

Spiking Neural Networks in Vertical Federated Learning: Performance Trade-offs

Maryam Abbasihafshejani

Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas

Anindya Maiti

Department of Computer Science
University of Oklahoma
Norman, Oklahoma

Murtuza Jadliwala

Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas

Abstract—Federated machine learning enables model training across multiple participants while preserving data privacy. Vertical Federated Learning (VFL) handles scenarios in which participants have different feature sets for the same samples. Although Spiking Neural Networks (SNNs) offer efficiency advantages over Artificial Neural Networks (ANNs), their applicability in a VFL scenario remains unexplored. This paper examines SNNs in VFL, implementing and evaluating two architectures—with and without model splitting—using CIFAR-10 and CIFAR-100 datasets with VGG9 and ResNet models. The evaluation results show that SNNs achieve an accuracy comparable to that of ANNs in VFL while being significantly more energy efficient.

I. INTRODUCTION

Federated Learning (FL) enables collaborative machine learning while preserving data privacy by avoiding direct data exchange among participants/users [1], [2], [3], [4]. Vertical Federated Learning (VFL) [5] is a specialized form of FL in which each participant has a different set of features for the same samples, which makes it essential for privacy-sensitive domains such as finance and healthcare. Although VFL has been widely studied with Artificial Neural Networks (ANNs) [6], its integration with Spiking Neural Networks (SNNs) remains underexplored. Neuromorphic computing provides energy-efficient alternatives to traditional deep learning, particularly edge computing [7], [8], [9], [10], supported by hardware advances such as IBM's TrueNorth [11] and Intel's Loihi [12]. SNNs, which leverage spike-based communication, mimic biological neurons and offer computational efficiency [13], making them suitable for FL in resource-constrained settings. We design two distinct architectures for training SNN models in the vertical federated learning setting (VFL-SNN): one with model splitting, where different segments of the SNN are processed and stored across different participants, and another without model splitting, where the entire model, albeit distributed, is collaboratively updated without segmenting the model. These approaches balance privacy and efficiency for different operational needs. We evaluate VFL-SNN models on the CIFAR-10 and CIFAR-100 datasets by using SNN adaptations of VGG9 and ResNet. Our analysis compares the accuracy and energy consumption of the model with traditional ANNs in similar VFL setups. Our evaluation results show that VFL-SNN models achieve accuracy comparable to ANNs while significantly reducing energy consumption, confirming their potential in VFL. This research advances FL applications and highlights the promise of energy-efficient neural networks for privacy-sensitive collaborative

learning. A detailed version of this article is available on: arXiv:2407.17672.

II. BACKGROUND AND RELATED WORK

A. SNNs and FL

SNNs consist of an *encoder* that converts input data into spike trains, an *SNN model* that processes spikes using neuronal dynamics, and a *decoder* that translates spike-based outputs [14], [15]. Encoding methods such as *rate coding* and *Poisson-distributed spike trains* map input intensity to spike frequency, while models like the *Leaky Integrate-and-Fire*, *Spike Response*, and *Izhikevich Model* regulate spiking activity [16], [17]. Decoding approaches, including *rate-based* and *temporal decoding*, reconstruct spike train information [18].

FL enables collaborative model training without data sharing, classified as *Horizontal Federated Learning (HFL)* and *Vertical Federated Learning (VFL)*. HFL involves participants with similar feature spaces but different samples, updating models using algorithms such as FedAvg and FedProx [19], [20], [21]. In VFL, participants hold different feature sets for the same samples and can be implemented *with* or *without model splitting* [22], as outlined earlier. SNNs have been incorporated into HFL by adjusting the learning parameters [23], refining batch normalization [24], and introducing asynchronous learning for scalability [25]. Techniques like slimmable neural networks improve federated frameworks [26]. However, VFL with SNNs remains underexplored. To address this, we adapt VFL training algorithms for SNNs, investigating both split and non-split architectures and comparing them with ANN models.

