Modeling and Optimizing Performance Bottlenecks for Neuromorphic Accelerators

Jason Yik*, Walter Gallego Gomez†, Andrew Cheng*, Benedetto Leto†, Alessandro Pierro‡§, Noah Pacik-Nelson¶‖, Korneel Van den Berghe**, Vittorio Fra†, Andreea Danielescu¶††, Gianvito Urgese†, Vijay Janapa Reddi*

*Harvard University †Politecnico di Torino ‡Intel §LMU Munich ¶Accenture Labs ‖BootLoop AI **TU Delft ††Wordly

***Abstract*—Neuromorphic accelerators offer promising platforms for machine learning (ML) inference by leveraging event-driven, spatially-expanded architectures that naturally exploit unstructured sparsity through co-located memory and compute. However, their unique architectural characteristics create performance dynamics that differ fundamentally from conventional accelerators. Existing workload optimization approaches for neuromorphic accelerators rely on aggregate network-wide sparsity and operation counting, but the extent to which these metrics actually improve deployed performance remains unknown. This paper presents the first comprehensive performance bound and bottleneck analysis of neuromorphic accelerators, revealing the shortcomings of the conventional metrics and offering an understanding of what facets matter for workload performance. We present both theoretical analytical modeling and extensive empirical characterization of three real neuromorphic accelerators: Brainchip AKD1000, Synsense Speck, and Intel Loihi 2. From these, we establish three distinct accelerator bottleneck states, memory-bound, compute-bound, and traffic-bound, and identify which workload configuration features are likely to exhibit these bottleneck states. We synthesize all of our insights into the floorline performance model, a visual model that identifies performance bounds and informs how to optimize a given workload, based on its position on the model. Finally, we present an optimization methodology that combines sparsity-aware training with floorline-informed partitioning. Our methodology achieves substantial performance improvements at iso-accuracy: up to 3.86× runtime improvement and 3.38× energy reduction compared to prior manually-tuned configurations. This work lays the groundwork for system-level optimization of neuromorphic accelerators, and it provides architectural insights and a systematic optimization framework that enables principled performance engineering for this class of architectures.**

I. Introduction

Neuromorphic accelerators employ a distinct architectural approach for executing sparse neural network inference, characterized by spatially-expanded designs where each logical neuron maps to a dedicated physical compute unit on-chip. This contrasts with conventional accelerators that time-multiplex logical neurons across shared arithmetic units [4], [24], [40], [41]. The neuromorphic approach co-locates memory with processing units (neurocores) and leverages event-driven execution to exploit activation sparsity without requiring structured sparsity patterns. Recent deployments have demonstrated substantial performance and energy benefits for specific sparse workloads compared to edge GPU baselines [1], [30], [38], [46], [62].

However, the unique architectural characteristics of neuromorphic accelerators create performance dynamics that are poorly understood. While recent algorithmic work has focused on optimizing neural networks for neuromorphic deployment [10], [32], [33], [38], [43], [44], [59], [60], these efforts rely on high-level performance proxies, primarily network-wide sparsity and aggregate operation counting, without testing how well such proxies actually inform deployed performance. This gap is critical: no systematic analysis has established which factors actually drive neuromorphic accelerator performance and how to optimize for them.

In this paper, we address this gap by presenting a "bound and bottleneck" analysis [25] of neuromorphic accelerator performance, mirroring methodology which produced the widely-used roofline performance model for conventional architectures [56]. Our analysis aims to answer three key questions about neuromorphic systems: (1) What core operations bound and bottleneck performance? (2) Which workload configurations result in different bounds and bottlenecks? (3) How does one optimize a given workload using an understanding of its bottleneck and performance bounds?

Our study proceeds by first using a simple analytical architecture and workload model to provide theoretical insights into how memory, compute, and traffic operations scale with workload sparsity and parallelization on a neuromorphic chip. Next, we verify and substantiate the insights by extensively profiling three real neuromorphic accelerators: the edge-focused Brainchip AKD1000 [7], the event-camera specialized Synsense Speck [28], and the research-oriented Intel Loihi 2 [20]. Based on the modeling and profiling insights, we synthesize the floorline model, an analog to the roofline model, which visually indicates the performance bounds of a neural network architecture and informs how to optimize any trained network instantiation.

Our analytical modeling and profiling reveal several important insights about neuromorphic accelerator performance. Most critically, we find that conventional network-wide performance proxies are insufficient for neuromorphic architectures due to neurocore-level load imbalance; instead, neurocore-aware metrics are necessary for understanding whether performance will improve. We also observe that current neuromorphic implementations show limited ability to exploit weight sparsity for convolutional networks (CNNs), suggesting that

recently proposed CNN weight pruning approaches [10], [43], [44] may require architectural modifications to be effective. For bounds and bottlenecks, we identify memory accesses during synaptic operations (synops) as the usual dominant workload cost, consistent with prior circuit-level analysis [11], [14], [52]. However, we importantly uncover that certain workloads can be bound by neuron computes or message traffic, necessitating different optimization approaches than the synops-bound state.

Based on these observations, we develop a two-stage optimization methodology that applies our performance model insights. The first stage co-optimizes network accuracy and sparsity during training to leverage the fundamental sparsity benefits in neuromorphic architectures. The second stage uses our architecture-aware floorline performance model to iteratively optimize neurocore partitioning and mapping. Applied to our three platforms at iso-accuracy operating points, this approach achieves notable performance improvements over baseline configurations: up to 4.29× runtime and 4.36× energy improvements from sparsity optimization, with additional gains of up to 1.83× runtime and 1.52× energy from architecture-aware optimization on Loihi 2. The combined two-stage optimization yields up to 3.86× runtime and 3.38× energy improvements compared to prior manually-tuned configurations for the studied workloads.

In summary, this work studies the performance bounds and bottlenecks of neuromorphic accelerators with the following contributions:

- We present analytical modeling to provide insights for how network sparsity and parallelization configurations affect memory, compute, and traffic bottlenecks.
- We empirically demonstrate that real neuromorphic accelerator workloads can be in any of the three bottleneck states, under configurations expected by the analytical model, and characterize the performance boundaries that result from the bottlenecks.
- We propose the floorline performance model, a visual model which synthesizes the relationships between performance bounds and bottlenecks, and informs how to understand and optimize a workload's performance given its location on the model.
- We develop and validate a generic two-stage optimization methodology that utilizes sparsity-aware training and floorline-informed partitioning, achieving up to 3.86× runtime and 3.38× energy improvements over prior configurations.

II. Background

This section provides the architectural background needed to understand neuromorphic accelerator performance characteristics and the unique challenges they present for performance modeling. In addition, we review related work and identify the research gaps that our paper uniquely addresses.

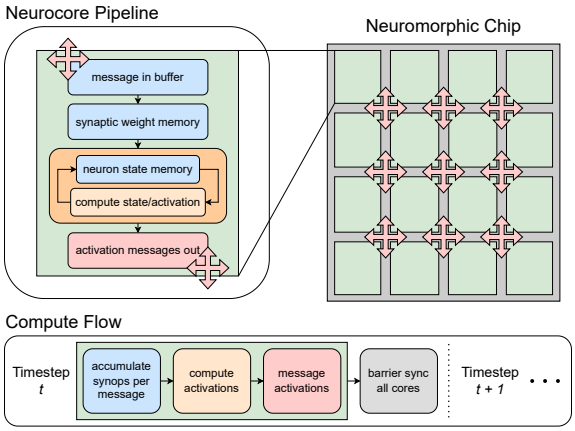


Fig. 1: The general macro-architecture and compute flow of neural network execution on neuromorphic accelerators. Blue components denote co-located memory, orange denotes compute, and red denotes inter-core communication via the NoC.

A. Neuromorphic Accelerator Architecture

Figure 1 illustrates the neuromorphic accelerator architecture, which is applicable across current designs [2], [7], [12], [15], [20], [28], [29], [31], [37], [53][1]. The architecture operates at two levels: the chip-level organization (right) is an array of **neurocores** connected via a network-on-chip (NoC), and each neurocore (left) uses a pipeline structure to process neural computations.

The execution pipeline of each **neurocore** contains three memory components, a compute stage, and message output stage. The **message in buffer** stores input activation messages from other neurocores, the **synaptic weight memory** holds the weights for neurons assigned to this neurocore, and the **neuron state memory** maintains persistent state for stateful neuron models. The compute stage generates new activations based on the inputs, which are then formatted as sparse **activation messages** and sent via the NoC to downstream neurocores for processing in the subsequent timestep.

