Distributed Neuromorphic Edge Computing: Theory and Applications in Environmental Monitoring

Arman Ferdowsi and Atakan Aral
Scientific Computing Group, University of Vienna, Austria
{arman.ferdowsi, atakan.aral}@univie.ac.at

2025 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3315-5559-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/EDGE67623.2025.00031

***Abstract*—Neuromorphic computing has emerged as a promising paradigm for energy-efficient AI by mimicking brain-like event-driven processing. In parallel, geo-distributed sensor networks at the edge play a critical role in environmental monitoring and disaster response, requiring real-time, low-power analytics across remote locations.**

In this work, we combine these thrusts and propose the NeuEdge architecture for geo-distributed neuromorphic edge intelligence. We develop two distributed spiking neural network (SNN) paradigms: (1) *Federated SNNs*, where multiple edge SNN models are trained locally on sensor data and periodically synchronized, and (2) *Split SNNs*, where a single SNN is partitioned across networked edge nodes that communicate spikes. We formally define both models and present distributed learning algorithms for each. Theoretical results are derived on communication efficiency, convergence guarantees, and scalability with network size. We then implement simulations on synthetic and real environmental datasets to evaluate both approaches for tasks like environmental event detection and disaster prediction. The federated SNN approach is shown to achieve accuracy comparable to a centralized neuromorphic model while significantly reducing communication vs. raw data offloading. The split SNN approach enables real-time collaborative inference across sensor nodes, outperforming single-node baselines in prediction accuracy for geographically distributed phenomena.

***Index Terms*—Neuromorphic edge computing, distributed algorithms, federated learning, spiking neural networks, environmental monitoring, geographically distributed sensors.**

I. Introduction

Large-scale sensor networks for environmental monitoring and disaster response generate vast streams of data from *geographically distributed* sources. Examples include remote arrays of environmental sensors (for climate, water, and seismic activity) and IoT devices for wildfire or flood detection. Processing this data centrally is challenging due to bandwidth, latency, and reliability constraints in the field. *Edge computing* mitigates these issues by pushing computation closer to data sources. However, conventional edge AI methods, like deep neural networks on GPUs, remain power-hungry and may struggle with intermittent connectivity and limited on-site power supply. This motivates exploring new computing architectures that are inherently efficient and event-driven.

Neuromorphic computing is an emerging paradigm that directly addresses energy and efficiency challenges by mimicking biological brains [1]. In neuromorphic processors, information is processed using *Spiking Neural Networks* (SNNs) implemented in specialized hardware, achieving large reductions in power consumption compared to conventional AI hardware [2]. Neuromorphic hardware often consumes mere microjoules per inference, less than the cost of sending a single 32-bit packet over radio [3], [4]. By co-locating processing and memory and employing an event-driven design that remains largely idle without relevant input spikes, such architectures are particularly suited to battery- or solar-powered sensing deployments. This enables real-time analytics within tight energy budgets, substantially reducing both power draw and communication overhead. For instance, Intel's Loihi neuromorphic chip consumes up to $27\times$ less power than a standard edge AI accelerator on vision tasks [5]. Neuromorphic systems operate asynchronously and transmit information only when significant events (spikes) occur, making them well-suited for sparse, event-driven sensor data common in environmental monitoring (e.g., a spike indicating a sudden temperature increase or seismic tremor) [6].

A further consideration, on the other hand, is the need for training and adaptation of spiking models in situ rather than relying on any pre-trained centralized model [7]. Environmental conditions vary widely across locations and over time, so a static model trained on one dataset will likely underperform when deployed in a different context without retraining. Each edge node observes a unique, non-iid slice of the environment, which can lead to model insularity if not addressed [8]. Simply aggregating all raw sensor data to a cloud server for centralized training is impractical in these settings, not only because of intermittent connectivity, but also due to the prohibitive energy cost of transmitting large volumes of raw data. Instead, learning must occur either locally at the edge or through collaborative schemes that respect data locality. This is especially true for spiking neural networks, which cannot directly leverage off-the-shelf trained models from traditional deep learning. The non-differentiable spike activations in SNNs demand specialized training techniques, meaning that effective SNN models often need to be trained (or at least fine-tuned) on representative data from each deployment context.

These challenges motivate *distributed training* paradigms. Indeed biological brains inspire the combination of *distributed sensing* and *neuromorphic processing*. The nervous system is inherently distributed: separate brain regions process different sensory inputs and yet collectively form a coherent perception and response. This suggests that a network of

This research was funded in part by CHIST-ERA-22-SPiDDS-07 (TROCI Project) and by the Austrian Science Fund (FWF) [10.55776/I6647].

2767-9918/25/$31.00 ©2025 IEEE
DOI 10.1109/EDGE67623.2025.00031

neuromorphic edge devices could be interconnected to act like a "distributed brain", collaboratively analyzing sensor data across a geographical area. Prior work has begun to identify the opportunity of neuromorphic edge intelligence for rural environmental monitoring. In particular, Aral [9] argues that neuromorphic computing at the edge can tackle the energy, communication, and computation constraints in remote sensing scenarios and highlights challenges such as *distributed model synchronization* and *edge-only learning*.

Building on these insights, we introduce *NeuEdge*, a *geo-distributed neuromorphic edge computing* framework. In NeuEdge, multiple neuromorphic devices (edge SNN nodes) are deployed near sensors (e.g., in a forest or along a river) and are connected via a communication network (wired or wireless). These devices collectively run spiking neural networks that analyze local sensor signals and also exchange information with each other. The goal is to enable accurate, real-time inference, like detecting an environmental event or predicting a disaster, without relying on cloud connectivity, and to continually learn from local data in a decentralized fashion. NeuEdge directly tackles three main challenges prevalent in environmental monitoring. First, it reduces the heavy communication overhead inherent in centralizing large volumes of sensor data, which is critical given limited or intermittent connectivity in remote areas. Second, it mitigates energy constraints by using low-power, event-driven neuromorphic hardware, allowing sustained operation on battery or solar sources. Third, it supports real-time local inference and distributed learning to capture heterogeneous and dynamic conditions across geographically dispersed sensors, improving both responsiveness and accuracy in environmental event detection.

