



Article

# EdgeMap: An Optimized Mapping Toolchain for Spiking Neural Network in Edge Computing

Jianwei Xue , Lisheng Xie, Faquan Chen, Liangshun Wu , Qingyang Tian, Yifan Zhou, Rendong Ying and Peilin Liu \*

School of Electronic and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; xuejw2019@sjtu.edu.cn (J.X.); 15955192071@sjtu.edu.cn (L.X.); fqchen1998@sjtu.edu.cn (F.C.); wuliangshun@sjtu.edu.cn (L.W.); tqyang\_sjtu@sjtu.edu.cn (Q.T.); yifanzhou-0713@sjtu.edu.cn (Y.Z.); rdying@sjtu.edu.cn (R.Y.)

\* Correspondence: liupeilin@sjtu.edu.cn

**Abstract:** Spiking neural networks (SNNs) have attracted considerable attention as third-generation artificial neural networks, known for their powerful, intelligent features and energy-efficiency advantages. These characteristics render them ideally suited for edge computing scenarios. Nevertheless, the current mapping schemes for deploying SNNs onto neuromorphic hardware face limitations such as extended execution times, low throughput, and insufficient consideration of energy consumption and connectivity, which undermine their suitability for edge computing applications. To address these challenges, we introduce EdgeMap, an optimized mapping toolchain specifically designed for deploying SNNs onto edge devices without compromising performance. EdgeMap consists of two main stages. The first stage involves partitioning the SNN graph into small neuron clusters based on the streaming graph partition algorithm, with the sizes of neuron clusters limited by the physical neuron cores. In the subsequent mapping stage, we adopt a multi-objective optimization algorithm specifically geared towards mitigating energy costs and communication costs for efficient deployment. EdgeMap—evaluated across four typical SNN applications—substantially outperforms other state-of-the-art mapping schemes. The performance improvements include a reduction in average latency by up to 19.8%, energy consumption by 57%, and communication cost by 58%. Moreover, EdgeMap exhibits an impressive enhancement in execution time by a factor of 1225.44×, alongside a throughput increase of up to 4.02×. These results highlight EdgeMap’s efficiency and effectiveness, emphasizing its utility for deploying SNN applications in edge computing scenarios.

**Keywords:** edge computing; spiking neural networks; neuromorphic hardware; mapping



**Citation:** Xue, J.; Xie, L.; Chen, F.; Wu, L.; Tian, Q.; Zhou, Y.; Ying, R.; Liu, P. EdgeMap: An Optimized Mapping Toolchain for Spiking Neural Network in Edge Computing. *Sensors* **2023**, *12*, 6548. <https://doi.org/10.3390/s23146548>

Received: 28 June 2023  
Revised: 13 July 2023  
Accepted: 18 July 2023  
Published: 20 July 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The growing popularity of edge devices, such as wearable electronics, the Internet of Things devices, industrial machinery, and smartphones connected to the internet, has been driving the development of intelligent applications [1]. Edge computing has emerged as a promising approach to optimize overall system performance by reducing latency and conserving bandwidth through processing data closer to the source rather than relying on centralized nodes. As a result, there is a growing interest in endowing edge computing with greater intelligence [2].

Spiking neural networks (SNNs) demonstrate significant potential in providing edge devices with intelligent capabilities due to their low energy consumption and powerful biomimetic intelligence features [3]. An increasing number of SNNs have exhibited exceptional computational performance in various edge computing applications, including object recognition [4], speech recognition [5], and robotics control [6]. Through the utilization of SNNs, edge devices can be transformed into more intelligent nodes capable of executing complex intelligent applications with enhanced accuracy and efficiency.

Recently, neuromorphic hardware designs, such as TrueNorth [7], Loihi [8], and SpiNNaker [9], have recently emerged. These designs feature multiple interconnected neuromorphic cores (NCs) through a network-on-a-chip (NoC), enabling the energy-efficient and high-performance execution of SNNs [10]. To bridge the gap between SNNs and hardware implementation, various mapping schemes have been proposed [11–13]. These schemes typically involve two stages: partitioning and mapping. During the partitioning stage, the computation graph of SNN neurons is divided into clusters under the constraints of the neuromorphic hardware. The mapping stage then places these clusters onto physical neuron cores to achieve different optimization objectives.

However, while these mapping schemes enhance the performance, scalability, and accuracy of SNNs on neuromorphic hardware, they primarily focus on neuromorphic hardware deployed at the server end at present. This approach assumes a stable environment and unlimited computational resources, which is not reflective of edge computing scenarios. Implementing SNNs on compact edge devices, thus, presents unique challenges, making it a more complex and rigorous task than server-end deployments. These challenges may include:

- **Energy constraints:** Edge computing devices, typically standalone units, operate under stringent energy constraints, making power efficiency a critical factor in their operation. Different mapping schemes can lead to substantial disparities in energy consumption, significantly affecting the performance and viability of these devices. These realities underscore the urgent need for mapping schemes that thoroughly consider energy efficiency.
- **Real-time processing:** Many edge computing applications require real-time or near-real-time processing, necessitating low latency and efficient network communication. However, achieving these performance characteristics without compromising other essential attributes, such as energy efficiency, is a challenging balancing issue.
- **Complex environments:** Edge computing scenarios are inherently complex and diverse, often with varying constraints and requirements. Edge environments, unlike server-end environments, can have much more diverse and demanding constraints and requirements.

Drawing upon the aforementioned challenges, this paper aims to explore the optimal mapping toolchain for SNN applications under multiple edge computing environments. We develop a partitioning algorithm for various SNNs to enable rapid and resource-efficient partitioning on edge devices. The mapping optimization strategy is then applied to meet edge computing requirements. The main contribution of this paper is to provide a more efficient and flexible deployment of neuromorphic computing in edge computing applications while also improving the overall system performance.

- We propose a novel partitioning method inspired by the streaming-based approach [14]. This method efficiently partitions SNNs into multiple neuron clusters, significantly reducing partitioning time while minimizing the number of spikes between clusters. Importantly, our approach takes into account the fan-in and fan-out limitations of neuromorphic hardware, ensuring compatibility with edge device constraints.
- We propose a multi-objective optimization method to place neuron clusters onto neuromorphic devices. This method addresses the diverse challenges in edge computing, including bandwidth limitations, communication costs, and energy consumption. Our approach, while successfully reducing energy consumption and latency, crucially enhances the overall performance of neuromorphic computation in edge environments.
- We propose a versatile mapping toolchain suitable for a wide range of neuromorphic edge devices. By conducting extensive evaluations of various SNNs and scales on NoC-based hardware, we demonstrate significant enhancements in edge computing performance. In addition, our toolchain can accommodate the diverse constraints and requirements of different edge computing scenarios.

This paper is organized as follows: Section 2 provides a review of the related work. In Section 3, we discuss the necessary background for understanding the subsequent sections. Section 4 delves into the design methodology of EdgeMap. Section 5 outlines the setup used for our evaluations. In Section 6, we present a detailed account of our experimental results. Finally, in Section 7, we conclude this paper.

## 2. Related Work

Recently, SNNs have garnered interest due to their energy efficiency and computational capabilities [3]. However, executing SNNs on neuromorphic hardware is challenging because of their complex connections and hardware limitations. Many efforts have been made to bridge this gap, with current SNN mapping schemes generally falling into three categories: dedicated hardware mapping, general mapping for crossbar-based hardware, and other flexible mapping schemes.

The first category involves dedicated hardware design schemes specifically tailored for specialized SNN solutions. Examples of such mapping schemes include SentryOS [15] for  $\mu$ Brain [16], the Corelet toolchain [17] for TrueNorth [7], and the LCompiler [18] for Loihi [8]. These complete toolchains aim to optimize core utilization to the maximum extent. The SentryOS toolchain comprises a SentryC compiler, which partitions SNN network structures into multiple subnetworks, and a SentryRT real-time manager responsible for sorting and computing different subnetworks in real-time. Corelet, a dedicated programmable toolchain for TrueNorth, encapsulates biological details and neuron complexity before proposing an object-oriented Corelet language to compose and then execute SNNs. The LCompiler framework maps SNNs onto Loihi neuromorphic hardware by presenting a data flow graph with logical entities describing an SNN instance, such as compartments, synapses, input maps, output axons, and synaptic traces. The framework first transforms SNNs into microcodes and translates SNN topology into a connection matrix. Next, it employs a greedy algorithm to map logical entities to hardware before generating the bitstream for Loihi cores. In the study by Wang et al. [19], they presented an innovative mapping method specific to the Tianjic neuromorphic chip [20]. This approach comprises two stages: logical mapping and physical mapping. In the logical mapping stage, the authors introduce a closed-loop mapping strategy that employs an asynchronous 4D model partition. For the physical mapping stage, a Hamilton loop algorithm is applied. The advantage of this method is its ability to achieve high resource utilization and processing efficiency, providing an interesting perspective for the efficient deployment of neural networks on neuromorphic hardware. All of these mapping toolchains are designed for specific targets, restricting their universal applications. This restricts their adaptability for more general-purpose use.

The second category focuses on crossbar-based SNN topology schemes, which are more general mapping techniques. These schemes include state-of-the-art mapping techniques, such as NEUTRAMS [21], SpiNeMap [13], SNEAP [22], and DFSynthesizer [23]. NEUTRAMS is a co-design toolchain that partitions SNNs to meet hardware constraints and optimizes their mapping onto neuromorphic hardware. SpiNeMap adopts a greedy approach, loosely based on the Kernighan–Lin graph partitioning algorithm [24], to minimize inter-cluster spike communication, and it utilizes a PSO algorithm [25] to place each cluster on a physical core. SNEAP [22] uses the METIS graph partition algorithm to partition the SNN computation graph with the objective being the communication cost, then it uses a greedy algorithm to become the final mapping results. DFSynthesizer [21] is an end-to-end framework for mapping machine learning algorithms. It decomposes and partitions SNNs into clusters before exploiting the rich semantics of synchronous data flow graphs to explore different hardware constraints that influence mapping performance. These existing toolchains are designed mainly for cloud servers and do not consider the constraints of complex computing environments like edge computing. This could limit their applicability in resource-constrained edge scenarios where environmental factors come into play.

Various works have been proposed to address specific challenges in mapping schemes. For instance, Jin et al. [26] aimed to address the challenge of mapping large-scale SNNs onto neuromorphic hardware. Their approach, which employs a Hilbert curve and a force-directed algorithm to optimize the mapping stage, achieves remarkable time degradation when deploying large-scale SNNs. Similarly, Nair [27] suggested mapping recurrent neural networks to in-memory computing neuromorphic chips by modifying the ANN to an RNN unit with an adaptive spiking neuron model and scaling it to fit the chip. Additionally, Migspike [28] was introduced to address reliability concerns resulting from the probability of accumulating faults, and it uses a node-level recovery mechanism for neuron failures by placing spare neurons into each neuron core, implemented through a mapping method for defective neurons.

In summary, various schemes have been proposed to optimize the mapping of neuromorphic hardware to achieve different optimization goals. To build upon these current mapping schemes, the objective of this paper is to introduce a novel mapping toolchain that is customized for complex edge computing scenarios with limited energy and computing resources that also require low latency.

### 3. Background and Motivation

This section begins with a brief overview of SNNs, emphasizing their unique attributes that facilitate highly efficient computational patterns and capabilities. Subsequently, we provide an introduction to edge computing, delving into the complexities of its inherent environments and the accompanying requisites.

