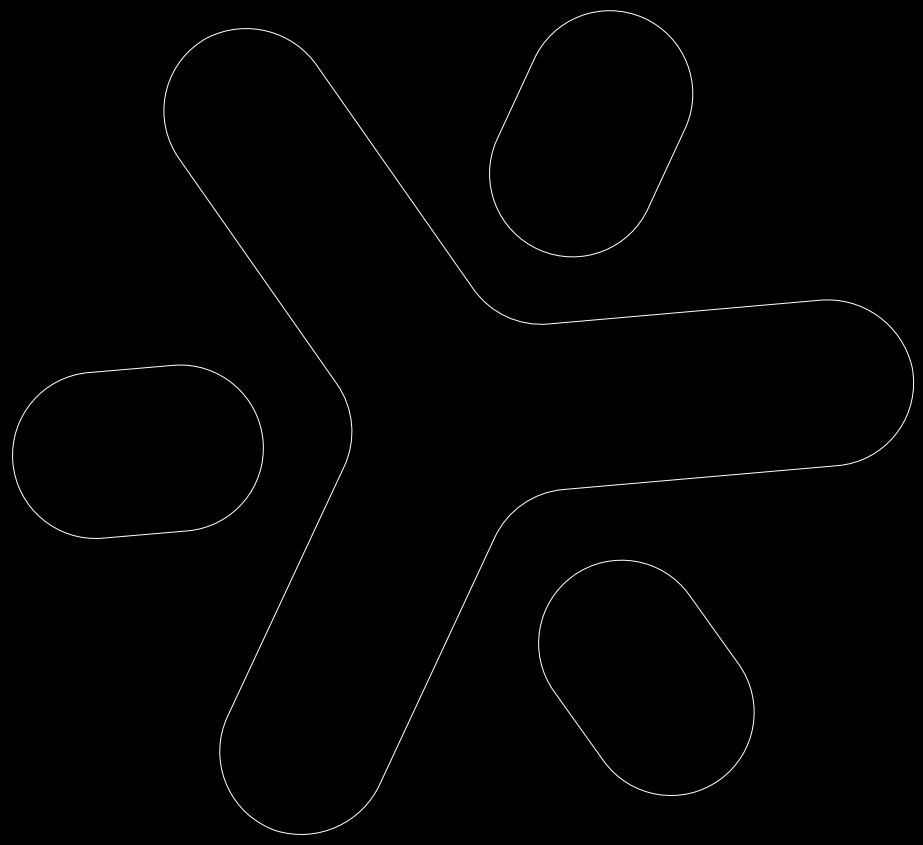
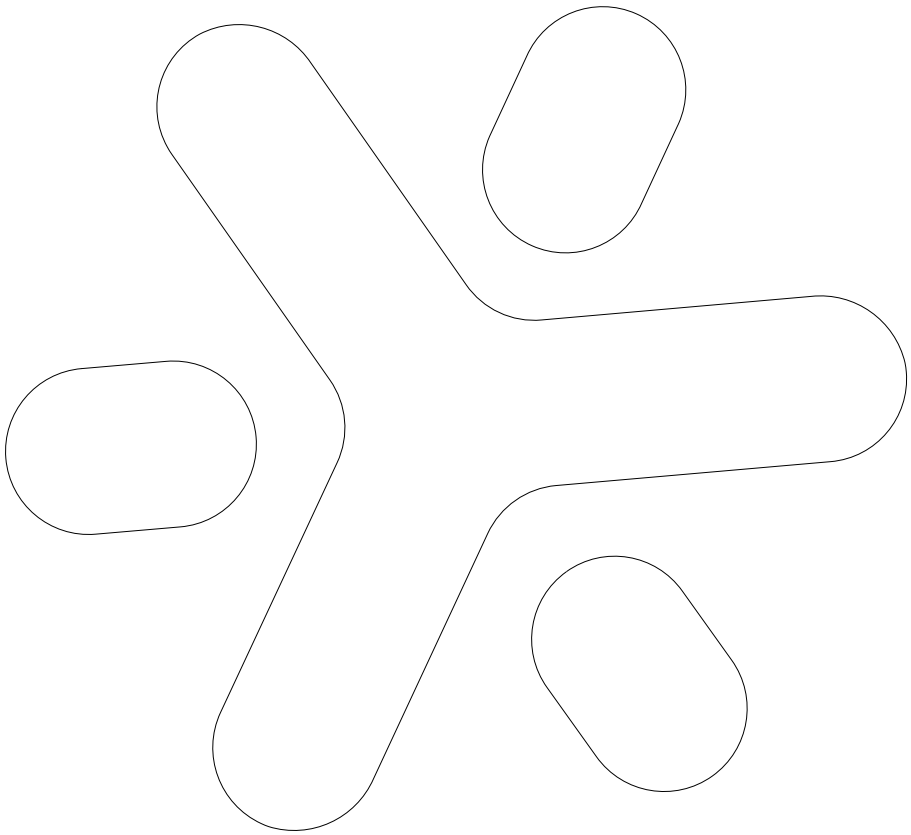
brainchip™

