

SparseDVFS: Sparsity-aware DVFS for Energy-Efficient Edge Inference

Ziyang Zhang
Politecnico di Milano
Milan, Italy
ziyang.zhang@polimi.it

Luca Mottola*
Politecnico di Milano
Milan, Italy
RI.SE Sweden
Kista, Sweden
luca.mottola@polimi.it

Zheshun Wu
Harbin Institute of Technology
Shenzhen, China
wuzhsh23@gmail.com

Jie Liu
Harbin Institute of Technology
Shenzhen, China
jieliu@hit.edu.cn

Tian Wang
Institute of Artificial Intelligence and
Future Networks, Beijing Normal
University
Zhuhai, China
tianwang@bnu.edu.cn

Abstract

We present SparseDVFS, a sparsity-aware DVFS technique operating at fine-grain for energy-efficient edge inference. Our key insight is that operator sparsity is a primary metric for hardware frequency modulation. Traditional model-level DVFS, on the other hand, is often too coarse-grained to capture intra-inference variations in sparsity and computational intensity, whereas fine-grained operator-level scaling suffers from prohibitive performance degradation due to significant hardware switching latency. By distinguishing between compute-bound dense operators and memory-bound sparse operators, SparseDVFS applies specialized frequency configurations to maximize energy efficiency. We achieve this with a combination of three techniques: (1) an *offline modeler* that establishes a mapping between operator sparsity and optimal frequency configurations via white-box timeline analysis; (2) a *runtime graph partitioner* that aggregates operators into super-blocks, balancing scaling granularity and DVFS switching latency through a latency amortization constraint; and (3) a *co-governor* that employs a scaling engine to jointly synchronize CPU, GPU, and memory clock frequencies and a look-ahead instruction queue to eliminate antagonistic effects between independent component governors and hide hardware switching latencies. Extensive evaluations show that SparseDVFS improves energy efficiency by 78.17% with

only a 12.8% latency penalty and reduces DVFS switching overhead by 8.5 $\times$, while ensuring thermal stability.

CCS Concepts: • Hardware → Power and energy; • Computing methodologies → Modeling and simulation.

Keywords: Edge Inference; Sparsity; Power Management; Fine-grained DVFS

ACM Reference Format:

Ziyang Zhang, Luca Mottola, Zheshun Wu, Jie Liu, and Tian Wang. 2027. SparseDVFS: Sparsity-aware DVFS for Energy-Efficient Edge Inference. In *Proceedings of the 32nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '27)*, April 11–15, 2027, Heraklion, Greece. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3845814.3850057>

1 Introduction

The proliferation of deep neural networks (DNNs) is driving an unprecedented demand for high-performance edge inference [13, 19, 33, 45, 48, 82]. As these models grow in complexity, from deep convolutional neural networks (CNNs) [20] to intricate Transformer-based architectures like Vision Transformers (ViT) [12, 68], deploying them on resource-constrained edge devices presents a formidable challenge. Edge devices [52] widely used in autonomous systems [55, 62] and IoT applications [41, 72], operate under strict energy envelopes and thermal constraints, necessitating sophisticated system-level power management strategies.

Dynamic voltage and frequency scaling (DVFS) [31, 40, 59, 66, 70, 83] stands as a standard for managing power and performance trade-offs in modern platforms. By dynamically adjusting the operating frequency and voltage of the CPU, GPU, and memory subsystems, systems can align power consumption with the instantaneous computational demands of the workload. In the context of edge inference, effective DVFS is

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.

ASPLOS '27, Heraklion, Greece

© 2027 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3012-2/2027/04

<https://doi.org/10.1145/3845814.3850057>

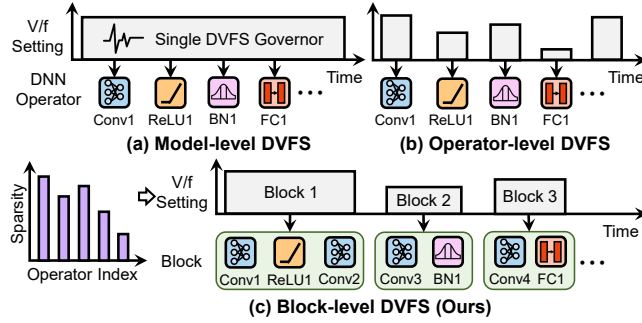


Figure 1. Comparison of DVFS governor granularities for edge DNN inference. (a) Model-level DVFS governor employs a single V/f setting. (b) Operator-level fine-grained DVFS governor. (c) Our sparsity-aware block-level DVFS governor.

critical. However, traditional DVFS cannot accurately capture DNN characteristics, such as sparsity [13, 39, 65], leading to suboptimal frequency allocation that either causes performance degradation or significant energy waste.

