Modeling Structured vs Unstructured and Dual vs Single Sided Sparsity in DNNs

Yicheng Huang
huanyic1@mit.edu

Nathan Mustafa
nmustafa@mit.edu

I. Introduction

Modern neural network accelerators can exploit sparsity in various ways to reduce energy consumption and computational cost. Some architectures focus on weight sparsity, others on activation sparsity, and some aim to leverage both simultaneously. A key design dimension is whether sparsity is handled using a fixed, structured approach or a flexible, unstructured approach. Unstructured sparsity offers maximum flexibility in pruning, or lack thereof, but typically imposes additional hardware overhead, while structured sparsity simplifies hardware design and improves efficiency at the cost of reduced pruning flexibility and increased overhead.

This project investigates a central set of questions: Under what conditions do different sparsity types—single-sided vs. dual-sided, structured vs. unstructured—deliver the best energy and computational performance? By identifying the regimes in which each sparsity strategy excels, we aim to provide insights that can better guide the design of future hardware accelerators.

We also examine how specific hardware factors, namely increased on-chip RAM and memory, affect the tradeoff between structured and unstructured sparsity optimizations as well as the impact of memory size on meta-data overhead costs as a whole.

To address these questions, we conducted experiments on ResNet50 using the Sparseloop hardware simulator. Our results show that structured sparsity consistently outperforms unstructured sparsity in terms of energy efficiency, from low to very high sparsity levels, even though both approaches achieve the same reduction in computation count. Furthermore, dual-sided sparsity (exploiting both weight and activation sparsity) outperforms single-sided sparsity even at low sparsity levels, as it can take advantage of inherent sparsity in both operands rather than just one.

It is important to note, however, that while structured sparsity offers substantial hardware benefits, it also requires significant preprocessing effort to prune weights or activations into a structured form (e.g., 2:4, 2:6 sparsity patterns), which may require additional software and computational resources during inference preparation.

Finally, we noticed that increased RAM size decreased energy usage for structured and unstructured strategies to a similar degree. This observation motivated us to dig deeper and realize that data movement costs and storage requirements are magnitudes more significant than the same costs and requirements for meta-data.

II. Related Works

The exploration of sparsity in deep neural networks (DNNs) has become a central topic in model compression, hardware acceleration, and energy efficiency. Among foundational contributions, Song Han et al.'s "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding" [1] stands out as a landmark study. Han's work introduced a three-stage pipeline: magnitude-based pruning of redundant weights, quantization of the remaining weights to reduce precision, and Huffman coding for further compression. On benchmark models such as AlexNet and VGG-16, Han et al. reported compression rates of 35× to 49× with no loss of accuracy. Importantly, the pruning used in Deep Compression was unstructured single sided, removing individual weights without considering their memory arrangement. This approach maximizes the number of prunable parameters, often achieving the highest theoretical compression and FLOP reduction. However, the resulting sparse matrices are irregular, making it difficult for hardware like GPUs and CPUs to efficiently exploit the sparsity, often resulting in large discrepancies between theoretical and real world speedups, as general-purpose hardware lacks efficient support for random sparse operations.

To address the hardware limitations of unstructured sparsity, researchers turned to structured sparsity, which prunes at coarser granularity — such as blocks, channels, or groups of weights — thus aligning better with hardware. A prominent example is NVIDIA's Ampere and Hopper Tensor Cores, which implement 2:4 structured sparsity, where two out of every four contiguous weights are pruned, typically using magnitude-based criteria. This structured pattern enables up to 2× theoretical throughput improvements in matrix multiplication operations, with empirical benchmarks showing 1.44–1.78× speedup in practice depending on the workload and kernel implementation [3].

This type of single sided structured sparsity balances compression and hardware efficiency, maintaining regularity in memory accesses and allowing highly optimized sparse matrix multiply units to skip pruned weights. It also makes deployment practical on commercial GPUs without requiring major software changes. On the downside, structured sparsity imposes rigid pruning patterns, reducing flexibility and often

requiring more careful tuning or retraining to avoid accuracy loss compared to unstructured pruning. As a result, the maximum compression ratio is usually lower (typically 1.5–2×), and it is less effective on layers that are already compact or sensitive to pruning, such as small transformers or bottleneck layers.

