SNNcut: An Efficient Partitioning Method for Large-Scale Spiking Neural Networks Using Spike-Sharing

Qinghui Xing , Ouwen Jin, Zhuo Chen, Xin Du, Ming Zhang, Shuiguang Deng, *Senior Member, IEEE*, Shuibing He, Ying Li, and Gang Pan, *Member, IEEE*

***Abstract*—Neuromorphic systems, designed to execute Spiking Neural Networks (SNNs) efficiently, feature a many-core architecture equipped with a Network-on-Chip (NoC) to facilitate spike communication. The distributed memory and decentralized computation require partitioning SNNs into multiple subnetworks, each executed on a neuromorphic core in parallel. However, the increasing scale of SNNs leads to high inter-core spike traffic and expensive computational cost of partitioning algorithms. To address these challenges, we propose SNNcut, an efficient partitioning method for large-scale SNNs using spike-sharing. This approach fully exploits spike-sharing by using one NoC packet to deliver multiple spikes of information to reduce spike traffic. Meanwhile, instead of exhaustively searching over neurons in SNN, SNNcut indexes neurons based on the layer connection pattern and clusters sequentially, which significantly reduces the computational complexity and ensures the efficiency of large-scale SNN partitioning. Experimental results demonstrate that SNNcut achieves an average reduction in spike traffic of 98.97%, compared to 74.98% from state-of-the-art existing methods. In addition, the SNNs partitioned by SNNcut require only $5.7\%\sim15.7\%$ neuromorphic core consumption of previous works. In a large-scale scenario, existing methods take more than 100 hours to accomplish the partitioning, while SNNcut only needs 10 minutes. We also validate the scalability of SNNcut by testing on edge cases with 3.2B neurons and 902B synapses.**

***Index Terms*—Neuromorphic System, Spiking Neural Networks, Partitioning, Hilbert Space Filling Curve.**

I. Introduction

Spiking Neural Networks (SNNs) [1], [2] have garnered increasing attention due to their biological plausibility and potential for facilitating efficient computing. An SNN consists of neurons that communicate asynchronously through discrete spikes via inter-neuron synapses. In recent years, there has been significant progress in the development of neuromorphic systems which are specifically designed for executing SNN efficiently. Notable examples include Darwin3 [3], Loihi [4], Loihi2 [5], SpiNNaker [6], SpiNNaker2 [7], Tianjic [8], TrueNorth [9], and BrainScaleS [10]. These systems typically feature homogeneous many-core architecture, with each neuromorphic computing core being identical in resource configuration (including memory capacity and computational units) and capable of simulating multiple neurons. The inter-core communications are facilitated through Network-on-Chip (NoC) [11] to transmit spike messages, while the localized memory paradigm ensures all other data required for computation—including fan-in (dendritic) connections and weights, fan-out (axonal) connections, and neuron state variables—are stored exclusively in on-chip memory.

Many neuromorphic systems have developed compilers or toolkits for deploying SNNs onto neuromorphic hardware [12]–[14]. The deployment pipeline explicitly or implicitly works as follows. At first, all neurons of SNN are **partitioned** into distinct neuron clusters such that each cluster serves as an atomic workload unit mapped to a single neuromorphic computing core. This one-to-one correspondence between clusters and cores ensures that each cluster's memory requirements—including axon memory (post-synaptic connections), dendrite memory (pre-synaptic connections), and neuron state memory—strictly adhere to the local on-chip memory capacity constraints of a single neuromorphic computing core. Then, each cluster is mapped to an available neuromorphic computing core. The partitioning process must simultaneously address two optimization objectives: 1) minimizing inter-core spike traffic, which remains a dominant contributor to elevated energy consumption and runtime latency [15]–[17]; 2) reducing the number of required neuron subnets/clusters. By minimizing the core count allocated to individual SNNs, partitioning algorithms free up more cores for the concurrent execution of multiple neural networks.

The recent studies on SNN partitioning typically treat it as a generic graph-partitioning task and either employ greedy strategies [18], [19] or adopt heuristic algorithms for graph-partitioning like the Kernighan-Lin (KL) algorithm [20]–[22]. However, existing works are primarily developed for relatively small networks, where the search space is limited and computational costs remain manageable. This limitation stems from two critical issues: First, both greedy strategies and KL algorithms rely on exhaustive neuron traversal for optimization. The KL algorithm incurs prohibitive $O(N^3)$ complexity due to iterative vertex swapping, while greedy methods face $O(N \cdot C \cdot Cost)$ computational overhead (where N is neuron count, C is cluster count, and $Cost$ represents objective function complexity). Second, greedy and KL algorithms perform localized searches guided by specific optimization objectives. As neuron/synapse counts increase, the solution space explodes exponentially, trapping these methods in suboptimal local minima due to insufficient exploration capacity. With the rapid scaling of SNNs [23]–[25], these limitations become critical bottlenecks. This scale expansion introduces three fundamental **challenges** for partitioning: 1) *Communication Overhead Explosion*. The surge in inter-neuron spikes causes prohibitively high inter-core communication demands. [15]–[17]. 2) *Proliferation of Neuron Clusters*. Massive synaptic connections in large-scale models exacerbate memory contention, forcing partitioners to create a large number of clusters. 3) *Scalability Limitations*: As neuron/synapse counts grow exponentially, existing partition strategies based on exhaustive search become computationally impractical. This necessitates fundamentally new partitioning approaches with improved time efficiency while maintaining solution quality.

To address these challenges, this paper presents a novel partitioning method for large-scale SNNs, named SNNcut, to reduce communication overhead and lower core consumption. SNNcut greatly exploits the spike-sharing mechanism of neuromorphic hardware, which *allows neurons within the same core to share the NoC spike packets sent from their common pre-synaptic neurons* [3]–[7]. In the meantime, SNNcut is designed to maintain computational efficiency. SNNcut primarily focuses on clustering neurons with common pre-synaptic neurons together in order to maximize the utilization of spike-sharing to reduce spike traffic and core consumption. To achieve this, SNNcut partitions SNN in a layer-wise manner

Qinghui Xing, Ouwen Jin, Zhuo Chen, Xin Du, Ming Zhang, Shuiguang Deng, Shuibing He, Ying Li, and Gang Pan are with Zhejiang University, Hangzhou, China (Email: gpan@zju.edu.cn).
Corresponding author: Gang Pan.

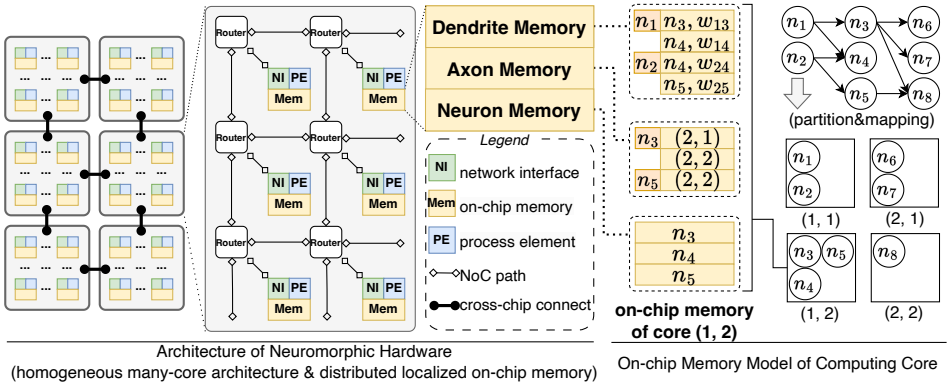


Fig. 1: Neuromorphic system architecture. Illustration of the typical homogeneous many-core design connected by a Network-on-Chip. Each core's distributed, localized on-chip memory (Dendrite, Axon, Neuron Memory) necessitates SNN partitioning, as the full network often exceeds the capacity of a single core.

and maps neurons into a one-dimensional sequence based on the layer connection pattern, such that the adjacent neurons on the sequence would have a higher probability of sharing the same pre-synaptic neurons. Then, SNNcut clusters neurons by segmenting along the sequence under resource constraints. SNNcut significantly reduces the computational complexity and ensures the efficiency of large-scale SNN partitioning. The layer-wise partitioning design also provides support for incremental compilation, which is essential for rapid deployment cycles in model development. The contributions of this paper are summarized as follows.

- We propose SNNcut, an efficient partitioning method for large-scale SNNs using spike-sharing. SNNcut primarily maps neurons into a one-dimensional sequence and segments along the sequence to cluster neurons together. SNNcut ensures a high-quality solution by leveraging spike-sharing to a great extent. Meanwhile, SNNcut eliminates the exhaustive searching over neurons in SNN and significantly reduces the computational complexity of partitioning.
- We propose the indexing rules to map neurons in different layers in a way that the utilization of spike-sharing can be fully exploited. For example, we propose to employ Hilbert Space Filling Curve (HSFC) to index neurons in convolutional layers since HSFC has the ability to preserve the locality of neurons with common pre-synaptic neurons. We also propose an FFD-based algorithm to index neurons of fully-connected layers.
- We have conducted extensive experiments on 6 SNN workloads with different scales and structures. The experimental results demonstrate that SNNcut consistently achieves improved spike traffic reduction and reduced core consumption over existing methods.

II. Background and Related Works

A. Preliminary of SNN

In SNN, neurons are the basic computation units with private states such as membrane potential, and the synapses connect neurons with different weights. The execution of SNNs involves multiple timesteps. At each timestep, all neurons 1) update their membrane potentials based on activation value, 2) send spikes to their post-synaptic neurons if the membrane potentials reach the threshold, and 3) receive spikes sent from pre-synaptic neurons and update the activation value with associated weights. The computation of each timestep remains consistent, and each neuron computes in parallel.

To obtain a well-trained SNN, ANN-to-SNN conversion is one of the most prevalent strategies and achieves the best performance in most tasks [26]. However, since ANN-to-SNN conversion is essentially derived from rate-coding (e.g., representing real value with the number of spikes), neurons in ANN-converted SNNs often exhibit much higher firing rates and longer running timesteps [16]. The growing spikes lead to increased communication demand at runtime, which impairs the energy efficiency and latency of the neuromorphic system.

B. Neuromorphic System

1) Homogeneous Many-Core Architecture

As shown in Fig. 1, most neuromorphic chips adopt a homogeneous many-core architecture that arranges cores in a two-dimensional array, and Network-on-Chip is equipped to support the inter-core communication. To enable a larger scale workload as well as to improve parallelism, most neuromorphic systems concatenate multiple chips together by building cross-chip communication channels on the border [3], [6], [7]. Assuming all cores are arranged in X columns and Y rows, the neuromorphic hardware can be modeled as a list of neuromorphic cores with coordinates as follows:

$$\mathcal{H} = \{(x, y) | x \in [1, X]; y \in [1, Y]; x, y \in \mathbb{N}^+\} \quad (1)$$

2) Localized On-Chip Memory Model

Most neuromorphic architectures adopt a distributed memory paradigm where each computing core maintains localized on-chip memory for complete neuronal connectivity and state variables. As depicted in Fig. 1, the localized on-chip memory records dendritic fan-in connections (including pre-synaptic weights), axonal fan-out routing information, and somatic state parameters (e.g., membrane potentials, firing thresholds) with dedicated SRAMs [3]–[7]. Therefore, the capacity of these on-chip memories decides how many neurons and connections one core can fit. Before introducing the on-chip memory organization, let us denote the neurons within a core as $\mathcal{N}_{neuron}$ and the union of pre-synaptic neurons of $\mathcal{N}_{neuron}$ as $\mathcal{N}_{pre}$.