B. Batch Normalization Through Time (BNTT)

The non-differentiability of spiking neurons makes SNN training challenging. To address this, [24] introduced *Batch Normalization Through Time (BNTT)*, which operates in the temporal dimension by associating a learnable parameter γ with each time step. BNTT allows neurons to dynamically regulate their spiking rate, improving the accuracy and efficiency of training. During forward propagation, the BNTT layer is applied after each convolutional or linear layer as follows:

$$v_i^t = \lambda v_i^{t-1} + \gamma_i^t \left(\frac{\sum_{j \in n} w_{ij} o_j^t - \mu_i^t}{\sqrt{(\sigma_i^t)^2 + \epsilon}} \right) + BNTT_{\gamma_i^t} \sum_{j \in n} w_{ij} o_j^t \quad (1)$$

where μ_i^t and σ_i^t are the mini-batch mean and variance at time t , o_j^t is the binary output of neuron j , and v_i^t is the membrane potential of neuron i . The leak factor λ controls the decay and n represents the input neurons. The learnable parameter γ_i^t is updated during back-propagation. During testing, the global mean and variance are estimated using an exponential moving average from the training rounds.

III. VERTICAL FEDERATED TRAINING USING SNNs

Efficient SNN training begins with input encoding. Since this work focuses on VFL with image data, we use *rate encoding*, which maps pixel intensity to spike rates. The VFL system consists of K participants $\{P_k\}_{k=1}^K$ and a server S , with data distributed among participants as feature partitions $\mathbf{x}_{i,k}$. VFL training follows two approaches: *with model splitting* and *without model splitting*. We first describe the VFL-SNN training with model splitting (Algorithm 1) and then the approach without model splitting (Algorithm 2).

A. VFL-SNN with Model Splitting

In VFL with model splitting, each participant k trains a local *bottom model* $\mathcal{G}_k$, parameterized by θ_k , using its local data $\mathbf{x}_{i,k}$. A *top model* $\mathcal{F}$, parameterized by ψ , runs on the server and aggregates outputs from the bottom models $\mathcal{G}_k$ to compute the final prediction by minimizing a joint task loss $\mathcal{L} = \text{Loss_Function}(\mathcal{F}(\psi; \text{Concat}(o_1(\theta_1; x_{i,1}), \dots, o_K(\theta_K; x_{i,K}))), y_i)$, where $\text{Loss_Function}()$ is a loss function such as cross-entropy, $o_k(\theta_k; x_{i,k})$ represents the output of the k -th bottom model for the i -th data sample, and $\text{Concat}()$ denotes the concatenation of outputs from all bottom models. The server computes the gradient $\frac{\partial \mathcal{L}}{\partial \psi}$, updating the top model as $\psi = \psi - \eta_2 \times \frac{\partial \mathcal{L}}{\partial \psi}$. Simultaneously, it calculates $\frac{\partial \mathcal{L}}{\partial o_k}$, which is sent to each participant for back propagation.

Each participant updates its parameters using $\frac{\partial \mathcal{L}}{\partial \theta_k} = \frac{\partial \mathcal{L}}{\partial o_k} \times \frac{\partial o_k}{\partial \theta_k}$, but since o_k is a threshold function, its derivative is undefined at spike times. This is approximated using a piecewise linear or exponential function [27], allowing the update $\theta_k = \theta_k - \eta_1 \times \frac{\partial \mathcal{L}}{\partial \theta_k}$. This VFL setup enables secure SNN training, as only the server accesses labels, enhancing privacy.

B. VFL-SNN without Model Splitting

VFL without model splitting involves each participant running a bottom model $\mathcal{G}_k$, but unlike model splitting, there is no trainable top model on the server. Instead, the server applies a simple aggregation function, such as summation, to combine outputs $o_k = \mathcal{G}_k(\theta_k; \mathbf{x}_{i,k})$ from all participants' bottom models to compute the final output using a joint task loss function, given by $\mathcal{L} = \text{Loss_Func}(\sum_{k=1}^K o_k, y_i)$. This approach reduces the server's computational burden and enhances privacy, as it merely aggregates outputs without additional processing. The server then computes the gradient of the joint loss $\mathcal{L}$ with respect to each output o_k , i.e., $\frac{\partial \mathcal{L}}{\partial o_k}$, and sends these gradients to the corresponding participants. Each participant uses these gradients to backpropagate, following the chain rule, to compute $\frac{\partial \mathcal{L}}{\partial \theta_k}$. Finally, participants update