Beyond weight pruning, recent advances have targeted activation sparsity, which naturally arises in ReLU-based networks where negative activations are zeroed out, or in attention models with masking. Yang Wang et al.'s "Dual-Side Sparse Tensor Core" introduced a hardware-software co-design that simultaneously exploits both weight and activation sparsity (dual-sided sparsity) in tensor core operations. Their system achieved a theoretical speedup of 3.97x on ResNet-18 and a 10.26x theoretical overall speedup on BERT-base Encoder [2].

This dual-sided sparsity unlocks multiplicative gains because it benefits from both static (weight) and dynamic (activation) sparsity. It is particularly advantageous in transformer architectures and convolutional networks with ReLU functions, where activations are naturally sparse. However, the main challenge is that activation sparsity is dynamic and input-dependent, requiring sophisticated runtime support and hardware modifications to detect and skip zero activations on the fly. Without efficient runtime management, the overhead of tracking dynamic sparsity can offset the gains, and integrating dual-sided sparsity often demands architectural changes beyond what off-the-shelf GPUs currently provide.

Overall, these works highlight the evolution from unstructured to structured and dual-sided sparsity in both algorithm and hardware design. While unstructured sparsity offers maximal pruning flexibility and theoretical compression, its irregularity limits hardware acceleration. Structured sparsity aligns well with current hardware, offering practical speedups at the cost of reduced flexibility and maximum compression. Dual-sided sparsity represents the frontier, combining the best of both worlds but requiring hardware-software co-design to fully realize its potential. This paper aims to systematically compare these paradigms and analyze their trade-offs in terms of cycles and energy utilization in specific workloads.

III. Project and Setup

In this section, we describe the experimental setup we used to evaluate the trade-offs between dual-sided vs. single-sided sparsity and structured vs. unstructured sparsity optimizations. Our experiments are conducted using the Sparseloop hardware simulator, which allows us to simulate the workload of running inference on a ResNet50 neural network under various sparsity regimes. We model the workload on a representative hardware architecture that includes DRAM memory, a shared local DRAM buffer across four cores, SRAM registers, and a multiply-accumulate (MAC) computation core. All computation and memory operations in our setup use 8-bit data precision, consistent with common high-efficiency inference hardware.

We specifically aim to examine how different sparsity strategies interact with this hardware architecture and how they influence energy consumption, compute cycles, and overall efficiency. To simulate single-sided and dual-sided sparsity optimizations, we leverage the sparsity skipping functionalities provided by the Sparseloop library. This allows us to selectively specify which parts of the workload (weights, activations, or both) should be considered for sparsity skipping. For example, to evaluate single-sided sparsity, we configure the simulator to skip computations involving zero-valued weights, effectively ignoring any multiplications where the weight is zero. For dual-sided sparsity, we enable skipping on both the weights and activations, ensuring that computations involving either zero weights or zero activations are avoided. This dual-sided setup enables the simulator to bypass a larger fraction of unnecessary operations, potentially offering greater energy and cycle savings.

However, these optimizations come with important trade-offs. Skipping computations on zero-valued data—whether weights or activations—requires additional hardware control logic to detect zero values and manage the flow of data. When a zero is detected, the hardware must continue iterating through the data until it finds a productive (nonzero) computation. In the case of dual-sided sparsity, this overhead increases because the system must check both data streams simultaneously. The key implication is that although skipping saves computation and energy at the high level, it also introduces nontrivial per-operation overhead in the form of additional control circuitry and logic transitions. This raises a critical question: under what conditions is the added hardware overhead of dual-sided sparsity justified by the gains in reduced computation and energy savings?

Alongside these sparsity dimensions, we also explore the trade-offs between structured and unstructured sparsity optimizations. A wide variety of techniques exist within both categories, each with varying degrees of flexibility, hardware cost, and metadata overhead. In our experiments, we modeled the unstructured sparsity optimization using the Dual-Sided Tensor Core (DSTC) scheme, which employs bitmasks to filter out unproductive computations during matrix multiplications. DSTC represents a highly flexible form of unstructured sparsity and is widely used in modern GPU and accelerator designs. For structured sparsity, we modeled the Nvidia Structured Tensor Core (STC) scheme, which offers highly efficient sparsity support for matrices with known, regular sparsity patterns, such as 2:4 or 2:8 sparsity (i.e., two zeros in every four or eight elements, respectively).

The fundamental difference between these two approaches lies in the predictability and guarantees provided by the data they operate on. Structured sparsity optimizations, such as STC, can exploit the fact that the sparsity pattern is known in advance. For example, in a 2:4 sparsity setup, the hardware can immediately skip the next two elements once it has encountered two nonzero values in a group of four, without the need for further checks. This leads to minimal overhead and highly efficient compute skipping. In contrast, unstructured sparsity provides no such guarantees, as the location of nonzero values is arbitrary. As a result, unstructured sparsity incurs additional

costs to handle irregular data patterns, including the need for indexing metadata, dynamic control logic, and complex data access patterns. This flexibility allows unstructured methods to achieve higher pruning rates, but it comes at the price of increased hardware complexity and potential performance penalties, especially at moderate sparsity levels.

Overall, our setup enables a comprehensive investigation of how these different sparsity strategies interact with hardware, where their advantages and limitations lie, and how the balance between flexibility and hardware efficiency shapes the ultimate performance and energy outcomes.

IV. Results and Discussion

In this section we discuss the results of evaluating the sparsity optimizations described in the previous section with regards to latency, computations, and energy.

A. Unstructured vs. Structured

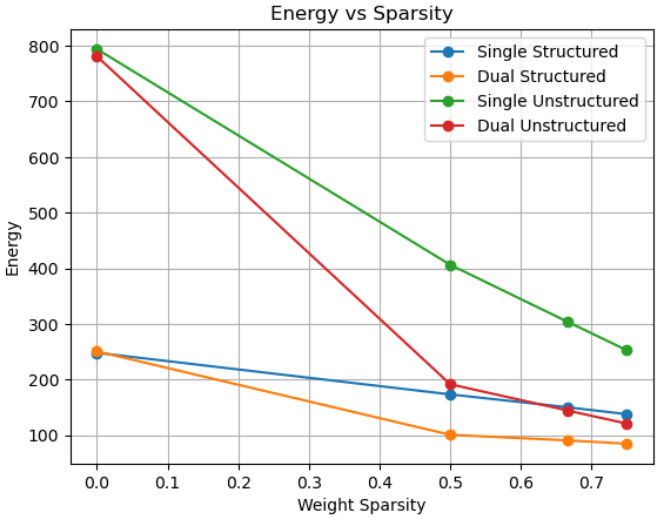

Fig. 1. Dual sparsity Outperforms single-sided across both and structured generally works better than unstructured.

In terms of energy efficiency, our results consistently show that structured sparsity optimizations outperform unstructured sparsity optimizations across almost all tested configurations. While dual-sided unstructured sparsity can match the performance of single-sided structured sparsity at sparsity levels exceeding 50%, a direct comparison between matched configurations—i.e., single-sided unstructured versus single-sided structured, and dual-sided unstructured versus dual-sided structured—reveals that structured sparsity offers nearly 2× energy savings across a wide range of sparsity levels. This suggests that the regularity and predictability of structured patterns provide a significant energy advantage over the more flexible but metadata-heavy unstructured approaches.

An important observation from our experiments is the high energy overhead of unstructured sparsity at low to moderate sparsity levels. For reference, running ResNet50 on our baseline architecture with no sparsity optimizations incurs a fixed energy cost of $276.58\mu J$ per inference. Under these conditions, single-sided unstructured sparsity does not achieve any energy savings over the baseline until the model reaches approximately 75% sparsity—highlighting the substantial overhead that unstructured methods must overcome before they deliver net gains. Dual-sided unstructured sparsity shows somewhat better performance, achieving energy benefits at around 40% sparsity, but still requires relatively high sparsity levels before its flexibility translates into meaningful efficiency improvements.

Notably, as sparsity increases toward the high end of the spectrum, the energy gap between structured and unstructured sparsity begins to narrow and eventually converges. This convergence is expected and reflects the fact that, at extreme sparsity, the control and metadata overhead inherent to unstructured methods becomes less significant relative to the dramatic reduction in overall computations. At that point, unstructured sparsity can fully exploit its flexibility without being penalized by control costs, allowing it to close the efficiency gap with structured approaches.

When we turn to the compute-level analysis, we observe that both unstructured and structured sparsity configurations result in the same total number of computations (i.e., multiply-accumulate operations) being skipped. This is an important confirmation of correctness; since both methods apply sparsity skipping at the same logical level, they must ultimately reduce the same number of operations. The distinction between the two lies not in whether operations are skipped, but in how the skipping is implemented and detected at the hardware level, which is revealed in the cycle count.