In detail, *Dendrite Memory* is composed of several memory blocks, each associated with a pre-synaptic neuron $n^i_{pre} \in \mathcal{N}_{pre}$. The memory block of n^i_{pre} records the post-synaptic connections (i.e., the indices of post-synaptic neurons in $\mathcal{N}_{neuron}$ and corresponding synaptic weights) of n^i_{pre}. Fig. 1 presents an example, where the *Dendrite Memory* of core $(1, 2)$ records two blocks, each of which is uniquely identified by the pre-synaptic neurons (n_1 and n_2), and holds the associated neuron indices and synaptic weights. For n_1, the block records its connections to n_3 and n_4. The block of n_2 follows the same principle.

Axon Memory also comprises several memory blocks, and each block is indexed by a neuron $n^i \in \mathcal{N}_{neuron}$. For each neuron n^i, its corresponding memory block stores the coordinates of neuromorphic cores that contain the post-synaptic neurons of n^i. For example, in Fig. 1, core $(1, 2)$ has two axon memory blocks. The one indexed by n_3 records core $(1, 1)$ and core $(2, 2)$ at which the post-synaptic neurons of n_3 are located. The same applies to n_5. Neuron n_4 does not need

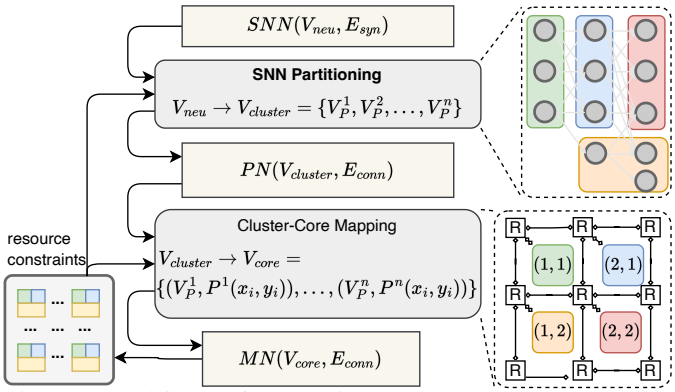


Fig. 2: Workflow of deploying SNN onto neuromorphic hardware. The process involves two main stages: (1) SNN Partitioning, which divides the SNN (V_{neu}) into resource-constrained clusters ($V_{cluster}$), and (2) Cluster-Core Mapping, which assigns each cluster to a physical core (V_{core}) on the hardware.

axon memory since it has no post-synaptic neurons. When neuron n^i fires, the routing destination of the spike packet is retrieved from its axon memory block, all neurons share the *Axon Memory*, and the total size of their axon memory blocks must not exceed the capacity of *Axon Memory*.

Similarly, the state variables of each neuron are stored in *Neuron Memory* and identified by the neuron index $n^i \in \mathcal{N}_{neuron}$. The capacity of *Neuron Memory* is determined by a specific boundary, limiting it to a fixed number of neurons up to a predefined maximum(N_{neu}).

We denote the size of the dendrite memory block of n^i_{pre} as $den_{n^i_{pre}}$ and the size of the axon memory block of neuron n^i as $axon_{n^i}$. A formal definition of the core memory constraints can be expressed as follows:

$$|\mathcal{N}_{neuron}| \leq N_{neu} \tag{2}$$

$$\sum_{n^i_{pre} \in \mathcal{N}_{pre}} den_{n^i_{pre}} \leq N_{den} \tag{3}$$

$$\sum_{n^i \in \mathcal{N}_{neuron}} axon_{n^i} \leq N_{axon} \tag{4}$$

where N_{axon} and N_{den} denote the capacity of *Axon Memory* and *Dendrite Memory*, respectively.

C. Deployment Workflow

To deploy an SNN onto neuromorphic hardware, the network has to be partitioned into smaller workloads since a single core cannot fit the entire network. The deployment workflow is illustrated in Fig. 2 and mainly comprises two steps: 1) partitioning the original SNN into disjoint clusters, each constrained by the memory capacity of a single neuromorphic core, and 2) mapping each cluster to an available core for runtime computation.

1) Partitioning

SNN can be represented as a Directed Acyclic Graph (DAG) ($SNN(V_{neu}, E_{syn})$) with neurons as vertices and synapses as edges. Formally, SNN partitioning requires partitioning V_{neu} into disjoint clusters ($V_{cluster} = \{V_P^1, V_P^2, ..., V_P^n\}$) with each cluster meets the memory constraints presented in Eq. (2)(3)(4), under two optimization objectives: 1) inter-cluster spike traffic minimization; and 2) cluster count minimization.

$$V_{neu} \rightarrow V_{cluster} = \{V_P^1, V_P^2, ..., V_P^n\}, \bigcup_{i=1}^{n} V_P^i = V_{neu} \tag{5}$$

The partitioned network is denoted as $PN(V_{cluster}, E_{conn})$. Two clusters are considered connected if there are synapses across them.

TABLE I: Involved symbols and the definitions

Symbol	Definition
$SNN(V_{neu}, E_{syn})$	DAG of SNN
V_{neu}	neurons in SNN
E_{syn}	synapses in SNN
$PN(V_{cluster}, E_{conn})$	DAG of partitioned SNN
$V_{cluster}$	neuron clusters
V_P^j	the j-th neuron cluster in $V_{cluster}$
E_{conn}	connections among neuron clusters
$\mathcal{N}^i_{neuron}$	set of neurons in V_P^i
$presyn(n)$	pre-synaptic neurons of neuron n
$\mathcal{N}^j_{pre}$	union of $presyn(n)$ of all neurons in $\mathcal{N}^i_{neuron}$
$MN(V_{core}, E_{conn})$	the mapped PN
V_{core}	neuron clusters with allocated computing cores
$P^i(x_i, y_i)$	the coordinates of allocated core for cluster V_P^i
N	the number of neurons in SNN
f	average fan-in of neurons in SNN
C	the number of clusters in PN
L	the number of layers in SNN
s_n	firing rate of neuron n

2) Mapping

After partitioning, each cluster will be allocated to an available neuromorphic computing core in $\mathcal{H}$.

$$V_{cluster} \rightarrow V_{core} = \{(V_P^1, P^1(x_1, y_1)), ..., (V_P^n, P^n(x_n, y_n))\}, \quad P^1(x_1, y_1), ..., P^n(x_n, y_n) \in \mathcal{H} \tag{6}$$

We denote the mapped network as $MN(V_{core}, E_{conn})$, and its connections/edges are directly inherited from PN.

D. Related Works

Prior efforts on SNN partitioning can be broadly classified into heuristic algorithms (often adapted from classical graph partitioning) and greedy strategies tailored to neuromorphic systems. We summarize key approaches in both categories and discuss their limitations in handling large-scale SNNs, especially under hardware constraints like limited on-chip memory and spike communication overhead, which motivate our proposed solution.

Heuristic-Based SNN Partitioning. SNN partitioning can be viewed as an edge-cut graph partitioning problem on the neuron connectivity graph. The goal is to assign neurons (graph vertices) to cores such that inter-core synaptic connections (edge cuts) are minimized [27]–[38]. This problem is NP-hard, so past SNN compilers [20]–[22] have resorted to graph-partitioning heuristics, most notably the Kernighan–Lin (KL) algorithm. KL iteratively swaps pairs of neurons between clusters to reduce a cost function (e.g. inter-cluster spike count). This heuristic was employed in SNN mapping frameworks like SpiNeMap [20] and the method by Song et al. [21] with the number of crossing spikes as the objective. While KL-based techniques can indeed lower inter-core spike traffic, they ignore neuromorphic memory constraints. In practice, moving a single neuron can trigger a cascade of axon memory overflow checks on all cores hosting its pre-synaptic neurons (see Fig. 3). As a result, each swap incurs additional overhead of $O(f)$ memory validations (where f is the neuron's fan-in) at least, inflating the algorithm's complexity to $O(N^3 f)$. This exponential increase makes straightforward KL partitioning infeasible for large networks. Researchers have introduced more advanced graph partitioning methods (e.g. multi-level partitioners like METIS [39] and streaming heuristics like LDG [31] and Fennel [27]) to improve scalability. However, these generic algorithms focus on balancing partitions and minimizing cut edges, and they do not account for neuromorphic-specific issues such as dynamic axon memory per neuron or the spike-sharing communication mechanism. In other words, traditional graph partitioners cannot directly satisfy the unique constraints of SNN partitioning.

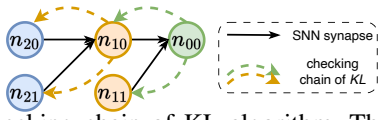


Fig. 3: The checking chain of KL algorithm. The unscalable validation overhead of KL-based partitioning. This illustrates how swapping a single neuron (part of the KL algorithm) triggers a complex cascade of $O(f)$ memory validation checks across all cores hosting its pre-synaptic neurons, leading to prohibitive computational complexity.

TABLE II: Comparison with related works

Methods	Neuron	Dendrite	Axon	Complexity
KL [20]–[22]	✓	×	×	$O(N^3 f)$
EdgeMap [19]	✓	✓	×	$O(NC\|E_{conn}\|))$
DFSynthesizer [18]	✓	✓	×	$O(NC\log(C))$
SNNcut	✓	✓	✓	$O(L\log(N_{layer}))$

N_{layer}: average # neurons per layer

Greedy SNN Partitioning Approaches. In parallel, several SNN partitioning methods use greedy, locally optimal strategies to incrementally build a solution. These approaches assign neurons to clusters based on immediate cost benefits, rather than exhaustive global optimization. For example, EdgeMap [19] introduces a dual-objective cost function that balances inter-core communication latency against computational load. Likewise, DFSynthesizer [18] prioritizes placing neurons into clusters with high synaptic connectivity to maximize hardware resource utilization. By designing such problem-specific cost metrics, greedy methods can incorporate domain knowledge (e.g. network topology or hardware limits) into partitioning decisions. Unfortunately, these methods struggle to scale to very large SNNs. The time complexity grows with both the number of neurons and clusters, leading to prohibitive runtimes on big networks. In fact, EdgeMap [19] and DFSynthesizer [18] reportedly required over 100 hours to partition VGG19 of 16 million neurons (see Section V). Moreover, the quality of greedy partitions degrades as network size increases: when scaling from 1 million to 3.5 million neurons, the fraction of inter-core spike traffic more than doubled (from 10% to 21%) for DFSynthesizer, and rose from 25.7% to 57.6% for EdgeMap. This drop in performance is attributed to the exponential growth of the search space, which causes greedy optimizers to get trapped in suboptimal local minima.

Our Approach – SNNcut. SNNcut differentiates itself from prior work by explicitly addressing the above limitations. It is the first SNN partitioning method to account for the per-neuron axon memory constraint during clustering. Instead of relying on iterative swapping or per-neuron greedy assignment, SNNcut uses a constructive partitioning strategy that leverages the *spike-sharing* mechanism inherent in many neuromorphic architectures [3]–[7]. In essence, neurons are clustered sequentially according to their layer-wise connectivity patterns. By preserving this sequential order, neurons with similar connectivity naturally form groups, thus maximizing the chance that neurons sharing the same presynaptic inputs will be assigned together. This ordered clustering leverages spike-sharing (delivering a single spike to activate multiple synapses), significantly reducing inter-core communication overhead. By co-designing the partitioning algorithm with SNN-specific characteristics, SNNcut avoids both the heavy search overhead of KL-style heuristics and the scalability pitfalls of naive greedy methods, achieving efficient partitioning even for extremely large SNNs.