Algorithm 1 Training VFL-SNN with Model Splitting.

```

1: Input:
2:   Dataset:  $\mathcal{D}$  (participant  $P_k$  privately holds  $\mathbf{x}_{i,k}$ , server  $S$  privately holds  $y_i$ )
3:   Hyperparameters: learning rates ( $\eta_1$  and  $\eta_2$  for bottom and top models, respectively), number of epochs ( $E$ ), number of time stamps ( $T$ )
4: Output:
5:   Trained local/bottom models  $\mathcal{G}_k$  defined by parameters  $\theta_k$  ( $\forall k = 1 \dots K$ ) and trained global/top model  $\mathcal{F}$  defined by parameter  $\psi$ 

6: procedure VFL-SNN ()
7:   Participants  $P_1, P_2, \dots, P_K$  initialize model parameters  $\theta_1, \theta_2, \dots, \theta_K$ , respectively, and server  $S$  initializes parameter  $\psi$  with random weights.
8:   while stopping epoch ( $E$ ) not met do
9:     for each batch  $b$  of samples do
10:      for each participant  $P_k, k \leftarrow 1$  to  $K$  do  $\triangleright$  can be executed in parallel
11:         $o_k \leftarrow$  perform BNTT-FPropagateParticipant ( $\mathcal{G}_k$ )
12:      end for
13:       $o_{all} \leftarrow \text{Concat}(o_1, o_2, \dots, o_K)$   $\triangleright$  concatenate local/bottom model outputs
14:       $o_{final} \leftarrow$  perform BNTT-FPropagateServer ( $\mathcal{F}, o_{all}$ )
15:       $\mathcal{L} \leftarrow \text{Loss\_Function}(o_{final}, y)$ 
16:       $S$  computes  $g_{top}^S \leftarrow \frac{\partial \mathcal{L}}{\partial \psi}$   $\triangleright$  compute loss and backpropagate top model
17:       $S$  updates  $\psi \leftarrow \psi - \eta_2 g_{top}^S$   $\triangleright$  update the top model
18:      for each  $k \leftarrow 1$  to  $K$  do
19:         $S$  computes  $g_k \leftarrow \frac{\partial \mathcal{L}}{\partial o_k}$  and forwards it to each  $P_k$ 
20:        Each  $P_k$  executes ParticipantBP ( $\mathcal{G}_k, g_k, o_k$ )
21:      end for
22:    end while
23:  end procedure

25: procedure BNTT-FPROPAGATEPARTICIPANT ( $\mathcal{G}_k$ )
26:   for time step  $t \leftarrow 1$  to  $T$  do
27:      $o \leftarrow \text{PoissonGenerator}(b)$   $\triangleright$  Convert pixels to spike trains
28:     for layer  $\leftarrow 1$  to  $L_{\mathcal{G}_k} - 1$  of  $\mathcal{G}_k$  do
29:       Forward propagate as outlined in Equation (1)
30:     end for
31:     Accumulate membrane potential at the last layer ( $L_{\mathcal{G}_k}$ )
32:   end for
33:   return the accumulated membrane potential  $o_k$ 
34: end procedure

35: procedure BNTT-FPROPAGATESERVER ( $\mathcal{F}, o_{all}$ )
36:   for layer  $\leftarrow 1$  to  $L_{\mathcal{F}} - 1$  of  $\mathcal{F}$  do
37:     Forward propagate as outlined in Equation (1)
38:   end for
39:   Accumulate membrane potential at the last layer ( $L_{\mathcal{F}}$ ) into  $o_{acc}$ 
40:   return  $o_{final} \leftarrow$  Averaging the firing rate
41: end procedure

42: procedure PARTICIPANTBP ( $\mathcal{G}_k, g_k, o_k$ )
43:    $P_k$  computes  $g_{bot}^k \leftarrow g_k \cdot \frac{\partial o_k}{\partial \theta_k}$   $\triangleright$  compute loss and backpropagate bottom model gradient
44:    $P_k$  updates  $\theta_k \leftarrow \theta_k - \eta_1 g_{bot}^k$   $\triangleright$  update the top model
45: end procedure

```

their model parameters as $\theta_k = \theta_k - \eta_1 \times \frac{\partial \mathcal{L}}{\partial \theta_k}$ to optimize their local models.

