Accelerating LLM Inference Using Sparsity

Kira Selby, Varun Khare

NimbleEdge

October 2025

1 Introduction

Large Language Models (LLMs) have quickly grown to dominate the machine learning space, and are rapidly rising in popularity in consumer applications across a wide range of fields. The power of these models can be staggering, but they come at a commensurate cost. Frontier models have hundreds of billions of parameters, and can only be run using specialized GPU farms, whose electricity consumption is so high that it is actively reshaping power grid infrastructure [Chen et al., 2025, Bogmans et al., 2025]. Even deploying smaller models for local applications requires a great deal of engineering and logistics in order to manage the memory and computation requirements needed to run these models efficiently.

As such, performance optimizations have become absolutely vital in making practical and effective use of the power that LLMs grant. Gone are the days when production-grade models and libraries could use high-level code, with uncompressed weights. Now, bespoke GPU kernels, low-rank adapters, and low-precision quantized weights are the norm wherever LLMs are deployed. In low-resource environments, however, even this may not be enough. When it comes to edge computing and on-device inference, squeezing every last bit of performance out of limited hardware is of utmost importance. As such, sparsity is becoming an increasingly important part of the conversation when it comes to optimization.

The high sparsity of neural networks is a property that has been observed ever since the beginning of deep learning as a field [Han et al., 2015, Frankle and Carbin, 2019] - but in the modern era, it is more important than ever. Transformers in particular have been the subject of an immense body of research in terms of sparsity. This paper will not seek to summarize the whole history of this body of research, but will instead narrow its focus on the most promising modern techniques for harnessing sparsity in LLM inference, with a particular focus on low-resource settings. We will discuss promising future directions of

research, and outline possible ways in which these existing techniques can be combined to leverage sparsity to its fullest for on-device computation.

At NimbleEdge, our mission is to make production-grade LLM capabilities accessible in edge computing environments. We believe the path forward lies not in any single technique, but in the systematic integration of complementary sparsity methods into a cohesive framework. This paper synthesizes the current state of the art, outlines our own contributions and areas of active research and proposes a practical roadmap for achieving this vision.

2 Notation

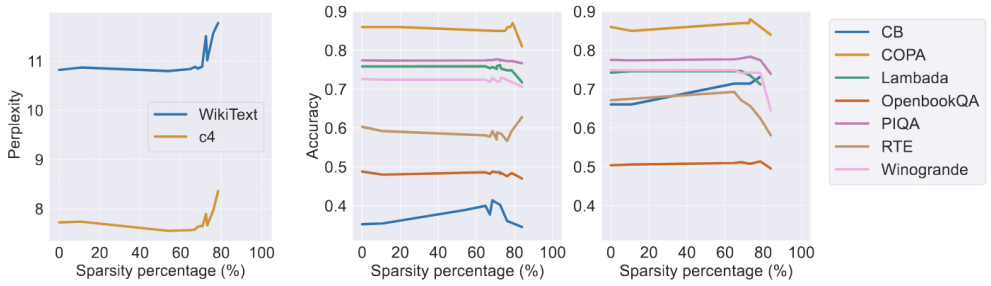
The transformer architecture used by all frontier LLMs consists of two primary components: an attention block, and a feedforward multi-layer perceptron (MLP) block. We shall discuss both sparse MLP and sparse attention methods, and thus this section will seek to establish a consistent notation for each going forward.

Multi-Headed Attention Given an input $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$, $\mathbf{x_t} \in \mathbb{R}^d$, the multi-headed attention (MHA) operation constructs parallel sets of *queries*, *keys* and *values* using n_h learned matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_h}$. The attention is then computed for each head in parallel, and the output is assembled using $W^O \in \mathbb{R}^{d \times d}$.

$$\begin{aligned} Q_i, K_i, V_i &= XW_i^Q, XW_i^K, XW_i^V \\ \text{MHA}(X) &= \sum_{i=1}^{n_h} \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_h}}\right) V_i W_i^O \end{aligned} \tag{1}$$