State of the art. Existing work includes DVFS governors [18, 28, 30, 40, 70, 83, 84] for on-device DNN inference to improve energy efficiency. However, as illustrated in Figure 1, these solutions face a fundamental trade-off between scaling granularity and system-level overhead. Model-level DVFS, shown in Figure 1(a), commonly employed by stock Linux governors [46] and learning-based methods like zTT [30] or GearDVFS [40], assign a static frequency for the entire inference process. This is too coarse-grained to capture intra-model workload fluctuations where computational intensity varies significantly between layers. Conversely, operator-level DVFS techniques like Ascend-DVFS [70], shown in Figure 1(b), attempts to track these variations at the finest grain, yet they remain practically infeasible on commercial hardware. Our benchmarks in Section 2 notably reveal that frequency switching latencies can even exceed the execution time of lightweight operators, causing transition penalties to dominate the timeline and overturn energy savings. Furthermore, memory-centric sparse acceleration frameworks like Symphony [58] and Spatula [14] rely on cycle-accurate emulators or custom ASIC architectures, making direct deployment on COTS edge boards methodologically infeasible.

Table 1 presents a comparison between SparseDVFS and related work across four dimensions. Unlike existing works that either suffer from high-overhead operator scaling, ignore dynamic activation sparsity, or rely on non-deployable custom hardware emulators, SparseDVFS operates purely as a software-level governor on stock COTS hardware. By bridging sparsity with power states, SparseDVFS achieves fine-grained energy modulation without requiring custom hardware modifications.

SparseDVFS. We conceive a block-level DVFS strategy, illustrated in Figure 1(c) that strikes a trade-off between the

Table 1. Comparison of SparseDVFS with related work.

Framework	DVFS Granularity	Sparsity Aware	Look Ahead	COTS
zTT [30]	Model-level	✗	✗	✓
GearDVFS [40]	Model-level	✗	✗	✓
Ascend-DVFS [70]	Operator-level	✗	✗	✓
Symphony [58]	Tensor-level	✓	✗	✗
Spatula [14]	Kernel-level	✓	✗	✗
CLONE [66]	Layer-level	✗	✗	✓
SparseDVFS	Block-level	✓	✓	✓

two extremes found in existing work (i.e., model-level DVFS and operator-level DVFS). By aggregating operators into sparsity-aware super-blocks, we enforce a latency amortization constraint that systematically balances the granularity of frequency scaling against frequency switching overhead. This enables SparseDVFS to achieve a significantly improved trade-off between energy efficiency and inference latency compared to existing DVFS strategies.

Achieving such a fine-grained, sparsity-aware DVFS operation involves the following three key challenges:

- **DVFS adjustment overhead:** transitioning from coarse-grained model-level scaling to fine-grained operator-level control implies frequent frequency switching. Our benchmarks in Section 2 reveal that GPU frequency switching latency for on-device DNN inference mostly ranges from 5ms to 10ms, but exceeds 20ms at extremely low frequencies. Since many lightweight DNN operators, such as ReLU [15, 35] and GELU [22], only take a few milliseconds to execute, operator-level DVFS strategy would cause the switching overhead to exceed the execution time, drastically increasing end-to-end latency.
- **Balancing latency and power:** the trade-off between latency and power is non-linear and governed by the Roofline Model [11, 24, 64, 75]. Our benchmarks in Section 2 reveal that for compute-bound operators (e.g., Conv2d [37]), higher frequencies yield linear speedups. For memory-bound sparse operators (e.g., LayerNorm [1]), however, the performance bottleneck shifts to memory bandwidth. In these cases, increasing GPU frequency beyond the ridge point primarily increases power consumption with diminishing energy efficiency gains.
- **Adaptability and generalization:** modern edge AI workloads exhibit extreme diversity [6, 8], spanning diverse architectures from traditional CNNs with high intra-model sparsity to emerging Transformer-based models characterized by dynamic attention patterns. Existing learning-driven DVFS governors, however,

treat DNN models as black boxes, necessitating prohibitive offline retraining and manual profiling whenever a new model architecture or hardware platform is introduced. This lack of architectural transparency and deterministic modeling renders current solutions difficult to generalize.