Main contribution:

- We propose and formalize two complementary paradigms for distributed SNN learning in NeuEdge: (a) *Federated SNNs*, where each edge node trains a local SNN on its sensor data and nodes periodically synchronize their models, and (b) *Split SNNs*, where a single logical SNN is split across multiple devices that communicate spikes in real time.
- We develop theoretical models for these paradigms, including definitions of the distributed training algorithms and analysis of their communication efficiency and convergence properties. In particular, we derive bounds showing how federated SNN training converges to a global optimum under standard assumptions, and we analyze the communication complexity of split SNN inference and learning as a function of network activity and topology.
- We demonstrate both approaches in simulation using spiking neural network libraries and apply them to environmental monitoring tasks. We use a synthetic dataset representing a distributed environmental event (for controlled experiments) and a public real-world sensor dataset that gives us insights into how the approach can be used for environmental monitoring, like water quality.
- Through experiments, we evaluate key metrics: prediction accuracy and F1 score. We also discuss latency (inference delay), energy consumption, and communication overhead.

Paper organization: Section II introduces the NeuEdge architecture and details of the federated and split SNN approaches. Section III provides theoretical analysis of communication requirements and convergence. Section IV describes the experimental setup and discusses results for both synthetic and real environmental data. Finally, Section V concludes with insights and future directions for research in geo-distributed neuromorphic edge computing.

II. NeuEdge Architectures for Distributed SNNs

NeuEdge encompasses a network of neuromorphic edge nodes, each capable of deploying SNN models and directly interfacing with local sensors. We consider two architectural paradigms for how these SNN models can be organized and trained across the network: (a) multiple independent SNNs that periodically share parameters (federated learning style), and (b) a single SNN that is partitioned across devices that continuously share spike data (model splitting).

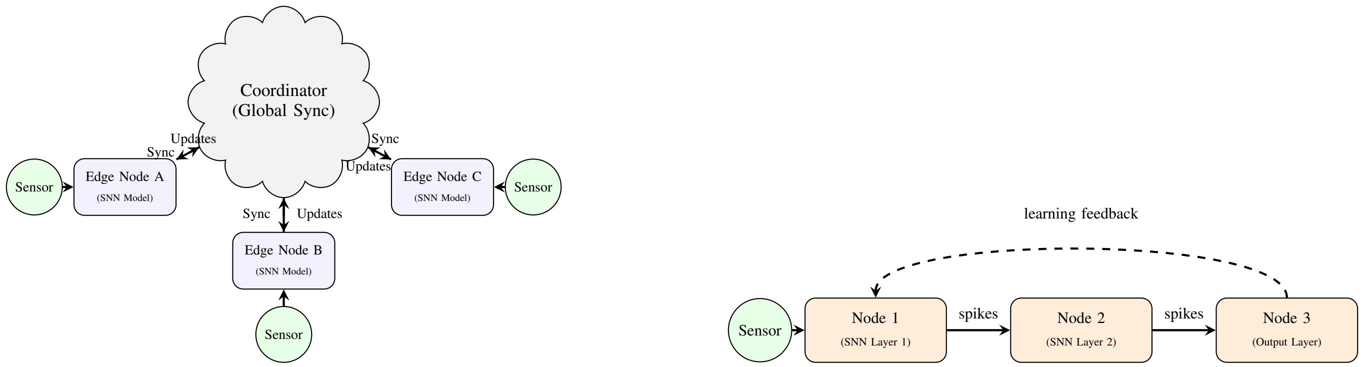
Fig. 1 gives a conceptual illustration of the two paradigms. In Fig. 1a, each edge node (e.g., a solar-powered neuromorphic board attached to an environmental sensor) runs its own SNN. These SNNs are trained locally and occasionally synchronize via a central coordinator or peer-to-peer averaging. In Fig. 1b, the SNN is divided into layers (or groups of neurons) distributed among the nodes; here the nodes must communicate at the granularity of individual spikes or activations during inference/training, effectively acting together as one larger neural network.

We now formalize each paradigm and present algorithms for training the SNN models in a distributed manner.

A. Federated SNNs

Federated learning is a paradigm in which multiple devices collaboratively train a model without sharing raw data, by periodically exchanging model updates [10]. This concept has been recently applied to spiking neural networks. Skatchkovsky *et al.* [11] and others [12] demonstrated that federated training can enable on-device learning in neuromorphic hardware, mitigating the limited data at each node by cooperative learning. In our federated SNN approach (Fed-SNN), each edge node has an SNN (with parameters W), which it trains using its locally observed sensor data (e.g., a stream of events from that location). We assume that each edge node employs an identical SNN architecture (i.e., the same graph structure and layer configuration), ensuring that the exchanged parameters are compatible for direct averaging and synchronization. Nodes periodically synchronize their models by exchanging parameters or weight updates, so that ultimately all nodes share a *consensus* model that has effectively learned from the union of data.