MLP After each attention block, an element-wise MLP is added to process each token representation individually. These were originally simple feedforward networks consisting of an "up" projection $W_{\text{up}} \in \mathbb{R}^{d \times d_{\text{ff}}}$, followed by a "down" projection $W_{\text{down}} \in \mathbb{R}^{d_{\text{ff}} \times d}$. Over time, however, they have increasingly become replaced by more sophisticated gated operators, which separate the up projection into an elementwise product of two components using an additional weight matrix $W_{\text{gate}} \in \mathbb{R}^{d \times d_{\text{ff}}}$.

$$\text{FFN}(X) = \left(\sigma_{\text{act}}(XW_{\text{gate}}) \odot XW_{\text{up}}\right) W_{\text{down}} \tag{2}$$



(a) Language Modeling (b) Zero-Shot(Left). Five-Shot(Right)

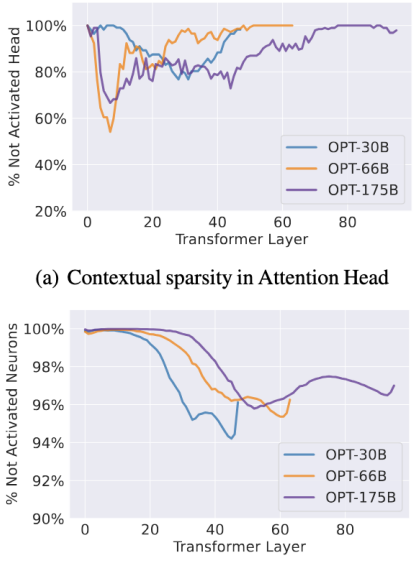
Figure 1: Accuracy of OPT using sparse predictors (figure taken from Liu et al. [2023])

3 Contextual Sparsity

Meta's OPT models may no longer be among the ranks of major players in frontier-grade LLMs, but they nevertheless provide a valuable case study in the potential applications of dynamic sparsity. A breakthrough paper by Liu et al. [2023] demonstrated that **over 95%** of weights in OPT's multilayer perceptron blocks were not activated for an average input token - in some cases rising even to 99% or higher for early layers! These numbers are staggering and, if leveraged properly, should give rise to massive performance gains. Between the authors of that paper, as well as followup papers such as Alizadeh et al. [2024], methods were proposed to do just that by exploiting **contextual activation sparsity**.

The main challenge for these approaches was to find ways to efficiently predict which neurons would be active in a given layer, without needing to compute the full layer. Liu et al. [2023] proposed to train lightweight predictor networks that acted as low-rank factorizations of the up projection matrix W_{up}. These "sparse predictors" would efficiently predict which neurons within the MLP layers would be active, allowing the use of sparse matrix multiplications to greatly accelerate the computation of the MLP block. Similar methods were also used to predict which attention heads contributed significantly to the overall output, and allow only the relevant heads to be used in the final computation.

When combined with various hardware optimizations and asynchronous look-ahead execution of the sparse predictor layers, this approach (known



(a) Contextual sparsity in Attention Head

(b) Contextual sparsity in MLP Block

Figure 2: Plots of contextual sparsity in OPT (figures taken from Liu et al. [2023])

as “Deja Vu”) led to 2-6x speedups in inference speed with little to no drop in accuracy (see Fig. 1 for details on accuracy results by sparsity level). This approach was quickly adopted by other researchers, and iterated on by papers such as LLM in a Flash [Alizadeh et al., 2024], PowerInfer [Song et al., 2023] and PowerInfer2 [Xue et al., 2024].

3.1 Neuron Concentration

One key phenomenon identified by Liu et al. [2023] and later expanded upon by [Song et al., 2023] is the fact that **neuron outputs are highly correlated**. The small fraction of neurons that activate for each given input is not random - far from it. Instead, a small group of “hot” neurons are active for almost all inputs, while the remaining “cold” neurons are active only very rarely.

This neuron concentration and correlation has been a subject of ongoing research here at NimbleEdge. We would like to unveil our own contribution to this research: a dynamic weight caching and paging operator for streaming weights in sparse operations in PyTorch. We see significant reduction in matrix multiplication for MLP[2] layers between subsequent model passes.