SparseDVFS is a fine-grained sparsity-aware DVFS for energy-efficient edge inference, predicated on the insight that operator sparsity is a primary metric for hardware frequency modulation. SparseDVFS overcomes the limitations of existing DVFS governors through three tightly coupled components. First, the *offline modeler* employs white-box timeline analysis [21] and thermal-aware power modeling [30] to construct a deterministic mapping between operator characteristics and optimal hardware states. Second, the *runtime graph partitioner* utilizes a greedy algorithm to dynamically aggregate operators into super-blocks [43], enforcing a latency amortization constraint that ensures computational gains are not offset by transition penalties. Third, the *co-governor* implements a frequency unified scaling engine (FUSE) strategy [83, 84] to synchronize CPU, GPU, and memory scaling as a coordinated frequency triplet effectively eliminating the antagonistic effects [81] inherent in independent component governors.

Experimental evaluations on the NVIDIA Jetson Orin Nano across diverse CNN- and Transformer-based models show that SparseDVFS achieves an average energy efficiency improvement of 78.17% over state-of-the-art solutions, with only a 12.8% latency penalty. By systematically balancing scaling granularity and DVFS governor costs, SparseDVFS reduces frequency switching overhead by up to 8.5 $\times$. It also maintains superior thermal stability under sustained workloads, effectively delaying the onset of thermal throttling.

This paper makes the following contributions:

1. We propose SparseDVFS, a sparsity-aware block-level DVFS that bridges the gap between coarse model-level and prohibitive operator-level scaling.
2. We develop a white-box offline modeler that provides a deterministic mapping between operator sparsity and optimal hardware states.
3. We introduce a runtime graph partitioner that greedily aggregates operators into super-blocks to amortize frequency adjustment overhead.
4. We design a co-governor to eliminate the antagonistic effects of independent governors.

2 Motivation

Our work is motivated by five key observations on the interaction between DNN operators and edge hardware, which we study *experimentally*. We use an NVIDIA Jetson Orin Nano with 8GB DRAM as a representative platform, as well as ResNet-18/101 [20] and ViT-B16/L16 [12] as workloads.

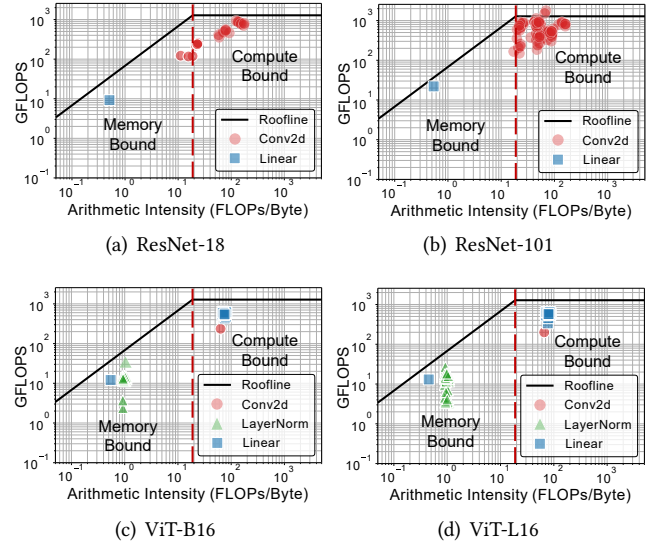


Figure 2. Roofline model across diverse DNN architectures.

2.1 Roofline Model Analysis

The Roofline Model [75] provides a fundamental characterization of how different operators interact with the hardware’s compute and memory limits. Figure 2 shows the performance (FLOPS) [25, 36, 63] against arithmetic intensity (FLOPs/byte) [53, 60, 61] for various operators in ResNet-18 and ViT-B16.1 The results reveal a bimodal behavior.