We define the following formal model: suppose there are K edge nodes. Node k has a local dataset D_k, which may be a



(a) Federated SNNs

(b) Split SNNs

Fig. 1. Illustration of the two NeuEdge paradigms. (a) *Federated SNNs*: each edge node trains its own local model on sensor data and periodically synchronizes model parameters with a coordinator (dashed lines show updates). (b) *Split SNNs*: a single SNN is partitioned across multiple nodes that exchange spikes (solid arrows) and learning signals (dashed arrow).

continuous data stream in practice, and maintains a local SNN with synaptic weight parameters W_k. Let $L_k(W)$ denote the loss function, like an error metric, for model parameters W on node k's data. In a supervised learning setting, $L_k(W) = \frac{1}{|D_k|}\sum_{(x,y)\in D_k} \ell(W;x,y)$, where ℓ is the sample-wise loss (e.g., misclassification error or spike timing prediction error) and (x,y) are input-output pairs. The global objective is to minimize the total loss $L(W) = \sum_{k=1}^{K} \alpha_k L_k(W)$, where $\alpha_k = \frac{|D_k|}{\sum_j |D_j|}$ (proportional to dataset size) gives each node's data appropriate weight.

Training proceeds in communication rounds $r = 1, 2, \ldots, R$. At the start, a global SNN model W^0 may be initialized, randomly or from a pre-trained baseline. In each round, each node k performs local training for some number of iterations or epochs on D_k, starting from the current global weights W^{r-1}. Let ΔW_k^r be the resulting weight update (difference between initial W^{r-1} and locally trained weights). Then the nodes share their updates or new weights with a central aggregator (or optionally with each other for peer-to-peer averaging). The aggregator computes an average to obtain the new global model:

$$W^r \;=\; W^{r-1} + \sum_{k=1}^{K} \frac{|D_k|}{\sum_j |D_j|}\,\Delta W_k^r,$$

which is essentially a weighted average of the local models (this is the *FedAvg* algorithm [13]). The new W^r is sent back to each node, and the process repeats. In practice, each node's local training can be as simple as a few epochs of stochastic gradient descent (SGD) on its SNN parameters, possibly using neuromorphic learning rules (e.g., event-driven random backpropagation [11] or local STDP-based rules with global feedback).

Algorithm 1 outlines the federated SNN training procedure. We highlight that during the training phase, only model parameters (synaptic weights) or their gradients are communicated, raw sensor data (spike trains, etc.) remain local, which is crucial for privacy (e.g., sensitive environmental data) and bandwidth efficiency. Each communication round in FedSNN

Algorithm 1 Federated SNN Training (FedSNN)

Require: K nodes with local datasets $D_1, \ldots, D_K$; initialization W^0

Require: Sync interval R rounds, local training epochs E per round

```
for each round r = 1 to R do
    for each node k = 1 to K in parallel do
        W_k ← W^{r-1}                          // start from global model
        for epoch e = 1 to E do
            for each local data batch (x, y) ∈ D_k:
                Compute local loss ℓ(W_k; x, y) and gradient
                Update W_k via SGD (or other learning rule)
        end for
        Send updated W_k (or ΔW_k = W_k − W^{r-1}) to server
    end for
    Server aggregates updates: W^r ← Σ_k (|D_k| / Σ_j |D_j|) W_k
    Server distributes new global model W^r to all nodes
end for
return final global model W^R
```

may occur infrequently, like once per day or when sufficient new data has been accumulated, thereby amortizing the communication cost over many local training events. Between rounds, each node can use its locally trained SNN to perform inference on new sensor inputs with minimal latency, since inference is entirely on-device, aside from sensing delays.

Federated SNNs leverage the fact that many environmental patterns, such as detection of a certain event, can be learned from data collected at one location and transferred to others. For instance, an acoustic impulse pattern associated with a landslide might be captured by one sensor's SNN; through federation, other sensors can incorporate this knowledge without ever exchanging raw acoustic signals, only the learned synaptic weight adjustments. This approach is robust in that each node maintains a functioning model even without connectivity (it just has a more limited experience). It also scales to a large number of nodes, since communication is mostly one-way model updates which can be aggregated from hundreds of nodes in parallel [13]. We note that techniques such as selective synapse sharing [11] can further reduce communication by only transmitting significant weight changes or a subset of the model.

One limitation of FedSNN is that it assumes each node can train a complete model. This may be problematic if each node only has a part of the input feature space (e.g., one sensor measures temperature, another measures humidity, and the target event depends on both). In such cases, no single node has the complete feature set to train on. Federated learning addresses data partitioned by samples (horizontal partition), but not by features (vertical partition). This motivates our second paradigm, Split SNNs, which is naturally suited to scenarios where different nodes observe different modalities or parts of the input.

B. Split SNNs

In the split SNN paradigm, we consider a single SNN that is divided into multiple segments (layers or sets of neurons) deployed across the edge nodes. Each node hosts one segment of the network and communicates spike events with the next segment over the network. This is analogous to the concept of *split learning* or model splitting in conventional deep learning [7], [14], wherein the early layers of a neural network reside on client devices and the later layers on a server; clients send intermediate activations (not raw data) to the server which continues the forward pass, then sends back gradients for the backward pass. Here, in a fully edge scenario, different segments of the SNN may reside on different peer devices, which could form a *chain* or a *distributed graph*.

A practical example is a multi-modal environmental sensing task: one node processes audio signals, like detecting thunder via an SNN trained on acoustic events, another processes visual events, like a neuromorphic vision sensor for lightning flashes, and their outputs feed into a third node that integrates both to decide if a lightning strike occurred. In this split architecture, no single node has the complete information; only the combination yields the full result. Traditional ANNs would send intermediate feature vectors among nodes; in SNN terms, the intermediate communication consists of spike trains.

It is worth noting, however, that, in practice, specific specialized layers or neuron groups may not be easily partitionable (e.g., layers that rely on hardware-specific dynamics). In such cases, a split learning pipeline can selectively offload only the portions or features that are inherently distributable while keeping specialized layers intact on a single node or gateway device.