IV. EVALUATION

A. Experimental Setup

We evaluated VFL-SNN on CIFAR-10 and CIFAR-100 datasets[28] using VGG9 [29] and ResNet-18 [30] in a VFL setup, with experiments conducted on an NVIDIA V100 GPU. Features were distributed among K participants, and both model splitting and non-splitting configurations were tested. VFL-SNN was adapted from the HFL-SNN codebase [23] with modifications for VFL. SNNs used the Leaky Integrate-and-Fire (LIF) neuron model with a 99% leakage rate and Poisson encoding [15], with hyperparameters optimized for VGG9 and ResNet-18. ANN-based VFL implementations achieved optimal accuracy with a learning rate of 0.001 and a weight decay of 5×10^{-4} .

TABLE I: Performance metrics comparison for CIFAR-10 and CIFAR-100 for VFL without model splitting.

Dataset	CIFAR-10				CIFAR-100			
	SNN-VGG9	SNN-ResNET18	ANN-VGG9	ANN-ResNET18	SNN-VGG9	SNN-ResNET	ANN-VGG9	ANN-ResNET18
Average Training Accuracy	80.38	88.61	92.79	75.48	65.35	65.40	67.32	66.56
Best Test Accuracy	77.95	74.10	80.35	77.09	50.93	44.01	49.07	44.50
Final Test Accuracy	77.64	72.66	79.6	75.51	50.41	42.65	48.72	43.81
Average Test Accuracy	71.02	71.23	77.14	67.65	44.92	40.13	44.10	38.85
Precision	0.777	0.740	0.804	0.774	0.521	0.449	0.507	0.4706
Recall	0.779	0.741	0.803	0.770	0.509	0.440	0.490	0.444
F1 Score	0.778	0.740	0.803	0.772	0.515	0.444	0.498	0.457

TABLE II: Performance metrics comparison for CIFAR-10 and CIFAR-100 for VFL with model splitting.

Dataset	CIFAR-10				CIFAR-100			
	SNN-VGG9	SNN-ResNET18	ANN-VGG9	ANN-ResNET18	SNN-VGG9	SNN-ResNET	ANN-VGG9	ANN-ResNET18
Average Training Accuracy	83.26	89.35	89.292	83.45	54.26	55.70	76.11	72.22
Best Test Accuracy	80.28	77.05	82.091	79.47	52.11	48.03	55.61	48.80
Final Test Accuracy	78.89	75.74	82.09	78.01	51.50	46.32	55.438	45.19
Average Test Accuracy	76.28	74.58	78.13	74.90	45.02	42.70	49.28	43.64
Precision	0.806	0.771	0.820	0.797	0.537	0.480	0.568	0.506
Recall	0.802	0.770	0.820	0.794	0.521	0.480	0.556	0.487
F1 Score	0.804	0.771	0.820	0.796	0.528	0.480	0.562	0.497

Algorithm 2 Training VFL-SNN without Model Splitting.

```

1: Input:
2:   Dataset:  $\mathcal{D}$  (participant  $P_k$  privately holds  $\mathbf{x}_{i,k}$ , server  $S$  privately holds  $y_i$ )
3:   Hyperparameters: learning rate ( $\eta$ ), number of epochs ( $E$ ), number of time stamps ( $T$ )
4: Output:
5:   Trained local model  $\mathcal{G}_k$  defined by parameters  $\theta_k$  ( $\forall k = 1 \dots K$ )

6: procedure VFL-SNN ()
7:   Participants  $P_1, P_2, \dots, P_K$  initialize model parameters  $\theta_1, \theta_2, \dots, \theta_K$ , respectively.
8:   while stopping epoch ( $E$ ) not met do
9:     for each batch  $b$  of samples do
10:      for each participant  $P_k, k \leftarrow 1$  to  $K$  do  $\triangleright$  can be executed in parallel
11:         $o_k \leftarrow$  perform BNTT-FPropagateParticipant ( $\mathcal{G}_k$ )
12:      end for
13:       $o_{sum} \leftarrow \sum_k (o_k)$   $\triangleright$  aggregate local model outputs
14:       $S$  computes  $\mathcal{L} \leftarrow$  Loss_Function( $o_{sum}, y$ )
15:      for each  $k \leftarrow 1$  to  $K$  do
16:         $S$  computes  $g_k \leftarrow \frac{\partial \mathcal{L}}{\partial o_k}$  and forwards it to each  $P_k$ 
17:        Each  $P_k$  executes ParticipantBP ( $\mathcal{G}_k, g_k, o_k$ )
18:      end for
19:    end for
20:  end while
21: end procedure

22: procedure BNTT-FPROPAGATEPARTICIPANT ( $\mathcal{G}_k$ )
23:   for time step  $t \leftarrow 1$  to  $T$  do
24:      $o \leftarrow$  PoissonGenerator( $b$ )  $\triangleright$  Convert pixels to spike trains
25:     for layer  $\leftarrow 1$  to  $L_{\mathcal{G}_k} - 1$  of  $\mathcal{G}_k$  do
26:       Forward propagate as outlined in Equation (1)
27:     end for
28:     Accumulate membrane potential at the last layer ( $L_{\mathcal{G}_k}$ )
29:   end for
30:   return the accumulated membrane potential  $o_k$ 
31: end procedure

32: procedure PARTICIPANTBP ( $\mathcal{G}_k, g_k, o_k$ )
33:    $P_k$  computes  $g_{loc}^k \leftarrow g_k \cdot \frac{\partial o_k^k}{\partial o_k}$   $\triangleright$  backpropagate bottom model gradient
34:    $P_k$  updates  $\theta_k \leftarrow \theta_k - \eta g_{loc}^k$   $\triangleright$  update the top model
35: end procedure