III. SNNcut

In this section, we present the design of SNNcut. As shown in Fig. 4 and Algorithm 1, SNNcut consists of three stages: 1) *topological sorting*: According to the definition of axon memory, the axon memory size of a neuron can be determined once its post-synaptic neurons have been clustered. Therefore, the partitioning process must be conducted in the topological order of SNN layers; 2) *spike-sharing-oriented neuron indexing*: For each layer, to better utilize the spike-sharing mechanism, neurons are indexed and mapped to a one-dimensional sequence such that the adjacent neurons in the sequence have more common pre-synaptic neurons. We design the HSFC-based and FFD-based neuron indexing algorithms for convolution layers and fully-connected layers, respectively; and 3) *binary segmentation of neuron sequence*: Finally, we propose binary segmentation (*BS_Segment*) presented in Algorithm 2 to efficiently segment the neuron sequence into multiple clusters under on-chip memory constraints as presented in Eq. (2)(3)(4).

Algorithm 1: SNNcut Overview

Input: SNN with L layers $\{layer_1, ..., layer_L\}$; neuromorphic core capacity $(N_{den}, N_{axon}, N_{neu})$
Output: Neuron Clusters $V_{cluster}$

```
/* topologically sort snn layers.   */
Sort SNN : {layer_1, ..., layer_L} in topological order;
V_cluster ← {};
/* For each layer, index neurons into
   sequence S, each neuron n has
   memory sizes (den_n, axon_n);     */
for layer l in sorted layers do
    if l.type lies in {convolution, pooling} then
        S ← index_neurons_with_HSFC(l);
    else if l.type lies in {fully_connected} then
        S ← index_neurons_with_FFD(l);
    else
        S ← default_order;
    /* segment to fit core capacity. */
    V^l_cluster = BS_Segment(S, N_den, N_axon, N_neu);
    Insert clusters in V^l_cluster to V_cluster;
Return V_cluster;
```

A. Fundamental Principle

The communication paradigm of representative neuromorphic architectures [3]–[7] reveals a common characteristic that enables a significant reduction of inter-core traffic. This mechanism operates as follows: *For synapses (i.e., their associated post-synaptic neurons) sharing a common pre-synaptic neuron, a single spike packet can trigger activation accumulation across all corresponding synapses*, provided they reside within the same core. We term this mechanism ***spike-sharing***, denoting that all associated synapses share one spike packet. The fundamental goal of SNNcut is to maximize the utilization of the spike-sharing mechanism, which is illustrated in Fig. 5. We achieve this by co-locating (i.e., clustering) post-synaptic neurons that share common pre-synaptic neurons. A higher degree of aggregation for these neurons directly translates to two key benefits: 1) maximized spike-sharing, which reduces inter-core spike traffic, and 2) reduced axon memory consumption for the pre-synaptic layer (as fewer core destinations need to be stored).

Better utilization of spike-sharing brings more efficient communication and reduced axon memory. Fig. 6 presents two partitionings where the second better utilizes spike-sharing. In the second partitioning, all post-synaptic neurons of n_0 are located in the same core, such that when n_0 fires, only one spike packet is transmitted on NoC. A neuron's Axon memory stores the coordinates of all cores that contain its post-synaptic neurons (as illustrated in Fig. 5). This creates a key optimization opportunity: if multiple post-synaptic neurons (e.g., n_1, n_2, n_3) of a single pre-synaptic neuron (e.g., n_0)

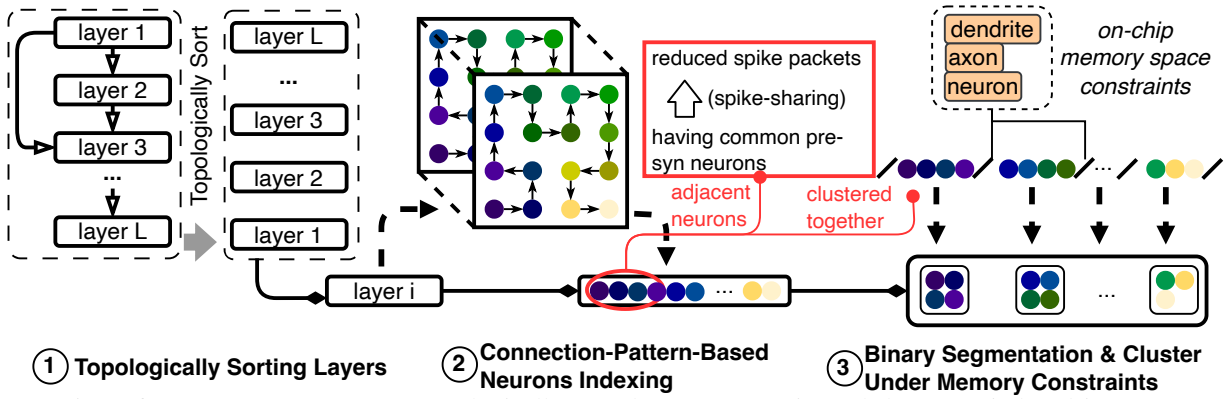


Fig. 4: Overview of SNNcut. (1) Layers are topologically sorted. (2) Neurons in each layer are indexed into a 1D sequence based on connection patterns (e.g., HSFC). (3) The sequence is segmented into clusters under hardware on-chip memory constraints, grouping adjacent neurons to maximize spike-sharing.

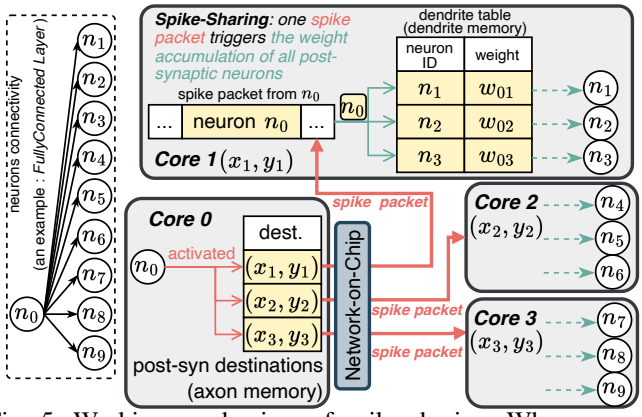


Fig. 5: Working mechanism of spike-sharing. When neuron n_0 is activated, only 3 NoC spike packets are generated, each triggers the weight accumulations of post-synaptic neurons $[n_1, n_2, n_3]$, $[n_4, n_5, n_6]$, and $[n_7, n_8, n_9]$, respectively. The post-synaptic neurons within the same core share 1 spike packet. The objective of SNNcut is to reduce the NoC spike traffic generated per neuron activation (i.e., firing) by exploiting the spike-sharing mechanism.

are all clustered into the same destination core (e.g., *Core 1*), the Axon memory for n_0 only needs to store the coordinate of *Core 1* (x_1, y_1) once.

SNNcut's strategy of *aggregating post-synaptic neurons that share common pre-synaptic neurons* is explicitly designed to exploit spike-sharing opportunities. By ensuring these *sibling* post-synaptic neurons are co-located, our method minimizes the number of distinct post-synaptic destination cores they are spread across. This directly collapses the number of coordinate entries each pre-synaptic neuron must store, alleviating the Axon memory bottleneck, which we identified as a primary driver for creating underutilized cores. This allows for a much higher neuron density per cluster, thereby reducing the total core consumption. Fig. 7 illustrates this with an example from our experimental results. Starting from layer l, *SNNcut-woS* (a partitioning method without effective utilization of spike-sharing, see Section IV) results in more cores being used compared to *SNNcut*. The neuron density of a core (or cluster) is co-limited by all three primary hardware memory constraints: Dendrite, Axon, and Neuron memory. A core becomes *full* (or *bounded*) when any one of these memory resources is exhausted. For each neuron, its Dendrite and Neuron memory demands are static, fixed by the SNN model. The Axon memory demand is dynamic, determined by the partitioning of post-synaptic neurons. SNNcut's core strategy is to minimize this dynamic Axon memory demand by aggregating *sibling*

Algorithm 2: BS_Segment

Input: neuron sequence S; core capacity $(N_{den}, N_{axon}, N_{neu})$
Output: neuron clusters of layer l: $V^l_{cluster}$

$V^l_{cluster} \leftarrow \{\}$; $start \leftarrow 0$;
while $start < |S|$: **do**
  $low \leftarrow start; high \leftarrow (|S|-1); split \leftarrow start;$
  while $low \leq high$ **do**
    $mid = \text{floor}((low + high)/2)$;
    if $\sum_{start}^{mid}(den_i) <= N_{den}$ *and* $\sum_{start}^{mid}(axon_i) <= N_{axon}$ *and* $mid - start + 1 <= N_{neu}$ **then**
      $split = mid$ $low = (mid + 1)$;
    else
      $high = (mid - 1)$;
  Record neuron cluster $S[start : split]$ in $V^l_{cluster}$;
  $start = (split + 1)$;
Return $V^l_{cluster}$;

post-synaptic neurons, while SNNcut-woS lacks such axon-memory-size-oriented optimization.

Layer $l+1 \Rightarrow$ **Layer** l: For neurons in layer l, their post-synaptic neurons lie in layer $l+1$. In topological order, layer $l+1$ is partitioned before layer l.

In the SNNcut-woS case (top row), partitioning layer $l+1$ without spike-sharing scatters its *sibling* post-synaptic neurons. This scattering inflates the Axon memory demands for layer l neurons (darker red circles). Consequently, partitioning layer l quickly saturates the Axon memory, creating numerous *axon-bounded* cores (red squares). These cores are inefficient, leaving Dendrite and Neuron memory underutilized.

In the SNNcut case (bottom row), our strategy proactively clusters these *sibling* neurons, which directly reduces the Axon memory demand for layer l neurons (lighter pink circles). This relieves the Axon memory pressure, ensuring it is no longer the bottleneck. Thus, SNNcut packs more layer l neurons into each core, filling the otherwise underutilized Dendrite or Neuron memory. This results in higher resource utilization and a dramatically lower total core count.

Layer $l \Rightarrow$ **Layer** $l-1$: This benefit propagates to layer $l-1$. The increased core count for layer l in the SNNcut-woS case further scatters the post-synaptic neurons of layer $l-1$, drastically increasing its axon memory demands. This forces SNNcut-woS to generate numerous cores for layer $l-1$, most of which become axon-bounded (red squares). In contrast, SNNcut's denser partition of layer l, combined with its aggregation of *sibling* post-synaptic neurons, ensures that

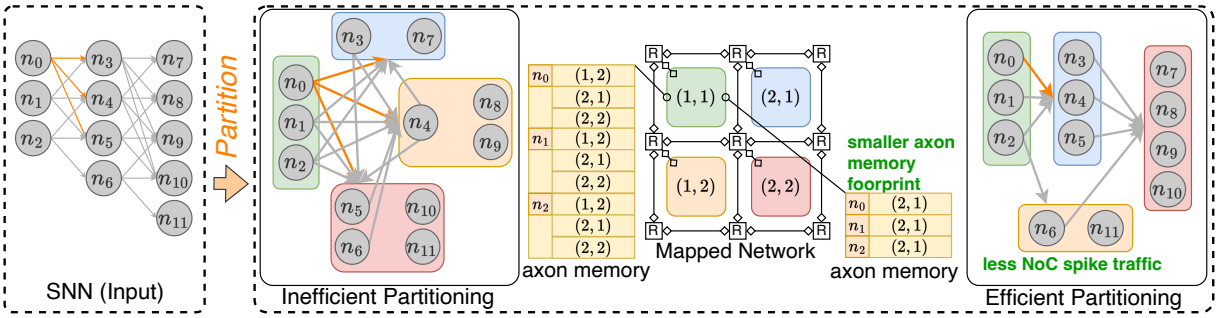


Fig. 6: Better utilization of spike-sharing brings more efficient communication and reduced axon memory. In the SNN, neuron n_0 has 3 post-synaptic neurons $[n_3, n_4, n_5]$. Left (Inefficient Partitioning): Scattering post-synaptic neurons (n_3, n_4, n_5) across multiple cores inflates spike traffic and axon memory for the pre-synaptic neuron (n_0). Right (Efficient Partitioning): SNNcut co-locates these neurons, reducing spike packets to one and minimizing axon memory, enabling less NoC spike traffic and smaller axon memory footprint.