Model definition: Consider an SNN divided into P layers (or functional blocks) labeled $1, \ldots, P$. For simplicity, assume a feed-forward layering (though split architectures can also apply to recurrent SNNs). Each layer p contains a set of spiking neurons with weights $W^{(p)}$ (connecting from layer $p-1$ to p). The layers are distributed across M nodes. For example, Node 1 might hold layers 1-2, Node 2 holds layer 3, Node 3 holds layers 4-5, etc, such that collectively layers 1-5 form the full network. When an input (e.g., sensor signal) is presented to the network, the neurons in layer 1 (on Node 1) produce spikes, which are transmitted to layer 2 (also on Node 1 if local, or to Node 2 if layer 2 is remote). Generally, whenever a neuron in layer p spikes, that event must be delivered to all neurons in layer $p+1$, wherever they reside. This can be implemented using an address-event representation protocol over IP or a custom routing scheme [15]. We assume the communication delay for a spike between nodes is small relative to the SNN processing timescale.

Training a split SNN requires coordinating updates across nodes. We focus on a distributed gradient descent approach, though one could also use local learning rules with occasional synchronization. In essence, split SNN training is equivalent to synchronous training of a single model on distributed hardware. One can perform forward propagation of a training sample's spikes through the network (with inter-node communication of spikes), compute the output loss, then perform backward propagation of gradients, where the error signals are communicated in the reverse direction. Conceptually, this is similar to how distributed deep learning is done across multiple devices, except that here the communication is event-driven.

Algorithm 2 provides a simplified view of one training iteration in a split SNN with two nodes for clarity (it can be generalized to M nodes). Node A holds the first part of the SNN (W^A) and Node B holds the second part (W^B). In the forward pass, Node A processes input x to produce spike outputs h_A which it sends to Node B. Node B continues the forward pass, producing output $\hat{y}$ and computing loss $\ell(\hat{y}, y)$. Then Node B computes the gradient of the loss w.r.t. its inputs (δ_B) and sends that back to Node A. This δ_B can be thought of as spikes/errors to be propagated backwards. Node A uses δ_B to update W^A via its local gradient. Node B updates W^B via its local gradient. In practice, this requires a differentiable spiking neuron model (e.g., using surrogate gradients [4]) or a local learning rule that achieves a similar effect.

In this way, training the split SNN yields the same result as training an equivalent integrated SNN on the combined data of all nodes. Importantly, none of the raw inputs x or local intermediate spikes in Node A are ever directly observed by Node B except via the transformed spiking pattern h_A. Similarly, Node A doesn't see Node B's portion of the network or the target y, only the feedback δ_B. This provides a degree of data privacy and modularity: each node's computations are hidden behind an interface of spike events. In sensitive environmental applications, such as private environmental audio recordings, this is beneficial.

An advantage of Split SNNs is that they naturally handle vertically partitioned data: if Node A had sensor modality 1 and Node B modality 2, neither alone could train a complete model, but together (with split learning) they can. The split approach also allows a single large SNN model to be spread in memory across multiple low-resource devices, meaning each device only needs to store its segment's weights.

Communication and runtime: Split SNNs require frequent communication: every spike that crosses a partition

Algorithm 2 Split SNN Training (for two-node case)

Require: Node A has weights W^A (layers $1..p$), Node B has W^B (layers $p+1..P$)
Require: Training sample (x, y)
Forward:
Node A: Input x (spike train or analog signal) into SNN_A (layers $1..p$)
Node A: Run SNN_A to produce output spike train h_A
Node A $\rightarrow$ Node B: Transmit h_A spikes to Node B
Node B: Receive h_A and feed into SNN_B (layers $p+1..P$)
Node B: Run SNN_B to produce output $\hat{y}$ (spike counts or membrane potentials)
Node B: Compute loss $\ell = \ell(\hat{y}, y)$ and output error $\delta_B = \partial\ell/\partial h_A$
Backward:
Node B: Compute gradient $\nabla_{W^B}\ell$ and update W^B
Node B $\rightarrow$ Node A: Send error signal δ_B back to Node A (e.g., as spike gradient)
Node A: Compute $\nabla_{W^A}\ell$ using δ_B and update W^A

boundary induces a network message. However, spikes are binary events and typically sparse in time, so the data volume can be low. We will analyze this trade-off in Section III. The end-to-end inference latency in a split SNN includes both neural processing time and network transmission delays. If the SNN requires T milliseconds (ms) to propagate through all layers (assuming sequential processing) and the inter-node link adds d ms of latency for each spike hop, a chain of M nodes would incur roughly $(M-1) \times d$ additional latency. In many cases, like short-range radio or wired links, d might be negligible (sub-millisecond), so the distributed SNN acts almost as a single device. In other cases, especially wireless sensor networks, latency and reliability must be considered (see Section III).

Finally, we note that hybrid approaches are possible: for instance, one might federate multiple split networks (if there are groups of nodes, each group running a split SNN, and groups exchange models occasionally). However, in this paper we focus on the two paradigms separately for clarity.

III. Theoretical Analysis

We now provide theoretical insights into the performance of the two distributed learning paradigms, focusing on communication efficiency, convergence properties, and scalability.

A. Convergence of Federated SNN training

Federated SNN training can be viewed as an instance of federated optimization, which has been studied extensively in the literature for conventional neural networks [13]. Under standard assumptions (convex or smooth loss surfaces, bounded gradients, etc.), one can derive convergence guarantees for FedAvg. While spiking neural networks are often non-convex and involve discrete events, recent work suggests that with surrogate gradient methods their training dynamics are similar to conventional neural nets [4]. We will assume that each local training step performs a (stochastic) gradient descent on $L_k(W)$ and that the global objective $L(W)$ is differentiable and L-smooth.