#### 3.1. A Brief Introduction to SNN

Inspired by mammalian brains, SNNs have become increasingly popular for use in AI applications due to their ability to process data in a more efficient manner than traditional neural networks. Moreover, SNNs are less prone to overfitting and are better at capturing nonlinear patterns, making them well-suited for a wide range of artificial intelligence applications [6,29,30].

A typical and fundamental SNN model, as illustrated in Figure 1, can be represented as a directed graph consisting of a series of connected layers. The model might also incorporate inter-layer connections, enhancing the network's connectivity and learning capabilities. Each neuron in an SNN is connected to a specific number of neurons, with adjustable connection weights, allowing the network to learn from data. During computation, each neuron in an SNN processes weighted stimulus spikes from neighbors, generating and transmitting spike outputs to output neurons. This iterative process continues until the output layer produces the final results.

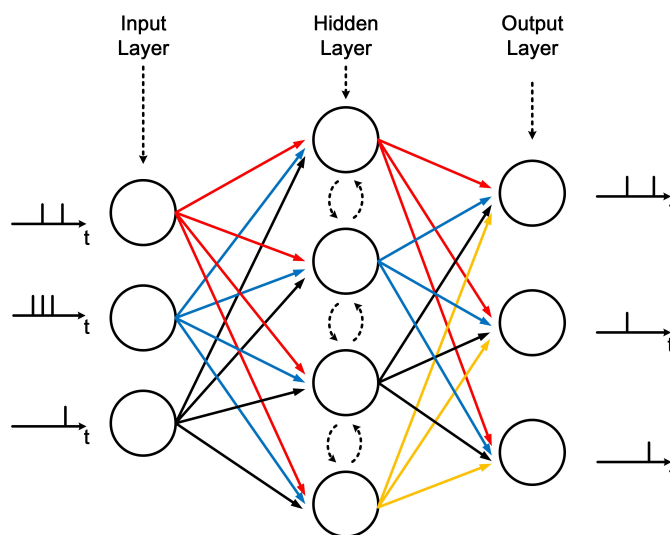
In this paper, we focus primarily on the inference of SNNs rather than training. This is because the training process is generally performed offline through powerful cloud data centers and is typically delay-tolerant [31], which we do not consider in this paper.

#### 3.2. Edge Computing

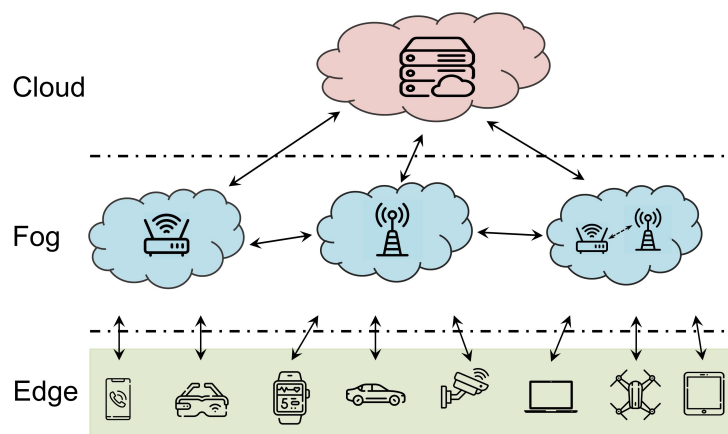
The edge–fog–cloud architecture, which has emerged as a popular paradigm in distributed computing, as depicted in Figure 2, has demonstrated its applicability across various domains for its significant advantages in managing computational resources and reducing communication latency [32]. This structure incorporates three layers: the edge, fog, and cloud layers, with the fog layer serving as a bridge between edge and cloud computing. Unlike traditional cloud computing, which predominantly depends on input data from edge devices such as images [33], speech [34], and videos [35], edge computing alleviates the burden on cloud computing centers by offering processing and storage services closer to end-users. However, progressing from the edge towards the cloud usually yields more sophisticated machine learning and data processing capabilities but introduces higher latency and communication overheads. The fog layer, featured in contemporary architectural designs, functions as a grid that mediates between the edge and the cloud,



helping to balance the advantages and drawbacks of the other two layers. This layer addresses concerns associated with reliability, load balancing, communication overhead, and data sharing. Particularly, the fog layer acts as a data collector close to the edge layer and as a data filter close to the cloud layer, streamlining the data flow in near real-time for efficient processing, compression, and transformation.



**Figure 1.** A three-layer SNN topology.



**Figure 2.** Example of the edge–fog–cloud architecture.

Despite the potential of the edge–fog–cloud architecture, the increasing demands of edge devices exert substantial pressure on the cloud and fog layers. This situation underscores the need for enhancing intelligence at the edge. A significant step towards this goal is the exploration and implementation of SNNs in edge computing. However, this integration comes with unique challenges. While SNNs offer high computational efficiency and biological realism, their inherent characteristics, such as temporal dynamics and spike-based communication, complicate their deployment. Limited hardware resources and the complexity of neuron connections pose additional difficulties in mapping SNNs onto neuromorphic hardware. The balance between enhancing intelligence for edge computing applications and navigating the constraints of edge devices forms the focal point of our discussion in this paper.

In addition, the adoption of edge computing itself comes with a unique set of challenges, primarily due to the constrained resources and the need for real-time data analysis and decision-making. The struggle is particularly evident in applications like IoT devices [36], autonomous vehicles [37], and smart city infrastructure [38]. To balance the

algorithm accuracy, energy consumption and communication efficiency amidst diverse devices and complex scenarios are significant difficulties. Individual edge computing scenarios with limited computational power were often unable to cope with large-scale data processing. Conversely, collaborative edge computing scenarios require efficient coordination schemes to manage variations in computational capabilities and device numbers. Consequently, conventional practices, such as deploying AI models or transferring data to the cloud for processing, prove inadequate for applications demanding real-time, high-intelligence performance.

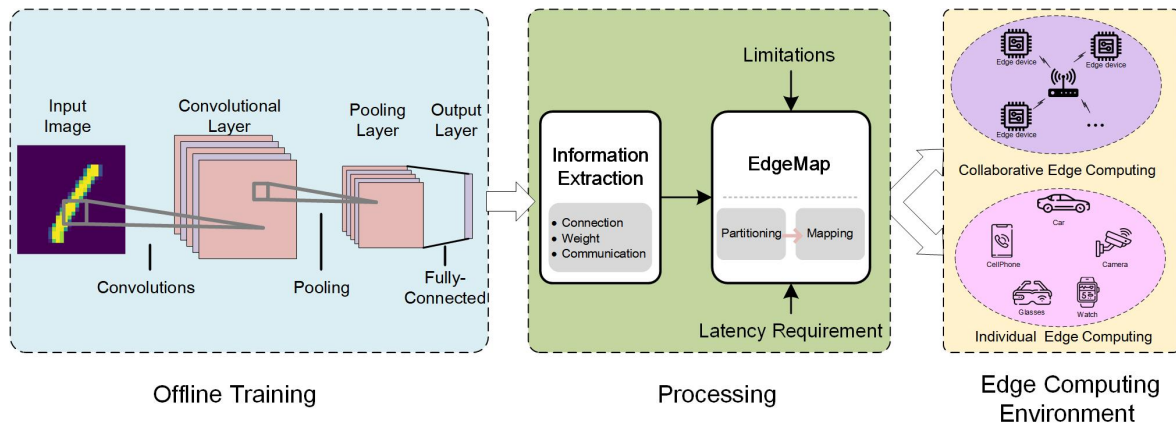
These constraints and the limitations of current solutions underscore the necessity of a novel approach. It is within this context that this work introduces a mapping toolchain specifically designed for deploying SNNs on edge computing devices. This innovation aims to bridge the gap between SNNs and edge computing, with an emphasis on low-latency and high-intelligence performance.

#### 4. Framework and Design Method

In this section, we present a comprehensive overview of EdgeMap, an optimized mapping toolchain specifically designed for SNNs in edge computing environments, which addresses the inherent limitations and challenges associated with edge computing scenarios.

##### 4.1. Framework Overview

The EdgeMap framework aims to enhance the performance of SNNs on neuromorphic hardware in edge computing scenarios. Current mapping approaches primarily focus on the forward direction of neural network inference, as the learning phase can be latency-insensitive and conducted through the cloud. As illustrated in Figure 3, the EdgeMap toolchain deploys a specific neural network through the following stages:



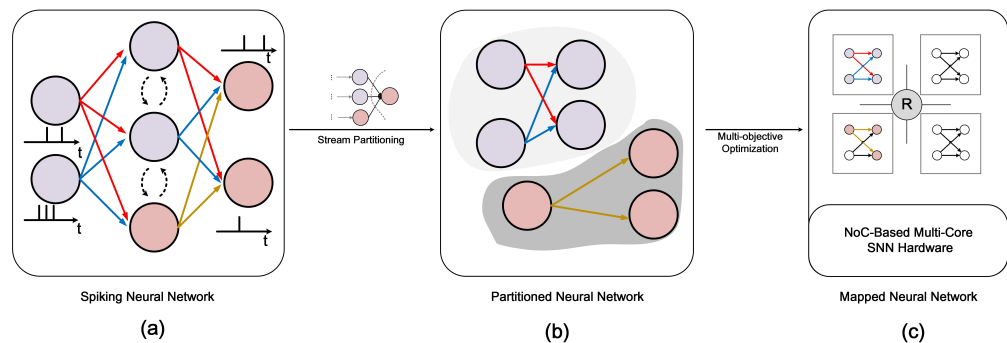
**Figure 3.** A high-level overview of EdgeMap.

- **Offline training stage:** The design and training of models with different network structures are crucial, considering the diverse application scenarios in edge computing. These structures must meet varied accuracy and scale requirements. The training process can be implemented using frameworks like Nengo [39], CARLsim [40], and Brain2 [41]. For clarity, we use the commonly adopted convolutional SNNs for recognition applications in Figure 3, primarily used for classification tasks [42].
- **Processing stage:** As shown in Figure 4, this stage is pivotal to the entire process. We start by extracting information from trained SNNs, such as neuron connection structure, weight information, and spike communication details. The SNN topology is represented as a computing graph, as depicted in Figure 4a. Subsequently, we divide the SNN into smaller neuron clusters for more manageable processing, demonstrated in Figure 4b. Finally, we map these partitioned neuron clusters onto the neuromorphic core, as illustrated in Figure 4c. This process forms the core function

of our EdgeMap toolchain. It optimizes hardware resource utilization and ensures computational efficiency.

- **Edge computing stage:** In this part, we briefly introduce two edge computing modes: individual edge computing scenarios and collaborative edge computing scenarios. The individual edge computing scenario involves a variety of computing modes and different types of devices. As for collaborative edge computing scenarios, multiple edge devices cooperate via network connections to accomplish more computing require tasks.

Based on the above introduction, the EdgeMap toolchain covers the entire process from training and processing to edge computing, adapting to the demands of various scenarios, and effectively improving the performance of SNNs on neuromorphic hardware in edge computing contexts.



**Figure 4.** Flowchart of the mapping. (a) The connection of SNNs. (b) The partitioned neuron clusters. (c) Neuron cluster to NCs mapping.

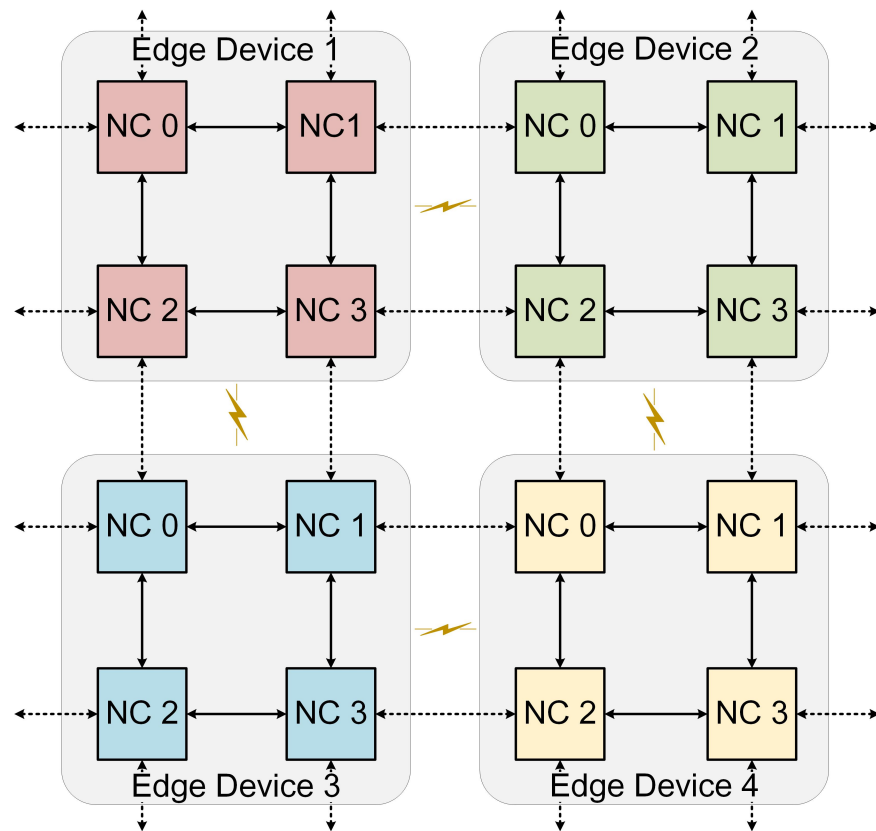
#### 4.2. Neuromorphic Edge Computing Hardware Model

In this paper, we focus on the deployment of SNNs on edge devices in two scenarios: individual edge computing and collaborative edge computing. The individual edge computing suits situations with a relatively small SNN scale or sufficient computing resources on a single edge device. Conversely, when a single edge device's computing resources are insufficient, collaborative edge computing becomes necessary, requiring multiple devices for computations.

To provide a clearer understanding of the edge computing modes, we devise a simplified hardware model as depicted in Figure 5. For the sake of clarity, we assume that each edge device consists of four neuromorphic cores interconnected through a 2D mesh. Although actual devices may have more cores, our model seeks to effectively clarify the system's fundamental architecture. Additionally, to emulate collaborative edge computing, we model four edge devices as interconnected via a mesh topology. The dashed lines in the figure represent the communication pathways between devices within a cooperative edge computing environment, with the fog layer being the primary facilitator for such interactions. For the purposes of our study, we make the following assumptions:

- The spike communication latency and energy consumption between devices are equivalent to those within the device's neuromorphic cores.
- The energy required for spike processing is uniformly distributed across all edge devices.
- The NCs across different devices possess equal computational capacity, including identical quantities of neurons and synapses.

By setting these assumptions, we establish a general framework to study the deployment of SNNs under various device and application scenarios. This enables us to gain a better understanding of the key aspects of cooperative edge computing and provides a more comprehensive evaluation of the performance of different mapping algorithms in an edge computing environment.



**Figure 5.** An illustration of the neuromorphic edge computing hardware model.

#### 4.3. Streaming-Based SNN Partition

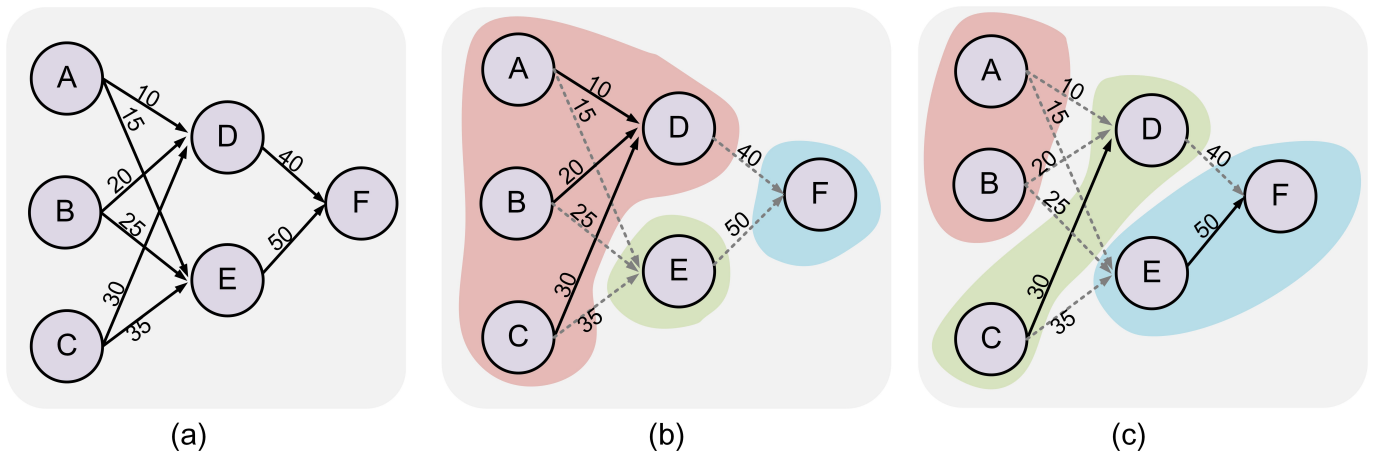
After training the SNNs, we use the resulting connectivity information to transform the SNN models into a computing graph  $G = \{V, E\}$ , where  $V$  is the set of neurons and  $E$  is the set of synapses. We then partition  $V$  into clusters  $\mathcal{C} = (C_1, C_2, \dots, C_k)$  to accommodate the constraints of neuromorphic hardware.

The partitioning process, as illustrated in Figure 4b and referred to as SNN partition, aims to partition SNNs into smaller neuron clusters. Our proposed SNN partition algorithm employs a streaming-based graph partitioning strategy, systematically assigning neurons to various clusters using a tailored gain function  $g$  to optimize both inter and intra-cluster communication costs based on the number of spikes transmitted.

In the context of our proposed SNN partition algorithm,  $k$  denotes the total number of clusters, which is no more than the number of NCs or edge devices. The gain function  $g(\mathcal{C})$  is the sum of all cluster gains  $h(C_i)$ , where  $h(C_i)$  is the gain of cluster  $C_i$ . The term  $\sum_{e \in e(X, Y)} w(e)$  represents the total number of spikes transmitted between  $X$  and  $Y$ . We use a specific convex increasing function  $c(x) = \alpha x^\gamma$ ,  $\alpha \geq 0, \gamma \geq 1$  to measure the cost of a cluster, with  $c(x) = x^2$  in our implementation, which quantifies the cost associated with the size of the cluster  $|C_i|$ . This approach allows us to balance the trade-off between the communication cost and cluster size, contributing to the efficient utilization of the NCs or edge devices.

$$\begin{aligned}
 g(\mathcal{C}) &= \sum_{i=1}^k h(C_i) \\
 &= \sum_{i=1}^k \sum_{e \in e(C_i, C_i)} w(e) - c(|C_i|) \\
 &= \left( \sum_{e \in E} w(e) - \sum_{i=1}^k \sum_{e \in e(C_i, V \setminus C_i)} w(e) \right) - \sum_{i=1}^k c(|C_i|)
 \end{aligned} \tag{1}$$

To illustrate the partitioning cost more clearly, we present a straightforward example of our proposed method in Figure 6. Subfigure (a) showcases the original SNN graph with the number of spikes indicated on each edge. Two distinct partition schemes are depicted in subfigures (b) and (c). Neurons that are colored the same belong to the same cluster. The gray dashed lines denote the connection edges that need partitioning, contributing to the communication cost of the neuron cluster. Scheme (b) results in a communication cost of 165 and a cluster size cost of 18, yielding a total gain of 183. On the other hand, scheme (c) incurs a communication cost of 155 and a cluster size cost of 12, resulting in a total gain of 167. Therefore, scheme (c) achieves a superior gain  $g$ , indicating its greater efficiency as a partitioning scheme.



**Figure 6.** Exhibition of partition strategies and corresponding cost analysis: (a) original SNN graph displaying the number of spikes, (b) partition result producing 3 clusters consisting of 4 neurons, 1 neuron, and 1 neuron, respectively; (c) alternative partition result resulting in 3 clusters, each containing 2 neurons.

In the partitioning process, a greedy approach is adopted. The gain of assigning neuron  $v_i$  to cluster  $C_j$  is defined as  $\delta g(v_i, C_j)$ . The objective is to allocate each neuron  $v_i$  to a cluster  $C_f$  that maximizes  $\delta g(v_i, C_f)$ , for all  $C_i \in \mathcal{C}$ .

$$\begin{aligned} \delta g(v_i, C_j) = & g(C_1, \dots, C_j \cup v_i, \dots, C_m, \dots, C_k) \\ & - g(C_1, \dots, C_j, \dots, C_m, \dots, C_k) \end{aligned} \quad (2)$$

In conclusion, our SNN partition algorithm provides an effective means of partitioning the SNN computation graph across multiple neuromorphic devices. By minimizing communication costs and balancing the size of each cluster, it ensures optimal performance while adhering to the device's constraints. This carefully calibrated approach leads to improvements in both computational efficiency and communication overhead, making it particularly beneficial for applications, such as real-time data processing and decision-making tasks. Examples of such applications include autonomous vehicles, smart home systems, and various edge computing scenarios.

Delving into the specifics of our method, the SNN partitioning algorithm we propose is detailed in Algorithm 1. Initially,  $k$  empty clusters are created, where  $k = \lceil |V|/N \rceil$ , and  $N$  is the maximum number of neurons a core can accommodate. Each neuron  $v_i$  is then allocated to its optimal cluster  $C_{final}$ , considering the hardware limit of crossbars. The `canAllocate( $v_i, C_j$ )` function checks the feasibility of allocating a neuron  $v_i$  to a cluster  $v_i$ , and `addVertexToCluster( $v_i, C_{final}$ )` subsequently adds the neuron to its assigned cluster.