```

B. Models and Splitting

Our evaluation compares VFL-SNN with and without model splitting, assessing VFL-ANN and SNN implementations of VGG9 and ResNet18 with two participants and an SNN time step of 32. Results (Tables I and II) show a minor 3% performance gap, with ANN VGG9 achieving 80.35% and

SNN VGG9 reaching 77.95% on CIFAR-10.

C. Number of Participants

We evaluated the stability of the model by varying the number of participants and distributing the image segments accordingly (Figures 2 and 3). For example, in a scenario with 2 participants, one participant processes the right side of the images, while the other handles the left side. As the number of participants increases, the accuracy of the ANN decreases more dramatically than that of the SNNs, indicating better scalability for the SNNs. For CIFAR-100 (Figure 3), the accuracy of VFL-ANN drops from 50% (two participants) to 43% (four participants). Also, the accuracy of VFL-SNN drops from about 50% to 45%, showing that its scalability is the same as VFL-ANN.

D. Time Steps

We analyze the impact of SNN time steps—defined as the interval at which SNNs process and transmit spike-based information—on the accuracy of VFL-SNN by varying them from 10 to 50 with three participants. As shown in Figure 1, the accuracy improved up until 30 time steps but showed minimal changes beyond that, indicating a saturation point.

E. Energy Consumption

We analyzed the energy consumption of each model layer during training to demonstrate the efficiency of SNNs, driven by their sparse neuronal activations. For comparison with ANN models, we recorded spike counts per layer in VFL-SNN and estimated energy consumption based on model operations. The energy required for operations (OPS) in convolutional layers is computed as $OPS = \left(\frac{NI-k+2p}{s} + 1 \right)^2 \times I \times k^2 \times O$, where k is the kernel size, I and O are input and output channels, and $M \times M$ is the feature map size. For fully connected layers, the energy is given as follows: $OPS = I \times O$.

By implementing vertical federated learning, which distributes features across multiple devices, the energy demand

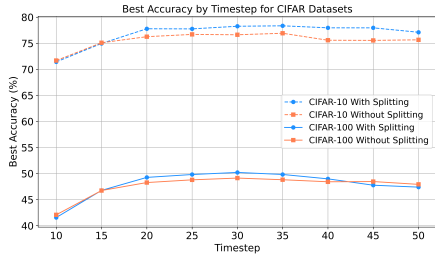


Fig. 1: Variations in accuracy with increasing time steps.

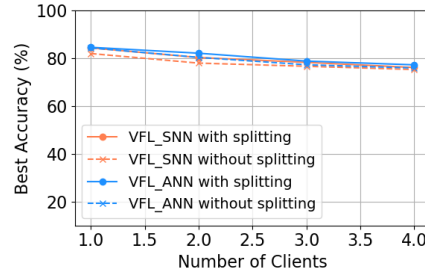


Fig. 2: CIFAR-10 classification accuracy across different numbers of participants.

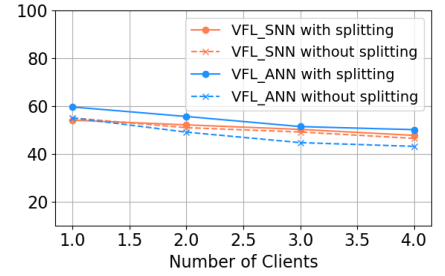


Fig. 3: CIFAR-100 classification accuracy across different numbers of participants.

on each device is reduced. Additionally, SNNs utilize binary spikes that convert the multiply accumulate (MAC) operation predominantly into an accumulation (AC) operation, greatly enhancing energy efficiency. Energy consumption for ANNs is estimated using $E_{ANN} = OPS \times E_{MAC}$, whereas for SNNs, it depends on spike rate R and time steps T , computed as $E_{SNN} = OPS \times R \times T \times E_{AC}$. To obtain the spike rate, we recorded the number of spikes during model training. To estimate the energy required for AC and MAC operations, we used the energy consumption data reported in [31] for 32-bit integer arithmetic operations in a 45 nm CMOS processor. In the estimation, the focus was on multiply and accumulate operations, excluding the energy costs associated with memory and peripheral circuits. Figure 4 shows the energy consumption of the VFL-SNN and VFL-ANN VGG9 models during training on CIFAR-10 with three participants. The SNN model, using 32 time steps, consumes an average of $1.41 \mu J$, significantly lower than the ANN's $64.98 \mu J$. With ANN using 34.7 times more energy, this highlights the efficiency of VFL-SNN, making it ideal for edge devices.

F. Training Time

SNNs provide energy efficiency but have longer training times due to spike processing. As shown in Figure 5, training VFL-SNN with model splitting took 123 s per participant per epoch (559 s total), whereas without splitting, it increased to 142 s but the total time reduced to 300 s because of the parallel execution of computations across participants and reduced communication overhead. In contrast, VFL-ANN required only 1.53 s per participant per epoch, highlighting the trade-off between efficiency and training speed.

V. DISCUSSION AND CONCLUSION

Integrating SNNs into VFL enhances efficiency, privacy, and energy savings, making them ideal for edge computing and IoT. Although slightly less accurate than ANNs, SNNs remain scalable. Training time is a challenge due to spike processing, but optimizations can improve viability. Our evaluations on CIFAR-10 and CIFAR-100 with VGG9 and ResNet models show that VFL-SNN achieves accuracy comparable to ANN with greater energy efficiency, reinforcing its potential for scalable and privacy-preserving FL.

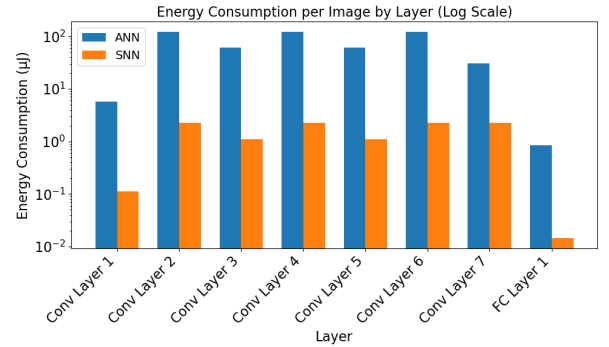


Fig. 4: Estimated training energy across layers for SNN (VFL-SNN) and ANN VGG9 model on CIFAR-10 dataset.

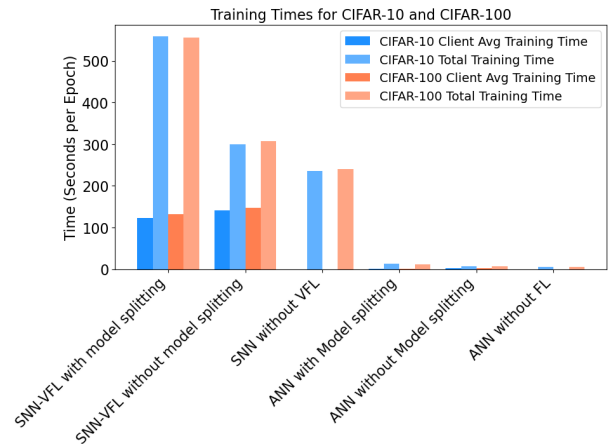


Fig. 5: Training time comparison across models.