The bottom of Figure 1 shows the temporal execution model where all neurocores operate in lockstep during discrete timesteps. Within each timestep, neurocores simultaneously: (1) accumulate the **synaptic operations (synops)** from each input message, (2) **compute activations**, (3) generate outgoing **activation messages**, and (4) participate in **barrier synchronization** before advancing to the next timestep. Synops for each message involves fetching weights, multiplying them by the input, and accumulating into a pre-activation for each neuron.

To map a neural network to the neuromorphic chip for inference, groups of neurons are logically **partitioned**, then physically **mapped** to a neurocore on the chip. For feedfor-

[1]Since the term 'neuromorphic' can be broadly applied, many other computing systems have been proposed and labeled under the term, such as compute-in-memory and crossbar array architectures [40], [41]. This study focuses on the macro-architecture shown in Figure 1.

ward ML networks, all neurons in a neurocore belong to the same algorithmic network layer.

The **timestep duration** is the time required for the slowest neurocore to complete its assigned computation, establishing the accelerator's throughput bottleneck. Since neurocores operate in parallel but barrier synchronize at each timestep, the maximum processing time across all active neurocores determines overall accelerator performance. This creates a load balancing challenge where uneven computational workload distribution across neurocores directly impacts accelerator throughput, regardless of total computational capacity.

The event-driven execution paradigm naturally exploits sparsity through two mechanisms. **Weight sparsity** reduces memory access and synop accumulation requirements within each neurocore, while **activation sparsity** reduces NoC message traffic and downstream synop requirements. Unlike conventional accelerators that may require structured sparsity, neuromorphic accelerators can natively leverage unstructured sparsity through their per-message weight fetching and activation message passing approaches, though our analysis reveals architecture-specific limitations for certain sparsity types.

The neuromorphic spatially expanded approach, where each logical neuron maps to a dedicated physical location in a neurocore [24], contrasts with conventional accelerators (GPUs, TPUs) that achieve parallelism through time-multiplexed execution across shared arithmetic units. Most critically for performance modeling, such exclusive use of on-chip co-located memory fundamentally alters the performance landscape compared to conventional accelerators. Unlike systems where off-chip memory bandwidth typically dominates performance, neuromorphic accelerators operate in a regime where on-chip memory access, local compute, and NoC communication costs are within the same order of magnitude [12], [52]. This creates a multi-dimensional performance space where the bottleneck can shift between intra-core synop memory access, intra-core activation computation, and inter-core communication traffic, depending on workload characteristics and system utilization.

B. Related Work

Prior work in neuromorphic performance falls into two main categories: performance estimation and hardware-software optimization. Performance estimation efforts have primarily relied on simulation-based approaches, including architectural simulators for specific designs [6], [58] and micro-operation cost estimation through circuit-level simulation [14], [52], [54] or system micro-benchmarks [16], [36]. However, the architectural simulator approaches either focus on single systems or rely on simplified assumptions (e.g., fixed-duration timesteps), while the micro-operation cost estimates fail to capture system-level performance characteristics of parallel workload execution across neurocores.

Optimization efforts have explored partitioning and mapping strategies for neuromorphic platforms [3], [23], [26], [27], [42], [49], [55], [57], [61], though most work remains limited to simulators without validation on real systems. Other recent works have demonstrated performance improvements through network sparsification [28], [33], [38], [46], but lack systematic characterization of sparsity effects or cross-platform generalizability.

TABLE I: Comparison of neuromorphic performance analysis features across prior work.

	Cross-Platform	Real Hardware	Arch-Aware Metrics	System Bottlenecks	Performance Framework	Optimization Method
Boyle et al. [6]	✓	✗	✗	✗	✗	✗
Tang et al. [52]	✗	✗	✗	✗	✗	✗
Ostrau et al. [36]	✓	✓	✗	✗	✗	✗
Yang et al. [61]	✗	✓	✓	✗	✗	✓
Pierro et al. [38]	✗	✓	✗	✗	✗	✓
This Work	✓	✓	✓	✓	✓	✓

As shown in Table I, existing work lacks cross-platform analysis using real hardware to characterize fundamental performance principles. Importantly, prior work has not investigated whether widely-used performance proxy metrics (network-wide sparsity, aggregate operation counts) truly reflect improvements in neuromorphic system performance, nor established a unifying performance framework capturing system bounds and bottlenecks. This paper addresses these gaps through the first systematic performance investigation with multiple real neuromorphic platforms.

III. Analytical Modeling

In this section, we use a simple analytical model to theoretically inform the relationships between neuromorphic workload configuration settings and fundamental accelerator bottlenecks. We use this analytical modeling to guide our hardware profiling in Section V.

The pipelined neurocore computing units have three core operations: (a) **synop accumulation**, (b) **activation computation**, and (c) **NoC activation messaging**. At the same time, the following features affect how many of each core operation is required: **weight sparsity** impacts the number of synop weight fetches and accumulations, **activation sparsity** impacts the number of synop inputs and NoC traffic, and **partitioning** affects the number of neurons in a neurocore that synops and activation computes are applied to, as well as the total neurocore utilization on-chip. Finally, neurocore **mapping** affects the traffic congestion and routing of activation messaging, though it does not affect the volume of total message traffic.

By varying workload configurations, the core operation counts will grow or shrink, and thus each of the three operations may bottleneck overall performance. Since the costs for synops, activation computes, and message routing are within an order of magnitude, we use asymptotic operation counts in order to compare the three [12], [52].

A. Model Formulation

Our model examines one layer, l_i, in a fully connected network which has N neurons. For simplicity, the previous and next layers connected to l, l_{i-1} and l_{i+1}, also have N neurons. The weight density in the synaptic connections is denoted w, such that the weight sparsity is $1-w$. We assume that m is the message density of both l_{i-1} and l_i, where the former impacts the input of l_i and the latter impacts the input of l_{i+1}.

B. Single Neurocore Per-Layer Operation

We first uncover the 'base case' bottleneck, by analyzing each layer mapped onto one neurocore without parallelization.

Accounting for activation sparsity, the number of inputs to l_i is mN, and each input triggers the sparse read of wN weights, leading to mwN^2 synops. Next, to count the minimum number of activation computes, we make the idealized assumption that an activation compute is only necessary if the neuron in l_i received at least one synop. Under uniform sparsity, for each of the mN inputs a, and for each of the N neurons b:

$$p(a \text{ does not message } b) = 1 - w \quad (1)$$

$$p(b \text{ receives no message}) = (1 - w)^{mN} \quad (2)$$

$$p(b \text{ is messaged}) = 1 - (1 - w)^{mN} \quad (3)$$

Therefore, the minimum neurocore activation computes is $N \cdot (1 - (1 - w)^{mN})$. In practice, without extremely high sparsity, for the usual neuron dimensions of ML network layers, each neuron is expected to receive at least one message, so neuron updates and writeback remain $\sim O(N)$. Thus, the sparsity-aware operation costs are:

(a) Synops: $O(mwN^2)$
(b) Act. computes: $\sim O(N)$
(c) Messages traffic to l_{i+1}: $O(mN)$

This indicates an operation bottleneck: **when sparsities are low, the quadratic N^2 scaling of synops will dominate.** As sparsity increases, and the synops are sufficiently scaled down, the system bottleneck may shift to activation computes or message traffic, which both scale linearly with N.

C. Parallel Utilization of Multiple Neurocores

Next, we introduce parallelization to the model, addressing the synops bottleneck by sharing the load among multiple neurocores. Here, we examine the case where a network layer is 'voluntarily' partitioned to multiple neurocores. This type of utilization contrasts with the 'involuntary' case, where greater utilization is forced as a layer's weight parameters scale beyond the memory limits of a neurocore, addressed next.

We retain neuron dimensions N and define the number of neurocores assigned to each layer C_{i-1}, C_i, and C_{i+1} of layers l_{i-1}, l_i, and l_{i+1}, respectively.

For the mN messages from l_{i-1}, by default, all must be broadcast such that each of the neurocores in l_i will receive them to update their mapped neurons. Using an idealized hardware assumption that a message does not need to be sent to a neurocore if all of the corresponding synapses to the neurons are pruned, then the probability that an activation misses the N/C_i neurons in a neurocore of l_i is $(1 - w)^{\frac{N}{C_i}}$. Again, in practice, this probability is very low, and the number of activations at each neurocore remains $\sim O(mN)$. Since (approximately) every message is duplicated to each neurocore in l_{i+1}, the partitioned operation costs are:

(a) Synops per neurocore: $O(mwN^2/C_i)$
(b) Act. computes per neurocore: $\sim O(N/C_i)$
(c) Message traffic to l_{i+1}: $O(mNC_{i+1})$

We identify two important implications of increasing neurocore utilization. First, **synops are processed in parallel across the active neurocores, meaning that this operation must be accounted for at the granularity of a neurocore, rather than in total.** Second, **increasing partitioning causes linearly increasing message traffic, and linearly decreasing neurocore synops.** This indicates that while partitioning can effectively address a synops bottleneck, it may shift the system to a traffic bottleneck.