Dense operators like Conv2d and Linear layers reside on the horizontal plateau of the Roofline, indicating they are compute-bound. Their performance scales linearly with frequency. In contrast, activation functions (ReLU, GELU) and normalization layers typically fall into the sloped region, identifying them as memory-bound. Crucially, as the GPU frequency increases from 306 MHz to 624.75 MHz, the ridge point, that is, where the memory-bound slope meets the compute-bound plateau, shifts accordingly. For these memory-bound operators, increasing frequency beyond a certain point yields diminishing returns in performance because the bottleneck is memory bandwidth, not compute cycles. Thus, running high-sparsity, memory-bound operators at peak frequency wastes significant energy.

2.2 Dynamic Sparsity Distribution

We analyze the sparsity distribution of operators across different models on the ImageNet-2012 validation dataset, consisting in about 50k images [35], using cumulative distribution function (CDF). As shown in Figure 3, the CDFs for ResNet-18/101 (ReLU) and ViT-B16/L16 (GELU) show distinct profiles. For ResNet, a substantial portion of operators exhibit high sparsity (>50%), with the median sparsity often exceeding a 0.5 factor. ViT models show a different but

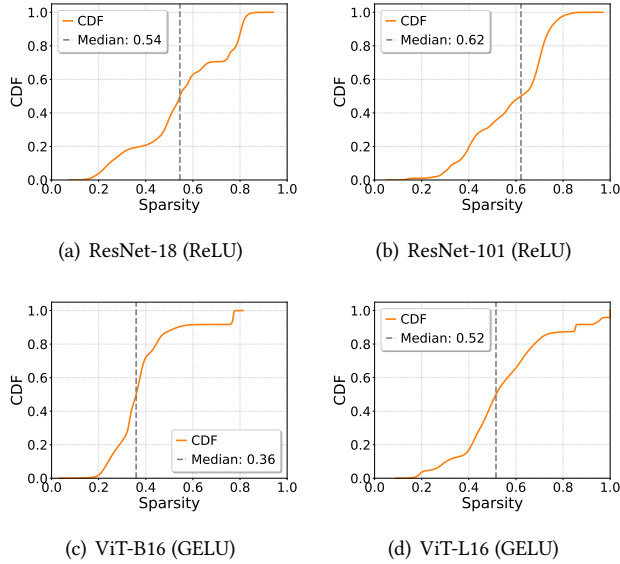


Figure 3. Dynamic sparsity distribution (CDF) for CNN and Transformer models on the ImageNet-2012 validation dataset. *A substantial portion of operators in ResNet and ViT exhibit high sparsity (>50%).*

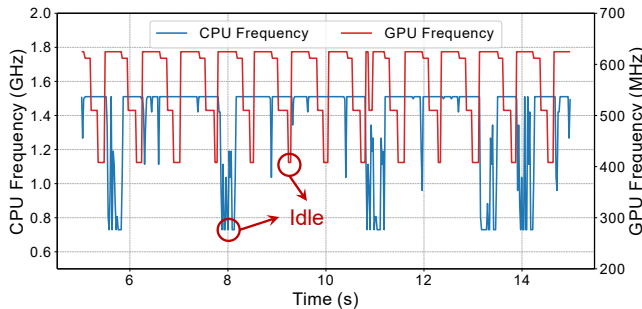


Figure 4. Runtime traces of CPU and GPU frequencies under default DVFS governors for ResNet-18. *Independent and often inverse fluctuations between operating frequencies of CPU and GPU create an antagonistic effect. CPU downclocking during GPU-intensive phases leads to wake-up latency spikes and pipeline stalls when data prefilling is requested.*

equally dynamic distribution, with sparsity varying significantly between Attention and MLP blocks.

The sparsity of activation functions thus changes dynamically with input data [13]. A static frequency setting cannot accommodate this variability. These results confirm that sparsity is a viable, high-entropy signal for modulating hardware performance. High-sparsity operators, that is, the right tail of the CDF, are prime candidates for frequency reduction, while low-sparsity operator, at the left tail of the CDF, require high performance states.