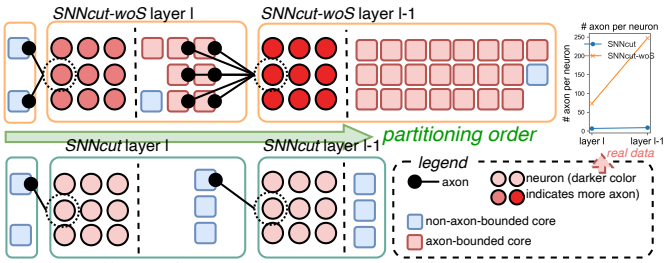


Fig. 7: Illustrations on SNNcut reducing core consumption by alleviating the axon memory bottleneck. The top row (SNNcut-woS) shows how failing to cluster post-synaptic neurons that share common pre-synaptic neurons leads to high axon memory demands (darker red circles), creating inefficient *axon-bounded* cores (red squares). The bottom row (SNNcut) demonstrates how proactively clustering those *sibling* neurons reduces axon demands (lighter pink circles), allowing for higher neuron density and fewer cores (blue squares). This advantage propagates from layer $l+1$ to $l-1$. (Note: The figure is plotted proportionately to experimental data.)

layer $l-1$ neurons also have a smaller Axon memory footprint. As a result, SNNcut significantly reduces the core count for layer $l-1$, creating efficient, non-axon-bounded cores (blue squares).

Targeted at the challenges of increased scale SNN partitioning, the fundamental principle behind SNNcut is *to layerwisely cluster the (post-synaptic) neurons that have common pre-synaptic neurons together, with a constructive strategy based on the layer connection pattern*. The reasons are detailed as follows:

Firstly, neurons of the same layer have the same connection pattern, which renders strategies searching at the neuron level redundant and inefficient. For example, the pre-synaptic connections of neurons in convolution layers lie within a *receptive-field* window. To speed up partitioning, we propose a constructive strategy concerning layer connection patterns and eliminate the redundant searching over individual neurons.

Secondly, to better exploit the utilization of spike-sharing, our constructive strategy encourages clustering post-synaptic neurons that have more common pre-synaptic neurons together such that both network latency and energy cost can be optimized through reduced spike traffic.

Lastly, with the aforementioned strategy, axon memory for the pre-synaptic neurons can be saved since their post-synaptic neurons are clustered into fewer cores, i.e., their destination cores decrease. That benefits the on-chip memory utilization and reduces the partitioned clusters.

B. Layer-wise Partitioning in Topological Order

When clustering neurons, the partitioning strategy must ensure that the memory size of the cluster fits within the capacity of a neuromorphic core. According to the core memory model introduced in Section II-B, the axon memory size of a neuron can only be determined once its post-synaptic neurons have been clustered. Therefore, the partitioning process must be conducted in the topological order of neurons in the SNN. Since neurons in the same layer have no connections between each other, there is no need to sort the entire graph of neurons. Instead, SNNcut ensures the order by topologically sorting the layers of the SNN and partitioning them in a layer-wise manner.

Unlike monolithic graph-partitioning methods (e.g., KL-based) that treat the SNN as a flat graph, the layer-wise partitioning strategy of SNNcut provides strong advantages for handling the online re-partitioning scenarios. In an adaptive SNN where the structure might change (e.g., through online learning), SNNcut would not require a full re-partitioning of the entire network. Instead, only the modified layers and their subsequent dependent layers would need to be re-processed. The partitioning results for all unchanged layers could be directly reused.

C. Spike-sharing-oriented Neuron Indexing

Unlike existing works that exhaustively search over individual neurons, we propose layer-specific strategies that index and map all neurons of the layer into a sequence and segment along the neuron sequence under on-chip memory constraints. Each round of segmentation packs as many neurons as possible until the memory capacity is reached to avoid memory space waste.

Based on the connection pattern, layer-specific neuron indexing strategies are proposed to maximize the number of common pre-synaptic neurons of adjacent neurons in the sequence. Since segmentation is carried out sequentially, adjacent neurons, i.e., neurons that share more common pre-synaptic neurons, can reside together in the same cluster to improve the utilization of spike-sharing.

Before introducing the indexing rules, we define the ***similarity*** (formally defined in Eq. (7); $|\cdot|$ denotes the L_0 norm) between two neurons as *the number of their common pre-synaptic neurons*.

$$sim(i,j) = |presyn(i) \cap presyn(j)| \tag{7}$$

1) Indexing Rule of Convolution Layers

Neurons in convolution layers are organized in three dimensions: width (W), height (H), and channel (C), and let us discuss the *similarity* distribution on WH plane first.

The similarity among neurons in the convolution layer exhibits a peak-shaped distribution, as shown in Fig. 8(a). This suggests that neurons located closer to each other on the two-dimensional WH plane tend to have higher similarity. Based

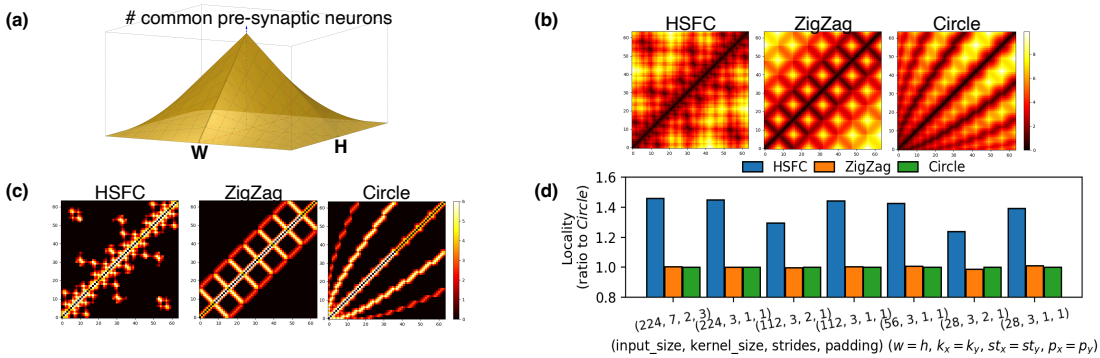


Fig. 8: Locality-preserving of different space-filling curves. (a) *Similarity* distribution of neurons in convolution layer. (b) Distance among points in different curves. (c) *Similarity* among neurons indexed by different curves (Diagonal values are set to 0). (d) Locality of different curves on representative configurations. Collectively, the results show HSFC (blue bar in (d)) better preserves locality for high-similarity neurons, validating its use for SNNcut.

on this observation, we propose to employ the Hilbert Space-Filling Curve (HSFC) to index neurons in convolution layers. HSFC is a space-filling curve that maps a two-dimensional array to a one-dimensional sequence while preserving locality; in other words, items that are adjacent in the two-dimensional space remain close to each other in the one-dimensional sequence [40]. It has been mathematically proven that HSFC exhibits superior *locality-preserving* properties compared to other space-filling curves [41]. Therefore, on the *WH* plane of the convolution layer, we index neurons according to the order defined by HSFC so that neurons with higher similarity are placed in adjacent positions in the resulting sequence, and thus are more likely to be grouped together during sequence segmentation.

We conduct a statistical analysis of different space-filling curves to demonstrate the suitability of HSFC for partitioning convolution layers. An 8×8 array was mapped into sequences using HSFC, ZigZag, and Circle curves. Fig. 8(b) shows the 2D Euclidean distance between points in those sequences (e.g., the distance between the i-th and j-th point is displayed at positions (i,j) and (j,i)). Darker points indicate smaller distances. Points near the diagonal correspond to distances between adjacent items in the sequence. In the HSFC heatmap, these points exhibit smaller distances, reflecting its superior locality-preserving property.

Furthermore, we index and map neurons in a convolution layer with 8×8 input shape, 3×3 kernel size, 1×1 stride, and 8×8 output shape, using the same curves, and measure the *similarity* (7) among neurons in the resulting sequences. The similarity distribution is shown in Fig. 8(c), where positions (i,j) and (j,i) represent $sim(i,j)$. Lighter points indicate higher similarity. For the HSFC-based indexing, points near the diagonal show higher values, indicating that HSFC effectively maps neurons with high similarity to nearby positions in the sequence.

Since the segmentation process is sequential, the distribution in Fig. 8(b) can be interpreted as the probability that two neurons are clustered together. Based on this, we provide a quantitative comparison of the effectiveness of different space-filling curves in facilitating spike-sharing by *masking* the two distributions shown in Fig. 8(b) and Fig. 8(c).

$$locality = \sum_{i<j,\forall i,j\in[1,N_l]} \frac{sim(i,j)}{||i-j||^{1.5}} \tag{8}$$

Specifically, we define *locality* (8) as the sum of the *similarity* between each pair of neurons, weighted by the inverse of their distance in the sequence. The evaluation of representative configurations is illustrated in Fig. 8(d). The results demonstrate that HSFC is better suited for partitioning convolution layers compared to other space-filling curves.

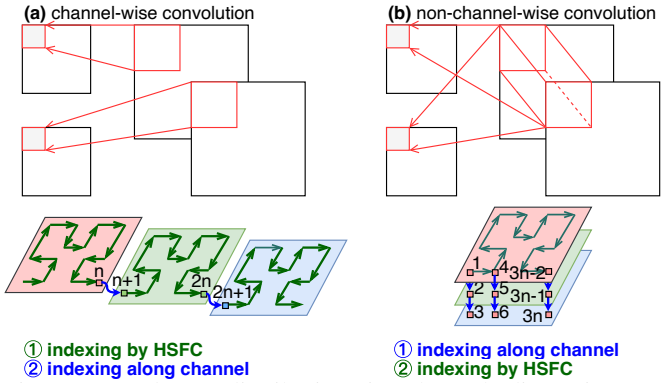


Fig. 9: *Similarity* distribution in channel dimension. (a) channel-last order: Used for channel-wise convolution. After indexing neurons in *WH* plane by HSFC, the sequences are concatenated along the channel dimension. (b) channel-first order: Used for non-channel-wise convolution. Clustering neurons with the same (w,h) together and then indexing the groups by HSFC.

For the channel dimension, the indexing rule depends on whether the connection is channel-wise. Fig. 9 shows the *similarity* distribution along the channel dimension. In channel-wise convolution layers (e.g., *DepthwiseConv2D* in Keras [42]), neurons across different channels do not share pre-synaptic neurons. These are indexed in a *channel-last* order (Fig. 9(a)), where the *WH* plane is first indexed using HSFC, and the sequences of different channels are then concatenated to form the final sequence. In contrast, for non-channel-wise layers, neurons with the same *WH* coordinates (w,h) but different channels share the same set of pre-synaptic neurons. Therefore, we adopt a *channel-first* indexing order (Fig. 9(b)), where neurons with the same *WH* coordinates (w,h) across different channels are grouped together first due to their high similarity, and these pre-clustered groups are subsequently indexed by HSFC based on their (w,h). This indexing strategy ensures that neurons with higher similarity are consistently placed close to each other in the sequence.

2) Indexing Rule of FC Layers

In fully connected (FC) layers, all neurons share the same set of pre-synaptic neurons, i.e., each neuron n has the same $presyn(n)$. Consequently, fewer clusters result in greater spike-sharing utilization and reduced axon memory usage. To maximize this benefit, neurons should be indexed such that the number of clusters is minimized after segmentation.