D. Forced Utilization from Network Width

Finally, we examine the workload configuration case where a network layer's width has scaled up, which forces 'involuntary' neurocore utilization due to the layer exceeding the synaptic memory capacity of one neurocore. Note that the prior 'voluntary' parallelization can be applied on top of this forced utilization.

We re-define N to be the maximum number of neurons that fit in a single neurocore. The new neuron dimension for all l is xN, where $x > 1$. Due to the fully connected weight layers, scaling neuron dimension by x leads to a x^2 scaling in number of synaptic weights, which leads to quadratic scaling in the number of neurocores necessary to minimally map each layer l: $C_{i-1,i,i+1} = O(x^2)$. This quadratic scaling of neurocores leads to a decrease in the number of neurons per neurocore in each layer, each neurocore maps $xN/x^2 = N/x$ neurons.

Counting operations for l_i, in each neurocore, each of the mxN inputs fetches wN/x synaptic weights for accumulation, and activation computes remain linear to the number of neurons N/x. Message traffic of l_i, which we consider across all neurocores, is mxN messages duplicated to x^2 neurocores in l_{i+1}. Overall:

(a) Synops per neurocore: $O(mwN^2)$
(b) Act. computes per neurocore: $\sim O(N/x)$
(c) Message traffic to l_{i+1}: $O(mx^3N)$

Thus, **increasing network width forces a quadratic increase in neurocore utilization, and a cubic increase in message traffic volume.** In addition, if each neurocore is filled to its capacity (i.e., no 'voluntary' partitioning is applied on top of 'involuntary' utilization), **synops per neurocore will not change as a function of network width.** Together, this suggests that wide network layers will be likely to cause NoC traffic to overtake the previously-identified dominant bottleneck of neurocore synops.

E. Key Modeling Insights

We summarize the findings into key Modeling insights (M), presenting guidelines to understanding the neuromorphic performance bottleneck implications across workload configurations:

M1. *Memory-bound*—Synops are a dominant bottleneck, scaling quadratically in network width until saturating the capacity of a neurocore. This bottleneck can be addressed by increasing weight and activation sparsity, or increasing partitioning. We refer to this as memory-bound since synop costs are mainly weight fetch and sum writeback [14], [52].

M2. *Compute-bound*—Activation computes are unlikely to bottleneck performance since they only scale linearly with the number of neurons in a neurocore. However, if sparsity and partitioning is large enough to shrink the synop bottleneck without triggering a traffic bottleneck, the workload may be compute-bound.

M3. *Traffic-bound*—High neurocore utilization from partitioning or forced from large network widths will expand the traffic volume linearly or cubically, leading to a traffic-bound state. But, an intelligent mapping of neurocores is also useful for mitigating the traffic bottleneck, i.e., by placing high-output neurocores on separate NoC router paths.

Moreover, we emphasize another key insight of our analytical model: synop and compute bottlenecks appear at the neurocore level, not the total network level. Therefore, total weight sparsity and activation sparsity will be inaccurate performance indicators if the resulting synops are unevenly distributed across neurocores, particularly for barrier-synchronized neuromorphic accelerators where total performance is dependent on the slowest neurocore. Aligning with the enumeration, we refer to this as **M0.**

In Section V, we substantiate each of these insights by extensively profiling real neuromorphic accelerators, which we briefly introduce next.

IV. Accelerators and Workloads

Accelerator	Task	Neuron Type	Network Architecture
AKD1000 [7]	RGB image classification [18]	ReLU	CNN [17]
Speck [28]	Event camera classification [35]	Spiking IF	CNN
Loihi 2 [20]	RGB video regression [5]	SD-ReLU [34]	CNN [46]
	Audio denoising [54]	ReLU	SSM [48]

TABLE II: Summary of accelerators and workloads under study.

Our cross-platform empirical investigation spans three neuromorphic accelerators with fundamentally different architectural implementations and target applications, allowing us to establish performance principles that generalize across neuromorphic designs rather than being specific to individual implementations. The platforms represent distinct points in the neuromorphic design space: edge inference acceleration (AKD1000), specialized sensor processing (Speck), and research programmability (Loihi 2). Table II summarizes the three accelerators and four workloads under study.

1) AKD1000: The Brainchip AKD1000 [7] is an edge inference accelerator with 80 neurocores optimized for ReLU activation sparsity in CNNs. This platform enables investigation of CNN weight sparsity limitations and load balancing effects. We characterize this accelerator using AkidaNet [8], a variant of the depthwise-separable MobileNetv1 CNN [17] using standard convolutions in early layers. The network is applied to classify Imagenette [18], a 10 class subset of the ImageNet dataset [13]

2) Speck: The Synsense Speck [28] is a specialized neuromorphic architecture with 9 neurocores optimized for event camera [19] processing using spiking integrate and fire neurons. In this study, this platform presents a highly specialized, micro-edge accelerator, which we demonstrate still conforms to the general trends as the other more general-purpose platforms. Unlike barrier synchronized accelerators (AKD1000 and Loihi 2), Speck neurocores operate asynchronously and enter idle states when no input is present [39]. Furthermore, there is no partitioning in network layers—each neurocore maps a full layer.

3) Loihi 2: The Intel Loihi 2 [20] is our primary characterization platform, containing 120 programmable neurocores and having extensive software instrumentation for microarchitectural analysis. The platform enables arbitrary network partitioning, detailed neurocore operation analysis, and validation of performance scaling principles across diverse network architectures and utilization regimes. We study two representative workloads: First, PilotNet [46], a CNN of sigma-delta ReLU (SD-ReLU) neurons for processing RGB video sequences into vehicle steering angle [5]. The sigma-delta neurons [34] leverage the high ReLU similarity between consecutive video frames to produce high activation (message) sparsity on-chip. Second, S5, a state space model [47] adapted for activation sparsity on Loihi 2 by adding ReLU activations [38], applied to an audio denoising task [54]. To avoid I/O latency bottlenecks [46], we study accelerator performance using input data loaded into neurocores on chip.

V. System Characterization

We substantiate the theoretical modeling insights from Section III using a broad empirical characterization of the real neuromorphic accelerators. We demonstrate the validity of the insights, in that each of the bottlenecks do appear under configurations expected by the model. Furthermore, we profile the relative performance impacts of each bottleneck on overall accelerator performance, extending the asymptotic trends to real performance boundary curves.

A. Characterization Methodology

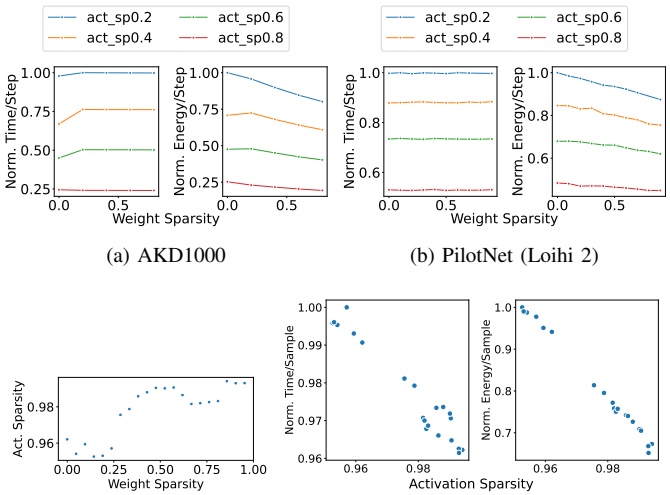
By manually configuring network weights, biases, and thresholds, or programming neurons directly (i.e., non-trained networks), we systematically profile the full range of workload sparsity and execution dynamics that a trained network can take.

Our profiling is structured at two levels of granularity. First, we examine network-wide weight sparsity and activation sparsity for all three accelerators, to investigate the shortcomings of total, high-level metrics (**M0**). Then, using the Loihi 2, we conduct a deeper dive into investigating workload configurations that exhibit the three bottleneck states of memory (**M1**), compute (**M2**), and traffic (**M3**) that are suggested by the analytical model.