2.3 CPU-GPU Antagonistic Effect

Runtime traces of CPU and GPU frequencies under conventional governors, such as the **simple_ondemand** [56] for GPU and Linux **schedutil** [17] for CPU, reveal a critical inefficiency. In Figure 4, CPU and GPU frequencies fluctuate independently and often inversely. For instance, during a GPU-intensive computation phase, the CPU utilization drops, causing the default governor to lower its frequency. However, when the GPU finishes and requests more data, the CPU is in a low-power state, causing a latency spike as the conventional governor ramps up its frequency.

This antagonistic effect [81] leads to pipeline stalls and energy waste. A precise synchronization between CPU and GPU transitions may, instead, ensure that the CPU is ready to feed the GPU exactly when needed, regardless of instantaneous utilization.

2.4 DVFS Switching Overhead

To quantify the overhead of DVFS, we benchmark four DNN models on the NVIDIA Jetson Orin Nano.

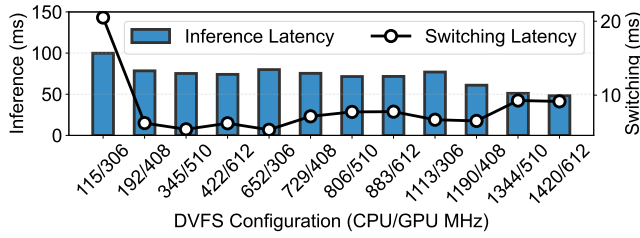
Model-level DVFS switching overhead. Figure 5 compares the single DVFS frequency switching latency against the total end-to-end inference latency for ResNet-18 across various CPU-GPU frequency configurations. For model-level DVFS, a single frequency switch incurs 5 ~ 10 ms of overhead (exceeding 20 ms at extremely low frequencies), while the entire inference pass takes 60 ~ 100 ms. In this coarse-grained regime, adjusting the frequency once per full inference represents a modest relative overhead (5 ~ 15%), making model-level DVFS feasible but incapable of capturing intra-inference sparsity variations.

Operator-level DVFS switching overhead. However, attempting to scale frequencies at fine operator granularity exposes a critical disparity. As shown in Table 2, single-operator switching latencies range from 4.6 ms to 14.6 ms. Crucially, this switching penalty drastically overshadows the execution time of lightweight operators (e.g., ReLU, BN, LayerNorm, and GELU take merely 0.07 ~ 1.9 ms), where a single transition takes up to 10× ~ 50× longer than the computation itself. Even for compute-dense operators like Conv2d and Linear layers (0.5 ~ 14.2 ms), the switching cost equals or exceeds the execution time.

Pipeline stalls and thermal leakage. For operator-level DVFS, invoking frequency switches per operator causes back-to-back hardware transition stalls across dozens or hundreds of sequential layers (e.g., 105 operators in ResNet-101 and 74 in ViT-L16). The cumulative switching overhead easily dominates the total execution timeline, doubling end-to-end latency and completely negating any energy dynamic savings. Furthermore, during these repeated, prolonged pipeline stalls, the chip remains powered in an active voltage state

Table 2. Operator-level inference latency, DVFS switching latency, and energy consumption benchmark across four DNN models on the NVIDIA Jetson Orin Nano.

Model	# Operator	Operator Type	Switching Latency (ms)	Inference Latency (ms)	Energy Consumption (mJ)
ResNet-18	10	Conv2d	5.6~8.7	1.5~4.2	22.5~67.5
	10	ReLU/BN	4.6~7.5	0.1~0.4	1.4~6.3
	1	Linear	5.3	1.2	18.0
ResNet-101	52	Conv2d	5.1~10.3	0.5~3.8	7.2~57.4
	52	ReLU/BN	4.7~8.2	0.07~0.2	0.71~3.6
	1	Linear	5.6	1.5	28.4
ViT-B16	14	Linear	6.1~9.7	1.2~5.5	18.6~82.4
	12	LayerNorm	5.2~8.9	0.3~0.8	4.9~12.7
	12	GELU	5.1~7.4	0.2~0.6	22.5~67.5
ViT-L16	26	Linear	6.7~14.6	2.3~14.2	36.8~213.2
	24	LayerNorm	5.5~7.9	0.5~1.9	7.4~23.5
	24	GELU	5.0~7.2	0.4~1.1	6.8~16.1