Since each neuron in an FC layer has the same number of pre-synaptic connections, the dendrite memory consumption den_n is uniform across neurons. Therefore, clustering neurons based on their varying axon memory demands $axon_n$ while

TABLE III: Hardware Configurations

Hardware	N_{den}	N_{neu}	N_{axon}	# Cores
Darwin3 [3]	3×2^{19}	2^{12}	2^{14}	2^{20}
Loihi [4]	2^{17}	2^{10}	2^{12}	3×2^{15}

TABLE IV: Workloads

Model	**#Neurons**	**#Synapses**	**Model**	**#Neurons**	**#Synapses**
AlexNet	864K	908M	VGG19	16M	19B
CNN2M*	2.5M	1.5B	ResNet152	35M	11B
ResNet18	3.5M	1.7B	CNN3B*	3.2B	902B

* synthetic models.

minimizing the number of clusters is essential for reducing both spike traffic and neuromorphic core usage. This problem corresponds to a variant of the one-dimensional bin-packing problem, which is NP-hard.

Several approximation algorithms and metaheuristics have been proposed for bin packing [43], [44]. To maintain computational efficiency, we index neurons in the FC layer in descending order of their axon memory sizes, inspired by the First-Fit-Decreasing (FFD) strategy [44].

D. Binary Segmentation of Neuron Sequence

The neuron sequence generated using the proposed indexing rules ensures that adjacent neurons in the sequence share more pre-synaptic neurons. Consequently, segmenting the sequence and grouping neurons within each segment leads to improved spike-sharing, thereby reducing runtime spike packet traffic and neuromorphic core consumption.

To ensure that each cluster fits within the on-chip memory constraints of a neuromorphic core as presented in Eq. (2)(3)(4), we propose a binary-search-based segmentation method (*BS_Segment*). As shown in Algorithm 1, starting from the beginning of the sequence, the method identifies the first neuron where the cumulative memory demand exceeds the core capacity. The sequence is split at this position, and the process is repeated on the remaining part until all neurons are assigned to clusters. *BS_Segment* ensures that at least one of the on-chip memory types—*Dendrite Memory*, *Axon Memory*, or *Neuron Memory*—is fully utilized in each cluster.

IV. Experimental Setup

A. Experimental Environment and Workloads

The experiments were conducted with the configurations of representative neuromorphic systems [3], [4] as illustrated in Table III. We have built a simulator in Python and conducted the experiments on an Ubuntu 20.04.6 LTS workstation with 40 CPU cores (Intel(R) Xeon(R) Silver 4210R CPU@2.40GHz) and 256GB memory. We have modified SNNToolbox [45] to convert ANN models in Keras into SNN models. Table IV lists the SNN models at different scales (number of neurons and synapses). We employ the method from [40] for mapping PN onto hardware due to its state-of-the-art performance and suitability for large-scale models.

B. Metrics

Due to the inaccessibility of neuromorphic hardware, we reference the evaluation methodology of [40] to evaluate the runtime performance of neuromorphic systems, covering the network interconnect energy and latency of spike communications. The metrics are briefly introduced in the following for clarification.

1) Spike Traffic

$$ST = \frac{\sum_{(i,j)\in E_{conn}} \sum_{n\in V_P^i \bigcap \mathcal{N}_{pre}^j} s_n}{|E_{syn}|} \tag{9}$$

s_n denotes the firing rate of neuron n. ST represents the overall traffic of spike packets normalized for evaluation. In the ideal situation, only one spike packet instead of duplicate traffic is transmitted when a neuron is activated, i.e., the *optimal bound* of ST:

TABLE V: Evaluation Parameters

Parameter	EN_p	EN_r	LA_p	LA_r
Value	1	0.1	$1e^{-4}$	$1e^{-5}$

$$ST_{ob} = \frac{\sum_{V_P^i \in V_{cluster}} \sum_{n\in V_P^i} s_n}{|E_{syn}|} \tag{10}$$

2) Core Consumption

$$NC = |V_{cluster}| \tag{11}$$

NC denotes the number of neuromorphic cores necessary for SNN simulation.

3) Spike Packet Latency

$$LT(P^i, P^j) = Lat(P^i, P^j) \times (\|P^i - P^j\|_1 \times LA_p + (\|P^i - P^j\|_1 + 1) \times LA_r) \tag{12}$$

$||\cdot||_1$ indicates the L1 norm. $LT(P^i, P^j)$ models the latency of transmitting a packet from core P^i to core P^j, which consists of two parts: the latency for NoC path transmitting and router forwarding. And LA_p and LA_r are the respective scaling factors. $Lat(P^i, P^j)$ represents the unit-hop-latency of all possible paths from P^i to P^j. The definition of $Lat(P^i, P^j)$ is shown in the Appendix.

4) Interconnect Energy

$$EC = \sum_{(P^i,P^j)\in E_{conn}} (w_{pkt}^{ij} \times \|P^i - P^j\|_1 \times EN_p + w_{pkt}^{ij} \times (\|P^i - P^j\|_1 + 1) \times EN_r) \tag{13}$$

EC accounts for the NoC interconnect energy for spike traffic transmission. EN_p and EN_r stand for the unit-hop energy of link transmission and router forwarding, respectively.

We set the evaluation parameters in LT (12) and EC (13) to the unit values listed in Table V for simplified implementation.

C. Approaches for Comparison

We evaluated the following approaches.

- **EdgeMap** [19]: EdgeMap iterates over neurons in SNN. For each neuron, it calculates an intricate cost function on every cluster and selects the one with the lowest cost. If no cluster is found, a new cluster is created.
- **DFSynthesizer** (referred as **DFSy** for convenience) [18]: DFSy also iterates over neurons in SNN and searches for the cluster with the highest resource utilization to pack the current neuron into. When there is no cluster that can accommodate the neuron, a new cluster is created.
- **SNNcut-woS (SNNcut without spike-sharing)**: It is for comparison with the proposed method. *SNNcut-woS* indexes neurons in all layers in default order instead of utilizing the proposed indexing rules.
- **SNNcut**: The proposed method that partitions SNN in a layer-wise manner, indexes neurons according to the rules in Section III, and segments the sequence with *BS_Segment* to produce the partitioned network.

V. Results and Analysis

A. Performance Evaluation

1) Spike Traffic

As shown in Table VI, our method reduces spike traffic by 98.97% on average, while the best of existing methods achieves a 74.98% average reduction. The reasons are discussed as follows. Firstly, existing greedy-based methods are primarily designed for models with limited scales (e.g., the largest model in [19] is no larger than AlexNet and is way much smaller than workloads in our experiments.). As the model scale increases, their performance degrades because of

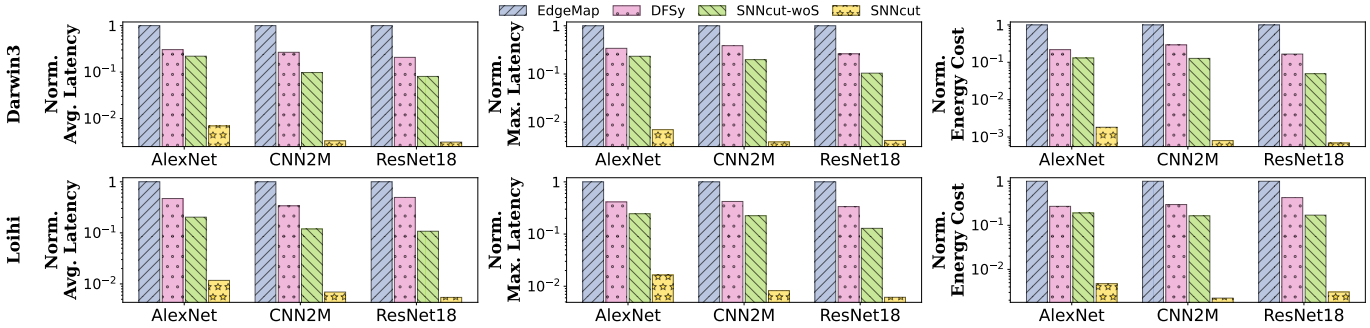


Fig. 10: Results of average/maximum latency (first column and mid column) and energy consumption (last column) under different hardware configurations (different rows). The values are normalized ratios to *EdgeMap*.

TABLE VI: Results of Spike Traffic(ST) ($\times 10^{-2}$)

Hardware	Model	*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
Darwin3	AlexNet	25.73	10.01	6.96	**0.3**
	CNN2M	42.32	22.31	15.32	**0.32**
	ResNet18	57.61	21.01	10.91	**0.43**
Loihi	AlexNet	71.51	28.61	25.21	**2.11**
	CNN2M	60.42	31.35	27.05	**1.31**
	ResNet18	63.13	36.82	29.58	**1.68**

TABLE VII: Results of Core Consumption(NC)

Hardware	Model	*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
Darwin3	AlexNet	14,362	5,602	4,268	**693**
	CNN2M	37,901	20,128	14,366	**1,161**
	ResNet18	59,391	21,782	11,931	**1,531**
Loihi	AlexNet	137,425	63,787	76,574	**10,009**
	CNN2M	226,113	113,741	106,026	**12,645**
	ResNet18	280,740	153,041	140,235	**16,544**

TABLE VIII: Scalability Experiments - Results of Spike Traffic(ST) ($\times 10^{-2}$)

Model	*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
VGG19	36.41	28.00	26.51	**0.31**
ResNet152	36.2	38.83	30.54	**1.00**
CNN3B	28.04	28.03	27.95	**0.54**

the explosion of solution space. Secondly, the cost function design is simplistic and lacks specificity for optimizations toward communication reduction, providing limited guidance for the searching process. We observed that under certain cost functions [18], existing methods tend to scatter neurons that could have shared spike packets into different clusters. In contrast, the superiority of SNNcut stems from its emphasis on reducing spike traffic through the spike-sharing mechanism.

2) Cores Consumption

Table VII shows that the core consumption of SNNcut is only $5.7\%\sim15.7\%$ of those required by the SOTA of existing approaches. As discussed in Section III-A, SNNcut saves core consumption since the cores that originally suffered from insufficient axon memory space could contain more neurons with reduced axon memory size.

3) Latency and Energy Consumption

Fig. 10 presents the evaluation of partitioned SNNs under different configurations in terms of average latency, max latency, and energy consumption, respectively. The Y-axes record the normalized values relative to the results of SNNcut. Based on the above discussions, SNNcut significantly reduces spike traffic and core consumption. The former promises fewer NoC spike packets, and the latter allows for mapping the connected cores to hardware positions with shorter distances [40]. With fewer spike packets traveling shorter distances, it is straightforward that SNNcut demonstrates better latency and interconnect energy cost.

4) Scalability

To verify the scalability of SNNcut, we conducted experiments on SNNs with $16M \sim 3.2B$ neurons and $19B \sim 902B$ synapses that are much larger than the workloads in existing works. We include CNN3B as an edge case, on which the number of partitioned clusters using SNNcut approaches the maximum core count of the existing neuromorphic system [3] as far as we are aware. In practice, it takes more than 100 hours for DFSynthesizer and EdgeMap to process those SNNs. Thus, we stop them early and continue partitioning the rest of the layers using SNNcut-woS. The results are presented in Table VIII, Table IX, and Fig. 11. SNNcut maintains a good performance in spike traffic, core consumption, latency, and energy consumption, even on CNN3B.