We use two performance metrics: **Time per Step** measures the average duration between consecutive inference outputs,

corresponding to timestep duration for synchronized accelerators (AKD1000, Loihi 2), and full sample processing time for the fully asynchronous Speck. **Energy per Step** combines average system power with execution time to evaluate efficiency. All measurements directly use the accelerators' telemetry and software instrumentation, except for the timing of Speck, which is measured in wall-clock time over a long experiment. **In our empirical study, we normalize all performance results and micro-operation measurements to focus on performance bound and bottleneck trends, rather than absolute system comparisons.**

B. *M0: Network-wide Weight Sparsity*



(a) AKD1000 (b) PilotNet (Loihi 2)

(c) Speck

Fig. 2: Weight sparsity performance of CNNs.

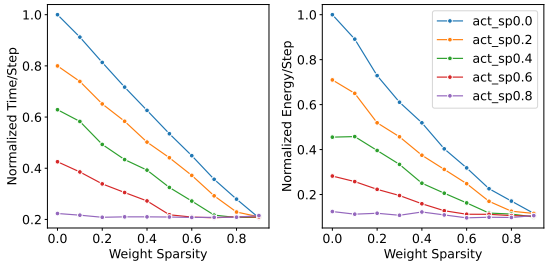


Fig. 3: S5 (Loihi 2) weight sparsity performance

We begin by studying whether total weight sparsity is an accurate performance indicator (**M0**), and surprisingly observe that for CNN workloads, weight sparsity does not yield substantial performance benefits.

Figures 2 and 3 profile the effects of weight sparsity on the timing and energy performance, by tracking uniformly increasing weight sparsity at a constant activation sparsity (act_sp). Since Speck cannot directly control for activation sparsity, we show the effect of increasing weight sparsity on activations, then re-visualize these networks with performance as a function of their activation sparsity.

In Figure 2, for the CNNs on AKD1000 and Loihi 2, there is no benefit for runtime, and a slight benefit for energy, likely due to compute efficiency for zero values. The magnitude of energy improvement is small, for instance, the energy benefit from increasing weight sparsity from 0.0 to 0.9 is smaller than the benefit of increasing activation sparsity from 0.2 to 0.4. Similarly, on the Speck, energy performance remains linear to activation sparsity, and there is no correlated performance benefit with increasing weight sparsity.

In contrast, Figure 3 shows that weight sparsity is supported for the linearly connected S5 network, and its benefit is roughly equal to the benefit of increasing activation sparsity. For example, the performance benefit by increasing weight sparsity from 0.0 to 0.2 with 0.0 activation sparsity (blue line), is about equal to the improvement of increasing activation sparsity from 0.0 to 0.2 with 0.0 weight sparsity (blue to orange line).

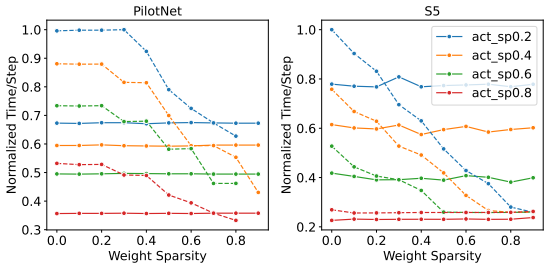


Fig. 4: Timing overhead of sparse weight support on Loihi 2. Solid lines use dense weight formatting, while dashed lines use sparse weight formatting.

The weight sparsity results in Figures 2 and 3 use the default weight formatting on Loihi 2: dense for CNNs, sparse for linearly connected networks. Figure 4 shows the implications of sparse weight formatting for PilotNet and S5. For the PilotNet CNN, the sparse weight format only begins to improve timing performance around 0.7 weight sparsity, under which the timing performance is much worse. For the S5 linearly connected network, the sparse weight format is beneficial around 0.2 weight sparsity. In the remainder of the paper, we use the default formatting.

The challenge of supporting weight sparsity in CNNs using the neuromorphic execution paradigm is that kernel weight fetches triggered for each input activation are small, so the overhead of decoding the sparse weight format can outweigh the benefit of fewer weight memory accesses. The profiling implies that specialized architecture components, potentially utilizing structured weight sparsity, may be necessary to support CNN weight pruning methods which have been intended for neuromorphic deployment optimization [10], [43], [44].

C. *M0. Network-wide Activation Sparsity*

Next, we repeat the investigation on total activation sparsity, particularly under load imbalanced conditions (**M0**).

Figure 5 shows the relationship between activation sparsity and performance. In this characterization, we skew load balance across neurocores by using different sparsity schedules:

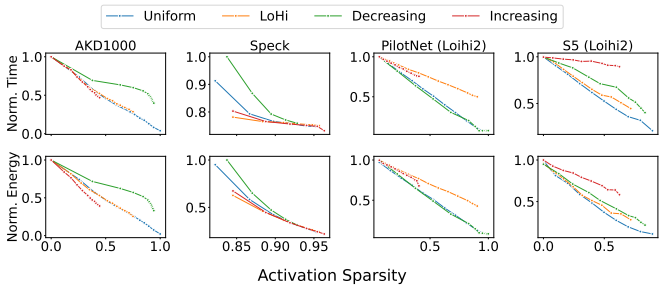


Fig. 5: Activation sparsity with varying sparsity schedules to change load balance

Uniform is consistent sparsity across all layers, LoHi is an alternating scheme of low and high sparsity, and Decreasing and Increasing linearly scale sparsity from first to last layer.

In the AKD1000 and Loihi 2 workloads, the sparsity is exactly programmed by explicitly toggling neuron activation messaging on and off. These workloads exhibit a linear correlation between sparsity and performance when the sparsity is Uniform across network layers. However, when the sparsity is non-uniformly applied in the LoHi, Decreasing, and Increasing schedules, the workload is no longer balanced, and the correlation breaks down. In the Speck, the sparsity schedule is programmed as spiking neuron thresholds per network layer, as exact sparsity cannot be induced. The LoHi and Increasing schedule curves have better performance at the same total sparsity, suggesting that the last network layer (which has the most weights) is the workload bottleneck.

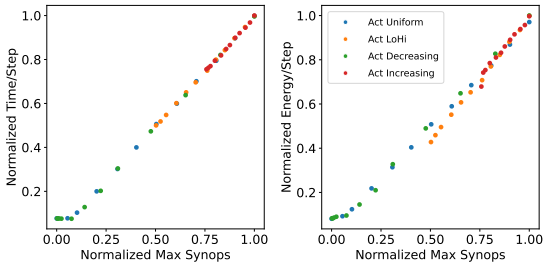
In effect, we confirm that total network measurements like activation sparsity can be correlated with performance benefits, but they are insufficient for fully understanding performance due to the parallelization of the network across neurocores.

*D. **M1.** Dominant Memory Bottleneck of Max SynOps*

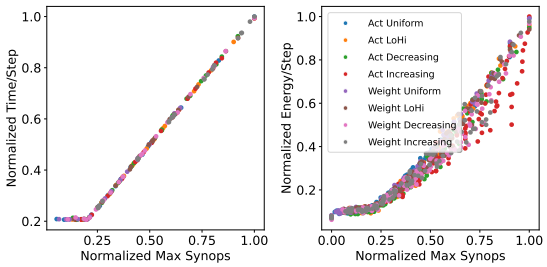
Next, we profile the three bottleneck states suggested by the analytical modeling in Section III, using a deeper dive into Loihi 2. We begin with the expected dominant bottleneck, neurocore synops (**M1**).

Figure 6 shows all PilotNet and S5 activation sparsity schedule configurations from Figure 5, plotted with performance as a function of the maximum synops of any active neurocore. Since the S5 network can also leverage weight sparsity (Figure 3), we additionally include S5 workloads with weight sparsity applied in imbalanced sparsity schedules. Here, all workloads for the same network architecture use the same, minimal neurocore partitioning.

Despite widely varying sparsity and load balance configuration, the timing performance of Loihi 2 is linear to the max synops metric, until reaching a performance floor when the max synops is sufficiently low. Energy, as well, is well-correlated with this metric. This profile not only verifies that max synops presents a performance bottleneck for Loihi 2, but it also demonstrates that this bottleneck results in a clear performance boundary: workload timing does not run faster than the linear max synops limit.