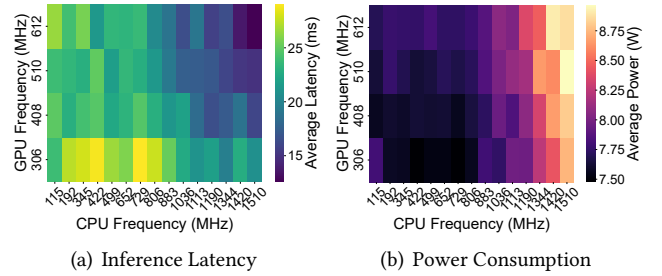
**Figure 5.** Comparison between DVFS switching overhead and end-to-end inference latency for ResNet-18.

while compute units sit idle. This extended dwell time amplifies static thermal leakage power, exacerbating chip heating and accelerating the onset of thermal throttling.

Compared to the switching overhead of operator-level DVFS and the operation oblivious of DNN characteristics in model-level DVFS, SparseDVFS aggregates operators into super-blocks based on their specific nature, enabling the execution time of single blocks to significantly exceed frequency switching overhead. This design amortizes hardware switching costs while capturing fine-grained model features.

2.5 Impact of Clock Frequency on Performance

We execute a grid search across the CPU and GPU frequency settings to characterize their joint impact on inference latency and power absorption consumption. Figure 6 indicates that inference latency is not uniformly sensitive to frequency scaling. For compute-bound phases, increasing GPU frequency yields significant latency reductions. As frequency increases, however, the system encounters a diminishing returns threshold where memory bandwidth becomes the primary bottleneck. At high GPU frequencies, the performance gain flattens out while power consumption continues to climb quadratically. We also observe a clear decoupling

**Figure 6.** The impact of CPU and GPU clock frequencies on system performance for ResNet-18. The heatmaps illustrate the non-linear sensitivity of (a) inference latency and (b) power consumption to frequency scaling.

between minimum latency and minimum energy consumption. While the highest frequency configuration minimizes latency, however, it may lead to sub-optimal energy consumption due to the non-linear increase in power.

By mapping specific operator characteristics, such as sparsity and arithmetic intensity, to the relationship between latency and power, SparseDVFS bypasses the limitations of conventional governors. It allows the system to proactively select the energy-optimal frequency that satisfies the target latency constraint, rather than blindly pursuing maximum performance at the cost of excessive energy consumption.

3 SparseDVFS

SparseDVFS sits between the deep learning framework (e.g., PyTorch) and the operating system interfaces. It intercepts the execution of DNN inference, analyzes the upcoming workload, and proactively adjusts hardware settings before execution begins. Unlike reactive governors that rely on trailing utilization indicators and thus suffer in performance