A naive implementation of contextual sparsity must perform full index-select operation at each step. These gather operations are memory intensive and i/o bound. If there is a high correlation between active weights at subsequent model passes, this process is wasteful. Instead, the previously active weights can be kept loaded in a cache, and only the *difference* between subsequent masks m_{t-1} and m_t needs to be loaded into memory.

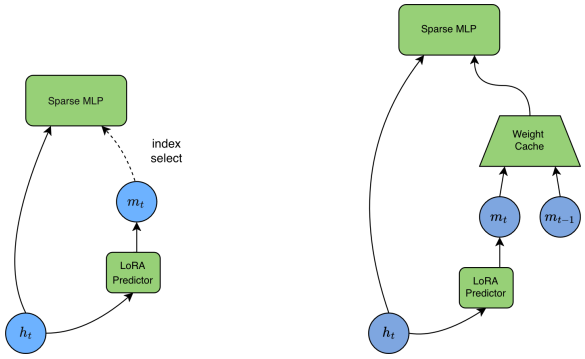


Figure 3: Left diagram shows a naive sparse mlp implementation, with full index select operation at each step. Right shows our modified version with weight caching.

We implement our weight cache using memcpy operations in c++, with up-projection weights stored in row-major format and down projection weights stored in column-major format. Our preliminary experiments validate this hypothesis, demonstrating up to 6.7x speedups in isolated index select operations compared to naive PyTorch implementations (from 29.89ms down to 4.46ms). Figure 4 shows the theoretical overall speedup from the weight cache for

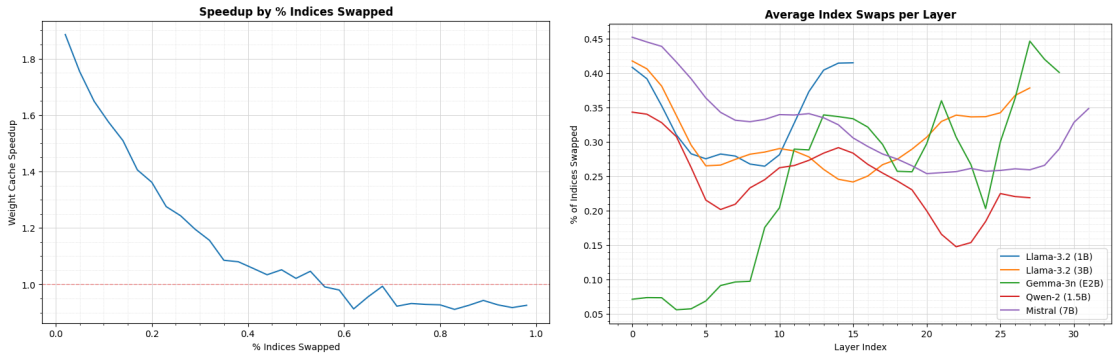


Figure 4: Overall speedup in MLP block operations as a function of index swaps per update. Right plot shows average index swaps for common models using sparsities derived from Fig 6.

the full sparse MLP computation as a function of the percent of index swaps at each step. Using custom fused sparse kernels and Torch OpenMP parallelization, we can see up to 5x faster CPU MLP inference compared to naive baselines (from 30.1ms down to 6.02ms).

4 Challenges

Unfortunately, these dynamic sparsity methods have a critical flaw. While OPT-175b displayed staggering sparsity levels that exceeded 99% in some layers, more recent models such as Llama, Mistral, or Gemma show vastly lower levels of sparsity. These newer models all have one thing in common - they all use activation functions such as SiLU or GeLU instead of the ReLU activation used by OPT. As shown in Fig. 5, these activation functions have long tails that extend smoothly well past zero - unlike ReLU, which imposes a hard cutoff. As such, ReLU naturally promotes highly sparse behaviour that SiLU or GeLU do not. Leveraging sparsity in networks with long-tail activations is a far more difficult prospect, as determining which neurons contribute negligibly to the overall output is far less clear. Sparsity values in these networks tend to be much lower as a result, and naive approaches can often face severe accuracy penalties when attempting to obtain sparse representations by discarding activations within the tail.