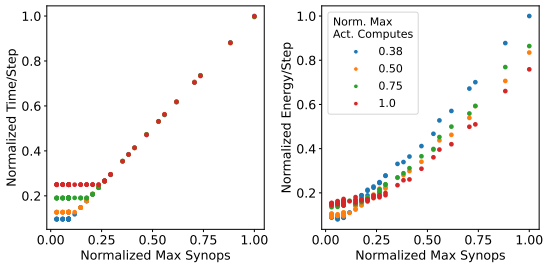


(a) PilotNet

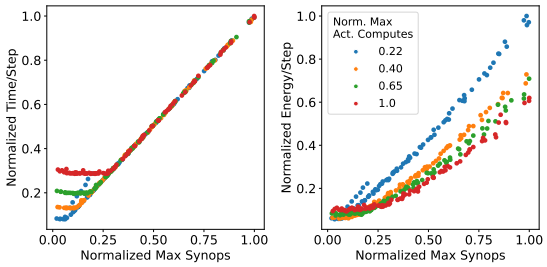


(b) S5

Fig. 6: Memory-bound bottleneck for all sparsity schedules of PilotNet and S5.



(a) PilotNet



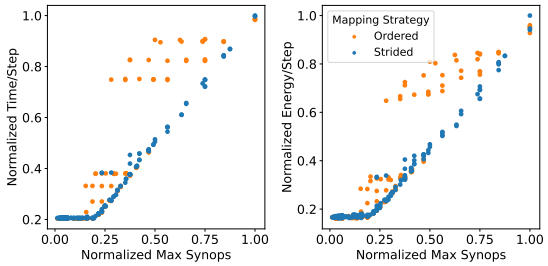
(b) S5

Fig. 7: Lowering the compute-bound floor by increasing partitioning.

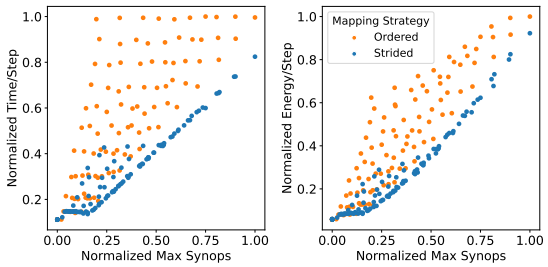
.

*E. **M2.** Compute Bottleneck Floor with Low Max Synops*

The timing floor that appears at low max synops is due to the workloads shifting from a memory-bound synops bottleneck state to an activation computation bottleneck state (**M2**). In Figure 7 (left), we verify this for PilotNet and S5 by demonstrating that the timing floor is lowered when the layer with the max activation computes is partitioned, which reduces the number of neurons in the compute-bottleneck neurocore



(a) PilotNet



(b) S5

Fig. 8: Traffic-bound comparison between ordered mapping (orange) and strided mapping (blue), using high neurocore utilization workloads.

.

and matches the expected behavior from Section III-C. Thus, the compute-bound timing floor is a variable performance boundary, dependent on the max activation computes of the workload partitioning configuration.

Figure 7 (right) also shows that energy curves diverge among the partitioning settings. As partitioning increases, lowering the max activation computes, more neurocores are utilized, raising power and thus raising energy as well. In Figure 7, the PilotNet utilization increases from 80 to 119 neurocores (highest to lowest timing floor). For the S5, the utilization increases from 27 to 60 neurocores. In general, the energy result implies that unless timestep duration is being improved by lowering the compute floor, an optimal network configuration should use the fewest neurocores possible, as adding cores leads to greater power.

*F. **M3.** Traffic Bottleneck with High Neurocore Utilization*

We address the third bottleneck state, traffic-bound (**M3**), by profiling workloads with high neurocore utilization.

For PilotNet, we reduce the convolution spatial dimensions and channels, shrinking the network to free more neurocores for greater (voluntary) partitioning (Section III-C). Scaling the network width does not significantly affect CNN utilization, as neurons maintain small receptive fields. The reduced PilotNet configuration when minimally partitioned uses 32 neurocores, and the network was partitioned to 116 neurocores. For reference, the prior full-size PilotNet profiling configurations use their minimum utilization of 96 neurocores.

For S5, we double the network width in order to increase (involuntary) utilization (Section III-D). This configuration

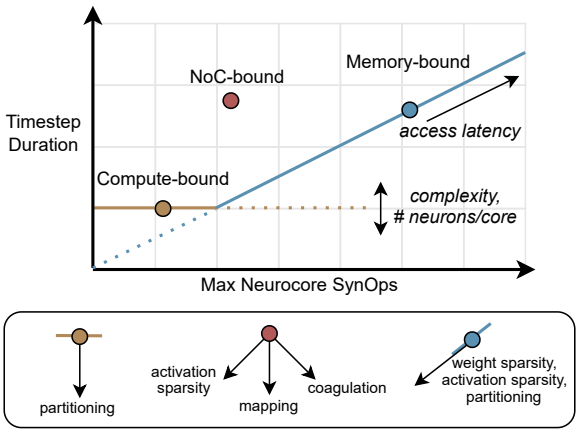


Fig. 9: The floorline model for understanding performance of neuromorphic systems. Bounds on the timestep duration are defined by memory access latency, or neuron activation computes with sufficiently low memory accesses. Above the bounds, applications are bottlenecked by message traffic in the NoC.

uses 76 neurocores, compared to the prior S5 profiling configurations which use 27 neurocores.

With the high-utilization network configurations, and still sweeping over sparsity configurations, Figure 8 shows the difference between a poor ordered mapping and a better strided mapping. The ordered mapping places neurocores sequentially on the chip, a scheme which has been previously used for Loihi 1 [27]. These workloads struggle with a traffic bottleneck since the highest output neurocores, belonging to the same network layer, are physically close to one another and create congestion on their shared NoC routers.

The improved strided mapping places neurocores in strided intervals, separating neurocores of the same network layers to different NoC routing paths. While the improved heuristic mapping does not fully address the traffic bottleneck for all workload configurations (i.e., strided points still rise above the memory-bound slope and compute floor), it is nevertheless effective at improving performance in all cases, and does not demonstrate negative side-effects such as raising the time or energy floor.

VI. Optimization Approach

Our performance bound and bottleneck analysis provides valuable insights into how one can improve the performance of a generic neuromorphic accelerator workload. Especially in the current landscape of nascent neuromorphic accelerator research, without standardized micro-architectural designs nor accurate performance estimation models, understanding the critical bottlenecks to look for and how to address these bottlenecks allows for actionable and generalizable performance optimization.

A. The Floorline Performance Model

The insights from the analytical modeling and empirical characterization of real neuromorphic accelerators have identified (1) three bottleneck states and (2) which configurations are

likely to exhibit each bottleneck. Moreover, the Loihi 2 characterization shows that the memory and compute bottlenecks result in clear performance boundaries across all sparsity and partitioning configurations for a given network architecture. We distill these insights into a visual model for understanding neuromorphic workload performance and optimization directions: the floorline performance model, shown in Figure 9.

In the floorline model plot (Figure 9, top), the horizontal 'intensity' axis is the maximum synops of any active neurocore, and the vertical 'performance' axis is timestep duration. The slope of the floorline is defined by the memory access latency in a neurocore, and represents the memory performance bound of a workload. When the maximum synops decreases, the performance reaches a floor, which is the compute boundary defined by the activation computation. The floor shifts up and down based on the complexity of the bottlenecking activation computation (i.e., number of instructions), and the number of neurons in the compute-bottlenecking neurocore.

The floorline also provides a guide to improving the performance of any neural network workload (Figure 9, bottom). A trained network, with its distinct sparsity, load balance, and partitioning configuration, will have performance (a) on the slope, (b) on the floor, or (c) above the floorline. These three locations correspond to (a) memory-bound, (b) compute-bound, and (c) traffic-bound states. Optimization of the workload then proceeds as directed by the modeling and characterization insights (**M1**, **M2**, **M3**):

(a) *Memory-bound optimization*—Reduce the maximum neurocore synops by increasing the weight sparsity and/or activation sparsity, or partitioning the synop-bottlenecking neurocore. This will move the workload down and left, improving performance while reducing the synops intensity (max neurocore synops).

(b) *Compute-bound optimization*—Reduce the maximum activation computes by partitioning the compute-bottlenecking neurocore. This will move the workload straight down, improving performance without necessarily affecting the synops intensity.

(c) *Traffic-bound optimization*—Reduce the activation message NoC traffic by increasing activation sparsity or coagulating the network into fewer neurocores. The former moves down and left, potentially reducing synops intensity, while the latter moves down and right, potentially increasing synops intensity but improving performance nonetheless since the workload is traffic-bound. Or, the workload performance can be optimized by improving the neurocore mapping, moving down since the synops intensity will not change.

The floorline differs from the conventional roofline model [56] to capture the unique performance dynamics of neuromorphic accelerators. First, the roofline relates throughput (y-axis) with operational intensity (x-axis) and bandwidth (roofline slope), while the floorline relates timestep duration with operation count and latency. This is suited for the sparse, event-based processing on neuromorphic accelerators. Second, conventionally, only one roofline ceiling is presented, representing the peak compute performance boundary of a system. In the floorline, the location of the compute floor boundary is highly variable based on the number of bottlenecking activation computes. Finally, while the roofline informs performance bounds, deeper micro-architectural and kernel-level insights are usually required to optimize a workload, moving it closer to the performance bounds. With the floorline, the location of a workload on the floorline can completely inform its bottleneck state and how to best optimize it, due to the specialized compute pattern of neuromorphic accelerators for neural network processing.

As for all models, the floorline uses assumptions which may not hold true for all potential neuromorphic accelerator designs. For example, the straight performance boundary lines are only realized if synop, activation compute, and messaging execution are sufficiently overlapped in a pipeline. If these operations were fully sequential, performance would be a function of the sum of the three, rather than a function of the individual bottleneck. Moreover, neuromorphic approaches like the Speck [39] and SENECA [59] have demonstrated asynchronous inference methods without timestepped neurocore synchronization, which disconnects the floorline model's performance metric of timestep duration. Though, the former is dependent on SNNs, and state of the art ML sequence processing requires timestepped token-by-token execution, while the latter relies on input message sorting which has not been demonstrated with multi-neurocore partitioning in a real accelerator.

B. Optimization Framework

We present an actionable, two-stage optimization procedure for improving the performance of a neuromorphic accelerator workload, involving a sparse-aware training stage, and a floorline-informed partitioning stage.

First, networks should be trained with as high sparsity as application accuracy allows. Techniques which increase sparsity approximately uniformly across network layers are preferable, as they will more likely lead to load-balanced configurations when deployed. Given that overall sparsity improvement is well-studied in ML network training and configuration, this provides a prudent first stage in optimization.

Second, with the network weights locked from training, we use the floorline model as a guide for optimizing the partitioning and mapping of the workload to the chip. Any network architecture may have unique performance boundaries, but one can trace along the boundaries using an iterative, backtracking procedure:

To begin, the network is initialized with the minimum possible neurocore utilization, as well as a good heuristic mapping, such as a strided mapping (Figure 8). These both increase the likelihood that the initialized network will be memory-bound by max synops. Under this assumption, after executing or simulating the network with sample data, we identify the neurocore which executed the most synops, and increase the partitioning of the layer that this neurocore belongs to. If the

memory-bound assumption is correct, then the workload will have traced down the memory slope of the floorline, and we repeat the max synops identification and partitioning.

If partitioning on the memory-bound assumption is not beneficial, then we backtrack the partitioning, as greater utilization without synops improvement is detrimental for power.

After backtracking, we shift the assumption to compute-bound, partitioning based on the max activation computations. Similarly, we can repeat until it is not beneficial, then backtrack. Finally, we may assume that the workload is NoC-bounded, and optimize mapping by moving the greatest message output neurocores onto separate router paths. In general, we can continuously cycle through each assumption and corresponding optimization step.

The iterative partitioning and mapping process stops when there are no further neurocores available on chip, or when energy starts to worsen due to larger utilization power compared to the timing benefits. The iterative process also stops if none of the three optimizations lead to performance improvements, indicating that the workload has reached its true performance boundary for its sparsity dynamics.

VII. Optimization Results

To demonstrate the potential speedup that iso-accurate networks can achieve with our optimization framework, we train and optimize a sweep of configurations using the same network architectures and inference tasks as before. Similarly to the characterization, we apply the first step of sparsity optimization across all three accelerators, training sparse networks with iso-accuracy to the baseline. Then, in the second optimization step we use Loihi 2 to further optimize the partitioning and mapping of the iso-accurate networks.

A. Optimization Methodology

As performance baselines, we use configurations which rely solely on the innate sparsity of ReLU and spiking activations from accuracy-only training, a strategy which has appeared in recent prior works with the AKD1000 [9] and Speck [39], and Loihi2 [45]. For PilotNet, we directly apply a prior manually-tuned configuration as the baseline [46].

For training in optimization stage 1, we use existing network training and quantization frameworks for each workload—MetaTF (AKD1000) [9], Sinabs (Speck) [51], Lava DL (PilotNet) [22], and SparseRNNs—and the following training procedures:

- **AKD1000**—The baseline AkidaNet network was trained with cross entropy loss only. To induce ReLU activation sparsity, we add the Tℓ1 regularizer [63], which we apply to the pre-trained baseline. All networks were also quantized and fine-tuned.
- **Speck**—The baseline spiking network is also trained with cross entropy loss only. To induce greater spiking sparsity, we add the synops regularization loss [50], training from scratch.
- **PilotNet**—For the PilotNet configurations, the same trained network weights are used, and sigma-delta thresholds are modified to adjust activation sparsity. The baseline is a uniform threshold setting across all network layers from prior work [46]. Our approach assigns thresholds uniquely to each layer in order to reach per-layer sparsity targets, in order to encourage more uniform activation sparsity.
- **S5**—We use weight pruning to sparsify the linearly connected network. With a fully dense baseline model, we prune the smallest 0.1 to 0.9 of weights away in one shot, and fine-tune. Networks include ReLU activations for innate activation sparsity which is not explicitly trained.

In optimization stage 2, we take the highest-accuracy networks from sparsity training and apply the backtracking partitioning approach described in Section VI-B.

For the S5 networks, due to quantization concerns on Loihi 2, we present deployed performance results with simulated message activity matching the per-layer activity measured during GPU inference.

B. Stage 1: Sparsity Optimization

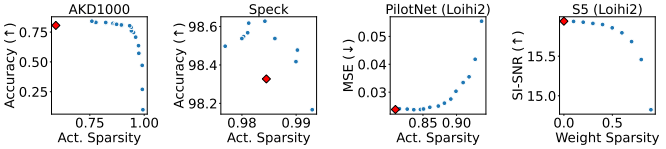


Fig. 10: Sparsity vs. accuracy of trained networks. Diamonds indicate the baselines which were not trained with explicit sparsity loss or were presented in prior work (PilotNet [46]).

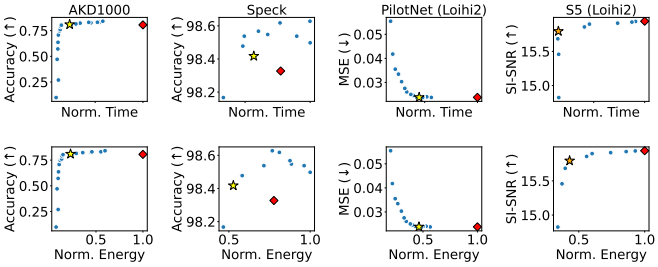


Fig. 11: Performance vs. accuracy of the sparse-trained networks and baselines. Diamonds indicate the baselines, while stars mark the highest-performance sparse network configurations which are iso-accurate to the baselines. For S5, the star highlights a high accuracy, high performance network, as no networks are iso-accurate.

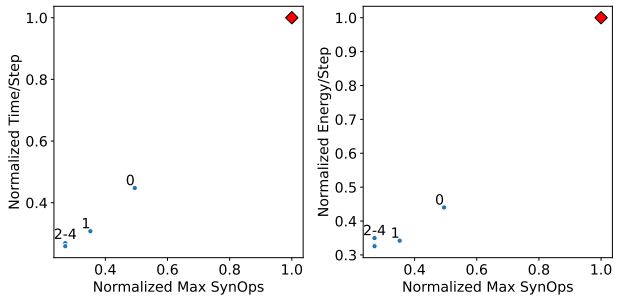
Figure 10 shows the accuracy achieved for the range of sparse-trained networks. For the AKD1000, PilotNet, and S5 networks, a clear Pareto curve emerges. For the Speck, which uses spiking neurons, the baseline trained without explicit sparsity regularization achieves considerable spike activation sparsity, but several sparse-trained networks achieve greater accuracy with greater activation sparsity.