Theorem 1 (Convergence of FedSNN): Suppose each local loss $L_k(W)$ is L-smooth and the stochastic gradients have bounded variance. Let η be a suitably small learning rate. Then the FedSNN algorithm (Algorithm 1) will converge to a critical point W^* of the global loss $L(W)$. Moreover, after R communication rounds, we have

$$\mathbb{E}[L(W^R) - L(W^*)] \;\leq\; \frac{C}{\eta R} + O(\sigma^2 \eta E),$$

for some constant C that depends on initial loss difference and σ^2 the variance of stochastic gradients, and E is the number of local epochs per round. By choosing η appropriately (decaying over rounds), the error can be made $O(1/R)$.

Proof Sketch: This result follows the outline of convergence proofs for FedAvg (e.g., [13] and refinements in [16]). The key is that at each round, the average update $\sum_k \frac{|D_k|}{|D|}(W_k - W^{r-1})$ is an unbiased estimate of the true gradient of L at W^{r-1}, assuming each node performs local SGD. The L-smoothness ensures descent, and additional error terms arise from doing $E > 1$ local steps without synchronization (leading to a term depending on data heterogeneity). For SNNs, the surrogate gradient method ensures we can treat the gradient as bounded. Thus, FedSNN converges in a similar manner to standard FedAvg. (For completeness, one could extend the formal proofs in [12] for spiking networks, but that is beyond our scope.) ■

In essence, Theorem 1 indicates that as the number of synchronization rounds increases, the federated model approaches the performance of a hypothetical centralized SNN trained on all the data. Practical studies have shown that even a handful of rounds can yield large gains over purely local training [11]. One must balance the frequency of synchronization (communication cost) with the speed of convergence.

Another consideration is *staleness*: if different nodes run local training at different speeds or communication is delayed, one might aggregate partially out-of-sync models. Whereas techniques like asynchronous update or staleness control [8] can be used, we assume synchronous rounds for simplicity.

B. Communication efficiency analysis

We now compare the communication overhead of the two paradigms, and also compare with a baseline scenario of centralizing all data for training at a server, which is often infeasible in practice but provides a reference point.

In federated SNN, communication occurs intermittently: each round, each node sends its model (W_k, size $|W|$ bytes) to the server and receives the global model ($|W|$ bytes). If using only weight differences or compressed updates, the size can be smaller, but let's assume full model exchange for worst-case analysis. Thus, per round, each node communicates $2|W|$ bytes (send and receive). Over R rounds, each node communicates $2R|W|$ bytes. The total network traffic (summed over all nodes) is $2KR|W|$ bytes (though the server is the hub so it sends/receives K times more than each node). Importantly, this does *not* depend on the amount of data $|D|$ at the nodes. In contrast, a non-federated approach that sends all raw data to a central server would send $\sum_k |D_k|$ data points, which could be orders of magnitude larger, especially if each data point is high-dimensional, e.g., lengthy time-series.

For a concrete example, suppose each node's SNN has $|W| = 10^5$ parameters (which at 1 byte each is 100 KB). With $K = 10$ nodes and $R = 20$ rounds, the total communication is $2 \cdot 10 \cdot 20 \cdot 100\text{KB} = 40,000\text{KB} \approx 40$ MB. If instead each node had to transmit its raw data, say $|D_k| = 100\text{k}$ samples of 1KB each (100 MB per node), that would be 1GB of raw data total. Thus, federated learning can drastically cut communication, making global training feasible over bandwidth-constrained links.

In split SNN, communication is continuous during operation. Every input sample (or every time step of a stream) causes spikes to be exchanged between partitions. The volume depends on the activity. Let us denote by $S_{i\to j}$ the number of spikes sent from node i to node j during the processing of one sample (or one unit time interval). Each spike might be represented by a small packet (with an address/ID, possibly timestamp). Assume each spike event is, say, 1 byte (in an ideal compressed form) or a few bytes. The total communication per sample is $\sum_{\text{edges } i\to j} S_{i\to j} \cdot$ (bytes per spike). If the network is mostly idle with occasional spikes, this could be very low. In the worst case of a very active SNN (all neurons firing at high rates), this could be higher than sending raw data. However, one reason to use SNNs is that they often exploit sparsity: neurons fire only on significant changes. For environmental monitoring, one might tune the SNN so that only anomalies or important features trigger spikes.

We can derive a simple bound: suppose the SNN has N neurons in total and each neuron emits at most r spikes during the processing of an input (or per unit time). If the N neurons are evenly divided among M nodes, then roughly $\frac{M-1}{M}$ fraction of spikes cross a boundary (this is a pessimistic estimate assuming inter-node connections are dense). Then the total spikes communicated per sample $\leq rN\frac{M-1}{M}$. For example, if $N = 1000$ neurons, $r = 5$ spikes each, $M = 5$ nodes, then $\approx 4/5$ of 5000 spikes are inter-node, i.e., 4000 spikes per sample. At a few bytes each, that's on the order of only 10–20 KB of data communicated per sample. If each sample is, say, a 1-second window of sensor signals originally maybe kilobytes of analog readings, this is quite efficient. On the other hand, federated learning would not send anything during inference; it sends model updates offline. So split vs federated communication is scenario-dependent: split sends smaller messages but continuously, federated sends larger messages but infrequently.

In a streaming scenario, federated might send, e.g., a model every hour, whereas split sends trickle of spike data every second. One can also reduce split SNN communication by introducing local event filtering: e.g., only significant spikes propagate. There is active research on communication-sparsity techniques for distributed SNNs [15].

We summarize the communication characteristics in Table I. Federated SNN communication grows with the model size and number of rounds, but *not* with number of samples. Split SNN communication grows with the number of spikes and network connections, i.e., with activity and network depth, but not directly with model size (a large model can still be communication-efficient if most of its neurons are quiet).

In large deployments, a hybrid may be optimal: one could use federated learning at a higher level (between clusters of sensors) and split learning within a small cluster of sensors measuring related features. federation reduces the need to constantly send raw data over long distances, while partitioning/splitting maximizes joint inference capability within a local area.