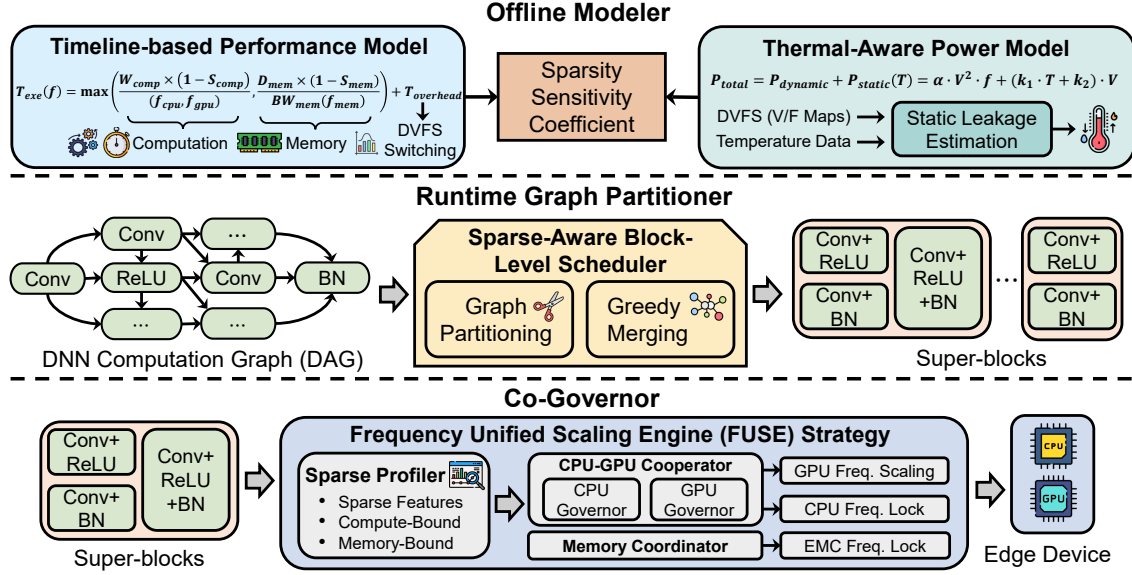


Figure 7. SparseDVFS overview. The system operates in three stages: (1) the offline modeler establishes a deterministic mapping between operator characteristics and the optimal CPU-GPU-memory frequency triplet through performance analysis and thermal-aware power modeling; (2) the runtime graph partitioner performs greedy operator aggregation to construct super-blocks, ensuring that fine-grained DVFS scaling satisfies an application-provided latency amortization constraint; (3) the co-governor executes coordinated frequency scaling across heterogeneous components via the frequency unified scaling engine (FUSE) strategy, while masking DVFS switching latency through a pipelined look-ahead mechanism.

because of the reasons outlined in Section 2, SparseDVFS uses a predictive model based on static graph analysis and dynamic sparsity profiling.

3.1 Architecture

Figure 7 shows the architecture of SparseDVFS, which consists of three stages. ❶ The offline modeler operates prior to deployment, characterizing the hardware’s behavior through performance and power models, thereby building a mapping between operator characteristics and the optimal CPU-GPU-memory frequency triplet. ❷ During inference, the runtime graph partitioner dynamically segments the DNN computation graph into super-blocks to amortize switching overheads. ❸ The co-governor enforces the optimal frequency configurations for the blocks across the CPU, GPU, and memory subsystems, utilizing a look-ahead mechanism to hide DVFS switching latency. This design ensures that SparseDVFS can deliver significant energy savings without compromising the strict latency requirements of edge inference.

3.2 Offline Modeler

This component aims to eliminate the cold-start latency and unpredictability inherent in existing black-box approaches by establishing a deterministic, physics-based mapping between operator characteristics and hardware states.

Timeline-based performance model. The execution performance of DNN operators on edge hardware is governed by

the interplay between computational throughput and memory bandwidth, classically described by the Roofline model. However, traditional Roofline formulations assume dense computation, failing to account for the dynamic workload reduction introduced by algorithmic sparsity.

To address this limitation, we formulate a timeline-based performance model that explicitly incorporates sparsity sensitivity coefficients to predict the execution time T_{exe} of an operator under varying frequency configurations. This model treats execution time not as a monolithic random variable, but as a convex piecewise linear function determined by the scarce resource that may represent a bottleneck.

We derive the execution time by calculating the maximum latency between the compute-bound execution path and the memory-bound data transfer path, augmented by a system overhead constant. For a given frequency triplet $f = [f_{cpu}, f_{gpu}, f_{mem}]$, the predicted execution time is

$$T_{exe}(f) = \max\left(\frac{W_{comp} \cdot (1 - S_{comp})}{P_{peak}(f_{cpu}, f_{gpu})}, \frac{D_{mem} \cdot (1 - S_{mem})}{B_{mem}(f_{mem})}\right) + T_{overhead} \quad (1)$$

where W_{comp} represents the theoretical computational workload (FLOPs) derived statically from the operator’s dimensions, while $S_{comp} \in [0, 1]$ is the computational sparsity coefficient, representing the fraction of zero-valued activations that the hardware can effectively skip.

The term $W_{comp} \cdot (1 - S_{comp})$ in Eq. (1) thus quantifies the effective computational load, revealing the execution

slack available for frequency downscaling in sparse layers. Conversely, the memory is governed by the total data volume D_{mem} and the storage sparsity coefficient S_{mem} , which captures the bandwidth savings dependent on the sparsity. For structured sparsity, $S_{mem} \approx S_{comp}$, whereas for unstructured sparsity, S_{mem} approaches zero due to random access inefficiencies. $\mathcal{P}_{peak}$ and $\mathcal{B}_{mem}$ denote the peak theoretical performance and memory bandwidth at respective frequencies, obtained via offline micro-benchmarking. Finally, the term $T_{overhead}$ accounts for constant system latencies such as kernel launch overhead and synchronization barriers.