Next, Figure 11 visualizes deployed performance as a function of accuracy, for all of the trained networks, while

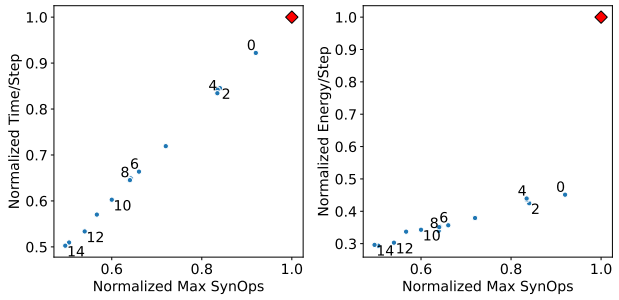
using constant deployment configurations of partitioning and mapping. Diamonds mark the baselines, and stars mark the highest-performance iso-accurate configurations. Respective time speedup and energy benefit of the iso-accurate workloads against the baselines are: AKD1000 $(4.29\times, 4.36\times)$, Speck $(1.01\times, 1.47\times)$, PilotNet $(2.23\times, 2.16\times)$. The pruned S5 networks do not achieve baseline iso-accuracy; though, at a loss of 0.15 dB SI-SNR, the 0.6 sparse pruned network achieves $(1.74\times, 2.32\times)$ deployed benefit.

We highlight the large benefit of the uniform sparsity target threshold setting for PilotNet, compared to the baseline's uniform threshold setting. The starred iso-accurate network differs slightly from the baseline in total activation sparsity ($1.24\times$ fewer messages), but our approach achieves much greater deployed performance ($> 2\times$), due to improved neurocore load balance.

C. Stage 2: Partitioning and Mapping Optimization



(a) PilotNet



(b) S5

Fig. 12: Partitioning and mapping optimization of PilotNet and S5. Diamonds mark the baseline performance from Figure 11, which use minimal partitioning. Numbers indicate the partitioning iteration steps, beginning with 0 as the minimally partitioned configuration.

Figure 12 shows the further partitioning optimization with the high accuracy sparse star configurations from Figure 11. In the figure, the diamonds mark the performance of the baselines (same as Figure 11 diamonds). For the S5, weight sparsity is used to pack the network into fewer neurocores compared to the dense baseline: in Figure 12b, the minimal partitioning marked with 0 uses 36 neurocores, while the diamond uses 73 neurocores, causing a large utilization power and energy difference.

The performance of the iterative partitioning procedure traces the memory performance boundary for both PilotNet and S5, which means that the optimization was achieved by solely attending to the maximum neurocore synops bottleneck. The PilotNet partitioning ended when no further neurocores were available, while the S5 partitioning ended when energy costs began to rise due to neurocore utilization power without sufficient timing improvements.

The time speedup and energy improvement from only partitioning for the PilotNet is $(1.73\times, 1.26\times)$, and for the S5 is $(1.83\times, 1.52\times)$. Combined with the sparsity improvements, the total improvements of the final optimized configurations against the baselines are $(3.86\times, 2.86\times)$ and $(1.99\times, 3.38\times)$ for the PilotNet and S5, respectively.

VIII. Conclusion

In this work, we presented the first systematic study of performance bounds and bottlenecks in real neuromorphic accelerators. Using analytical modeling and empirical characterization of three real accelerators, we established three neuromorphic bottleneck states (memory, compute, and traffic), and identified which sparsity and parallelization configurations that are likely to be in each bottleneck state. Based on our modeling insights, we synthesized neuromorphic accelerator bounds and bottlenecks into the floorline performance model, analogous to the roofline model for conventional systems, and developed an actionable two-stage optimization framework.

A. Discussion and Future Work

While we have substantiated the performance boundary trends and optimization methods of the floorline model using extensive characterization and validation on Loihi 2, further in-depth analysis is necessary with other neuromorphic accelerator designs to understand their specific performance bound and bottleneck implications. Currently, such analysis is limited by the research infrastructure and access of nascent neuromorphic accelerators.

Addressing limited research access, further micro-architectural profiling and simulation can inform the construction of an accurate Loihi 2 floorline performance simulator, which can be used as a software proxy for Loihi 2 in network training and compiling research. Such a tool would require not only neurocore-level modeling to inform memory and compute boundaries, but also NoC simulation in order to determine whether workloads have become traffic-bound.

Finally, while the present study has addressed single-chip systems, an important future direction for bottleneck analysis is the extension to multi-chip neuromorphic accelerators, which can include tens to thousands of chips in one system [21], [29]. Inter-chip communication latency has the potential to dominate the performance over all intra-chip operations, but as inter-chip traffic can also be reduced by network activation sparsity, the relative performance impacts may remain comparable.

REFERENCES

[1] S. Abreu, S. B. Shrestha, R.-J. Zhu, and J. Eshraghian, "Neuromorphic principles for efficient large language models on intel loihi 2," 2025. [Online]. Available: https://arxiv.org/abs/2503.18002

[2] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[3] 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, 2020.

[4] A. Basu, L. Deng, C. Frenkel, and X. Zhang, "Spiking neural network integrated circuits: A review of trends and future directions," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, 2022, pp. 1–8.

[5] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[6] J. Boyle, M. Plagge, S. G. Cardwell, F. S. Chance, and A. Gerstlauer, "Performance and energy simulation of spiking neuromorphic architectures for fast exploration," in *Proceedings of the 2023 International Conference on Neuromorphic Systems*, ser. ICONS '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3589737.3605970

[7] Brainchip, "Akida akd1000," https://brainchip.com/akida-neural-processor-soc/, 2025.

[8] ——, "Akidanet/imagenet inference," https://doc.brainchipinc.com/examples/general/plot_1_akidanet_imagenet.html#sphx-glr-examples-general-plot-1-akidanet-imagenet-py, 2025.

[9] ——, "Metatf documentation," https://doc.brainchipinc.com/index.html, 2025.

[10] B. Chakraborty, B. Kang, H. Kumar, and S. Mukhopadhyay, "Sparse spiking neural network: Exploiting heterogeneity in timescales for pruning recurrent SNN," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=0jsfesDZDq

[11] M. Dampfhoffer, T. Mesquida, A. Valentian, and L. Anghel, "Are snns really more energy-efficient than anns? an in-depth hardware-aware study," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 3, pp. 731–741, 2023.

[12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[14] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, aug 2016. [Online]. Available: https://dx.doi.org/10.1088/1741-2560/13/5/051001

[15] 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.

[16] W. G. Gomez, A. Pignata, R. Pignari, V. Fra, E. Macii, and G. Urgese, "First steps towards micro-benchmarking the lava-loihi neuromorphic ecosystem," in *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2023, pp. 462–469.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[18] J. Howard, "imagenette." [Online]. Available: https://github.com/fastai/imagenette/

[19] G. Indiveri and R. Douglas, "Neuromorphic vision sensors," *Science*, vol. 288, no. 5469, pp. 1189–1190, 2000. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.288.5469.1189

[20] Intel, "Taking neuromorphic computing to the next level with loihi 2," https://www.intel.com/content/www/us/en/research/neuromorphic-computing-loihi-2-technology-brief.html, 2021.

[21] ——, "Intel builds world's largest neuromorphic system to enable more sustainable ai," https://www.intel.com/content/www/us/en/newsroom/news/intel-builds-worlds-largest-neuromorphic-system.html, 2024.

[22] ——, "Lava software framework," https://lava-nc.org/, 2025.

[23] 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*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 419–432. [Online]. Available: https://doi.org/10.1145/3582016.3582038

[24] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[25] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Englewood Cliffs, NJ: Prentice-Hall, Englewood Cliffs, NJ, 1984, originally presented at the CMG International Conference, 1984 (conference paper), later expanded into the book.

[26] 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*, ser. GLSVLSI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 9–14. [Online]. Available: https://doi.org/10.1145/3386263.3406900

[27] 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," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 78–89. [Online]. Available: https://doi.org/10.1145/3192366.3192371

[28] Y. Man, O. Richter, G. Zhao, N. Qiao, Y. Xing, W. Dingheng, T. Hu, W. Fang, T. Demirci, M. Marchi, T. Yan, C. Nielsen, S. Sheik, C. Wu, Y. Tian, B. Xu, and G. Li, "Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip," *Nature Communications*, vol. 15, 05 2024.

[29] C. Mayr, S. Hoeppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," 2019.

[30] S. M. Meyer, P. Weidel, P. Plank, L. Campos-Macias, S. B. Shrestha, P. Stratmann, and M. Richter, "A diagonal structured state space model on loihi 2 for efficient streaming sequence processing," 2024. [Online]. Available: https://arxiv.org/abs/2409.15022

[31] O. Moreira, A. Yousefzadeh, F. Chersi, G. Cinserin, R.-J. Zwartenkot, A. Kapoor, P. Qiao, P. Kievits, M. Khoei, L. Rouillard, A. Ferouge, J. Tapson, and A. Visweswara, "Neuronflow: a neuromorphic processor architecture for live ai applications," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 840–845.