TABLE IX: Scalability Experiments - Results of Core Consumption(NC)

Model	*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
VGG19	457,659	333,913	321,580	**13,433**
ResNet152	237,048	253,637	208,315	**15,208**
CNN3B	16,000,334	15,993,447	15,950,202	**1033,301**

TABLE X: Impact of Hardware Configurations on Spike Traffic(ST) ($\times 10^{-2}$)

Config		*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
Axon (N_{axon})	4,096	56.14	17.80	12.72	**0.31**
	7,168	52.63	13.84	10.43	**0.30**
	10,240	39.86	11.82	8.82	**0.30**
	13,312	34.58	11.13	7.64	**0.30**
Dendrite (N_{den})	131,072	38.23	19.51	16.39	**2.06**
	491,520	37.62	14.23	11.75	**0.65**
	851,968	33.71	12.23	9.93	**0.44**
	1,212,416	29.75	11.13	8.41	**0.35**
Neuron (N_{neu})	1,024	25.73	10.08	7.56	**0.32**
	1,792	25.73	10.07	7.50	**0.31**
	2,560	25.73	10.06	7.47	**0.30**
	3,328	25.73	10.03	7.22	**0.30**
Default (Darwin3)		25.73	10.01	6.96	**0.30**

B. Impact of Hardware Configurations

We systematically evaluate SNN runtime performance under three key neuromorphic core configurations: axon capacity (N_{axon}), dendrite capacity (N_{den}), and neuron capacity (N_{neu}). The experiment adopts a single-variable methodology, varying one parameter while keeping others fixed, to assess four partitioning algorithms on AlexNet. Fig. 12, Table X, and Table XI illustrate the performance across different configurations. The results reveal that SNNcut demonstrates superior adaptability, significantly outperforming baseline methods across all hardware configurations. Another observation is that, like most SNN partitioning methods, SNNcut favors abundant memory resources. It is straightforward since the abundant memory space can contain more neurons and more synapses, such that inter-core spike traffic can be decreased with spike-sharing by clustering more neurons with common pre-synaptic connections together.

C. Spike-Sharing-Constrained Scenarios Analysis

A potential degradation case occurs when static constraints (Dendrite or Neuron Memory) become the bottleneck, restrict-

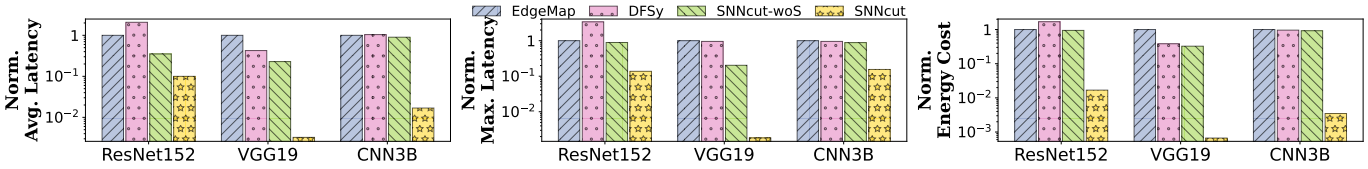


Fig. 11: Results of network scalability experiments. The values are normalized ratios to *EdgeMap*.

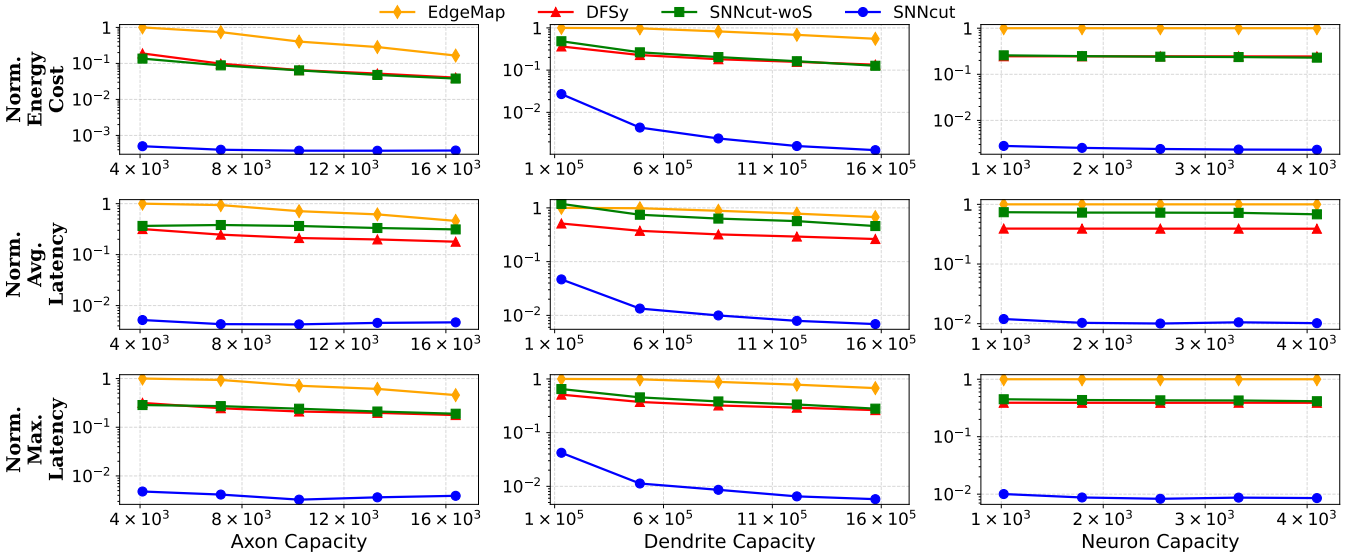


Fig. 12: Impact of hardware configurations. The first row shows the impact of axon capacity (ratio to *EdgeMap* with axon capacity as 2^{12}). The second row shows the impact of dendrite capacity (ratio to *EdgeMap* with dendrite capacity as 2^{17}). The last row shows the impact of neuron capacity (ratio to *EdgeMap* with neuron capacity as 2^{10}). The configurations other than the varying item are set according to the Darwin3 [3] in Table III.

TABLE XI: Impact of Hardware Configurations on Core Consumption(NC)

Config		*EdgeMap*	*DFSy*	*SNNcut-woS*	*SNNcut*
Axon (N_{axon})	4,096	110,673	39,742	33,135	**1,005**
	7,168	69,363	17,629	14,130	**793**
	10,240	35,720	10,548	8,307	**737**
	13,312	23,766	7,645	5,655	**711**
Dendrite (N_{den})	131,072	21,363	10,869	13,949	**7,685**
	491,520	21,116	7,976	7,810	**2,072**
	851,968	18,863	6,838	6,266	**1,221**
	1,212,416	16,554	6,176	5,183	**875**
Neuron (N_{neu})	1,024	14,362	5,657	4,803	**1,045**
	1,792	14,362	5,640	4,652	**834**
	2,560	14,362	5,632	4,592	**756**
	3,328	14,362	5,616	4,425	**718**
Default (Darwin3)		14,362	5,602	4,268	**693**

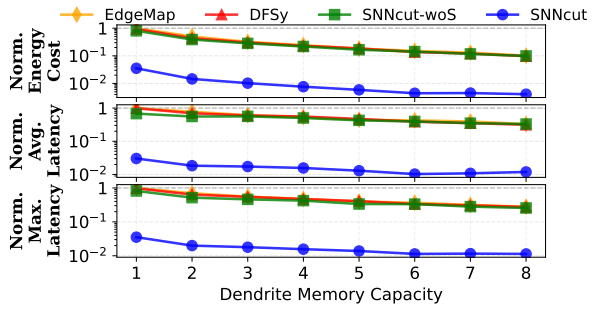


Fig. 13: Sensitivity experiment on Dendrite Memory bottleneck. The values (lower is better) are normalized to EdgeMap at 1024 dendrite memory capacity.

ing the aggregation of post-synaptic neurons, thus limiting the benefits of spike-sharing. We conduct experiments to analyze the advantages of SNNcut in these spike-sharing-constrained scenarios.

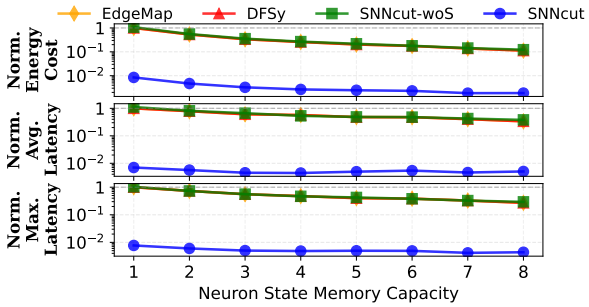


Fig. 14: Sensitivity experiment on Neuron State Memory bottleneck. The values (lower is better) are normalized to EdgeMap at 128 neuron memory capacity.

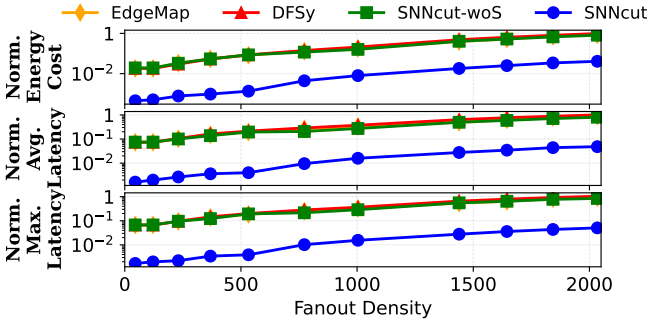


Fig. 15: Sensitivity experiment on fanout density. The values (lower is better) are normalized to EdgeMap at ~ 2000 fanout density. The y-axis is log-scaled.

To quantify this, we have expanded our analysis. In section V-B, the sensitivity analyses for dendrite memory capacity (from 1×10^5 to 16×10^5) and neuron memory capacity (from 1×10^3 to 4×10^3) were based on configurations from

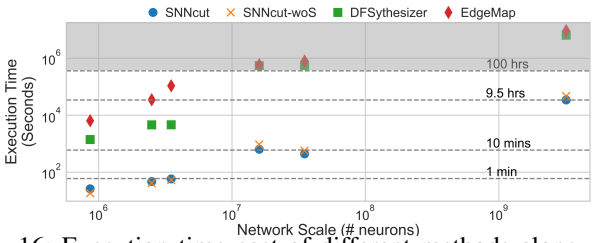


Fig. 16: Execution time cost of different methods along with network scale.

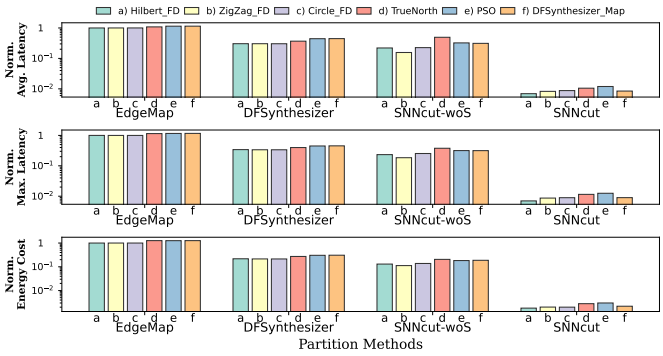


Fig. 17: Energy consumption and latency under different combinations of partitioning and mapping strategies. Y-axes show the ratio values relative to the results of *EdgeMap* with *Hilbert_FD* mapping strategy.

actual neuromorphic hardware [3]–[5]. We now focus on more constrained scenarios where Dendrite memory capacity ranges from 1024 to 8192, and Neuron memory capacity ranges from 128 to 1024.

Dendrite Memory Capacity: We conduct a sensitivity analysis on Dendrite memory capacity. The results are presented in Fig. 13. As dendrite capacity decreases, it becomes the primary bottleneck, which limits the density of post-synaptic neuron aggregation and thus restricts the benefits derived from spike-sharing. However, even at a 1024 capacity, SNNcut still demonstrated a 33× (28.3×) speedup on average (maximum) latency and 28.5× energy efficiency over SOTA methods.

Neuron Memory Capacity: We conducted a similar analysis for the Neuron memory capacity (Fig. 14). We observed an identical trend. As neuron memory becomes the bottleneck, it limits the neuron density of cores, which constrains the potential gains from spike-sharing. Nevertheless, even at the 128-neuron limit, SNNcut maintained a 142× (129.5×) speedup on average latency (maximum latency) and 118.1× energy efficiency.