Conventional history- or reactive-based predictors [34, 80] rely on past resource utilization to estimate future execution demand. However, they suffer from severe reactive lag during microsecond/millisecond-level intra-DNN operator shifts, as frequency scaling only occurs after performance degradation or power spike has already occurred. In contrast, SparseDVFS proactively intercepts compiled TensorRT or CUDA execution queues ahead of runtime. By inspecting incoming operator dimensions and dynamic sparsity profiles before kernel launch, SparseDVFS maps workloads to optimal frequency settings without incurring reactive overheads.

Table 3 shows the performance of the timeline-based accuracy performance model. We take the FLOPs-based predictor as the baseline, which utilizes theoretical computational workload to build a simple linear model for latency prediction. As shown, the FLOPs-based predictor only achieves 8.2%~18.4% accuracy across all models we test. By incorporating sparsity sensitivity and hardware-specific bottlenecks, our predictor achieves $\pm 10\%$ accuracies of 86.25%, 84.6%, 82.1%, and 80.8% for ResNet-18, ResNet-101, ViT-B16, and ViT-L16, respectively. This performance is even comparable with the black-box model nn-Meter, which relies on extensive kernel-level profiling. Note that our predictor is designed to be extremely lightweight that only requires 32 bytes, whereas nn-Meter’s size is nearly 856MB which is not suitable for online prediction on edge devices.

Thermal-aware power model. Accurate energy optimization in embedded AI chips requires a precise estimation of power consumption that accounts for the non-linear dependency of static leakage current on temperature. A simplistic model ignoring thermal feedback risks underestimating power draw during intensive inference workloads, leading to thermal throttling and performance jitter.

We employ a thermal-aware power model that decouples total power into dynamic switching power and temperature-dependent power absorption due to leakage current. The dynamic component is modeled as a function of the activity factor, supply voltage, and frequency, while the static component is modeled as a linear function of temperature, scaled by voltage. Power consumption is then:

$$\begin{aligned} P_{total}(f, T) &= P_{dynamic} + P_{static}(T) \\ &= \alpha(S_{comp}) \cdot V^2 \cdot f + (k_1 \cdot T + k_2) \cdot V \end{aligned} \quad (2)$$

where the activity factor α is modeled as a function of sparsity S_{comp} , as sparse data streams induce fewer bit-flips in the datapath, thereby reducing dynamic power. The coefficients k_1 and k_2 are device-specific thermal constants derived from regression analysis through offline profiling data, capturing the sensitivity of leakage current to junction temperature and the base leakage at a reference temperature, respectively.

In Figure 6, power consumption exhibits non-monotonic behavior under passive cooling. This result is explained by the dynamic temperature-leakage feedback loop inherent in Eq. (2). When operating under high frequencies, elevated power dissipation rapidly elevates the chip junction temperature T . Under passive cooling, heat accumulation causes $P_{static}(T)$ to increase non-linearly. This elevated thermal leakage, combined with quadratic voltage scaling, offsets the expected power reductions at intermediate operating states, giving rise to non-monotonic power shifts.

Our thermal-aware power model enables the system to perform proactive thermal management. When the device temperature T rises, the model predicts the super-linear increase in P_{static} , prompting the scheduler to select a voltage-frequency pair that not only satisfies the immediate latency constraint but also minimizes total energy $E = P_{total} \times T_{exe}$, as derived from Eq. (1), by potentially lowering voltage to reduce the thermal envelope. This feedback loop prevents the device from hitting hardware thermal limits.

Figure 8 presents a comparison between measured data and the prediction data of our offline modeler for ResNet-18. Figure 8(a) demonstrates that the timeline-based performance model accurately characterizes operator latency across the joint CPU-GPU frequency scaling space, with a mean error below 2%. Simultaneously, Figure 8(b) shows the prediction accuracy of the thermal-aware power model, which successfully captures the super-linear increase in power consumption driven by temperature-dependent static

Table 3. