNs, and stateless networks like CNNs and fully connected networks. In stateful networks, SNNs and RNNs are computed sequentially, with states stored for the next computation, making it more challenging to improve intra-operator parallelism. In contrast, CNNs and fully connected networks are stateless, allowing input samples to be split into multiple parts and computed independently, which makes it easier to improve intra-operator parallelism. However, for operators like matrix multiplication and element-wise operations, parallel execution can be achieved on different hardware threads using mature BLAS libraries. This allows for full hardware performance without the need for an elaborate execution plan.

To assess the inter-operator and intra-operator parallelism, we control the thread number of PyTorch's inter-operator and intra-operator thread pool for scheduling inter-operator and intra-operators separately. Although we choose the PyTorch framework as our experimental environment, other frameworks are also applicable, such as TensorFlow, which also has the same thread pool for scheduling inter-operators and intra-operators. For measuring the inter-operator and intra-operator parallelism, we fix the number of inter- and intra-operator threads to 1, respectively, and adjust the number of intra- and inter-operator threads from 1 to 12, respectively, to measure the training time for 10 iterations. Fig. 5 illustrates the inter- and intra-operator parallelism for the SNNBench workloads. As we analyze above, improving the inter-operator parallelism is hard, so we can find that increasing the thread number of inter-operators hardly changes the training time in all the workloads. However, we can find that increasing the thread number of intra-operators only improves the performance of Image-Backprop. For other workloads, more intra-operator threads have little impact and even downgrade the performance, which is counter-intuitive. To explain this phenomenon, we need to analyze the intra-operator execution at a finer granularity.

We count the time of every operator and compare the time executed in a different number of intra-operator threads. We select the most consumed operators in each workload and show the results in Fig. 6. We can find that the most consumed operators in the Image-Backprop workload have good scalability as the number of threads increases so that it can get better performance with more threads. As for other workloads, most workloads have worse performance overall, including Image-Conversion, Speech-LIF, Speech-LSNN, and Speech-LSTM workloads. Different operators have opposite reactions when increasing the number of threads; thus, the overall effect on performance is unpredictable. For the Image-STDP workload, the most three cost operators can be accelerated by increasing the number of threads, but the performance can no longer be improved when the number of