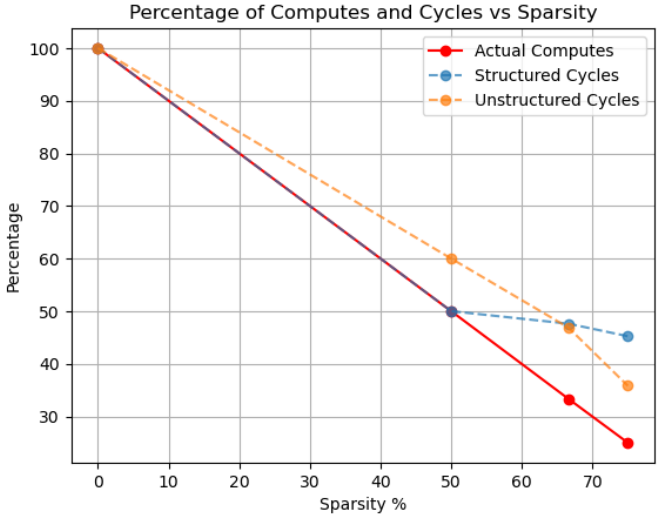

Fig. 2. Structured Sparsity requires more cycles at higher sparsities, indicating more overhead.

However, when examining cycle-level performance, a more nuanced picture emerges. Structured sparsity consistently outperforms unstructured sparsity at lower sparsity levels, benefiting from its ability to bypass redundant computations with minimal control overhead. In contrast, unstructured sparsity

begins to show a latency advantage as sparsity levels rise above approximately 66%, where its flexible skipping mechanisms can dynamically take advantage of increasingly sparse inputs. This crossover point highlights a key trade-off: while structured sparsity delivers reliable performance at moderate sparsity, unstructured sparsity becomes competitive—and sometimes superior—when extreme sparsity levels are reached and its additional control complexity is amortized over fewer active computations.

This result is fairly surprising as from general hardware theory, structured sparsity is supposed to have lower overhead because it has regular patterns, making it easier to skip on hardware. Thus, specialized hardware units can exploit this without complex indexing or irregular data movement whereas unstructured sparsity needs to track and process random zero patterns, reducing hardware utilization. So in theory, as we increase sparsity we expected structured sparsity to drop both compute and cycles efficiently and unstructured sparsity to retain more cycle overhead.

It's possible that structured kernels have an inherent baseline cycle cost per block, regardless of the number of nonzero elements, which prevents them from fully exploiting high sparsity. However the much more likely explanation is that we configured something incorrectly or that there exists a limitation in the simulation that we are not accounting.

Overall, the dominance of structured sparsity optimizations implies that machine learning hardware should opt towards structured optimizations as much as possible. However, as we discuss later, structured sparsity comes with significant drawbacks and overhead related to pruning costs and accuracy. Therefore, unstructured sparsity optimizations still carry significant merit.

B. Dual-Sided vs. Single-Sided

In our comparison of single-sided and dual-sided optimizations, we find that dual-sided optimizations outperform single-sided optimizations in general.

With regards to energy, dual-sided unstructured and structured sparsity outperform their single-sided counterparts at almost all sparsity levels - particularly at high sparsity where sparsity in both input and weights can be utilized to skip useless computations. As expected, however, at low sparsity levels, dual-sided sparsity's additional overhead to access both input operands causes it to have higher energy levels to the point that single-sided structured outperforms dual-sided structured optimizations. We were surprised that the dual-sided overhead costs get overtaken at such a low sparsity, only 5% for structured optimizations and immediately for unstructured optimizations. In general, we would've expected the dual-sided overhead to dominate at low sparsity making single-sided optimizations more performant up to a higher sparsity level.

The tradeoffs in latency and number of computes between single-sided and dual-sided optimizations are directly related to the amount of sparsity in the inputs and weights, as expected. Dual-sided optimizations are able to capitalize on sparsity within the inputs, not just the weights, in order to skip an additional number of cycles directly proportional to the sparsity in the inputs.