[32] B. Na, J. Mok, S. Park, D. Lee, H. Choe, and S. Yoon, "Autosnn: Towards energy-efficient spiking neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2201.12738

[33] K. K. Nazeer, M. Schöne, R. Mukherji, B. Vogginger, C. Mayr, D. Kappel, and A. Subramoney, "Language modeling on a spinnaker2 neuromorphic chip," in *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, 2024, pp. 492–496.

[34] P. O'Connor and M. Welling, "Sigma delta quantized networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=HkNRsU5ge

[35] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

[36] C. Ostrau, C. Klarhorst, M. Thies, and U. Rückert, "Benchmarking neuromorphic hardware and its energy expenditure," *Frontiers in Neuroscience*, vol. 16, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2022.873935

[37] 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.

[38] A. Pierro, S. Abreu, J. Timcheck, P. Stratmann, A. Wild, and S. B. Shrestha, "Accelerating linear recurrent neural networks for

the edge with unstructured sparsity," 2025. [Online]. Available: https://arxiv.org/abs/2502.01330

[39] O. Richter, Y. Xing, M. D. Marchi, C. Nielsen, M. Katsimpris, R. Cattaneo, Y. Ren, Y. Hu, Q. Liu, S. Sheik, T. Demirci, and N. Qiao, "Speck: A smart event-based vision sensor with a low latency 327k neuron convolutional neuronal network processing pipeline," 2024. [Online]. Available: https://arxiv.org/abs/2304.06793

[40] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[41] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," 2017. [Online]. Available: https://arxiv.org/abs/1705.06963

[42] W. Severa, F. Wang, Y. Ho, F. Rothganger, A. Daram, and E. Gonzalez, "Benchmarking spiking network partitioning methods on loihi 2," in *Proceedings of the Great Lakes Symposium on VLSI 2025*, ser. GLSVLSI '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 898–904. [Online]. Available: https://doi.org/10.1145/3716368.3735294

[43] J. Shen, Q. Xu, J. K. Liu, Y. Wang, G. Pan, and H. Tang, "Esl-snns: an evolutionary structure learning strategy for spiking neural networks," in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i1.25079

[44] X. Shi, J. Ding, Z. Hao, and Z. Yu, "Towards energy efficient spiking neural networks: An unstructured pruning framework," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=eoSeaK4QJo

[45] T. Shoesmith, J. C. Knight, B. Mészáros, J. Timcheck, and T. Nowotny, "Eventprop training for efficient neuromorphic applications," 2025. [Online]. Available: https://arxiv.org/abs/2503.04341

[46] S. B. Shrestha, J. Timcheck, P. Frady, L. Campos-Macias, and M. Davies, "Efficient video and audio processing with loihi 2," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 13 481–13 485.

[47] J. T. H. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," 2023. [Online]. Available: https://arxiv.org/abs/2208.04933

[48] J. T. Smith, A. Warrington, and S. Linderman, "Simplified state space layers for sequence modeling," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=Ai8Hw3AXqks

[49] 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 Trans. Embed. Comput. Syst.*, vol. 21, no. 3, May 2022. [Online]. Available: https://doi.org/10.1145/3479156

[50] M. Sorbaro, Q. Liu, M. Bortone, and S. Sheik, "Optimizing the energy consumption of spiking neural networks for neuromorphic applications," *Frontiers in Neuroscience*, vol. Volume 14 - 2020, 2020. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2020.00662

[51] Synsense, "Sinabs is not a brain simulator," https://sinabs.readthedocs.io/, 2025.

[52] G. Tang, A. Safa, K. Shidqi, P. Detterer, S. Traferro, M. Konijnenburg, M. Sifalakis, G.-J. van Schaik, and A. Yousefzadeh, "Open the box of digital neuromorphic processor: Towards effective algorithm-hardware co-design," 2023. [Online]. Available: https://arxiv.org/abs/2303.15224

[53] G. Tang, K. Vadivel, Y. Xu, R. Bilgic, K. Shidqi, P. Detterer, S. Traferro, M. Konijnenburg, M. Sifalakis, G.-J. van Schaik *et al.*, "Seneca: building a fully digital neuromorphic processor, design trade-offs and challenges," *Frontiers in Neuroscience*, vol. 17, p. 1187252, 2023.

[54] J. Timcheck, S. B. Shrestha, D. Ben Dayan Rubin, A. Kupryjanow, G. Orchard, L. Pindor, T. Shea, and M. Davies, "The intel neuromorphic dns challenge," *Neuromorphic Computing and Engineering*, vol. 3, no. 3, p. 034005, aug 2023. [Online]. Available: https://dx.doi.org/10.1088/2634-4386/ace737

[55] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva, "Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 3, pp. 317–329, 2016.

[56] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, Apr. 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[57] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 2, Oct. 2020. [Online]. Available: https://doi.org/10.1145/3418498

[58] L. Xie, J. Xue, L. Wu, F. Chen, Q. Tian, Y. Zhou, R. Ying, and P. Liu, "Spikenc: An accurate and scalable simulator for spiking neural network on multi-core neuromorphic hardware," in *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2023, pp. 357–366.

[59] Y. Xu, K. Shidqi, G.-J. van Schaik, R. Bilgic, A. Dobrita, S. Wang, R. Meijer, P. Nembhani, C. Arjmand, P. Martinello, A. Gebregiorgis, S. Hamdioui, P. Detterer, S. Traferro, M. Konijnenburg, K. Vadivel, M. Sifalakis, G. Tang, and A. Yousefzadeh, "Optimizing event-based neural networks on digital neuromorphic architecture: a comprehensive design space exploration," *Frontiers in Neuroscience*, vol. Volume 18 - 2024, 2024. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2024.1335422

[60] Y. Yan, H. Chu, Y. Jin, Y. Huan, Z. Zou, and L. Zheng, "Backpropagation with sparsity regularization for spiking neural network learning," *Frontiers in Neuroscience*, vol. 16, 2022. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.760298

[61] Y. Yang, Q. Fan, T. Yan, J. Pei, and G. Li, "Network group partition and core placement optimization for neuromorphic multi-core and multi-chip systems," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 6, pp. 3966–3981, 2024.

[62] J. Yik, K. Van den Berghe, D. den Blanken, Y. Bouhadjar, M. Fabre, P. Hueber, W. Ke, M. A. Khoei, D. Kleyko, N. Pacik-Nelson, A. Pierro, P. Stratmann, P.-S. V. Sun, G. Tang, S. Wang, B. Zhou, S. H. Ahmed, G. Vathakkattil Joseph, B. Leto, A. Micheli, A. K. Mishra, G. Lenz, T. Sun, Z. Ahmed, M. Akl, B. Anderson, A. G. Andreou, C. Bartolozzi, A. Basu, P. Bogdan, S. Bohte, S. Buckley, G. Cauwenberghs, E. Chicca, F. Corradi, G. de Croon, A. Danielescu, A. Daram, M. Davies, Y. Demirag, J. Eshraghian, T. Fischer, J. Forest, V. Fra, S. Furber, P. M. Furlong, W. Gilpin, A. Gilra, H. A. Gonzalez, G. Indiveri, S. Joshi, V. Karia, L. Khacef, J. C. Knight, L. Kriener, R. Kubendran, D. Kudithipudi, S.-C. Liu, Y.-H. Liu, H. Ma, R. Manohar, J. M. Margarit-Taulé, C. Mayr, K. Michmizos, D. R. Muir, E. Neftci, T. Nowotny, F. Ottati, A. Ozcelikkale, P. Panda, J. Park, M. Payvand, C. Pehle, M. A. Petrovici, C. Posch, A. Renner, Y. Sandamirskaya, C. J. S. Schaefer, A. van Schaik, J. Schemmel, S. Schmidgall, C. Schuman, J.-s. Seo, S. Sheik, S. B. Shrestha, M. Sifalakis, A. Sironi, K. Stewart, M. Stewart, T. C. Stewart, J. Timcheck, N. Tömen, G. Urgese, M. Verhelst, C. M. Vineyard, B. Vogginger, A. Yousefzadeh, F. T. Zohora, C. Frenkel, and V. J. Reddi, "The neurobench framework for benchmarking neuromorphic computing algorithms and systems," *Nature Communications*, vol. 16, no. 1, p. 1545, Feb. 2025.

[63] X. Yu and C. Tian, "Dual sparse training framework: inducing activation map sparsity via transformed $\ell 1$ regularization," 2024. [Online]. Available: https://arxiv.org/abs/2405.19652