So is activation sparsity dead in the modern era? Not at all. Two major schools of thought have emerged as possible solutions to this challenge. The first class of solutions, known as "relufication", involves using fine-tuning to reintroduce ReLU-based activations and improve sparsity. The second is to eschew expensive training, and find ways to leverage sparsity even when using long-tailed activations. We will discuss each in turn.

Relufication If the problem is that modern models do not use ReLU activations, then the most obvious solution is to simply reintroduce ReLU within those models. By surgically swapping the activation functions from SiLU or GeLU back to ReLU, and then fine-tuning

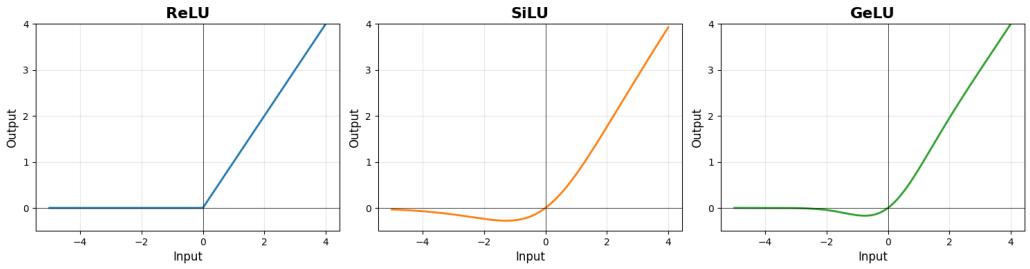


Figure 5: Plots of activation curves for ReLU, SiLU and GeLU.

on a 15 billion token corpus to restore performance, Mirzadeh et al. [2023] were able to regain 60% sparsity for Llama-7B, while sacrificing only around 1% accuracy across an array of 9 different zero-shot datasets. Many other papers then took up this mantle, and pushed these results even further. Song et al. [2025] observed that this 60% sparsity was still far below the >95% sparsity demonstrated in OPT, and proposed a multi-phase training programme incorporating auxiliary losses that help promote sparse activations. Their results on Llama2 (7B and 13B) show sparsities of almost 90%, with no appreciable drop in accuracy. Song et al. [2024] achieve similar results, reaching sparsities of 85-90% on Mistral-7B and -47B after training for 150 billion tokens using a modified gated structure where ReLU is applied to both the gate and up projections. Pretrained weights are currently available for ReluLlama (based on Llama-2 7B or 70B), ProSparse (based on Llama-2 7B or 13B), and TurboSparse (based on Mistral/Mixtral 47B).

But these approaches still have clear drawbacks. When seeking to deploy models for inference, being limited to only a handful of available models is restricting, and employing the same programmes to fine-tune other models is costly. Ideally, we would wish for methods that could recover sparsity in non-RELU-based LLMs *without* requiring this expensive fine-tuning.

Training-Free Approaches Without the hard threshold imposed by RELU, imposing sparsity becomes more complicated. Simply thresholding at zero (as ReLU does) leads to very poor results. Instead, most training-free approaches involve computing layerwise thresholds values applied to the activation norm for which the approximation error is within acceptable tolerances. To see this in practice, let us rewrite the equation for a gated FFN in a modified form by summing over the contributions from each individual neuron. We can then restrict our sum to only an index set $\mathcal{I}$ of "active" neurons, and thus arrive at a sparse formulation of the gated MLP operation:

$$\text{FFN}(X) = \sum_{i \in \mathcal{I}} \big(\sigma_{\text{act}}\left(XW_{\text{gate}}[:, i]\right) \odot \left(XW_{\text{up}}[:, i]\right)\big) W_{\text{down}}[i, :]$$

The index set $\mathcal{I}$ is typically found by fixing an activation threshold τ, and zeroing out all components whose activation norms fall below the threshold value. The first major approach