Fanout: We further analyzed the impact of high fan-out, as this directly strains Dendrite Memory. While the fan-out of typical SNNs is approximately 1000 [6], we configured the fan-out to range from 10 to 2000 to explore the impact of high fan-out on SNNcut. The results in Fig. 15 show that as fan-out increases, the performance of all methods, including SNNcut, degrades. This is due to two factors: 1) high fan-out inherently limits spike-sharing opportunities, and 2) it unavoidably increases the total volume of spike communication. However, even at a fanout of ∼ 2000, SNNcut still demonstrated a 18.2× (17.8×) speedup on average (maximum) latency and 20.3× energy efficiency over SOTA methods.

In summary, the experiments confirm that while static memory bottlenecks do restrict the gains of spike-sharing, our method remains highly effective and significantly outperforms existing SOTA approaches even in these highly constrained scenarios.

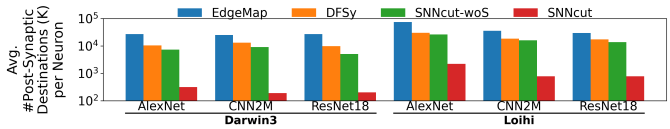


Fig. 18: The average number of post-synaptic destinations (or cores) per neuron.

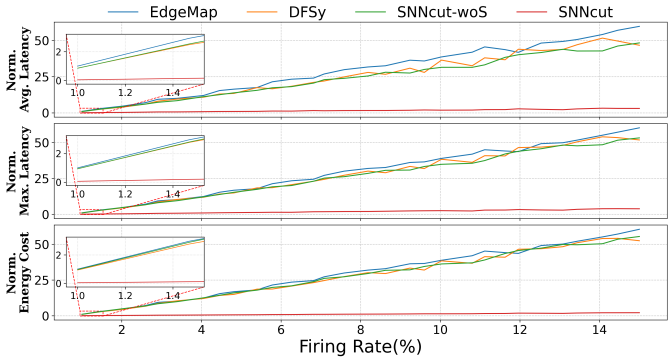


Fig. 19: Sensitivity experiment on firing rate. The values (lower is better) are normalized to EdgeMap.

D. Results on Execution Time

Fig. 16 illustrates the elapsed time that the evaluated approaches take to accomplish the partitioning of SNNs with different scales (indicated by the number of neurons). In practice, some methods fail to finish within a reasonable time, and the corresponding data points in the grey area above 100 hours in Fig. 16 represent our estimates rather than actual measurements. These estimates are calculated by stopping the algorithms at 100 hours and dividing the elapsed time by the percentage of layers partitioned up to that point. Note that these estimates might underestimate the actual execution time because as the number of clusters increases during partitioning, methods like DFSynthesizer [18] and EdgeMap [19] tend to spend more time on the rest iterations due to a larger searching space.

The results indicate that SNNcut is significantly more efficient than existing methods. For models with no more than $3.5M$ neurons, existing methods require hours or tens of hours to finish SNN partitioning, while SNNcut accomplishes it in under 1 minute. For models with more than $16M$ neurons, existing approaches fail to provide a solution within 100 hours, while SNNcut takes approximately 10 minutes and requires about 9.5 hours for partitioning SNN with $3.2B$ neurons.

E. Sensitivity on Mapping Strategies

To validate that SNNcut maintains advantages when it is integrated with any mapping strategy to build the complete workflow of SNN deployment, we conducted the experiments with different mapping approaches. In addition to the three mapping strategies introduced in [40] (*Hilbert_FD*, *ZigZag_FD*, and *Circle_FD*), *PSO* [21] and two greedy algorithms [46] and [18] are employed to map the AlexNet [47] processed by different partitioning methods.

Fig. 17 illustrates the evaluations of latency and energy consumption of four partitioning methods, each with six mapping strategies. The performance variations brought by mapping strategies are relatively small compared with partitioning strategies. Even in the worst case, SNNcut, in combination with *PSO* mapping [21], still outperforms the other partitioning methods with any mapping approaches.

F. Sensitivity on Firing Rates (Spike Patterns)

We theoretically analyze and experimentally evaluate the performance of SNNcut to validate its advantages across diverse SNNs exhibiting different firing rates or spike patterns.

Traffic Modeling. SNNcut is a compile-time optimization that

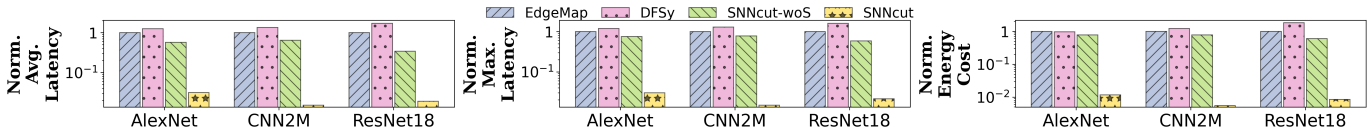


Fig. 20: Results on heterogeneous neuromorphic platforms. The values are normalized to EdgeMap.

strategically clusters neurons sharing common pre-synaptic neurons onto the same core. This strategy fundamentally reduces the number of NoC packets generated per neuron activation. We can model the total NoC spike traffic (T) as $T \propto \sum_i^{all\ neurons}(K_i \times \text{firing\_rate}_i)$, illustrated in Fig. 5, where K_i is the number of post-synaptic destinations (or cores) for neuron i. Our analysis in Fig. 18 confirmed that existing methods result in a high scatter ($K_i = M$), while SNNcut aims to reduce this to $K_i = N$, with $N < M$. This model theoretically guarantees that SNNcut has lower traffic ($N \times \text{firing\_rate} < M \times \text{firing\_rate}$) for any firing rate > 0.

Experimental Evaluation. To verify this, we conducted a new sensitivity analysis on the spike firing rate, setting the range covering typical firing rates of practical SNNs [48]. SNNcut achieves a $16.2\times$ ($14.1\times$) speedup on average (maximum) latency and $26\times$ energy efficiency improvement across all firing rates. The results (Fig. 19) demonstrate two key points: 1) Even at a very low firing rate ($\sim 1\%$), SNNcut still achieves a $17.7\times$ ($15\times$) speedup on average (maximum) latency and a $27.5\times$ energy efficiency improvement over SOTA methods, confirming its effectiveness in low firing rate scenarios. 2) As the firing rate increases, SNNcut's performance degrades at a significantly slower rate than SOTA methods. This experimentally validates the above traffic modeling, confirming that SNNcut's reduced packet generation per spike ($N < M$) provides a compounding benefit as spike activity increases.

In summary, the traffic modeling and experimental evaluation collectively demonstrate that: 1) SNNcut holds advantages in high-firing-rate scenarios, as its performance degrades more slowly with increasing activity; and 2) SNNcut's advantage at low firing rates is guaranteed, as it fundamentally generates fewer NoC spike packets per neuron activation.

G. Evaluation on Heterogeneous Neuromorphic Platforms

For scenarios where cores share the same architecture but have different on-chip memory capacities, SNNcut can be naturally generalized by enhancing *BS_Segment* (Algorithm 2) to become aware of the available counts of different core types, thereby dynamically generating clusters of varying sizes (all types of memories) to match this hardware heterogeneity.

We evaluated SNNcut's performance in heterogeneous neuromorphic platforms containing multiple neuromorphic hardware. Configurations mirrored mainstream chip specifications, including Loihi, Loihi2, SpiNNaker2, TrueNorth, BrainScaleS2, Darwin3 [3]–[5], [7], [9], [49]. The partitioning algorithms dynamically adjust on-chip memory constraints based on the available quantity of each core type. As shown in Fig. 20, SNNcut achieved an $49.9\times$ ($49.3\times$) speedup on average (maximum) latency and a $128\times$ energy efficiency improvement over SOTA methods on the heterogeneous neuromorphic platforms.

H. Evaluation on Real-World Hardware Platform

To validate our analytical metrics, we have added experiments on the Darwin3 hardware platform. We deployed the partitioned AlexNet [47] (using SNNcut and baseline methods) onto the hardware and measured the actual runtime latency and energy consumption.

The hardware results, presented in Table XII, corroborate the findings from our analytical model. SNNcut consistently demonstrates advantages. This real-world improvement is attributed to: 1) a drastic reduction in inter-core communication costs, achieved by maximizing spike-sharing, and 2) a reduced number of active cores, which is a direct consequence of our efficient, memory-aware clustering.

Method	# Cores	Time (ms) (Darwin3@$400MHz$,$1.8W$)
EdgeMap	14362	2688651
DFSy	5602	816595
SNNcut-woS	4268	618389
SNNcut	**693**	**23929**

TABLE XII: Real-world deployment results on the Darwin3 hardware platform. This table presents the measured latency for the partitioned AlexNet model. The results validate SNNcut's real-world performance advantage, which corroborates the findings from our analytical model.

VI. Discussion

A. Distinction from Runtime Re-partitioning Methods

The compile-time optimization provided by SNNcut and runtime re-partitioning strategies [50] (which adapt to dynamic spike patterns during learning) are **parallel** and **complementary**: SNNcut focuses on providing a one-time SNN partitioning based on SNN topology at compile-time; A runtime adaptation would aim to further refine the partition based on live spiking activity. For the latter techniques, SNNcut provides two key benefits:

1) It serves as an excellent initial partition, offering a high-quality starting point that vastly reduces the search space and potential overhead for any subsequent runtime optimizer.

2) SNNcut's layer-wise partitioning localizes re-partitioning to affected layers, dramatically reducing runtime adaptation costs by avoiding a full-network re-evaluation.

SNNcut's primary focus is to provide the best possible partition under a one-time compile-time cost, while also ensuring this partition is highly robust across dynamic spike patterns. The sensitivity analysis on firing rates (Fig. 19) validates this, demonstrating that SNNcut maintains its advantages over SOTA methods across diverse spike patterns.

B. Distinction from Packet Coalescing and Packet Transmission Scheduling

SNNcut is neither "packet coalescing" [51]–[53], which bundles multiple spikes into one packet, nor "packet transmission scheduling" [54], which delays the transmission time of spike packets; therefore, it introduces neither additional latency nor congestion bottlenecks. Fundamentally, SNNcut is a compile-time optimization that reduces NoC communication load without altering the runtime behavior of the NoC or the neuromorphic hardware.

VII. Conclusion

This paper presents SNNcut, an efficient partitioning method for large-scale SNNs on neuromorphic systems that significantly reduces runtime spike traffic and core consumption by innovatively utilizing spike-sharing. SNNcut offers a scalable solution for deploying large-scale SNNs on neuromorphic systems, maintaining high performance and efficiency as model scale increases. Extensive experimental results demonstrate the superiority of SNNcut over existing approaches in terms of task runtime performance and computational efficiency. In future work, we will focus on refining more partitioning rules and exploring additional optimization techniques to enhance the applicability and performance of

our method across diverse neuromorphic architectures and applications.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (No. 2025YFG0100700). 10.13039/501100001809-National Natural Science Foundation of China (**Grant Number**: 61925603 and 62172361). Major Projects of Zhejiang Province (**Grant Number**: LD24F02001)

APPENDIX

DEFINITION OF $Lat(P^i, P^j)$

Denoting $Rec(P^i, P^j)$ as the rectangle region with P^i and P^j as the two endpoints of the diagonal, $Lat(P^i, P^j)$ is defined as follows:

$$Lat(P^i, P^j) = \frac{\sum_{P \in Rec(P^i, P^j)} Con(P)}{size(Rec(P^i, P^j))} \tag{14}$$

where $size(Rec)$ gives the area of rectangle Rec and $Con(P)$ is the function that computes the congestion on the core at position P. The definition of $Con(P)$ is shown in Algorithm 3.