C. Scalability considerations

Scalability can refer to three aspects: scaling with the number of edge nodes, scaling with the size of the SNN model or data, and energy scalability.

Number of nodes K: Federated SNN training algorithms can in principle scale to very large K, since the server's aggregation is an $O(K|W|)$ operation and each node does $O(|W|)$ work. Techniques exist to reduce server load, like hierarchical federated learning, if K is extremely large [17]. The convergence rate may degrade if data is extremely non-iid across many nodes, but even then strategies like grouping or personalization can help. In contrast, Split SNNs typically involve a relatively small number of nodes forming a distributed model. While one could imagine a very large distributed SNN spanning dozens of hops, synchronization and reliability become problematic as M grows. Split learning is usually considered for small M (e.g., a client and a server, or a few clients in a chain). If one tries to make M large, the latency accumulates and the chance of some link failing is high. Thus, we expect federated approaches to handle larger networks of devices, while split approaches will be used in smaller clusters or between a few partners.

Model/Data size: Federated SNNs allow each node to hold the full model, so model size is limited by node memory but not by the network structure. If an SNN with millions of parameters can run on each neuromorphic chip, FedSNN can train it with more communication cost. Split SNNs allow an even larger model to be divided among nodes if one node cannot hold it entirely. This is useful if we want an extremely detailed model that no single low-power device could store. By splitting layers across devices (akin to how the SpiNNaker machine spreads a large neural network over many chips [3]), we achieve greater model capacity. The trade-off is again the communication overhead and complexity of coordination.

From a data perspective, if the total amount of data grows (more sensors or more time), federated learning can harness it by increasing rounds or local epochs; split learning can handle more data by continuous operation since inference is online, and learning can be continuous or done in batches. Neither inherently has a bottleneck in data size aside from the usual training time considerations.

TABLE I
COMMUNICATION CHARACTERISTICS OF FEDERATED VS SPLIT SNN PARADIGMS

Paradigm	Comm. per Training Round	Comm. per Inference
Federated SNN	$\approx 2K\|W\|$ bytes (model sync)	0 (local inference only)
Split SNN	0 (training is on combined model)	$\sim$ #spikes per sample $\times$ few bytes

One must also consider *energy scalability*. Neuromorphic hardware is extremely energy-efficient for spiking computation [4], often consuming microjoules per inference. Federated SNN adds some overhead for periodic communication as transmitting a model might cost as much energy as performing many inferences on-chip. Split SNN shifts some energy consumption to the communication channel (radio transmit/receive costs). If nodes are battery-powered, we need to ensure communication does not dominate the energy budget. Generally, sending a single 32-bit packet over radio can consume more energy than a million synaptic operations on a neuromorphic chip [4]. This favors minimizing communication events. FedSNN does infrequent but large transfers, which can be scheduled when solar power is available, etc. Split SNN does frequent tiny transfers; if using an always-on radio, the idle listening might waste energy. Careful protocol design, like event-triggered wake-up is required. Work by Liu *et al.* [15] optimized a wireless SNN implementation and found that distributing an SNN across multiple edge devices could still be energy-efficient overall, thanks to reduced local load per device and event-driven communication.

Proposition 1 (Communication-accuracy trade-off): For a given learning task, there exists a trade-off frontier between communication cost and accuracy. In general, solutions like federated SNN and split SNN lie on this frontier, achieving high accuracy with much lower communication than naive baselines, such as centralizing data or isolated local models. Federated SNN minimizes communication at the expense of requiring more local computation rounds, whereas split SNN minimizes latency and maximizes accuracy (by using all features in real-time) at the expense of more communication per inference. Any intermediate approach will have to balance these factors.

NeuEdge strikes a balance between communication cost and model accuracy by adjusting two levers: the frequency of federated synchronization (which impacts overall communication volume) and the sparsity of spike exchange in split SNN (which controls per-inference overhead). Hence, each deployment can tune these factors, e.g., fewer federated rounds but higher local epochs or more event-filtering in split SNN, to meet specific bandwidth, latency, or accuracy targets. While Proposition 1 is stated qualitatively, one could formalize it via multi-objective optimization: minimize $L(W)$ (error) and communication C. Federated and split are two strategies to reduce C while keeping error low. The proof is by construction of scenarios. We showed that local-only training (no communication) can severely hurt accuracy if data or features are partitioned, and conversely that communicating everything (centralized training) maximizes accuracy at cost C. Thus, there is no gold standard. One must exchange some information to achieve better performance, but NeuEdge paradigms significantly cut down the information exchanged (only weights or spikes, not raw signals).

IV. EXPERIMENTAL EVALUATION

In this section, we experimentally validate our proposed NeuEdge approaches on both a synthetic dataset and the Intel Lab sensor dataset [18].

A. Datasets and preprocessing

a) Synthetic dataset: We generate a two-dimensional dataset of size 5000, sampling each feature from a standard normal distribution. A binary label is assigned based on whether $x_1 + x_2 > 0$, after which random label noise (5%) is injected. For federated training, we split the dataset into two partitions (each comprising roughly half of the samples), and further subdivide each partition into an $80\,\%$ training set and $20\,\%$ test set. For the split-learning scenario, we partition the *features*, so that Node A sees only the first feature (x_1) and Node B sees only the second (x_2).

b) Intel Lab dataset: We use measurements from the public Intel Berkeley Research Lab deployment, which recorded temperature, humidity, light, and voltage across multiple mote sensors [18]. Following standard practices, we segment each sensor's time-series into fixed-size windows (64 samples per window) and normalize each dimension. We then artificially inject, very hard to distinguish, *faults* in some fraction (20%) of the windows. Each window is labeled *faulty* if its temperature readings have been perturbed; otherwise *normal*. We split each sensor's data into $70\,\%$ train and $30\,\%$ test windows. In the federated scenario, each sensor is treated as an autonomous node. In the split scenario, we select two sensors and emulate a two-node setup (Node A and Node B), each holding distinct subsets of the input features.