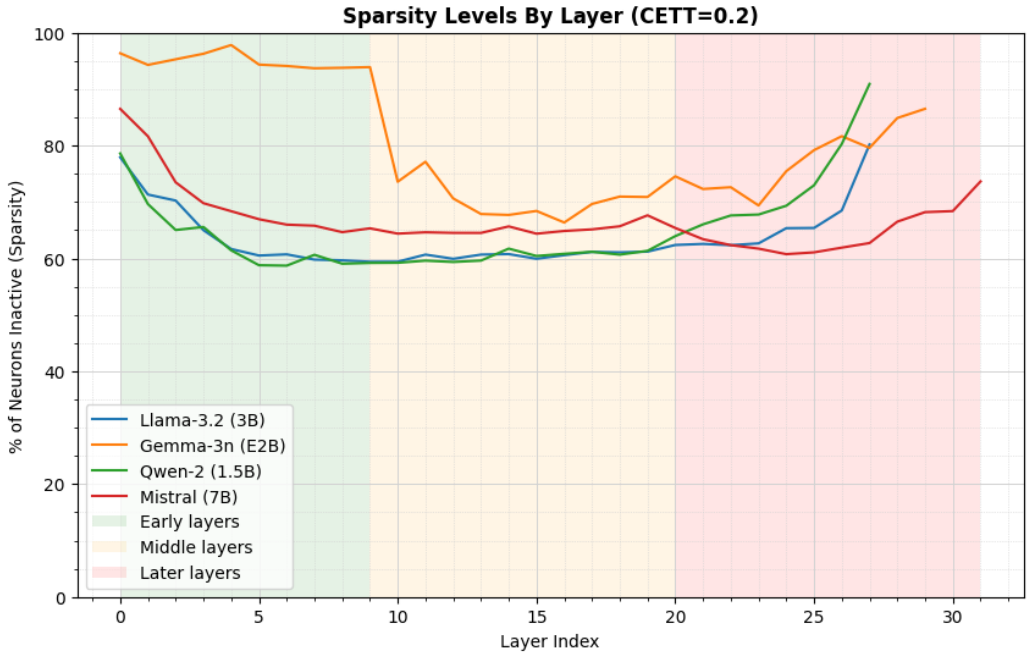


Figure 6: Plots of sparsity for common models using CETT-based thresholding (CETT=0.2). Note that the unusual behaviour of Gemma-3n on layers 0-9 is due to its own internal sparsity mechanism in those layers.

to use this framework was "Contextually Aware Thresholding Sparsity, or "CATS" [Lee et al., 2024], which proposed to apply the threshold τ to the norm of the gate activation output $|\sigma_{\text{act}}(XW_{\text{gate}})|$, and to find the threshold by precomputing these activation norms over a fixed dataset. By fixing a target sparsity p, they would take τ to be the p-th percentile of activation norms on the precomputed data. Using this approach, they were able to achieve 50% sparsity in Llama-2 7B and Mistral 7B without significant loss of downstream accuracy.

The major flaw in this initial formulation was that CATS only considered the output contribution of a given neuron to be based on the gate activation value $|\sigma_{\text{act}}(XW_{\text{gate}})|$. With a long-tailed activation, small but nonzero activations may be scaled by larger contributions from the 'up' and 'down' projections, and thus cannot simply be neglected. Zhang et al. [2024] proposed a corrected version of this approach which they named "Cumulative Errors of Tail Truncation" (or CETT), which incorporates the contributions of these terms.

CETT CETT proposed to apply the threshold τ to the full output contribution of each individual neuron, rather than solely the gate activations. Given the output vector n_i associated with the contribution from neuron i, the learned threshold τ would be applied to the L_2 norms $||n_i||_2$ in order to determine active neurons.

$$n_i(X) = \left(\sigma_{\text{act}}\left(XW_{\text{gate}}[:, i]\right) \odot \left(XW_{\text{up}}[:, i]\right)\right) W_{\text{down}}[i, :]$$

The CETT metric itself was defined by taking the sum over all neglected neurons whose outputs fell below a given threshold, and computing the fraction of the total MLP output norm lost using that sparse approximation.

$$\text{CETT}(X;\tau) = \frac{||\sum_{j\in\mathcal{D}} n_j(X)||_2}{||\sum_{j=1}^{d_{\text{ff}}} n_j(X)||_2} \quad \mathcal{D} = \{i|\ ||n_i||_2 < \tau\}$$

This metric could then be employed in reverse: by fixing an "acceptable" deviation in output norm (often 20%, or CETT=0.2), the threshold τ (and resulting sparsity ratio) can then be computed using binary search to keep the network within acceptable error bounds. We ran our own experiments using this approach (results shown in Figure 6) and found the results to be extremely promising: using the standard threshold of 0.2 CETT, we were able to recover upwards of 60% sparsity across four different commonly used models.