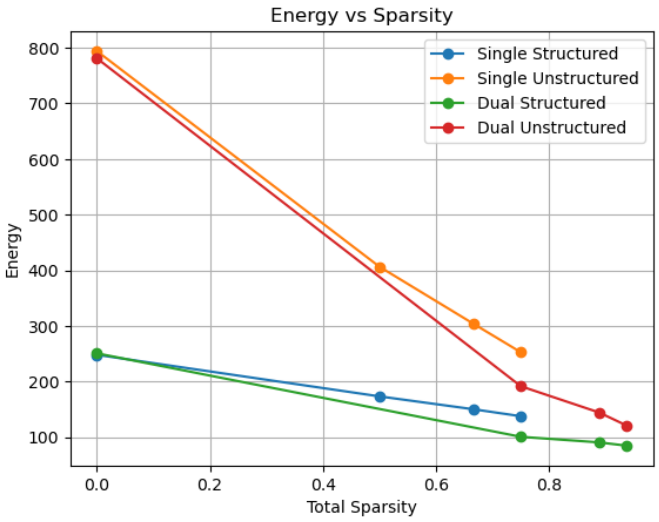

Fig. 3. Adjusting for total sparsity.

However, one could argue that this comparison is somewhat unfair. In the previous results, we assumed that input sparsity would scale proportionally with weight sparsity. This means that, for example, a dual-sided setup with 50% sparsity in both inputs and weights effectively performs only a quarter of the total computations, since $1 - (0.5)^2 = 75\%$ of operations are skipped. Similarly, a configuration with 25% sparsity on both sides corresponds to $1 - (0.25)^2 = 93.75\%$ overall sparsity—comparable to a highly sparse single-sided setup.

When we adjust for this effect, as shown in Figure 3, the overall trends remain consistent: structured sparsity continues to outperform unstructured sparsity, and dual-sided configurations still outperform single-sided ones. However, the performance advantage of dual-sided sparsity over single-sided sparsity becomes noticeably less pronounced.

That said, we argue that greater weight should be placed on the results from the initial analysis. Regardless of the amount of input sparsity present, single-sided approaches are fundamentally unable to exploit it. Therefore, dual-sided sparsity should not be penalized for its inherent structural advantage.

C. Impact of Memory on Sparsity Optimizations

As part of our evaluation of various sparsity optimizations, we wanted to measure the effect of hardware attributes on the efficacy of sparsity optimizations by seeing how increasing RAM affects the energy spent for unstructured vs. structured optimizations. Note that our interest in unstructured vs. structured stems from the fact that they have differing levels of reliance on meta-data which are stored in memory. Any comparison of dual-sided vs single-sided with respect to changing memory capacities would be uninteresting since dual-sided and single-sided optimizations don't care about

how the data is fetched, simply whether there are zeros in one or both operand. Similarly, comparing computations is not interesting since memory size doesn't affect how many computations are performed.

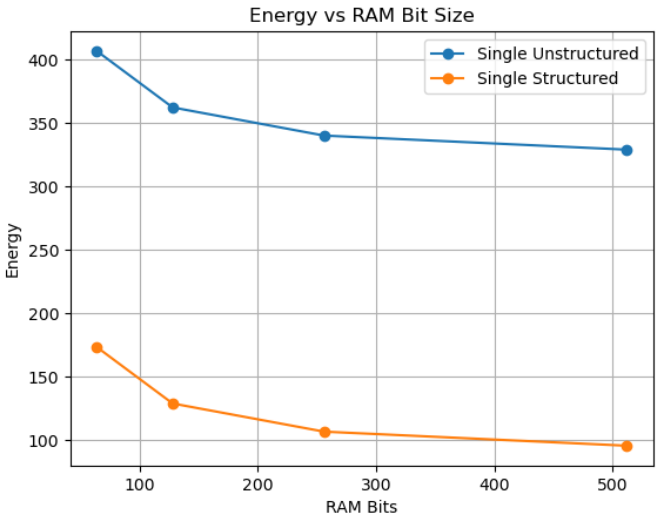

Fig. 4. Energy spent on inference as DRAM data and meta-data capacity are increased on a 50% sparse workload.

Our results show that when we increase the amount of meta-data and data capacity in RAM, the energy drops by a fixed amount independent of whether structured or unstructured sparsity optimizations are utilized. As RAM continues to be expanded, the improvement in energy asymptotically fades away.

We found this result to be surprising as we expected that unstructured sparsity and it's larger overhead would stand to benefit more from having additional space in memory to store it's larger meta-data information. Instead, both optimizations benefit equally - likely primarily from the fact that less energy is spent accessing disk.