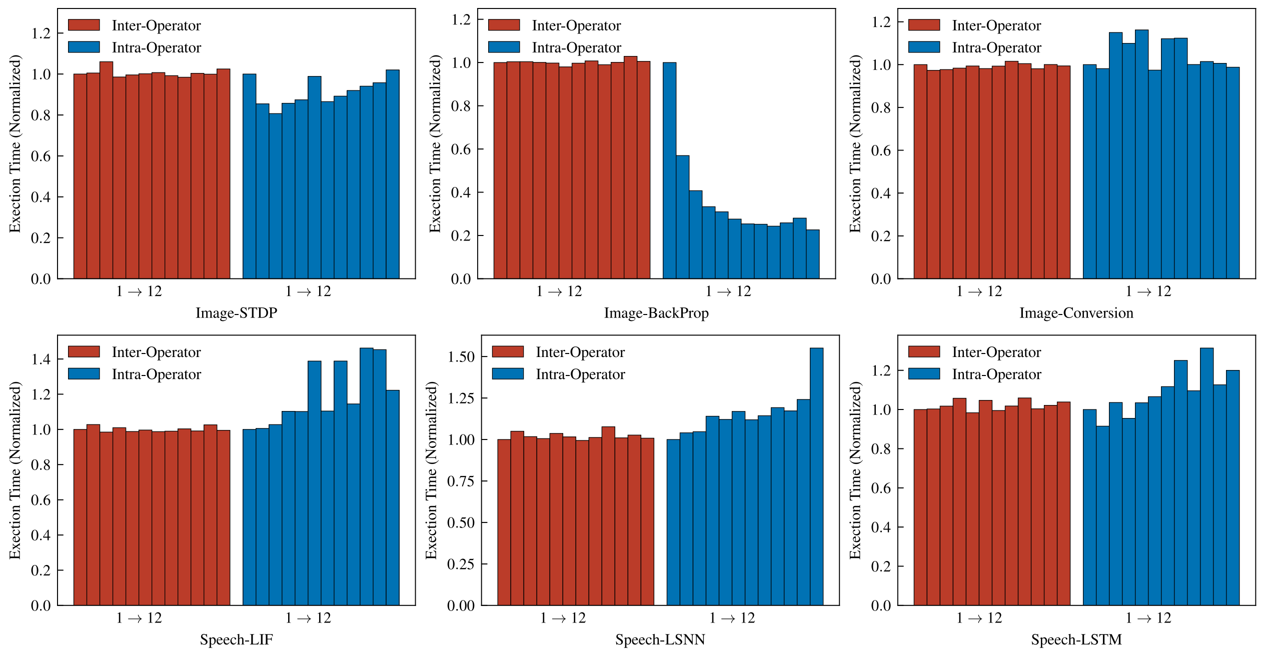


Fig. 5. Operator parallelisms of SNNBench workloads. All workloads are training workloads, without any inference workload. Image-Conversion and Speech-LSTM are included here for comparison to highlight the differences in operator parallelism between mature DNN training workloads and SNN training workloads.

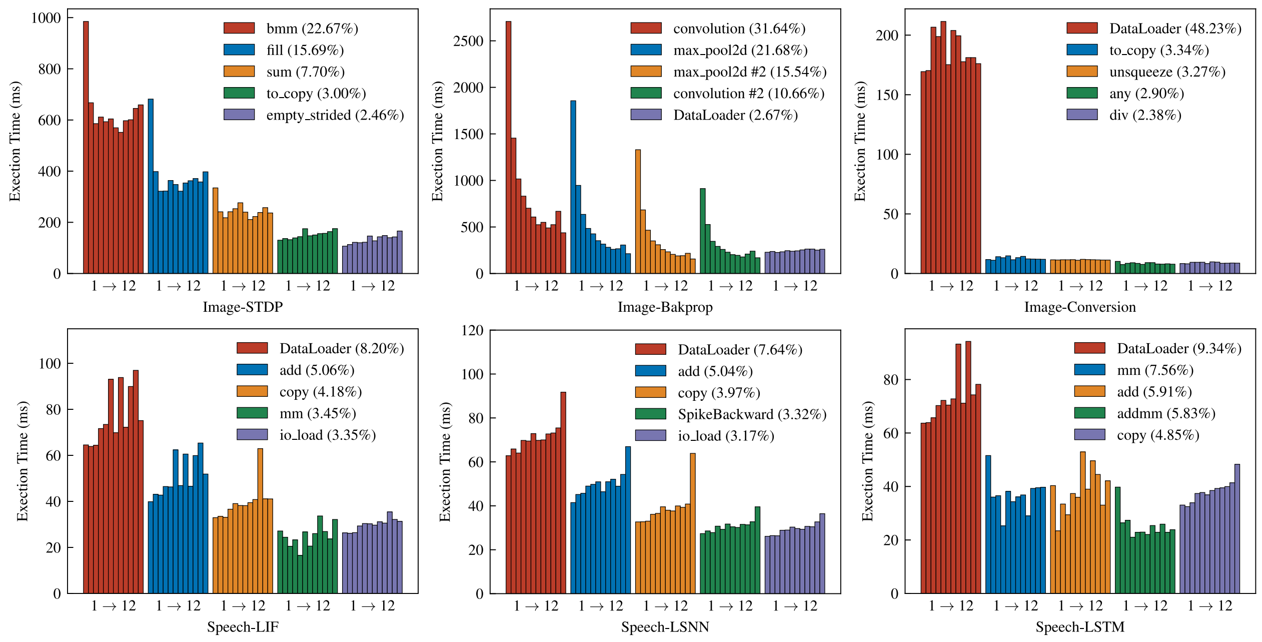


Fig. 6. Intra-operator executing on different threads. All workloads are training workloads, without any inference workload. Image-Conversion and Speech-LSTM are included here for comparison to highlight the differences in operator scalability between mature DNN training workloads and SNN training workloads. We select the most time-consuming operators, and the percentage numbers in brackets represent the proportion of time consumed by each operator.

threads exceeds three. This may be related to the input size of the three operators since the Image-STDP workload does not support batch training, so the data input size is small, and performance is best with three threads. If the data input size becomes larger, it will benefit from more hardware computing units, as we discussed in Section 4.4. From the result, the number of intra-operator threads that gains the best performance is also the thread number that the most cost operators get the best performance. These optimal numbers of threads are 3, 8, 6, 1, 1, and 2, respectively. Therefore, it cannot be simply said that the more hardware threads, the better. Different operators and different data input sizes correspond to different optimal thread numbers. In practice, we can perform an ahead small-size training for searching the optimal thread number before the long training job.