5 Recent Advances

By combining the advances in sparse inference detailed in Section 3 with the relufication and training-free approaches to reintroduce sparsity in modern LLMs discussed in Section 4, we can begin to see a clear path forward to leveraging sparsity in order to greatly accelerate LLM inference across the board. But the story doesn't end there. New ways to leverage sparsity are emerging with every passing month, and major model developers such as DeepSeek and Google have made some particularly interesting advancements which warrant discussion.

In particular, DeepSeek v3.2 [DeepSeek-AI, 2025] and Google's SparkTransformer [You et al., 2025] showcase several transferable techniques for leveraging sparsity at scale. While these models were trained with massive computational budgets, many of their core innovations can be adapted to more modest settings. We will take a look at each of these in turn, and examine how their findings can be adapted to further improve sparse inference.

5.1 Case 1: Deepseek Sparse Attention

The bulk of this work has focused on sparsity in transformer MLP layers, but finding ways to leverage sparsity within the attention layer is just as important! Deepseek has been responsible for a number of major revolutions in the structure of the attention mechanism, from the Multiheaded Latent Attention (MLA) of Deepseek v2 DeepSeek-AI [2024], to the "Deepseek Sparse Attention" (DSA) mechanism they recently unveiled with the release of Deepseek v3.2 [DeepSeek-AI, 2025].

The DSA mechanism (shown in Fig. 7) is complex and utilizes many different components, including compressed latent representations from MLA, modified RoPE (Rotary Positional Embeddings) implementations, and the single-headed keys and values from Multiheaded

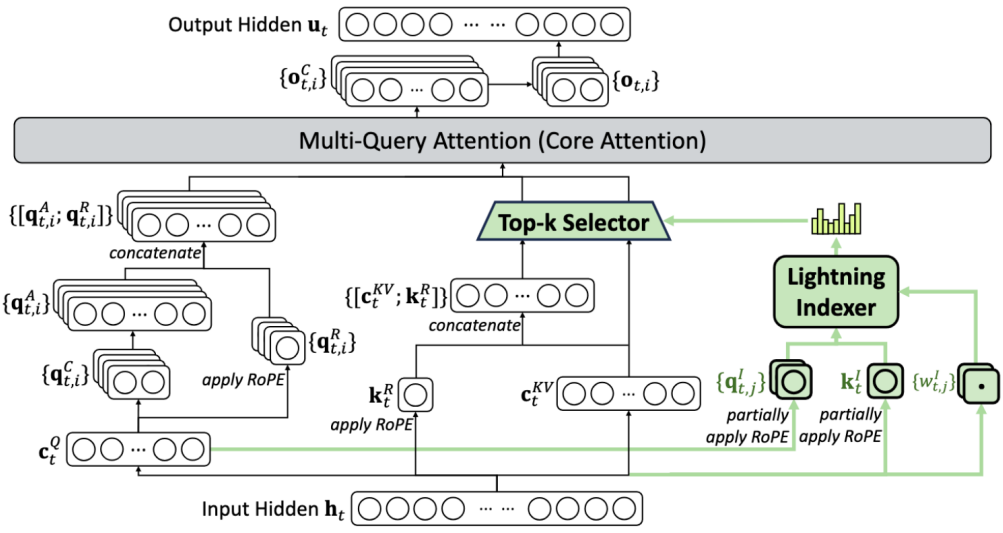


Figure 7: Architecture of Deepseek Sparse Attention (figure taken from DeepSeek-AI [2025])

Query Attention Shazeer [2019]. But the key innovation behind their sparse approach is surprisingly simple - and follows very similar lines to approaches we have already discussed.

The primary component of their sparse attention mechanism is a lightweight predictor known as the "lightning indexer", which is very similar in spirit to the sparse predictors proposed in Liu et al. [2023].

$$I_t = \sum_{j=1}^{n_I} w^I_{t,j} \text{ReLU}(q^I_{t,j} {K^I}^T) \tag{3}$$

The lightning indexer essentially computes a version of the full attention mechanism in an efficient, scaled-down fashion. It uses a much smaller number of heads, a simple RELU activation and low-precision quantization to compute an approximation to the full attention energies as efficiently as possible. These approximated energies would be far too inaccurate to use as a replacement for the full attention matrix, but by finding the top-k entries corresponding to each query $C_t = \{c_s | s \in \text{Top-k}(I_t)\}$, this lightning indexer can predict a sparse mask that can then be used by the full attention mechanism in order to narrow its search window from the 100,000+ tokens of the full context window to only the most relevant few (2048 by default).

This division of concerns between a full quadratic attention computation using the lightweight indexer combined with a constant-size window for the more powerful attention mechanism results in vastly accelerated inference in long-context settings. While DeepSeek builds this approach on top of their 671B parameter flagship "Terminus" model, there's no reason to believe the same techniques can't be leveraged in much smaller settings. The main barrier

that needs to be overcome here is training - the indexer itself used a very small training set (relative to the size of Deepseek 3.1's corpus), but this was followed by fine-tuning the full model with sparse attention incorporated for another 950B tokens. This is obviously not feasible for local inference setups - but there might be ways to scale down this fine-tuning to manageable levels, or even use a similar approach in a training-free fashion.

5.2 Case 2: Spark Transformer

The other major step forward to discuss is Google's Spark Transformer [You et al., 2025], which also leverages sparsity in novel ways to accelerate computation. Their approach involves defining sparse operators "SparkAttention" and "SparkFFN", which function somewhat similarly to the lightweight sparse predictors we have discussed previously.

The key difference is in how the sparse predictors are implemented - SparkTransformer proposes to partition the input query x into two sets of indices, $x[:r]$ and $x[r:]$. The first r indices would be used only to compute the sparse predictor, which would generate a sparse mask using a top-k operation. This sparse mask would then be applied to the full output computation using the remaining input indices $x[r:]$.

$$\begin{aligned}\text{SparkFFN}(x;k) &= \left(\sigma_{\text{act}}\left(\text{Top-k}\left[x[:r]W_{\text{gate}}\right]\right) \odot x[r:]W_{\text{up}}\right) W_{\text{down}} \\ \text{SparkAttn}(q,K,V;k) &= \left[\text{Top-k}^{-\infty}\left(q[:r]K[:,:r]^T\right) \odot \text{softmax}\left(q[r:]K[:,r:]^T\right)\right] V\end{aligned}$$

Unfortunately, the architecture described here required pretraining from scratch using the full 1T token corpus of Gemma-2, and is thus not immediately practical for smaller-scale inference. But there are still valuable lessons to take from it, particularly when it comes to SparkTransformer's second primary contribution: the statistical top-k mechanism.

In order to accelerate the computation of the top-k mask on GPU, You et al. propose a "Statistical Top-K" approximation using quantile function Q (the inverse of the cdf of the standard normal). By effectively modelling the activations of a batch as approximately normally distributed, a threshold θ can be computed using sample means and standard deviations as $\theta(x,k) = \mu(x) + n(k) \cdot \sigma(x)$, where $n(k) = \mathcal{Q}(1-k/d)$ is the number of standard deviations outside the norm needed to achieve a desired sparsity ratio k/d. This statistical approach eliminates the need for explicit sorting operations, making it significantly faster on parallel hardware.

This method is fully generalizable, and can be combined with other existing sparse approaches to leverage fast top-k operations on GPU. Notably, versions of both the index partitioning and statistical top-k operations were eventually used within Google's Gemma 3-n [Team, 2025] model released in June 2025.

6 Harnessing Sparsity

The escalating computational demands of modern LLMs have made sparsity not merely an optimization technique, but a fundamental requirement for practical deployment—particularly in resource-constrained environments. This paper has traced the evolution of sparse inference methods from the early successes with ReLU-based models like OPT to the sophisticated techniques now emerging for contemporary architectures using SiLU and GeLU activations.