We confirmed that the benefit is solely from being able to store more data in RAM by testing whether increasing just the meta-data RAM size had any impact. We found that increasing just the meta-data RAM size has no impact on energy or cycles for both unstructured and structured optimizations which confirms that the there is no apparent benefit related to sparsity optimizations in increasing RAM size. This is further confirmed by the fact that only changing the data RAM size produces identical results to changing both the meta-data and data RAM size.

We note that this is likely caused by the meta-data necessary for the ResNet workload being insufficiently large to require meta-data to be flushed to disk and so a larger workload is needed to see an affect in increasing/decreasing the meta-data RAM size. This is supported by the fact that decreasing meta-data DRAM size down to a single bit doesn't show any changes nor does decreasing the size of the SRAM SMEM Buffer.

Critically, however, increasing the size of the SRAM LRF very slightly (on the scale of $\leq 1\mu J$) decreases the energy usage when using an unstructured sparsity optimization. Similar to the DRAM, the decrease in energy usage asymptotically decreases as the size of the LRF increases.

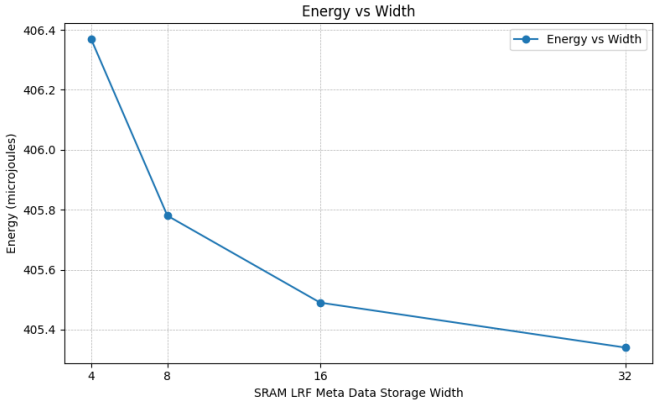

Fig. 5. Energy spent on inference using an unstructured sparsity optimization as the width of the LRF meta-data storage is increased and the depth is held at 32 bits.

The interesting insight this data provides is twofold. First, we see that the amount of space meta-data takes up is incredibly small. The fact that neither increasing the size of DRAM nor the SMEM buffer made an impact on energy means that sparse optimization meta-data on the Resnet architecture requires not even enough meta-data to fill up the SMEM buffer - let alone DRAM memory or disk. Hence, the primary focus should always be on data movement costs while meta-data movement costs do not need to be a primary concern.

Second, the improvement provided by expanding the LRF was nearly insignificant, only a $\frac{1}{400} = 00.25\%$ decrease in energy. Therefore, we see that hardware design decisions made around the meta-data storage - such as increasing memory sizes to better store meta-data - are not vital.

It's worth noting that this analysis is specific to the ResNet workload used for experimentation and that larger, modern workloads would require much more space and meta-data, in which case the effects of increasing meta-data storage sizes may become more significant. Nonetheless, our results show that data storage and data movement costs are magnitudes more significant than that of meta-data.

V. Limitations

It is critical to note the nuances in our results. Specifically, we note that while our results show that structured optimizations consistently outperform unstructured sparsity optimizations, there are significant constraints and overheads to using structured sparsity optimizations that are not considered in our evaluations.

First and foremost, pruning data into a structured format requires significant energy and time. The process of pruning can take several iterations of pruning large networks and re-evaluating the entire network. For pruning weights, this is

a (non-negligible) fixed overhead. However, doing the same pruning on inputs requires every inference to have significant overhead time prior to running the inference as evaluated in this paper. Additionally, the process of pruning into a structured format such as 2-4 or 2-6 can have significant impacts on the accuracy of a network that can easily outweigh the performance benefits of using structured optimizations during inference.

As a result, we argue that it can be unrealistic to utilize structured sparsity optimizations in all cases. For example, dual-sided structured sparsity requires extensive pruning on both the weights and inputs which would take an extensive amount of time per inference to ensure accuracy is maintained.

It is therefore possible that the flexibility and lack of overhead needed by unstructured sparsity optimizations make it more enticing than structured optimizations despite its energy and latency disadvantages.

REFERENCES

[1] Song Han, Huizi Mao, William J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," arXiv, 2016.

[2] Yang Wang and Chen Zhang and Zhiqiang Xie and Cong Guo and Yunxin Liu and Jingwen Leng, "Dual-side Sparse Tensor Core," arXiv, 2021.

[3] Jeff Pool, "Accelerating Sparsity in the Nvidia Ampere Architecture," Nvidia, 2021.