In this section, we analyze the scalability and parallelism potential of SNN workloads. We experiment with varying thread numbers for inter- and intra-operator parallelism and evaluate their impact on training time across different workloads. Our findings indicate that increasing inter-operator threads has negligible effects on training time. However, adjusting the number of intra-operator threads enhances the performance of the Image-Backprop workload but has little or negative impact on others. We observe that different operators and input sizes require different optimal thread numbers, and more hardware threads do not always guarantee better performance. To improve performance, we suggest employing a strategy that uses varying intra-operator parallel thread numbers for different operators. By optimizing thread count based on the specific operator and input size, each operator's performance can be enhanced, consequently boosting overall neural network training efficiency. In practice, a small-size training run can help determine the optimal thread number before starting a longer training job.

5. Conclusion

SNN, as a promising technology for AI, needs in-depth explorations and further developments to achieve both high performance and high model accuracy—two requisites for AI scenarios. Benchmarks play foundational roles in locating the bottlenecks and making improvements. Existing benchmarks either focus on brain science benchmarking, which has totally different targets and solutions, or cover partial aspects of AI without accuracy information and thus fail to fulfill the above two requisites. This paper proposes an end-to-end AI-oriented benchmarking methodology and presents SNNBench, the first end-to-end SNN benchmark suite. The end-to-end represents two-fold meanings: (1) SNNBench focuses on typical AI applications and covers both training and inference stages for end-to-end evaluation; (2) SNNBench provides an end-to-end training process, covering diverse training strategies and achieving a target accuracy. In total, SNNBench provides nine workloads and covers two learning paradigms, including supervised and unsupervised learning, four connection types, including one-to-one, fully connected, convolutional, and recurrent, and three learning rules, including STDP, backpropagation, and DNN-to-SNN. Our experiments on CPU and GPU show the effectiveness of SNNBench.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to express profound gratitude to Mr. Zhengxin Yang. His insightful discussions and valuable suggestions greatly contributed to the improvement of Fig. 1 in this paper.