B. Implementation and metrics

For *federated SNN*, we rely on a two-layer spiking MLP (with a threshold-based spiking hidden layer and a logistic output neuron) at each node, periodically synchronized using a weighted average of parameters (FedAvg). Each node performs multiple local epochs of mini-batch training before synchronization. For *split SNN*, we use a more advanced structure in which Node A is a deeper *spiking encoder* (outputting a vector of spikes), and Node B is a small MLP that combines Node A's spike vector with Node B's local features. The entire system is trained end-to-end via a surrogate-gradient approach that passes error signals back to Node A.

We evaluate performance in terms of classification accuracy (%) and F1 score. All experiments use cross-entropy loss, a moderate mini-batch size (e.g., 32), and a sufficient number of training epochs ($10 - 20$) to allow convergence.

C. Results on the Synthetic dataset

Table II summarizes the final performance of both federated SNN and split SNN on the synthetic data. Our federated approach achieves around $88.7\,\%$ accuracy and an F1 of 0.892, closely approaching a single centralized model. The split approach also performs well, at approximately $81.9\,\%$ accuracy and 0.821 F1. These results confirm that both paradigms successfully learn the latent decision boundary. The marginal gap between federated and split learning can be attributed to the fact that split nodes each receive only partial features and rely on the spike-based encoding to communicate the missing dimensions, whereas in federated learning each node eventually gains all feature knowledge through parameter sharing.

TABLE II
PERFORMANCE ON THE SYNTHETIC DATASET.

Method	**Accuracy (%)**	**F1 Score**
Federated SNN	88.7	0.892
Split SNN	81.9	0.821

Fig. 2 compares the final performance metrics for federated and split training on the synthetic dataset with the centralized SNN approach, where centralized training has been applied to all data. Federated SNN approaches the centralized model's performance, demonstrating the effectiveness of parameter-sharing under non-i.i.d. conditions. Meanwhile, split SNN achieves nearly the same final performance by dynamically fusing partial features from multiple nodes in a single end-to-end model.

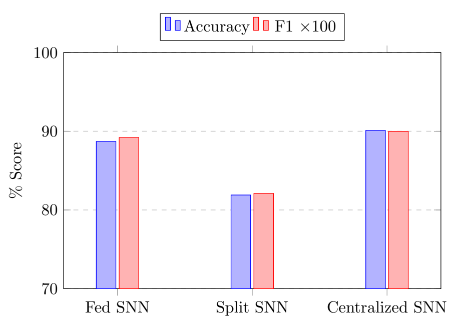


Fig. 2. Comparison of federated SNN, split SNN, and a centralized SNN reference in terms of Accuracy and F1. Both federated and split SNN approaches successfully combine knowledge from multiple nodes, approaching or matching the centralized baseline without centralizing raw data.

D. Results on the Intel Lab dataset

For the Intel Lab fault-detection task, we observe slightly lower absolute values of accuracy and F1 compared to the synthetic problem. Table III lists the performance of both paradigms. Federated SNN obtains about $74.4\,\%$ accuracy and 0.68 F1, while the split scenario reaches about $76.2\,\%$ accuracy and 0.7 F1.

TABLE III
PERFORMANCE ON INTEL LAB DATASET (ANOMALY DETECTION).

Method	**Accuracy (%)**	**F1 Score**
Federated SNN	74.4	0.68
Split SNN	76.2	0.7

Although these results appear not as high as one might initially expect, there are several confounding factors that can hamper anomaly classification in this dataset:

- The artificially injected "faults" (e.g., adding a constant offset to temperature) can overlap with natural variations in humidity or voltage, thereby creating borderline cases that are not easily separable by a threshold-based or spike-based model. More precisely, as the injected faults were deliberately made to be not very distinct from the real sensor readings, causing the noisy data to blend with the real pattern, the approach faced a considerable challenge in detecting anomalies.
- The Intel deployment is known to exhibit significant noise and missing readings, and combining multiple sensors can lead to highly non-i.i.d. data distributions. This challenges both federated parameter averaging and direct feature-splitting.

Nonetheless, these results underscore that even in complex real-world conditions, the proposed NeuEdge methodology remains nicely operational and yields useful detection models without transferring raw data. We stress that by further refining hyperparameters (e.g., deeper spiking layers, more local epochs, or domain-specific encoders), performance can be improved in real deployments. More specifically, according to Theorem 1, one can naturally expect that under smoothness and bounded gradient assumptions, the distributed federated solution converges to the same optimum that would be found by training on all data centrally. More specifically, if synchronization is carried out often enough and local updates faithfully approximate gradient steps, the FedSNN approach will asymptotically achieve the same performance as a hypothetical centralized SNN model.

E. Complementary discussion

In addition to the above findings, we highlight some conceptual discussion of latency, energy, and communication overhead.

a) Latency and energy considerations: Neuromorphic edge devices are typically designed for event-driven, low-latency inference. Because SNNs process and transmit information only when a relevant spike occurs, their on-device inference can be very fast and power-efficient, often running in real time or near real time. In a federated learning setting, each edge node performs local inference with almost no additional latency compared to a standalone model, since predictions are made entirely on-site. By contrast, a split learning approach may involve transmitting spikes between two or more networked nodes before a final output can be produced, potentially introducing extra delay proportional to the number of hops and the underlying communication protocol. Still, because spike events are typically sparse and require minimal payload, this overhead can remain small.