Activation Sparsity: Unlocking Efficient Deep Learning at Scale

V1 Last Updated:
18 March 2026



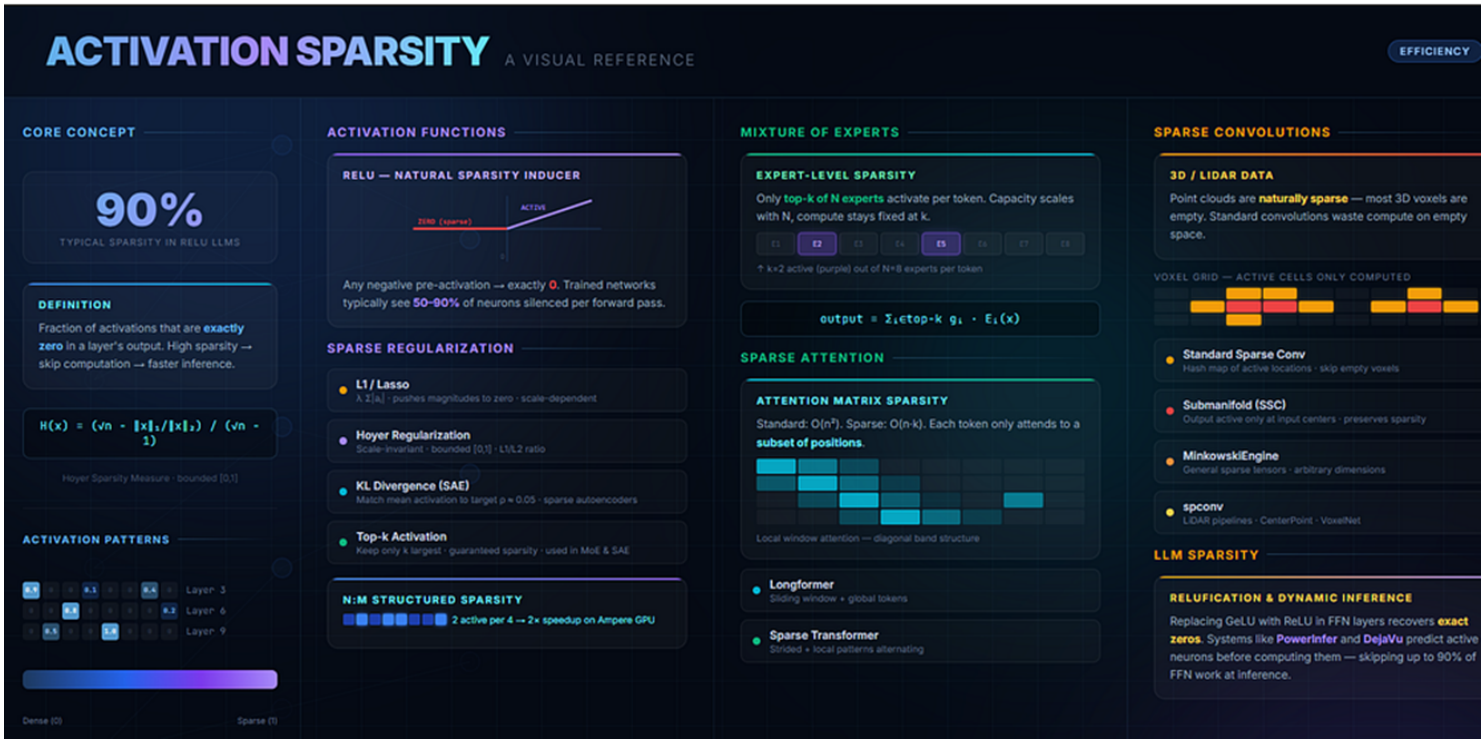
Contents



Introduction

Modern deep neural networks are powerful but computationally expensive. A forward pass through a large model involves billions of multiply-accumulate operations, most of which involve activations that are very small or exactly zero. This observation motivates **activation sparsity** — the idea that if most activations in a network are zero at any given time, we can skip the corresponding computations entirely, yielding significant speedups and memory savings without sacrificing accuracy.