VI. ACKNOWLEDGEMENT

This work was supported in part by the NSF under award 2332744. The views and findings solely reflect those of the authors and should not be seen as representative opinions of any funding agency.

REFERENCES

- [1] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype, and reality for data privacy and protection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3347–3366, 2021.
- [2] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Federated learning for data privacy preservation in vehicular cyber-physical systems," *IEEE Netw.*, vol. 34, no. 3, pp. 50–56, 2020.
- [3] G. Liu, C. Wang, X. Ma, and Y. Yang, "Keep your data locally: Federated-learning-based data privacy preservation in edge computing," *IEEE Netw.*, vol. 35, no. 2, pp. 60–66, 2021.
- [4] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6532–6542, 2019.
- [5] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE Trans. Knowl. Data Eng.*, 2024.
- [6] I. A. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *J. Microbiol. Methods*, vol. 43, no. 1, pp. 3–31, 2000.
- [7] A. Vitale, E. Donati, R. Germann, and M. Magno, "Neuromorphic edge computing for biomedical applications: Gesture classification using emg signals," *IEEE Sens. J.*, vol. 22, no. 20, pp. 19 490–19 499, 2022.
- [8] E. Covi, E. Donati, X. Liang, D. Kappel, H. Heidari, M. Payvand, and W. Wang, "Adaptive extreme edge computing for wearable devices," *Front. Neurosci.*, vol. 15, p. 611300, 2021.
- [9] J. Yoon, B. You, Y. Kim, J. Bak, M. Yang, J. Park, M. G. Hahm, and M. Lee, "Environmentally stable and reconfigurable ultralow-power two-dimensional tellurene synaptic transistor for neuromorphic edge computing," *ACS Appl. Mater. Interfaces*, vol. 15, no. 14, pp. 18 463–18 472, 2023.
- [10] X. Zhang, Z. Wu, J. Lu, J. Wei, J. Lu, J. Zhu, J. Qiu, R. Wang, K. Lou, Y. Wang *et al.*, "Fully memristive snns with temporal coding for fast and low-power edge computing," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2020, pp. 29–6.
- [11] F. Akopyan, "Design and tool flow of ibm's trueneuron: An ultra-low power programmable neurosynaptic chip with 1 million neurons," in *Proc. Int. Symp. Phys. Des. (ISPD)*, 2016, pp. 59–60.
- [12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, and *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [13] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, 2019.
- [14] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep neural networks with weighted spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018.
- [15] D. Heeger *et al.*, "Poisson model of spike generation," Stanford University, Tech. Rep. 5, 2000.
- [16] G. Tanoni, "A spiking neural network-based approach for binaural sound localization," Ph.D. dissertation, Polytechnic University of Marche, 2019.
- [17] C. Tan, M. Šarlija, and N. Kasabov, "Spiking neural networks: Background, recent development and the neucube architecture," *Neural Process. Lett.*, vol. 52, no. 2, pp. 1675–1701, 2020.
- [18] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, 2005.
- [19] T. Sun, D. Li, and B. Wang, "Decentralized federated averaging," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 4289–4301, 2022.
- [20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, vol. 2, 2020, pp. 429–450.
- [21] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440 [cs.LG]*, 2020.
- [22] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 8, 2022, pp. 8485–8493.
- [23] Y. Venkatesha, Y. Kim, L. Tassiulas, and P. Panda, "Federated learning with spiking neural networks," *IEEE Trans. Signal Process.*, vol. 69, pp. 6183–6194, 2021.
- [24] Y. Kim and P. Panda, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," *Front. Neurosci.*, vol. 15, p. 773954, 2021.
- [25] Y. Wang, S. Duan, and F. Chen, "Efficient asynchronous federated neuromorphic learning of spiking neural networks," *Neurocomputing*, vol. 557, p. 126686, 2023.
- [26] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Slimfl: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Trans. Netw.*, 2023.
- [27] 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, 2019.
- [28] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
- [29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556 [cs.CV]*, 2014.
- [30] B. Koonce and B. Koonce, "Resnet 50," *Convolutional Neural Networks with Swift for TensorFlow: Image Recognition and Dataset Categorization*, pp. 63–72, 2021.
- [31] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2014, pp. 10–14.