**Algorithm 1:** Flow-based SNN partition

---

**Data:** SNN computing graph  $G = \{V, E\}$ , neuromorphic core size  $N$   
**Result:** Clusters of neurons  $\mathcal{C} = (C_1, C_2, \dots, C_n)$

```

1 Function clusterNeurons( $V, N$ ):
2    $k \leftarrow \lceil |V|/N \rceil$ ; /* Initialize  $k$  empty clusters */
3   for  $i = 1 \rightarrow k$  do
4      $C_i \leftarrow$  empty cluster;
5      $\mathcal{C} \leftarrow \mathcal{C} \cup C_i$ ;
6      $g \leftarrow 0$ ; //  $g$  is the metric function of partition
7   end
8   /* Traverse all of the neurons */
9   for  $v_i \in V$  do
10     $final \leftarrow -1$ ; // final cluster of  $v_i$ 
11     $\delta g_{\max} \leftarrow -\infty$ ; /* Traverse all of the clusters */
12    for  $C_j$  in  $\mathcal{C}$  do
13      if canAllocate( $v_i, C_j$ ) then
14         $\delta g_{ij} \leftarrow$  Compute  $\delta g(v_i, C_j)$ ;
15        if  $\delta g_{ij} > \delta g_{\max}$  then
16           $\delta g_{\max} \leftarrow \delta g_{ij}$ ;
17           $final \leftarrow j$ ;
18        end
19      end
20    if  $final < 0$  then
21      /*  $v_i$  cannot be allocated to the existing clusters */
22       $C_{k+1} \leftarrow$  empty cluster;
23       $\mathcal{C} \leftarrow \mathcal{C} \cup C_{k+1}$ ;
24       $final \leftarrow k + 1, k \leftarrow k + 1$ ;
25    end
26    addVertexToCluster( $v_i, C_{final}$ );
27  end
28  return  $\mathcal{C}$ 

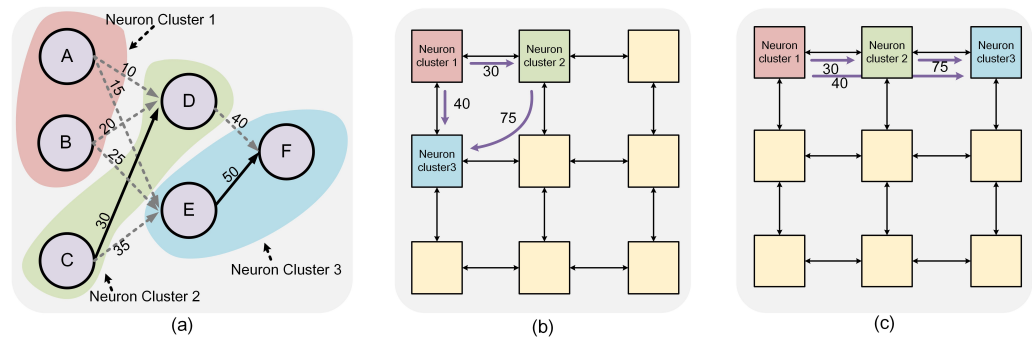
```

---

In the execution of our algorithm, we partition the SNN computation graph into manageable clusters for individual neuromorphic devices. Optimizing the gain function  $g$  reduces the communication overhead and ensures efficient utilization of hardware resources. The algorithm selects the cluster with the largest gain  $\delta g$  for each neuron  $v_i$ , maintaining a time complexity of  $O(|V||\mathcal{C}|)$ , linear to the size of the neural network and the cluster set.

#### 4.4. Multi-Objective Optimization Mapping

In this paper, we operate under the assumption that a sufficient number of physical cores are available on edge devices for the parallel processing of all neurons in the SNNs. The mapping stage, which is vital for multi-core collaborative computing, especially in edge scenarios, assigns partitioned neuron clusters to corresponding NCs as depicted in Figure 4c. As shown in Figure 7, the choice of mapping schemes can significantly influence energy consumption, average latency, and hop count. In Figure 7a, we display the neuron clusters post-partitioning. Subfigures (b) and (c) present two different mapping schemes. In scheme (b), the maximum communication throughput is 75 (spike). On the other hand, scheme (c) increases the maximum communication throughput to 115 (total spike from 2 to 3 and 1 to 3). However, it also results in higher congestion in the cluster 2 node, leading to increased latency. Therefore, the selection of the mapping scheme should strike a balance between these various factors to achieve the most efficient outcome.



**Figure 7.** Demonstration of neuron cluster mapping strategies for NCs: (a) partitioned neuron clusters, (b) mapping scheme with low congestion, and (c) mapping scheme with high congestion.

To effectively simulate the communication between NoC-based multi-core or multi-device neuromorphic computing, we developed a Noxim-based simulator [43]. This tool not only provides an accurate model for communication within NoC-based multi-core neuromorphic hardware but also supports various edge computing scenarios, facilitating collaboration among multiple devices.

**Mapping definition:** Placing the neuron cluster set  $\mathcal{C} = (C_1, C_2, \dots, C_k)$  onto the physical computing hardware with a set of hardware computing cores  $\mathcal{P} = (P_1, P_2, \dots, P_k)$ . This association can be denoted as  $\mathcal{C} \rightarrow \mathcal{P}$  and is detailed through a logical matrix,  $m_{ij}$ , belonging to  $\{0, 1\}^{k \times k}$ . The element  $m_{ij}$  within the matrix is defined as follows:

$$m_{ij} = \begin{cases} 1 & \text{if partition } C_i \in \mathcal{C} \text{ is mapped to device } P_j \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

and the mapping constraints are the following.

- (1) A neuron cluster can be mapped to only one computing device, thereby establishing a unique mapping relationship. Mathematically, this constraint can be expressed as follows:

$$\sum_j m_{ij} = 1 \quad \forall i \quad (4)$$

- (2) A computing device can accommodate, at most, one neuron cluster, ensuring exclusivity of the mapping process. This can be formally stated as follows:

$$\sum_i m_{ij} \leq 1 \quad \forall j \quad (5)$$

In this paper, we primarily aim to maximize the performance of SNNs on edge devices while bolstering their energy efficiency. These objectives are primarily driven by the need to satisfy the stringent energy constraints and meet real-time processing demands, which are discussed in Section 1. This is under the assumption provided in Section 4.2 that the spike processing speed and energy consumption are constant. The real-time performance and energy consumption are typically influenced by the mapping results in the context of NCs. It should be acknowledged that optimizing one objective may degrade others, leading to a multi-objective optimization problem. To address this, we consider several optimization objectives concurrently, intending to meet the diverse resource requirements that are characteristic of edge computing scenarios. The following are some of the critical objectives:

- (1) **Total energy consumption:** This objective captures the cumulative energy consumed by all spikes on the hardware, providing a direct reflection of the energy consumption. It is computed as

$$E = \sum_{i=1}^{N_s} [(h_i - 1) * e_w + h_i * e_s] \quad (6)$$

where  $N_s$  is the total number of spikes,  $e_w$  and  $e_s$  represent the energy consumption of spikes on the communication and cores, respectively.

- (2) Communication cost: This is the total communication count of the NCs, acting as a reflection of the system's communication efficiency. This metric has substantial implications for performance and power consumption. It is computed as

$$M = \sum_{i=1}^k \sum_{j=i}^k s_{ij} * h_{ij} \quad (7)$$

where  $k$  is the number of clusters,  $s_{ij}$  is the number of spikes between clusters  $i$  and  $j$ , and  $h_{ij}$  is the hop distance between clusters  $i$  and  $j$  (Manhattan distance).

- (3) Throughput: This metric represents the maximum number of spikes traveling on a link between any two neighboring clusters. As a pivotal metric directly influencing the system's performance, it is computed as follows:

$$\begin{aligned} Throughput &= \max Load_{ij}, 1 \leq i < j \leq k, |i - j| = 1 \\ Load_{ij} &= \text{Total spikes through channel}_{ij} \end{aligned} \quad (8)$$

where  $Load_{ij}$  indicates the total number of spikes transmitted between clusters  $i$  and  $j$ .

- (4) Average Hop: This metric offers insights into the efficiency of communication and overall performance. It denotes the average number of hops a spike requires to travel between neuron clusters, which mirrors the communication efficiency of the edge computing system. It is computed as follows:

$$H_{av} = M / \sum_{j=i}^k s_{ij} \quad (9)$$

where  $M$  is the total communication cost from Equation (7).

- (5) Average latency: This metric represents the average delay experienced by spikes on the NoC. The latency is critical to system responsiveness and the real-time nature of inference computations, particularly in edge computing applications requiring swift responses. It can be computed as follows:

$$E = \sum_{i=1}^{N_s} [(h_i - 1) * l_w + h_i * l_s] \quad (10)$$

where  $h_i$  is the hop count for a spike from source to destination,  $l_w$  is the communication delay, and  $l_s$  is the processing delay of the hops.

- (6) Average congestion: This metric represents the average communication pattern of the NoC, which is the connection network in the edge scenarios. It is especially critical for applications demanding real-time responses. It is computed as follows:

$$M_{ac} = \sum_{(x,y) \in S} Con(x,y) / N \quad (11)$$

where  $Con(x,y)$  computes the congestion in the router, with coordinates of  $(x,y)$ . It is computed as follows:

$$Con(x,y) = \sum_{e_{i,j} \in E_P} (w_P(e_{i,j}) * Expe(x,y, P(c_i), P(c_j))), \quad (12)$$

where  $Expe$  is the function that computes the expected value of the number of spikes that pass through the coordinates [26].

To tackle the complexities of these scenarios, we propose a multi-objective optimization algorithm based on NSGA-II [44], tailored specifically to edge computing applications where energy consumption and communication costs are vital factors. The optimization

target, designed to simultaneously minimize the total communication costs and energy costs, is expressed in Equation (13).

$$\underset{x}{\operatorname{argmin}}(M, E), s.t. \quad x \text{ is a valid mapping scheme.} \quad (13)$$

In order to find the best optimization solution, Algorithm 2 is used to optimize Equation (13). Firstly, we initialize the population with input size  $N$  (line1~5). A quick non-dominated sorting is then performed on the initial population  $P$  (line 6). The offspring are generated using  $P$  (line 7) by selection, crossover, and mutation. We then iterate for several rounds until the convergence criteria are met (line 8~19). In each round, we first perform the parent-child merge, then perform the fast non-dominated sorting. The top best  $N$  individuals are selected and then generate offspring for the next round based on these  $N$  individuals. Once the convergence criteria are met, individuals with rank 0 (non-dominated solutions) are filtered from the population ( $P$ ) and added to the non-dominated (ND) population (line 20~25). The non-dominated (ND) population is finally returned (line 26).