6.1 Our Approach

So how do we take these disparate pieces and work towards a unified framework for sparse inference using transformers? At NimbleEdge, we are systematically working towards this goal with a focus on combining multiple complementary techniques into a unified framework. Our repository[1] aims to provide not just isolated implementations, but a complete toolkit for sparse inference.

Our current research priorities include:

- Validating weight caching optimizations for neuron concentration patterns (preliminary results show 6.7x speedups in index operations)
- Translating these theoretical performance gains into the long-tailed activation setting using both relufied models and CETT-based thresholding
- Developing fused kernels that combine sparse prediction with cached weight access
- Exploring lightweight attention indexers for long-context edge deployment

Early results are promising, but significant engineering challenges remain in creating a production-grade system that seamlessly integrates these optimizations. We are releasing our implementations and benchmarks incrementally as they mature.

6.2 Looking Forward

As major model developers like DeepSeek and Google increasingly build sparsity into their core architectures—with DeepSeek v3.2 achieving remarkable long-context efficiency and Gemma 3-n incorporating statistical top-k operations—sparsity is transitioning from a research curiosity to a production necessity. The 2-6x speedups demonstrated by these techniques are not merely incremental improvements; they represent the difference between feasible and infeasible deployment in edge computing scenarios.

However, most existing implementations focus on individual techniques in isolation. The next frontier is systematic integration: combining predictive masking, weight caching, statistical

[1] https://github.com/NimbleEdge/sparse_transformers

top-k operations, and hardware-aware kernels into unified frameworks that work across diverse model architectures.

At NimbleEdge, we are committed to advancing this integration challenge through open-source development and community collaboration. We invite researchers and practitioners to contribute to this effort—the future of edge AI depends on making sparse inference not the exception, but the standard.

References

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, S. Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. In *ACL*, 2024. URL https://arxiv.org/pdf/2312.11514.

Christian Bogmans, Patricia Gomez-Gonzalez, Ganchimeg Ganpurev, Giovanni Melina, Andrea Pescatori, and Sneha Thube. Power Hungry. *IMF Working Papers*, 2025(081):1, 4 2025. ISSN 1018-5941. doi: 10.5089/9798229007207.001. URL http://dx.doi.org/10.5089/9798229007207.001.

Xin Chen, Xiaoyang Wang, Ana Colacelli, Matt Lee, and Le Xie. Electricity demand and grid impacts of ai data centers: Challenges and prospects. *arXiv preprint arXiv:2509.07218*, 2025.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

DeepSeek-AI. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention, 2025.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019. URL https://arxiv.org/abs/1803.03635.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

Donghyun Lee, Je-Yong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. Cats: Contextually-aware thresholding for sparsity in large language models, 2024. URL https://arxiv.org/abs/2404.08763.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/liu23am.html.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C. del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. *ArXiv*, abs/2310.04564, 2023. URL https://api.semanticscholar.org/CorpusID:263830421.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert, editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 2626–2644, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.180/.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu, 2023.

Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. Turbo sparse: Achieving llm sota performance with minimal activated parameters, 2024. URL https://arxiv.org/abs/2406.05955.

Gemma Team. Gemma 3n. 2025. URL https://ai.google.dev/gemma/docs/gemma-3n.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. Powerinfer-2: Fast large language model inference on a smartphone, 2024. URL https://arxiv.org/abs/2406.06282.

Chong You, Kan Wu, Zhipeng Jia, Lin Chen, Srinadh Bhojanapalli, Jiaxian Guo, Utku Evci, Jan Wassenberg, Praneeth Netrapalli, Jeremiah J. Willcock, Suvinay Subramanian, Felix Chern, Alek Andreev, Shreya Pathak, Felix Yu, Prateek Jain, David E. Culler, Henry M. Levy, and Sanjiv Kumar. Spark transformer: Reactivating sparsity in ffn and attention, 2025. URL https://arxiv.org/abs/2506.06644.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu2 wins: Discovering efficient activation functions for sparse llms. *ArXiv*, abs/2402.03804, 2024. URL https://api.semanticscholar.org/CorpusID:267499856.