Algorithm 3: Function $Con(P)$, spike traffic of neuromorphic core at $P : (x, y)$

Input: $P : (x, y)$, $MN(V_{core}, E_{conn}, W_{pkt})$
Output: $Con(P)$

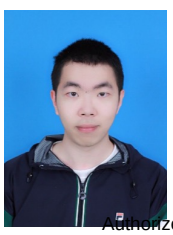
Define array A with the same shape as the minimum bounding rectangle of V_{core};
for *point* P^i *in* A **do**
    $A[P^i] = 0$;
for $e(P_s, P_d)$ *in* E_{conn} *and weight* w *in* W_{pkt} **do**
    $G = [G_1, G_2, ..., G_M] \leftarrow$ Group all cores within $Rec(P_s, P_d)$ according to their *Manhattan Distance* to P_s;
    for $G_m \in G$: **do**
        for *core* P^i *in* G_m : **do**
            $A[P^i] = A[P^i] + \frac{w}{|G_m|}$;
Return $A[P^i_{core}]$;

REFERENCES

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
[2] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.
[3] D. Ma, X. Jin, S. Sun, Y. Li, X. Wu, Y. Hu, F. Yang, H. Tang, X. Zhu, P. Lin *et al.*, "Darwin3: a large-scale neuromorphic chip with a novel isa and on-chip learning," *National Science Review*, vol. 11, no. 5, p. nwae102, 2024.
[4] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.
[5] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer, and M. Davies, "Efficient neuromorphic signal processing with loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2021, pp. 254–259.
[6] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.
[7] S. Höppner, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, M. Stolba, F. Kelber, B. Vogginger, F. Neumärker, G. Ellguth *et al.*, "The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing," *arXiv preprint arXiv:2103.08392*, 2021.
[8] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He *et al.*, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.
[9] M. V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W. P. Risk, J. Kusnitz, C. O. Otero, T. K. Nayak, R. Appuswamy *et al.*, "Truenorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.
[10] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, "The brainscales-2 accelerated neuromorphic system with hybrid plasticity," *Frontiers in Neuroscience*, vol. 16, p. 795876, 2022.
[11] X. Liu, W. Wen, X. Qian, H. Li, and Y. Chen, "Neu-noc: A high-efficient interconnection network for accelerated neuromorphic systems," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 141–146.
[12] C.-K. Lin, A. Wild, G. N. Chinya, T.-H. Lin, M. Davies, and H. Wang, "Mapping spiking neural networks onto a manycore neuromorphic architecture," *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 78–89, 2018.
[13] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.
[14] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Proceedings of the 9th conference on Computing Frontiers*, 2012, pp. 183–192.
[15] S. Li, S. Guo, L. Zhang, Z. Kang, S. Wang, W. Shi, L. Wang, and W. Xu, "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*, 2020, pp. 9–14.
[16] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *European conference on computer vision*. Springer, 2020, pp. 388–404.
[17] X. Du, M. Wang, Z. Lu, Q. Duan, Y. Liu, J. Feng, and H. Wang, "Hrcm: A hierarchical regularizing mechanism for sparse and imbalanced communication in whole human brain simulations," *IEEE Transactions on Parallel and Distributed Systems*, 2024.
[18] S. Song, H. Chong, A. Balaji, A. Das, J. Shackleford, and N. Kandasamy, "Dfsynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 3, pp. 1–35, 2022.
[19] J. Xue, L. Xie, F. Chen, L. Wu, Q. Tian, Y. Zhou, R. Ying, and P. Liu, "Edgemap: an optimized mapping toolchain for spiking neural network in edge computing," *Sensors*, vol. 23, no. 14, p. 6548, 2023.
[20] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2019.
[21] S. Song, M. L. Varshika, A. Das, and N. Kandasamy, "A design flow for mapping spiking neural networks to many-core neuromorphic hardware," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.
[22] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.
[23] Y. Hu, L. Deng, Y. Wu, M. Yao, and G. Li, "Advancing spiking neural networks toward deep residual learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.
[24] Z. Huang, X. Shi, Z. Hao, T. Bu, J. Ding, Z. Yu, and T. Huang, "Towards high-performance spiking transformers from ann to snn conversion," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 10 688–10 697.
[25] K. You, Z. Xu, C. Nie, Z. Deng, Q. Guo, X. Wang, and Z. He, "Spikezip-tf: conversion is all you need for transformer-based snn," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, pp. 57 367–57 383.
[26] T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang, "Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks," *arXiv preprint arXiv:2303.04347*, 2023.
[27] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming graph partitioning for massive scale graphs," in *Proceedings of the 7th ACM international conference on Web search and data mining*, 2014, pp. 333–342.
[28] Y. Akhremtsev, P. Sanders, and C. Schulz, "High-quality shared-memory graph partitioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2710–2722, 2020.
[29] L. Wang, Y. Xiao, B. Shao, and H. Wang, "How to partition a billion-node graph," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 568–579.
[30] F. Bourse, M. Lelarge, and M. Vojnovic, "Balanced graph edge partition," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1456–1465.
[31] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1222–1230.
[32] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.
[33] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs," *IBM Journal of Research and Development*, vol. 17, no. 5, pp. 420–425, 1973.
[34] M. Holtgrewe, P. Sanders, and C. Schulz, "Engineering a scalable high quality graph partitioner," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–12.
[35] S. Gong, Y. Zhang, and G. Yu, "Hbp: Hotness balanced partition for prioritized iterative graph computations," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1942–1945.

[36] C. Martella, D. Logothetis, A. Loukas, and G. Siganos, "Spinner: Scalable graph partitioning in the cloud," in *2017 IEEE 33rd international conference on data engineering (ICDE)*. Ieee, 2017, pp. 1083–1094.
[37] W. Fan, M. Liu, P. Lu, and Q. Yin, "Graph algorithms with partition transparency," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1554–1566, 2021.
[38] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," in *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, 1996, pp. 35–es.
[39] ——, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
[40] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "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, Volume 3*, 2023, pp. 419–432.
[41] M. Bader, *Space-filling curves: an introduction with applications in scientific computing*. Springer Science & Business Media, 2012, vol. 9.
[42] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.
[43] C. Munien, S. Mahabeer, E. Dzitiro, S. Singh, S. Zungu, and A. E.-S. Ezugwu, "Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study," *IEEE Access*, vol. 8, pp. 227 438–227 465, 2020.
[44] C. Munien and A. E. Ezugwu, "Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 636–663, 2021.
[45] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017.
[46] J. Sawada, F. Akopyan, A. S. Cassidy, B. Taba, M. V. Debole, P. Datta, R. Alvarez-Icaza, A. Amir, J. V. Arthur, A. Andreopoulos *et al.*, "Truenorth ecosystem for brain-inspired computing: scalable systems, software, and applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 130–141.
[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
[48] J.-J. Lee, W. Zhang, and P. Li, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 317–330.
[49] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, "The brainscales-2 accelerated neuromorphic system with hybrid plasticity," *CoRR*, vol. abs/2201.11063, 2022. [Online]. Available: https://arxiv.org/abs/2201.11063
[50] M. Rizk, K. J. Martin, and J.-P. Diguet, "Run-time remapping algorithm of dataflow actors on noc-based heterogeneous mpsocs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3959–3976, 2022.
[51] X. Zhao, D. Kaeli, Z. Wang, L. Eeckhout *et al.*, "Intra-cluster coalescing and distributed-block scheduling to reduce gpu noc pressure," *Ieee Transactions on Computers*, vol. 68, no. 7, pp. 1064–1076, 2019.
[52] K. H. Kim, R. Boyapati, J. Huang, Y. Jin, K. H. Yum, and E. J. Kim, "Packet coalescing exploiting data redundancy in gpgpu architectures," in *Proceedings of the International Conference on Supercomputing*, 2017, pp. 1–10.
[53] J. Kloosterman, J. Beaumont, M. Wollman, A. Sethia, R. Dreslinski, T. Mudge, and S. Mahlke, "Warppool: Sharing requests with inter-warp coalescing for throughput processors," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 433–444.
[54] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *25 years of the international symposia on Computer architecture (selected papers)*, 1998, pp. 195–201.

Qinghui Xing is currently pursuing a PhD degree in School of Computer Science and Technology, Zhejiang University. His research interests focus on neuromorphic computing and computer architecture.



Ouwen Jin is currently a doctoral student at the College of Computer Science and Technology, Zhejiang University. His research interests include brain-inspired computing and neuromorphic computing hardware architectures. His work focuses on optimizing the compilation, deployment, and execution efficiency of spiking neural networks on neuromorphic hardware. He has published as the first author at ASPLOS.

Zhuo Chen is currently a Ph.D. student in the School of Computer Science at Zhejiang University. His research focuses on neuromorphic computing systems, with particular emphasis on the design, implementation, and application of neuromorphic chips. His work aims to bridge the gap between biological neural processing and artificial intelligence hardware through brain-inspired computing architectures.

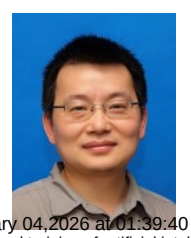
Xin Du is an Assistant Professor at School of Software Technology, Zhejiang University, China. He received a Ph.D. computer science degree from Fudan University in 2024. His research interests include service computing, distributed systems, and brain-inspired computing. He has published several papers in flagship conferences and journals, including IEEE ICWS, the IEEE Transactions on Parallel and Distributed Systems, etc. He has received the Best Student Paper Award of IEEE ICWS 2020 and IEEE ICWS 2023.

Ming Zhang is a software engineer with College of Computer Science and Technology, Zhejiang University, China. He received his Ph.D. in computer science from Zhejiang University in 2019. His research interests focus on building basic software tools for neuromorphic computers.

Shuiguang Deng (Senior Member, IEEE) received the B.S. and PhD degrees in computer science from Zhejiang University, China, in 2002 and 2007, respectively. He is currently a full professor at the College of Computer Science and Technology, Zhejiang University. He previously worked with the Massachusetts Institute of Technology in 2014 and Stanford University in 2015 as a visiting scholar. His research interests include edge computing, service computing, cloud computing, and business process management. He serves for the journal IEEE Transactions on Services Computing, Knowledge and Information Systems, Computing, and IET Cyber-Physical Systems: Theory & Applications as an associate editor.

Shuibing He is a Professor with the College of Computer Science and Technology at Zhejiang University (ZJU), China, where he leads the Intelligent Storage and Computing Systems (ISCS) Laboratory. He also serves as the Vice President of Zhejiang Lab and the Deputy Director of the Zhejiang Key Laboratory of Big Data Intelligent Computing. His research interests include storage systems, intelligent computing, computer architecture, and high-performance computing. Dr. He serves as an Associate Editor for the IEEE Transactions on Computers (TC) and previously held the same role for the IEEE Transactions on Parallel and Distributed Systems (TPDS) from 2018 to 2022. Additionally, he has served as the Program Chair of NAS 2024, General Chair of ChinaSys 2024, and a Program Committee Member for conferences such as ICDCS, SRDS, ICPP, IPDPS, and CLUSTER.

Ying Li received the B.S., M.S., and Ph.D. degrees in computer science from Zhejiang University, China, in 1994, 1997, and 2000, respectively. He is currently an Associate Professor at the College of Computer Science, Zhejiang University. He is leading several research projects supported by the National Natural Science Foundation of China. His research interests include service computing, process mining, and compiler technology.



Gang Pan received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor at the College of Computer Science and Technology, Zhejiang University. He has published more than 100 refereed papers and was a visiting scholar at the University of California, Los Angeles, from 2007 to 2008. His research interests include pervasive computing, computer vision, and pattern recognition.