---

**Algorithm 2:** Multi-objective optimized mapping for edge computing

---

```

Data: Population size  $N$ 
Result: Non-dominated population  $ND$ 
/* Initialize population */
1  $P \leftarrow$  population with  $N$  individuals;
2 for  $i = 1 \rightarrow N$  do
3    $P[i].solution \leftarrow$  randomly initialized solution;
4    $P[i].rank \leftarrow -1$ ; // no dominance relationship initially
5 end
6  $P \leftarrow$  Quick non-dominated sort( $P$ );
/* Create a new population by selection, crossover, and mutation */
7  $Q \leftarrow$  Make new population( $P$ );
/* Iterate over the population */
8 while divergence criteria not met do
9    $R \leftarrow P \cup Q$ ; //  $R[i]$  is the set of individuals with rank  $i$ 
10   $R \leftarrow$  Quick non-dominated sort( $R$ );
11   $P \leftarrow \{\}, i \leftarrow 0$ ;
12  while  $(P \cup R[i]).size \leq N$  do
13     $P \leftarrow P \cup R[i]$ ;
14     $i \leftarrow i + 1$ ;
15  end
16   $R[i] \leftarrow$  Crowding distance sort( $R[i]$ );
17   $P \leftarrow P \cup R[i][N - P.size]$ ;
18   $Q \leftarrow$  Make new population( $P$ );
19 end
/* Divergence criteria met */
20  $ND \leftarrow \{\}$ ;
21 for  $p \in P$  do
22   if  $p.rank = 0$  then
23     add  $p$  to  $ND$ ;
24   end
25 end
26 return  $ND$ 

```

---

Upon obtaining the Pareto solution space through Algorithm 2, a location placement algorithm is needed to acquire the solution space, the task of which is performed by

**Algorithm 3.** The goal of this algorithm is to convert each solution  $Y_i$  from Algorithm 2 into a final mapping solution, represented as  $X = (X_1, X_2, \dots, X_k)$ .

$$\left(\frac{c_1}{\max c_1}\right)^2 + \left(\frac{c_2}{\max c_2}\right)^2 \quad (14)$$

Let us recall that each  $Y_i$  is an integer array of size  $k$ , where  $1 \leq y_{ij} \leq j + 1$ . The metric for selection is defined in Equation (14), where  $\max c_1$  and  $\max c_2$  represent the maximum values of the metrics computed during the execution of Algorithm 2.

$$Y_i = (y_{i1}, y_{i2}, \dots, y_{ik}), 1 \leq i \leq \text{population size} \quad (15)$$

However,  $Y_i$  does not represent the final mapping solution. Each final mapping solution, denoted by  $X = (X_1, X_2, \dots, X_k)$ , is constructed from  $Y_i$ . Here,  $X$  is a sorted array ranging from 1 to  $k$ . The procedure for constructing  $X$  from  $Y_i$  is detailed in Algorithm 3. The outcome of this procedure is that each cluster  $C_i$  from  $\mathcal{C}$  is mapped onto a physical core  $P_{x_i}$  in  $\mathcal{P}$ . This final mapping is essential for the optimization of the communication cost and energy cost in edge computing applications.

---

**Algorithm 3:** Procedure for the optimal mapping result

---

**Data:** A particle/individual  $Y = (y_1, y_2, \dots, y_k)$   
**Result:** A mapping scheme  $X = (x_1, x_2, \dots, x_k)$

- 1  $used \leftarrow$  All false Boolean arrays with a size of  $k$ ;
- 2  $X \leftarrow$  All zero integer arrays with a size of  $k$ ;
- 3 **for**  $i = k \rightarrow 1$  **do**
- 4      $rank \leftarrow y_i - 1$ ;
- 5      $pos \leftarrow -1$ ;
- 6     **while**  $rank > 0$  **do**
- 7          $pos \leftarrow pos + 1$ ;
- 8         **while**  $used[pos]$  **do**
- 9              $pos \leftarrow pos + 1$ ;
- 10         **end**
- 11          $rank \leftarrow rank - 1$ ;
- 12     **end**
- 13      $used[pos] \leftarrow True$ ;
- 14      $x_i \leftarrow pos$ ;
- 15 **end**
- 16 **return**  $X$

---

## 5. Evaluation Methodology

### 5.1. Experimental Environment

In this section, we describe the configuration of the edge computing environment used for the comprehensive evaluation of EdgeMap. We conducted all experiments on a system with an Intel i7-8700k CPU, 32 GB of RAM, and an NVIDIA RTX2080 GPU operating on Ubuntu 20.04. CARLsim [40] serves as our choice for the training and simulation of SNNs. In order to gauge performance, we employ the advanced, modified, cycle-accurate Noxim framework [43] to generate more accurate test data.

We leverage the software to simulate a hypothetical neuromorphic edge computing environment, using it as a benchmark to measure the performance of EdgeMap and other toolchains. The parameters of the target hardware are outlined in Table 1. The NCs in the edge devices are configured with 256 neurons and 64K synapses. The rest of the parameters are mentioned in Equations (6) and (10) and will be utilized in the experiments conducted throughout this paper.



**Table 1.** Parameters of the target neuromorphic edge devices.

Parameter	Neurons/Core	Synapses/Core	$e_s$	$e_w$	$l_s$	$l_w$
Value	256	64k	1	0.1	1	0.01

### 5.2. Evaluated Applications

In this subsection, we delve into the efficiency evaluation of EdgeMap by considering various applications represented by SNNs. All applications are designed for edge computing and smaller edge devices, and are listed in Table 2; this table encapsulates vital details, including the input size, SNN connection topology, total neuron count, total synapses, and the whole spike count of the SNNs.

In this paper, we selected a suite of applications to evaluate EdgeMap. For classification tasks on the MNIST dataset [45], we used two different networks: MLP (multi-layer perceptron) and LeNet [46], both processing  $28 \times 28$ -pixel images (referred to as MNIST-MLP and MNIST-LeNet). In addition, we used an MLP network to process  $28 \times 28$  pixel images from the Fashion MNIST dataset [47], referred to as Fashion-MNIST. The Heart Class application focuses on ECG-based heartbeat classification using images of  $42 \times 42$  pixels [48]. For the CIFAR-10 classification task, we evaluated four different SNNs of varying complexity. These include LeNet [46] (CIFAR10-LeNet), AlexNet [49] (CIFAR10-AlexNet), VGG11 [50] (CIFAR10-VGG11), and ResNet [51] (CIFAR10-ResNet), all processing  $32 \times 32 \times 3$ -pixel images. All of these applications are converted into spike-based models using the CNN-to-SNN conversion tool [52], and they are used to perform inference computations.

### 5.3. Evaluated Metrics

We evaluate the mapping toolchain for all SNN applications using the following metrics as listed below. These metrics are crucial as they directly reflect energy consumption and real-time processing capabilities, providing valuable insights into the system's ability to meet the unique demands and constraints of edge computing scenarios.

- Energy consumption: This reflects the energy consumed by the neuromorphic hardware, a critical factor in edge device computing. It is modeled in Equation (6).
- Communication cost: This represents the cost of multi-chip or multi-devices edge computing scenarios. It is modeled in Equation (7).
- Throughput: This refers to the throughput of each application on the multi-core neuromorphic hardware. It is modeled in Equation (8).
- Average hop: Serves as an important indicator of communication efficiency in NoC-based neuromorphic hardware, which is modeled in Equation (9). We further analyze the maximum hop count across the mapping schemes to capture extremes.
- Average latency: This metric is indicative of the time delay in processing and is essential for time-sensitive edge computing applications. It is modeled in Equation (10); we also assess the maximum latency observed in the mapping schemes.
- Average congestion: This metric provides insight into the load balance across the network. It is modeled in Equation (11), and we extend our analysis to the maximum congestion experienced in the mapping schemes.
- Execution time: This metric measures the duration required for a mapping toolchain to produce a mapping result, encompassing both the partitioning and mapping stages.

### 5.4. Comparison Approaches

In our experiments, we conducted a comprehensive evaluation of various mapping toolchains, including our proposal, EdgeMap, which encompasses both the partitioning and mapping components. All of the toolchains we compare are summarized in Table 3. In these evaluations, we use SpiNeMap as a baseline for all metrics.

**Table 2.** Application used for the evaluation of EdgeMap.

Applications	MNIST-MLP	MNIST-LeNet	Fashion-MNIST	Heart Class	CIFAR10-LeNet	CIFAR10-AlexNet	CIFAR10-VGG11	CIFAR10-ResNet
Topology	$28 \times 28 \times 1$	$28 \times 28 \times 1$	$28 \times 28 \times 1$	$42 \times 42 \times 1$	$32 \times 32 \times 3$	$32 \times 32 \times 3$	$32 \times 32 \times 3$	$32 \times 32 \times 3$
Neuron number	<i>MLP</i> <sup>1</sup>	<i>CNN</i> <sup>2</sup>	<i>MLP</i> <sup>3</sup>	<i>CNN</i> <sup>4</sup>	<i>CNN</i> <sup>5</sup>	<i>CNN</i> <sup>6</sup>	<i>CNN</i> <sup>7</sup>	<i>CNN</i> <sup>8</sup>
Synapses	1193	7403	1393	17,001	11,461	794,232	9,986,862	9,675,543
Total spikes	97,900	377,990	444,600	773,112	804,614	39,117,304	47,737,200	48,384,204
	358,000	1,555,986	10,846,940	2,209,232	7,978,094	574,266,873	796,453,842	5,534,290,865

<sup>1</sup> Feedforward(784-100-10). <sup>2</sup> Conv((5,5),(1,1),6)-AvgPool(2,2)-Conv((5,5),(1,1),16)-AvgPool(2,2)-FC(500)-FC(10). <sup>3</sup> Feedforward(784-500-100-10). <sup>4</sup> Conv((5,5),(1,1),6)-AvgPool(2,2)-Conv((5,5),(1,1),16)-AvgPool(2,2)-FC(10). <sup>5</sup> Conv((5,5),(1,1),6)-AvgPool(2,2)-Conv((5,5),(1,1),16)-AvgPool(2,2)-FC(500)-FC(10). <sup>6</sup> Conv((11,11),(4,4),96)-Maxpool(2,2)-Conv((5,5),(1,1),256)-Maxpool(2,2)-Conv((3,3),(1,1),384)-Conv((3,3),(1,1),256)-Maxpool(2,2)-FC(4096-4096-10). <sup>7</sup> Conv((3,3),(1,1),64)-MaxPool(2,2)-Conv((3,3),(1,1),128)-MaxPool(2,2)-Conv((3,3),(1,1),256)-Conv((3,3),(1,1),256)-MaxPool(2,2)-Conv((3,3),(1,1),512)-Conv((3,3),(1,1),512)-MaxPool(2,2)-Flatten-FC(4096-4096-10). <sup>8</sup> Conv((3,3),(1,1),64)-MaxPool(2,2)-Conv((3,3),(1,1),128)-MaxPool(2,2)-Conv((3,3),(1,1),256)-Conv((3,3),(1,1),256)-MaxPool(2,2)-Conv((3,3),(1,1),512)-Conv((3,3),(1,1),512)-MaxPool(2,2)-FC(4096-4096-10).

**Table 3.** Comparison of different mapping toolchains.

Toolchains		SpiNeMap [13]	SNEAP [22]	DFSynthesizer [23]	NEUTRAMS <sup>4</sup> [21]	EdgeMap
Partition	Algorithm	Kernighan–Lin(KL)	METIS	Greedy algorithm <sup>1</sup>	Kernighan–Lin(KL)	Streaming-based
	Objective	Communication cost	Communication cost	Resource Utilization.	Communication cost	Communication cost
Mapping	Algorithm	PSO	SA <sup>2</sup>	LSA <sup>3</sup>	–	NSGA-II
	Objective	Communication cost	Average Hop	Energy consumption	–	Communication cost & Energy consumption

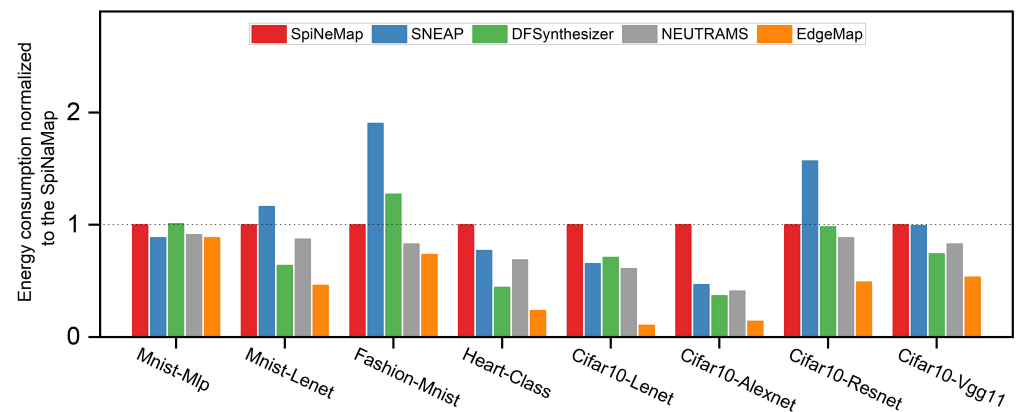
<sup>1</sup> This algorithm is a modified greedy algorithm. <sup>2</sup> SA is short for simulated annealing. <sup>3</sup> LSA is short for the local search algorithm. <sup>4</sup> In NEUTRAMS, the mapping stage is not mentioned, we use sequential mapping as the default.