Activation sparsity is distinct from *weight sparsity* (pruning), which removes parameters from the model permanently. Instead, activation sparsity is *dynamic* — it varies per input, per layer, and per forward pass. A network can be dense in its weights but still exhibit highly sparse activations, and this sparsity can be exploited at inference time.



A poster summary of this article

Chapter 1

What Is Activation Sparsity?

Given a layer's output tensor $\mathbf{a} \in \mathbb{R}^n$, the activation sparsity is simply the fraction of elements that are zero (or near-zero):

$$\text{sparsity} = \frac{|\{ i : a_i = 0 \}|}{n}$$

A sparsity of 0.9 means 90% of activations are zero — only 10% carry meaningful signal. In such a regime, any downstream computation multiplied by a zero activation is wasted work.

Why Does Sparsity Arise Naturally?

The most common activation function in modern networks — **ReLU (Rectified Linear Unit) — is defined as:**

$$\text{ReLU}(x) = \max(0, x)$$

ReLU is a natural sparsity inducer: any negative pre-activation becomes exactly zero. In practice, well-trained networks with ReLU activations routinely exhibit 50–90% activation sparsity across layers. This is not a bug — it is a feature. Sparse activations are associated with more disentangled, interpretable representations and improved generalization.

Chapter 2

Methods to Encourage or Exploit Activation Sparsity

1. ReLU and Variants

The simplest way to get sparse activations is to use ReLU. Unlike sigmoid or tanh, which squash values but rarely produce exact zeros, ReLU hard-zeros all negative inputs.

Variants that modulate sparsity:

- **Leaky ReLU / PReLU:** Allow small negative values through, reducing sparsity slightly but avoiding dead neurons.
- **ReLU6:** Clips activations at 6, useful for quantization but similar sparsity to ReLU.
- **Dying ReLU problem:** If a neuron's pre-activation is always negative, it produces zero for every input and never updates — it "dies." This is a degenerate form of sparsity that hurts capacity.

2. Top-k Activation

Rather than letting sparsity emerge naturally, **top-k activation** enforces it explicitly. After computing a layer's output, only the top-k largest activations are kept; the rest are zeroed out.

$$a_i^{sparse} = \begin{cases} a_i & \text{if } a_i \in \text{top-}k\ (a) \\ 0 & \text{otherwise} \end{cases}$$

This guarantees a fixed sparsity level regardless of input. Top-k is widely used in:

- **Mixture of Experts (MoE):** Only the top-k experts are activated per token, keeping compute constant as model capacity scales.
- **Sparse attention mechanisms:** Only attending to the top-k most relevant tokens.
- **k-sparse autoencoders:** A key tool in mechanistic interpretability research (e.g., Anthropic's sparse autoencoders), where top-k activation enforces a fixed number of active features per input.

3. Mixture of Experts (MoE)

MoE layers replace a single dense feed-forward network with a collection of N expert networks, of which only k are activated per input token. A router (a small learned network) assigns each token to its top-k experts.

$$\text{output} = \sum_{i \in \text{top-}k} g_i \cdot E_i(\text{x})$$

where g_i are the gating weights and E_i are the expert networks.

Benefits:

- Model capacity scales with N experts, but compute scales with k (typically k=2).
- Extremely sparse activation at the expert level — only 2/N experts fire per token.

Challenges

- **Load balancing:** Without auxiliary losses, the router tends to collapse onto a few popular experts. Techniques like auxiliary load-balancing losses and expert capacity constraints address this.
- **Communication overhead** in distributed settings, where experts may reside on different devices.

MoE is used in production at scale — models like Mixtral, GPT-4 (reportedly), and Switch Transformer all use MoE layers.

4. Sparse Regularization on Activations

Rather than forcing sparsity architecturally, regularization can encourage it during training by penalizing dense activations.

L1 regularization on activations:

$$\mathcal{L}_{\text{reg}} = \lambda \sum |a_i|$$

Penalizing the absolute magnitude of activations pushes many of them toward zero.