References

[1] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (1997) 1659–1671.
[2] K. Roy, A. Jaiswal, P. Panda, Towards spike-based machine intelligence with neuromorphic computing, Nature 575 (2019) 607–617.
[3] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J.M. Bower, M. Diesmann, A. Morrison, P.H. Goodman, F.C. Harris, et al., Simulation of networks of spiking neurons: a review of tools and strategies, J. Comput. Neurosci. 23 (2007) 349–398.
[4] R.A. Tikidji-Hamburyan, V. Narayana, Z. Bozkus, T.A. El-Ghazawi, Software for brain network simulations: a comparative study, Front. Neuroinform. 11 (2017) 46.
[5] S.J. Van Albada, A.G. Rowley, J. Senk, M. Hopkins, M. Schmidt, A.B. Stokes, D.R. Lester, M. Diesmann, S.B. Furber, Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model, Front. Neurosci. 12 (2018) 291.
[6] T.J. Sejnowski, C. Koch, P.S. Churchland, Computational neuroscience, Science 241 (1988) 1299–1306.
[7] A.L. Hodgkin, A.F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. 117 (1952) 500.
[8] C. Koch, I. Segev, Methods in Neuronal Modeling: From Ions To Networks, MIT Press, 1998.
[9] N. Brunel, M.C. Van Rossum, Lapicque's 1907 paper: from frogs to integrate-and-fire, Biol. Cybernet. 97 (2007) 337–339.
[10] M. Davies, Benchmarks for progress in neuromorphic computing, Nat. Mach. Intell. 1 (2019) 386–388.
[11] C. Ostrau, J. Homburg, C. Klarhorst, M. Thies, U. Rückert, Benchmarking deep spiking neural networks on neuromorphic hardware, in: International Conference on Artificial Neural Networks, Springer, 2020, pp. 610–621.
[12] S.R. Kulkarni, M. Parsa, J.P. Mitchell, C.D. Schuman, Benchmarking the performance of neuromorphic and spiking neural network simulators, Neurocomputing 447 (2021) 145–160.
[13] P.U. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Front. Comput. Neurosci. 9 (2015) 99.
[14] N. Brunel, X.J. Wang, What determines the frequency of fast network oscillations with irregular neural discharges? I. Synaptic dynamics and excitation-inhibition balance, J. Neurophysiol. 90 (2003) 415–430.
[15] R.A. Tikidji-Hamburyan, J.A. Martínez, C.C. Canavier, Resonant interneurons can increase robustness of gamma oscillations, J. Neurosci. 35 (2015) 15682–15695.
[16] J. Gray, Benchmark Handbook: For Database and Transaction Processing Systems, Morgan Kaufmann Publishers Inc, 1992.
[17] J. Zhan, L. Wang, W. Gao, R. Ren, Benchcouncil's view on benchmarking ai and other emerging workloads, 2019, arXiv preprint arXiv:1912.00572.
[18] Z. Jiang, W. Gao, F. Tang, L. Wang, X. Xiong, C. Luo, C. Lan, H. Li, J. Zhan, Hpc ai500 v2, 0: The methodology, tools, and metrics for benchmarking hpc ai systems, in: 2021 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2021, pp. 458–477.
[19] C.D. Schuman, S.R. Kulkarni, M. Parsa, J.P. Mitchell, B. Kay, et al., Opportunities for neuromorphic computing algorithms and applications, Nat. Comput. Sci. 2 (2022) 10–19.
[20] J.H. Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Front. Neurosci. 10 (2016) 508.
[21] A. Tavanaei, M. Ghodrati, S.R. Kheradpisheh, T. Masquelier, A. Maida, Deep learning in spiking neural networks, Neural Netw. 111 (2019) 47–63.
[22] Y. Jin, W. Zhang, P. Li, Hybrid macro/micro level backpropagation for training deep spiking neural networks, Adv. Neural Inf. Process. Syst. 31 (2018).
[23] W. Zhang, P. Li, Temporal spike sequence learning via backpropagation for deep spiking neural networks, Adv. Neural Inf. Process. Syst. 33 (2020) 12022–12033.
[24] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (2019) 1997–2017.
[25] D. Simon, Evolutionary Optimization Algorithms, John Wiley & Sons, 2013.
[26] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Process. Mag. 29 (2012) 141–142.
[27] P. Warden, Speech commands: A dataset for limited-vocabulary speech recognition, 2018, arXiv preprint arXiv:1804.03209.
[28] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.J. Nam, et al., Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (2015) 1537–1557.
[29] M. Davies, N. Srinivasa, T.H. Lin, G. Chinya, Y. Cao, S.H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al., Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro 38 (2018) 82–99.
[30] A.F. Agarap, Deep learning using rectified linear units (relu), 2018, arXiv preprint arXiv:1803.08375.
[31] Y. Cao, Y. Chen, D. Khosla, Spiking deep convolutional neural networks for energy-efficient object recognition, Int. J. Comput. Vis. 113 (2015) 54–66.
[32] P.U. Diehl, D. Neil, J. Binas, M. Cook, S.C. Liu, M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, in: 2015 International Joint Conference on Neural Networks, IJCNN, IEEE, 2015, pp. 1–8.
[33] PyTorch Documentation, b. Reproducibility. URL: https://pytorch.org/docs/stable/notes/randomness.html.
[34] H. Hazan, D.J. Saunders, H. Khan, D. Patel, D.T. Sanghavi, H.T. Siegelmann, R. Kozma, Bindsnet: A machine learning-oriented spiking neural networks library in python, Front. Neuroinform. 89 (2018).
[35] F. Zenke, S. Ganguli, Superspike: Supervised learning in multilayer spiking neural networks, Neural Comput. 30 (2018) 1514–1541.
[36] PyTorch Documentation, a. Profiler. URL: https://pytorch.org/docs/stable/profiler.html.
[37] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[38] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.
[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).
[40] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, W. Maass, Long short-term memory and learning-to-learn in networks of spiking neurons, Adv. Neural Inf. Process. Syst. (2018) 787–797.