## 6. Results and Discussion

In this section, we present and discuss the results of our experiments, exploring the effectiveness of different mapping schemes in various applications. We evaluate their performance based on critical metrics, including energy consumption, communication cost, throughput, spike hop, spike latency, congestion, and execution time. These metrics are essential to comprehending the operational efficiency of each scheme under edge computing conditions.

### 6.1. Energy Consumption Analysis

Figure 8 illustrates the energy consumption of different mapping schemes. As shown in the figure, EdgeMap demonstrates a clear advantage over other schemes in all applications, notably achieving an approximate 89% reduction in energy consumption compared to the baseline SpiNeMap in the CIFAR10-LeNet application. This considerable reduction is attributed to EdgeMap's advanced optimization strategies that focus on energy consumption and communication costs during the mapping stage. It is worth mentioning that these optimization policies are not only unique to EdgeMap but also present a substantial advancement over similar strategies applied in previous work, such as the methods used in SpiNeMap, SNEAP, DFSynthesizer, and NEUTRAMS.



**Figure 8.** Normalized energy consumption analysis across different mapping schemes.

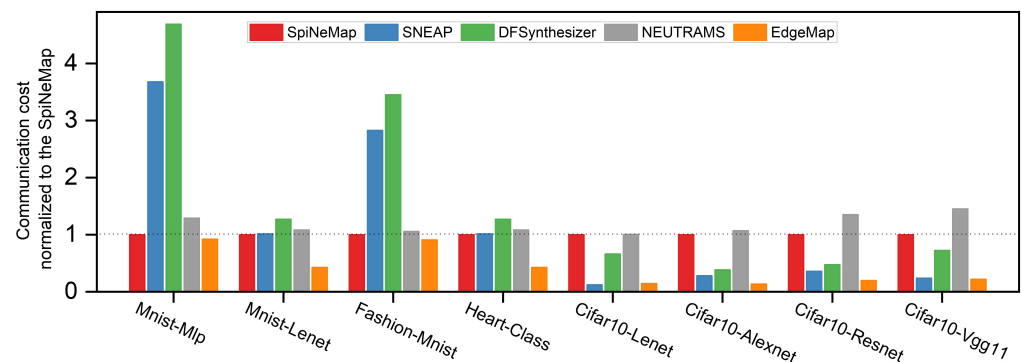
In relatively simple SNN structures used in the MNIST-MLP and Fashion-MNIST applications, all mapping schemes show comparably the same energy consumption. This might be a consequence of fewer neurons and connections involved in these applications. Nonetheless, even in these simple scenarios, EdgeMap still provides a notable improvement in energy consumption (12% decrease of SpiNeMap), which validates the effectiveness of its optimization techniques.

EdgeMap outperforms other mapping schemes in terms of average energy consumption, surpassing SpiNeMap by 57%, SNEAP by 66%, DFSynthesizer by 63%, and NEUTRAMS by 33%. This improvement is not incidental but is the result of EdgeMap's specialized optimization policies, which are particularly designed for edge devices. It is important to note here that these policies include methods that significantly reduce the total number of spikes on the devices during the partitioning stage. This is a fundamental advancement over traditional methods and contributes significantly to the energy-saving performance of EdgeMap. Furthermore, the robustness of EdgeMap's performance across a diverse range of SNNs with different scales shows EdgeMap's generalizability to a broad array of SNN structures and edge devices.

In summary, EdgeMap's excellence in managing energy consumption distinguishes it from other mapping schemes evaluated in this paper. Its superior efficiency makes it a compelling choice for edge devices, even when dealing with complex SNNs that have a large number of neurons and connections.

### 6.2. Communication Cost Analysis

Figure 9 provides a comparison of the normalized communication costs across different mapping schemes, each measured relative to SpiNeMap. Clearly, EdgeMap emerges as the most effective scheme, indicating a significant improvement in reducing communication costs. In fact, when compared to SpiNeMap, SNEAP, DFSynthesizer, and NEUTRAMS, EdgeMap manages to reduce communication costs by 58%, 73.9%, 78.9%, and 65.8%, respectively. This substantial reduction is primarily the result of EdgeMap's unique optimization approach, as outlined in Algorithm 1, which strategically modulates interactions among the neuron clusters. This technique represents an advancement over the existing methods employed by the compared schemes.



**Figure 9.** Normalized communication costs across different mapping schemes.

Moreover, our analysis shows that EdgeMap's performance improvement is not constant but scales proportionally with the complexity of the SNNs. That is, the larger the scale of the SNNs, the more pronounced the performance benefits of our optimization algorithm. This correlation suggests that EdgeMap's optimization strategies are well-suited to complex SNNs that typically pose challenges for existing mapping schemes. This adaptive nature of performance enhancement is of great importance in an edge computing environment. It ensures that the deployment of complex SNNs does not translate into excessive communication overhead. It is worth noting that this advantage is not merely theoretical, but it has been validated across various scales of SNNs and in different edge computing scenarios, reinforcing the robustness and generalizability of our results.

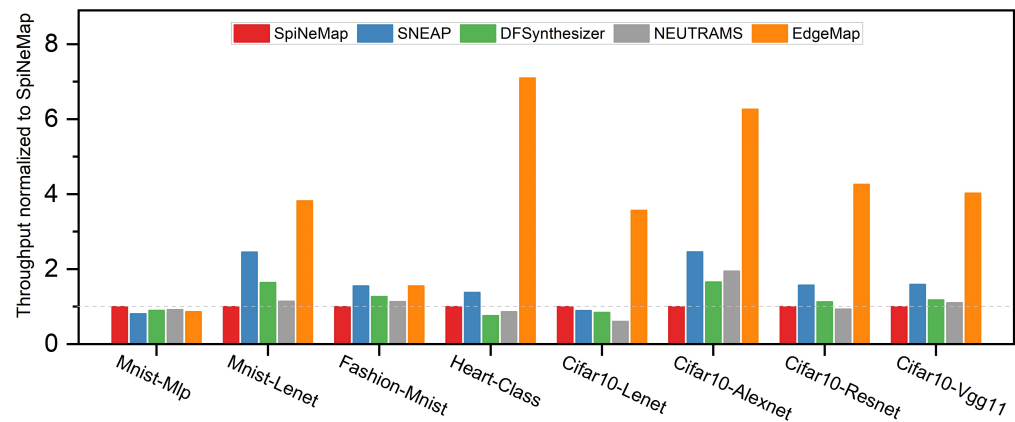
In conclusion, the performance advantage demonstrated by EdgeMap in terms of communication cost reduction is a testament to the effectiveness of its novel optimization approach. This not only separates EdgeMap from other mapping schemes but also underscores its suitability for complex SNNs in an edge computing environment.

### 6.3. Throughput Analysis

Figure 10 offers a comparative analysis of the normalized throughput for each application across the mapping schemes under evaluation, with each scheme normalized to SpiNeMap. This comparison is pivotal for understanding the operational efficiency of each scheme within the edge computing context. The analysis reveals that EdgeMap surpasses all other mapping schemes in terms of throughput performance. Notably, EdgeMap's throughput is  $4.02\times$  greater than that of SpiNeMap, and outperforms SNEAP, DFSynthesizer, and NEUTRAMS by  $2.52\times$ ,  $3.43\times$ , and  $3.65\times$ , respectively.

The remarkable throughput advantage of EdgeMap is primarily due to its focus on multi-objective optimization during the mapping process, as detailed in Algorithm 2. This distinguishing factor sets EdgeMap apart from the other schemes, which do not typically incorporate this aspect into their mapping methodologies. By focusing on multi-objective optimization, EdgeMap can effectively harness the computational capabilities of edge devices, facilitating a significant improvement in throughput, satisfying the real-time processing requirement. As the complexity of the SNNs increases, the throughput advantage

of EdgeMap becomes more pronounced, suggesting that our toolchain is particularly suited for large-scale, complex SNN applications.

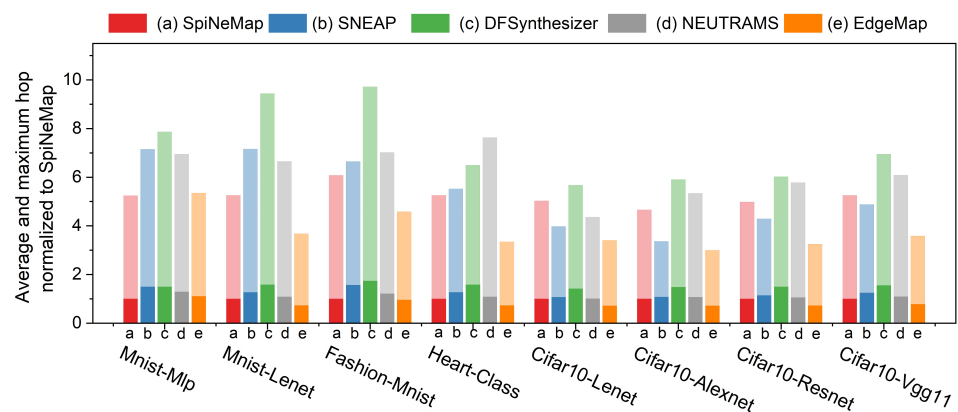


**Figure 10.** Normalized throughput across different mapping schemes.

Overall, these results affirm EdgeMap’s effectiveness in delivering superior throughput performance and its scalability in handling complex SNNs, making it a highly competitive solution in the field of SNN mapping on edge devices.

#### 6.4. Spike Hop Analysis

Figure 11 presents the normalized average and maximum hops for each application under each of the evaluated mapping schemes, normalized to SpiNeMap. A single color represents each mapping scheme, the darker shade indicates the average hop, and the lighter shade denotes the maximum hop. The hop count serves as a crucial indicator of collaborative capability in edge computing scenarios.



**Figure 11.** Normalized average and maximum hop analysis across different mapping schemes.

EdgeMap consistently outperformed all other mapping schemes across all applications, highlighting its benefits in edge computing scenarios. Specifically, EdgeMap’s average hop count is 19.4% lower than that of SpiNeMap, and its maximum hop count is 29.9% lower. Moreover, when compared with SNEAP, DFSynthesizer, and NEUTRAMS, EdgeMap reduces the average hop count by 36.5%, 48.1%, and 27.3%, respectively, while its maximum hop count is decreased by 27.8%, 48.1%, and 41.9%.