- **Hoyer regularization:** As discussed in the sparsity literature, the Hoyer measure can be applied to activation vectors rather than weights, encouraging a high ratio of L1 to L2 norm — a signature of sparsity — in a scale-invariant way.
- **KL-divergence sparsity (sparse autoencoders)**: In autoencoders, a KL divergence penalty encourages the mean activation of each hidden unit to match a target low probability ρ (e.g., 0.05):

$$\mathcal{L}_{\text{KL}} = \sum_j \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

This is the classic **sparse autoencoder** formulation from Ng et al., now widely used in mechanistic interpretability to discover monosemantic features in language models.

5. Sparse Attention

In standard Transformer self-attention, every token attends to every other token — an $O(n^2)$ operation. **Sparse attention** restricts each token to attend to only a subset of positions, introducing activation sparsity in the attention matrix.

Patterns of sparse attention:

- **Local/sliding window attention (Longformer):** Each token attends to a fixed window of neighboring tokens.
- **Strided attention (Sparse Transformer):** Alternating layers use local and strided patterns.
- **Learned sparse attention (Routing Transformer, Reformer):** The model learns which tokens to attend to, using clustering or hashing.
- **Top-k attention:** Retain only the top-k attention weights per query, zeroing out the rest.

Sparse attention reduces memory and compute from $O(n^2)$ to $O(n \log n)$ or $O(n \cdot k)$, enabling processing of much longer sequences.

6. Sparse Convolutions

In domains like **3D point clouds**, **LiDAR data**, and **medical imaging**, input data is naturally sparse — the majority of voxels or spatial locations are empty. **Standard convolutions** waste compute by processing these empty regions. Sparse convolutions only compute outputs at locations where the input is non-zero.

How sparse convolution works:

A standard convolution slides a kernel across all spatial positions. A sparse convolution maintains a **hash map** of active (non-zero) input locations and only computes outputs for positions in the kernel's receptive field of active inputs.

$$y[\mathbf{p}] = \sum_{\mathbf{k}} W[\mathbf{k}] \cdot x[\mathbf{p} + \mathbf{k}] \quad \text{only if } \mathbf{p} \text{ is active}$$

Key implementations:

- **MinkowskiEngine:** A general-purpose sparse tensor library for sparse convolutions on arbitrary-dimensional data.
- **spconv (Sparse Convolution library):** Widely used in autonomous driving pipelines (e.g., CenterPoint, VoxelNet).
- **TorchSparse:** Optimized sparse convolution for mobile and embedded inference.

Submanifold Sparse Convolutions (SSC): A variant where output sparsity mirrors input sparsity — an output location is active only if the center of the kernel falls on an active input location (not just any position in the receptive field). This preserves sparsity across many layers, preventing the "densification" problem where sparse inputs gradually produce dense outputs.

Applications:

- 3D object detection in autonomous vehicles (LiDAR point clouds)
- Volumetric medical image segmentation (CT/MRI scans)
- Indoor scene understanding
- Event camera data processing

7. Dynamic Sparse Inference (N:M Sparsity)

Hardware vendors, particularly NVIDIA with their **Ampere and Hopper architectures**, have introduced support for **structured N:M sparsity** — a pattern where exactly N out of every M consecutive values are non-zero (commonly 2:4 sparsity).

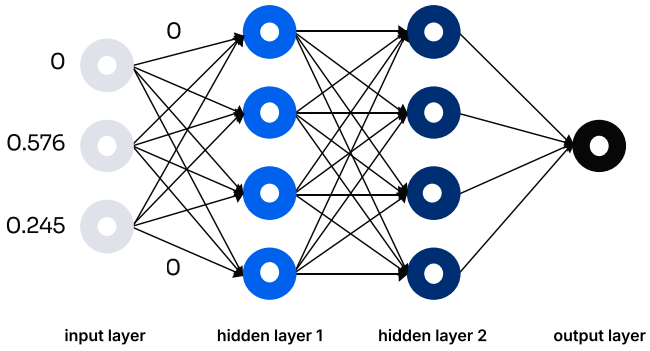
This structured pattern allows the hardware's sparse tensor cores to skip zero-valued computations in a predictable, hardware-friendly way, achieving up to **2× speedup** on matrix multiplications with minimal accuracy loss.

While primarily a weight sparsity technique, it can also be applied to activations when combined with appropriate quantization and pruning schedules during training.

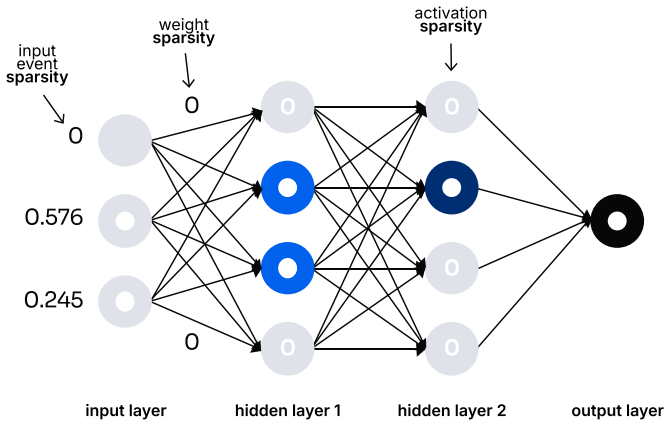
Hardware Considerations

Exploiting activation sparsity for real speedups requires hardware that can efficiently skip zero computations. This is non-trivial:

- **Dense GPU kernels** process all elements uniformly; zero multiplications are still computed.
- **Sparse BLAS routines** (e.g., cuSPARSE) help but have high overhead for moderate sparsity levels — benefits typically only materialize above ~90–95% sparsity.
- **Sparse tensor cores (NVIDIA Ampere+)** handle structured sparsity natively.
- **Neuromorphic and custom AI chips** (e.g., Cerebras, Groq, BrainChip Akida) are designed from the ground up to benefit from activation sparsity.
- **Event-driven architectures** (spiking neural networks) only communicate when a neuron "fires," making sparsity a first-class citizen.



Dense Execution



Sparse (Event-Driven) Execution

Chapter 3

Activation Sparsity in Large Language Models

Recent work has revealed that large language models exhibit substantial activation sparsity in their feed-forward (FFN) layers. The FFN in a Transformer typically uses ReLU or GeLU activations, and studies have found:

- **In ReLU-based FFN layers**, sparsity can exceed **90%** per token.
- **Even GeLU** (which is smooth and doesn't produce exact zeros) shows **near-zero** activations for a large fraction of neurons.
- The **"Relufication"** trend — replacing GeLU with ReLU in LLMs — has been explored (e.g., in Falcon, and proposed in papers like "ReLU Strikes Back") specifically to recover exact sparsity and enable sparse inference.

This has spawned practical systems like **PowerInfer** and **DejaVu**, which predict which neurons will be active before computing them, and skip the inactive ones — achieving significant inference speedup on commodity hardware.

Summary

Method	Sparsity Type	Key Use Case
ReLU activation	Natural, dynamic	General deep learning
Top-k activation	Enforced, dynamic	MoE, sparse autoencoders
Mixture of Experts	Expert-level	Scalable LLMs
Sparse regularization	Encouraged	Representation learning
Sparse attention	Position-level	Long-context Transformers
Sparse convolution	Spatial	3D/LiDAR perception
N:M structured sparsity	Structured	Hardware-accelerated inference

Activation sparsity sits at the intersection of efficiency, interpretability, and biological plausibility. As models grow larger and deployment constraints tighten, leveraging sparsity — whether naturally occurring or deliberately induced — will remain one of the most important levers for scalable, efficient deep learning.

Documentation:

https://brainchip.com/technology/

https://brainchip.com/akida-exploits-sparsity-for-low-power-in-neural-networks/

https://htor.inf.ethz.ch/sparsity-in-dl/

Slides created with the assistance of Claude

Reference for Sparse AutoEncoder

Biography

Ali Kayyam, Principal Research Scientist

Ali Kayyam is currently a principal research scientist at BrainChip.

He received his BS and MS degrees in computer engineering from the Petroleum University of Technology, Tehran, Iran, 2001 and Shiraz University, Shiraz, Iran, 2004, respectively. He received his PhD degree in computational neurosciences from the Institute for Studies in Fundamental Sciences (IPM) in Tehran, 2009.

He was a postdoctoral scholar at iLab, University of Southern California, Los Angeles from March 2010 to August 2014. He was an assistant professor at the University of Wisconsin, Milwaukee, and then at the University of Central Florida before joining industry in 2018.

His research interests include computer vision, machine learning, and neurosciences with particular emphasis on visual attention, visual search, active learning, scene and object recognition, and biologically plausible vision models.

brainchip™

V1 Last Updated:
18 March 2026