From an energy standpoint, neuromorphic hardware is known for very low per-inference power consumption. Hence, the chief additional factor comes from radio or network transmission, which must be carefully managed in split learning to avoid undue power costs. Neuromorphic hardware platforms like Intel's Loihi [3] and IBM's TrueNorth [19] have demonstrated energy consumption in the range of microjoules per inference, significantly lower than conventional GPU or CPU-based approaches. For instance, Loihi consumes up to 27$\times$ less power than a standard edge AI accelerator when executing spiking workloads [5].

b) Communication overhead: In federated SNNs, communication is infrequent and typically limited to exchanging model updates or parameters during specific synchronization rounds. This means that, once trained, each node operates almost independently, reducing the volume of data transmitted over the network. In contrast, split SNNs continuously transmit spike-based activations among distributed layers or segments of the model. While these spike messages are usually small and sparse, their cumulative volume can grow with high activity or a large number of distributed nodes. Nonetheless, in practical environmental sensing scenarios, where the frequency and amplitude of events tend to be moderate, spike-based communication can still compare favorably to streaming raw sensor signals or large feature maps. In both paradigms, a key design objective is to keep these transmissions minimal while leveraging neuromorphic hardware's strengths in real-time and event-driven inference.

c) Implications for environmental monitoring: Many environmental and disaster-response applications, such as flood or wildfire detection, benefit from having low-latency alerts and low-power sensing nodes that can run for extended periods under battery or solar constraints. Neuromorphic edge systems align well with these demands, as they bring computation close to the source and avoid costly data transfers to the cloud. In a federated framework, diverse sensor nodes located across different geographic regions can train collectively to capture varying environmental conditions, leading to more robust models without exchanging sensitive or bandwidth-intensive data. Similarly, split SNN architectures enable collaborative inference over multiple modalities (e.g., combining audio spikes from one sensor with temperature spikes from another), effectively constructing a distributed *edge brain*. These approaches have the potential to reduce false alarms, spot complex multi-sensor anomalies, and deliver alerts quickly, ultimately enhancing situational awareness and responsiveness in critical scenarios.

V. Conclusion

We have presented *NeuEdge*, a framework uniting neuromorphic computing and geo-distributed edge intelligence, and explored two key paradigms within it: *Federated SNNs* and *Split SNNs*. Through algorithms, theoretical analysis, and experiments, we demonstrated that:

- *Federated SNNs* enable multiple neuromorphic edge devices to collaboratively learn a global brain-inspired model without exchanging raw sensor data. This approach addresses data scarcity at individual nodes and adapts to non-iid data distributions, all while keeping communication efficient and sporadic. We proved that under reasonable assumptions this algorithm converges to an optimal solution similarly to traditional federated learning, which we validated by reaching high accuracy in our experiments.
- *Split SNNs* allow a single spiking neural network to be geographically distributed across sensor nodes, supporting scenarios where each node has only part of the sensory input. Our results showed that split SNNs achieve desirable performance. The communication overhead scales with spike activity, which in event-driven systems can be very low.

These findings highlight the potential of NeuEdge for real-world environmental monitoring and disaster response applications. By leveraging the inherent event-driven efficiency of SNNs and appropriate distributed strategies, one can build intelligent sensing networks that are low-latency, energy-frugal, and privacy-preserving. For example, a forest wildfire detection system might use split SNNs to combine smoke sensor spikes and temperature sensor spikes from nearby points to improve reliability, and use federated updates to adapt the detectors to different forest conditions over time.

References

[1] D. R. Muir and S. Sheik, "The road to commercial success for neuromorphic technologies," *Nature Communications*, vol. 16, no. 1, p. 3586, 2025.

[2] G. Zhang, L. Feng, F. Zhou, Z. Yang, Q. Zhang, A. Saleh, P. K. Donta, and C. K. Dehury, "Spiking neural networks in intelligent edge computing," *IEEE Consumer Electronics Magazine*, pp. 1–9, 2024.

[3] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[4] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay *et al.*, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[5] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[6] A. Ferdowsi, M. Függer, T. Nowak, U. Schmid, and M. Drmota, "Faithful dynamic timing analysis of digital circuits using continuous thresholded mode-switched odes," *Nonlinear Analysis: Hybrid Systems*, vol. 56, p. 101572, 2025.

[7] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[8] A. Aral, M. Erol-Kantarci, and I. Brandić, "Staleness control for edge data analytics," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–24, 2020.

[9] A. Aral, “The promise of neuromorphic edge ai for rural environmental monitoring,” *Environmental Data Science*, vol. 3, p. e34, 2024.

[10] S. A. Tumpa, S. Singh, M. F. F. Khan, M. T. Kandemir, V. Narayanan, and C. R. Das, “Federated learning with spiking neural networks in heterogeneous systems,” in *Proceedings of the 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2023, pp. 1–6.

[11] N. Skatchkovsky, H. Jang, and O. Simeone, “Federated neuromorphic learning of spiking neural networks for low-power edge intelligence,” in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8524–8528.

[12] Y. Wang, S. Duan, and F. Chen, “Efficient asynchronous federated neuromorphic learning of spiking neural networks,” *Neurocomputing*, vol. 557, p. 126686, 2023.

[13] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. 20th Intl. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[14] D. Yu, X. Du, L. Jiang, W. Tong, and S. Deng, “Ec-snn: Splitting deep spiking neural networks for edge devices,” in *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024, pp. 5389–5397.

[15] Y. Liu, Z. Qin, and G. Y. Li, “Energy-efficient distributed spiking neural network for wireless edge intelligence,” *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 1874–1888, 2024.

[16] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[17] S. Ahmad and A. Aral, “Hierarchical federated transfer learning: A multi-cluster approach on the computing continuum,” in *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 1163–1168.

[18] I. B. R. Lab, “Intel lab data,” http://db.csail.mit.edu/labdata/labdata.html, accessed: 2025-05-03.

[19] J. Paul, “How brain-inspired ai chips are changing the game: The rise of neuromorphic computing,” 2024.