With the scaling of the SNNs, both average and maximum hops increase correspondingly. However, EdgeMap effectively keeps this increase within a manageable range, primarily due to the mapping strategy deployed in Algorithm 2, which emphasizes the spatial proximity of interconnected neurons. This approach curbs the exponential growth of hop counts as the network expands, thereby contributing significantly to EdgeMap’s

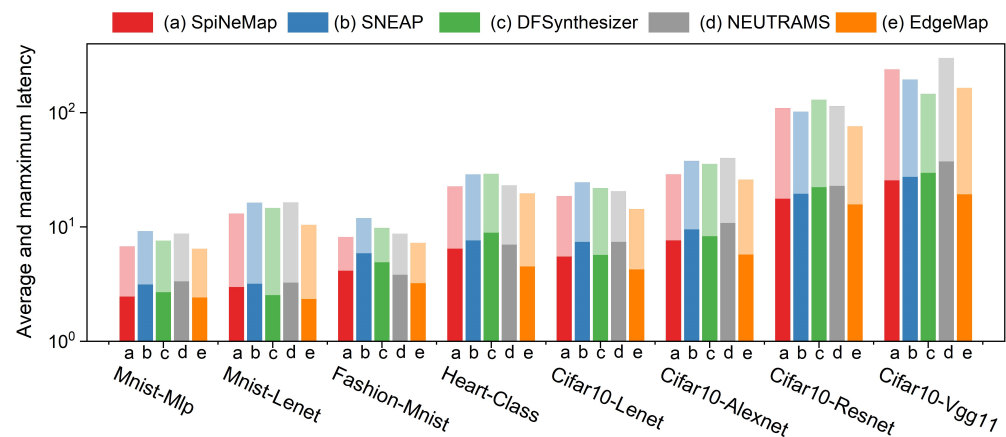


efficient performance within edge computing environments. Notably, this control over hop counts is instrumental in reducing latency, making EdgeMap an ideal choice for real-time applications where responsiveness is key.

In summary, the results underscore EdgeMap's efficacy in maintaining efficient communication among neurons in large-scale SNN applications, resulting in lower latency, and making it a compelling choice for edge computing scenarios.

### 6.5. Spike Latency Analysis

Figure 12 illustrates the comparative analysis of both average and maximum spike latencies across applications for the evaluated mapping schemes. Consistent color patterns are used to denote individual mapping schemes, with darker shades representing average latency, and lighter shades indicating maximum latency. Our evaluation reveals that EdgeMap consistently outperforms other schemes, demonstrating lower average and maximum latencies. Specifically, EdgeMap's average latency is 19.8% lower than that of SpiNeMap, while its maximum latency is reduced by 22.5%. Comparatively, EdgeMap's average latency reduces by 33.4%, 29.2%, and 35.5% against SNEAP, DFSynthesizer, and NEUTRAMS, respectively. Similarly, its maximum latency is lower by 28.6%, 22.9%, and 33.7%.



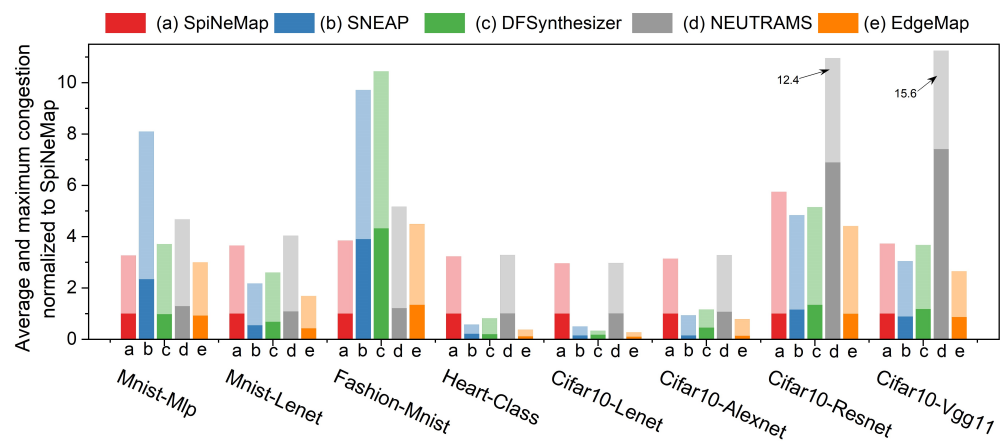
**Figure 12.** Average and maximum latency analysis across different mapping schemes.

The latency in each application arises from two components: the processing time for spikes and the spike communication latency. As the scale of the SNN increases, so does the hardware scale and, thus, the transmission time, leading to increased latency. However, EdgeMap's optimization of communication costs effectively reduces congestion, resulting in a decrease in latency.

In summary, EdgeMap demonstrates superior performance by consistently maintaining lower latency levels across all applications. This is primarily achieved through its strategic optimization, which effectively mitigates congestion, enabling EdgeMap to maintain optimal performance even as the network scales. These features underscore EdgeMap's suitability for edge computing environments, where low latency is crucial.

### 6.6. Congestion Analysis

Congestion is a crucial metric that reflects the collaborative capability of edge computing scenarios, and as such, it plays a pivotal role in our analysis. As Figure 13 demonstrates, EdgeMap consistently outperforms other mapping schemes across all applications, exhibiting superior performance. Each mapping scheme in the figure is represented by a specific color, with darker shades indicating average congestion and lighter shades representing maximum congestion.



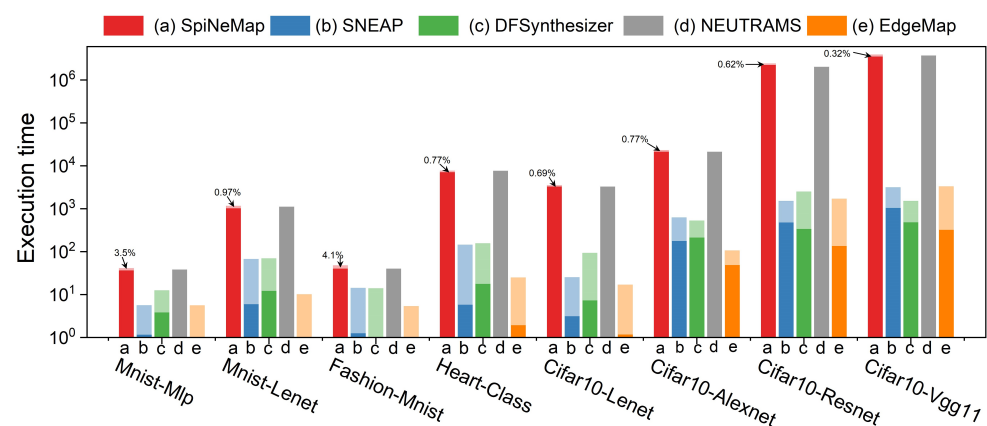
**Figure 13.** Normalized average and maximum congestion analysis across different mapping schemes.

Specifically, EdgeMap outperforms other mapping schemes across all applications. Specifically, the average congestion of EdgeMap is 39.5% lower than that of SpiNeMap, while its maximum congestion is reduced by 40.8%. Furthermore, compared to SNEAP, DFSynthesizer, and NEUTRAMS, EdgeMap's average congestion decreases by 47.8%, 42.6%, and 81.2% respectively, with maximum congestion reduced by 47.5%, 43.7%, and 74%. Interestingly, NEUTRAMS exhibits inferior performance, which can be attributed to its methodology. This scheme optimizes the communication cost only during the partitioning stage, mapping neuron clusters onto devices sequentially. As a result, it suffers from high congestion.

In summary, EdgeMap is shown to be a highly effective mapping scheme, significantly reducing congestion in edge computing applications. This finding underlines EdgeMap's capabilities in boosting performance and efficiency in edge scenarios, providing valuable insights for researchers in the field.

### 6.7. Execution Time Analysis

Figure 14 provides a comparison of the execution time across different applications for the evaluated mapping schemes. We divided the total execution time into two stages: the partitioning stage and the mapping stage. A uniform color scheme is employed to represent each mapping scheme, with darker shades indicating the partition time and lighter shades denoting the mapping time.



**Figure 14.** Execution times across different mapping schemes.

Interestingly, the execution times of SpiNeMap and NEUTRAMS are relatively similar, which can be attributed to their identical partitioning algorithms and objectives. However, NEUTRAMS does not incorporate a separate mapping stage. The partition time dominates

the execution time, accounting for 96.5% of the total in SpiNeMap and rising to 99.68% for the CIFAR10-VGG11 application as the complexity of SNNs increases.

EdgeMap's execution process differs significantly from other schemes, with the mapping stage accounting for a considerable portion of the total execution time. For the MNIST-MLP network, the mapping time essentially occupies the entire execution time. As the complexity of the SNNs increases, so does the time spent on mapping. For instance, the mapping time constitutes 90.3% of the total execution time for the CIFAR10-VGG11 network. Furthermore, EdgeMap outperforms SpiNeMap by an impressive factor of  $1225.44\times$  in the average execution time. The partitioning stage and the mapping stage are faster by  $11268.9\times$  and  $131\times$ , respectively. Despite having a comparable total execution time to SNEAP, EdgeMap demonstrates superior efficiency, with partition times being  $3.36\times$  faster and mapping times  $0.82\times$  as fast. When compared with DFSynthesizer and NEUTRAMS, EdgeMap's average total execution time is more efficient, specifically  $0.94\times$  and  $1113\times$ , respectively. The partition time for DFSynthesizer is  $2.1\times$  longer, further emphasizing EdgeMap's efficacy.

## 7. Conclusions

In this paper, we introduced EdgeMap, an optimized toolchain for mapping SNNs onto neuromorphic hardware within edge computing scenarios. EdgeMap demonstrates significant improvements in energy efficiency, communication cost reduction, and throughput enhancement. It also successfully minimizes both spike latency and congestion, highlighting its superior real-time performance, which is crucial for edge applications. Through a two-stage mechanism, partitioning and mapping, EdgeMap ensures rapid execution speed and is highly adaptable to edge computing conditions. Remarkably, its robust performance remains consistent even as the network complexity escalates, reinforcing EdgeMap's supremacy over other contemporary mapping methods.

Despite its significant advantages, EdgeMap still has its limitations. Currently, our SNN mapping is largely based on static optimization, concentrating primarily on fixed network structures. However, edge computing scenarios often require dynamic adjustments in response to environmental changes or task requirements. This dynamic aspect of edge computing presents a challenge that EdgeMap, in its current form, may not handle directly. Additionally, we make an assumption that all edge devices have equal computational power and overlook geographical factors. This assumption may not reflect real-world scenarios. Variations in hardware can cause differences in the computational power of edge devices. Similarly, the geographical location can significantly affect communication latency and data transfer efficiency. These factors could increase the complexity of mapping, indicating an area where our EdgeMap method may require enhancements.

Moving forward, our goal is to incorporate more flexible optimization objectives within the EdgeMap toolchain to accommodate the needs of increasingly complex computing environments. We also aim to explore methods for mapping SNNs with online learning capabilities onto edge devices. In conclusion, EdgeMap shows promise as a strategy for efficiently mapping SNNs onto edge devices. Although it has demonstrated exceptional results, improvements and adaptations are necessary to reflect real-world edge computing scenarios more accurately.

**Author Contributions:** Conceptualization, R.Y.; Methodology, J.X.; Software, L.X.; Validation, J.X. and Y.Z.; Formal analysis, F.C.; Investigation, F.C.; Data curation, L.W.; Visualization, Q.T.; Supervision, R.Y. and P.L.; Project administration, P.L.; Funding acquisition, P.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by Science and Technology Innovation (STI) 2030–Major Projects 2022ZD0208700.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data will be made available upon reasonable request to the authors.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Ghosh-Dastidar, S.; Adeli, H. Spiking neural networks. *Int. J. Neural Syst.* **2009**, *19*, 295–308. [[CrossRef](#)] [[PubMed](#)]
2. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [[CrossRef](#)]
3. Taherkhani, A.; Belatreche, A.; Li, Y.; Cosma, G.; Maguire, L.P.; McGinnity, T.M. A review of learning in biologically plausible spiking neural networks. *Neural Netw.* **2020**, *122*, 253–272. [[CrossRef](#)] [[PubMed](#)]
4. Wang, Y.; Xu, Y.; Yan, R.; Tang, H. Deep Spiking Neural Networks with Binary Weights for Object Recognition. *IEEE Trans. Cogn. Dev. Syst.* **2021**, *13*, 514–523. [[CrossRef](#)]
5. Bittar, A.; Garner, P.N. Surrogate Gradient Spiking Neural Networks as Encoders for Large Vocabulary Continuous Speech Recognition. *arXiv* **2022**, arXiv:2212.01187.
6. Bing, Z.; Meschede, C.; Röhrbein, F.; Huang, K.; Knoll, A.C. A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks. *Front. Neurobot.* **2018**, *12*, 35. [[CrossRef](#)] [[PubMed](#)]
7. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.V.; Merolla, P.; Imam, N.; Nakamura, Y.Y.; Datta, P.; Nam, G.; 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.* **2015**, *34*, 1537–1557. [[CrossRef](#)]
8. Davies, M.; Srinivasa, N.; Lin, T.; Chinya, G.N.; Cao, Y.; Choday, S.H.; Dimou, G.D.; Joshi, P.; Imam, N.; Jain, S.; et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. [[CrossRef](#)]
9. Furber, S.B.; Lester, D.R.; Plana, L.A.; Garside, J.D.; Painkras, E.; Temple, S.; Brown, A.D. Overview of the SpiNNaker System Architecture. *IEEE Trans. Comput.* **2013**, *62*, 2454–2467. [[CrossRef](#)]
10. Rajendran, B.; Sebastian, A.; Schmuker, M.; Srinivasa, N.; Eleftheriou, E. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Process. Mag.* **2019**, *36*, 97–110. [[CrossRef](#)]
11. Galluppi, F.; Davies, S.; Rast, A.; Sharp, T.; Plana, L.A.; Furber, S. A hierarchical configuration system for a massively parallel neural hardware platform. In Proceedings of the 9th Conference on Computing Frontiers, Cagliari, Italy, 15–17 May 2012; pp. 183–192.
12. Rueckauer, B.; Bybee, C.; Goettsche, R.; Singh, Y.; Mishra, J.; Wild, A. NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi. *ACM J. Emerg. Technol. Comput. Syst.* **2022**, *18*, 48:1–48:22. [[CrossRef](#)]
13. Balaji, A.; Catthoor, F.; Das, A.; Wu, Y.; Huynh, K.; Dell’Anna, F.; Indiveri, G.; Krichmar, J.L.; Dutt, N.D.; Schaafsma, S. Mapping Spiking Neural Networks to Neuromorphic Hardware. *IEEE Trans. Very Large Scale Integr. Syst.* **2020**, *28*, 76–86. [[CrossRef](#)]
14. Tsourakakis, C.E.; Gkantsidis, C.; Radunovic, B.; Vojnovic, M. FENNEL: Streaming graph partitioning for massive scale graphs. In Proceedings of the Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, 24–28 February 2014; pp. 333–342. [[CrossRef](#)]
15. Varshika, M.L.; Balaji, A.; Corradi, F.; Das, A.; Stuijt, J.; Catthoor, F. Design of Many-Core Big Little Brains for Energy-Efficient Embedded Neuromorphic Computing. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, 14–23 March 2022; Bolchini, C., Verbauwhede, I., Vatajelu, I., Eds.; IEEE: Antwerp, Belgium, 2022; pp. 1011–1016. [[CrossRef](#)]
16. Stuijt, J.; Sifalakis, M.; Yousefzadeh, A.; Corradi, F.  $\mu$ Brain: An event-driven and fully synthesizable architecture for spiking neural networks. *Front. Neurosci.* **2021**, *15*, 664208. [[CrossRef](#)] [[PubMed](#)]
17. Amir, A.; Datta, P.; Risk, W.P.; Cassidy, A.S.; Kusnitz, J.A.; Esser, S.K.; Andreopoulos, A.; Wong, T.M.; Flickner, M.; Alvarez-Icaza, R.; et al. Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores. In Proceedings of the 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, 4–9 August 2013; pp. 1–10. [[CrossRef](#)]
18. Lin, C.; Wild, A.; Chinya, G.N.; Lin, T.; Davies, M.; Wang, H. Mapping spiking neural networks onto a manycore neuromorphic architecture. In Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, 18–22 June 2018; pp. 78–89. [[CrossRef](#)]
19. Wang, S.; Yu, Q.; Xie, T.; Ma, C.; Pei, J. Approaching the mapping limit with closed-loop mapping strategy for deploying neural networks on neuromorphic hardware. *Front. Neurosci.* **2023**, *17*, 1168864. [[CrossRef](#)]
20. Deng, L.; Wang, G.; Li, G.; Li, S.; Liang, L.; Zhu, M.; Wu, Y.; Yang, Z.; Zou, Z.; Pei, J.; et al. Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid-State Circuits* **2020**, *55*, 2228–2246. [[CrossRef](#)]
21. Ji, Y.; Zhang, Y.; Li, S.; Chi, P.; Jiang, C.; Qu, P.; Xie, Y.; Chen, W. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, 15–19 October 2016; pp. 21:1–21:13. [[CrossRef](#)]
22. Li, S.; Guo, S.; Zhang, L.; Kang, Z.; Wang, S.; Shi, W.; Wang, L.; Xu, W. SNEAP: A fast and efficient toolchain for mapping large-scale spiking neural network onto NoC-based neuromorphic platform. In Proceedings of the 2020 on Great Lakes Symposium on VLSI, Virtual Event, China, 7–9 September 2020; pp. 9–14.

23. Song, S.; Chong, H.; Balaji, A.; Das, A.; Shackleford, J.A.; Kandasamy, N. DFSynthesizer: Dataflow-based Synthesis of Spiking Neural Networks to Neuromorphic Hardware. *ACM Trans. Embed. Comput. Syst.* **2022**, *21*, 27:1–27:35. [[CrossRef](#)]
24. Kernighan, B.W.; Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **1970**, *49*, 291–307. [[CrossRef](#)]
25. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, 27 November–1 December 1995 pp. 1942–1948. [[CrossRef](#)]
26. Jin, O.; Xing, Q.; Li, Y.; Deng, S.; He, S.; Pan, G. Mapping Very Large Scale Spiking Neuron Network to Neuromorphic Hardware. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2023, Vancouver, BC, Canada, 25–29 March 2023; Aamodt, T.M., Jerger, N.D.E., Swift, M.M., Eds.; ACM: New York, NY, USA, 2023; Volume 3, pp. 419–432. [[CrossRef](#)]
27. Nair, M.V.; Indiveri, G. Mapping high-performance RNNs to in-memory neuromorphic chips. *arXiv* **2019**, arXiv:1905.10692.
28. Dang, K.N.; Doan, N.A.V.; Abdallah, A.B. MigSpike: A Migration Based Algorithms and Architecture for Scalable Robust Neuromorphic Systems. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 602–617. [[CrossRef](#)]
29. Cheng, X.; Hao, Y.; Xu, J.; Xu, B. LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, Yokohama, Japan, 11–17 July 2020; , 2020; pp. 1519–1525. [[CrossRef](#)]
30. Hazan, H.; Saunders, D.J.; Khan, H.; Patel, D.; Sanghavi, D.T.; Siegelmann, H.T.; Kozma, R. BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. *Front. Neuroinform.* **2018**, *12*, 89. [[CrossRef](#)]
31. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans. Wirel. Commun.* **2019**, *19*, 447–457. [[CrossRef](#)]
32. Firouzi, F.; Jiang, S.; Chakrabarty, K.; Farahani, B.J.; Daneshmand, M.; Song, J.; Mankodiya, K. Fusion of IoT, AI, Edge-Fog-Cloud, and Blockchain: Challenges, Solutions, and a Case Study in Healthcare and Medicine. *IEEE Internet Things J.* **2023**, *10*, 3686–3705. [[CrossRef](#)]
33. Zhang, H.; Jolfaei, A.; Alazab, M. A face emotion recognition method using convolutional neural network and image edge computing. *IEEE Access* **2019**, *7*, 159081–159089. [[CrossRef](#)]
34. Yang, S.; Gong, Z.; Ye, K.; Wei, Y.; Huang, Z.; Huang, Z. EdgeRNN: A compact speech recognition network with spatio-temporal features for edge computing. *IEEE Access* **2020**, *8*, 81468–81478. [[CrossRef](#)]
35. Bilal, K.; Erbad, A. Edge computing for interactive media and video streaming. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; pp. 68–73.
36. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [[CrossRef](#)]
37. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [[CrossRef](#)]
38. Wang, B.; Li, M.; Jin, X.; Guo, C. A reliable IoT edge computing trust management mechanism for smart cities. *IEEE Access* **2020**, *8*, 46373–46399.
39. Bekolay, T.; Bergstra, J.; Hunsberger, E.; DeWolf, T.; Stewart, T.C.; Rasmussen, D.; Choo, F.; Voelker, A.; Eliasmith, C. Nengo: A Python tool for building large-scale functional brain models. *Front. Neuroinform.* **2013**, *7*, 48. [[CrossRef](#)]
40. Niedermeier, L.; Chen, K.; Xing, J.; Das, A.; Kopsick, J.; Scott, E.; Sutton, N.; Weber, K.; Dutt, N.D.; Krichmar, J.L. CARLsim 6: An Open Source Library for Large-Scale, Biologically Detailed Spiking Neural Network Simulation. In Proceedings of the International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, 18–23 July 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–10. [[CrossRef](#)]
41. Stimberg, M.; Brette, R.; Goodman, D.F. Brian 2, an intuitive and efficient neural simulator. *eLife* **2019**, *8*, e47314.
42. Wei, D.; Wang, J.; Niu, X.; Li, Z. Wind speed forecasting system based on gated recurrent units and convolutional spiking neural networks. *Appl. Energy* **2021**, *292*, 116842.
43. Catania, V.; Mineo, A.; Monteleone, S.; Palesi, M.; Patti, D. Noxim: An open, extensible and cycle-accurate network on chip simulator. In Proceedings of the 2015 IEEE 26th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Toronto, ON, Canada, 27–29 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 162–163.
44. Deb, K.; Agrawal, S.; Pratap, A.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [[CrossRef](#)]
45. Deng, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [[CrossRef](#)]
46. Rueckauer, B.; Liu, S.C. Conversion of analog to spiking neural networks using sparse temporal coding. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
47. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
48. Xu, X.; Liu, H. ECG heartbeat classification using convolutional neural networks. *IEEE Access* **2020**, *8*, 8614–8619. [[CrossRef](#)]
49. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
50. Igloukov, V.; Shvets, A. Ternaunet: U-net with VGG11 encoder pre-trained on imagenet for image segmentation. *arXiv* **2018**, arXiv:1801.05746.



51. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
52. Rueckauer, B.; Lungu, I.A.; Hu, Y.; Pfeiffer, M.; Liu, S.C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* **2017**, *11*, 682. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.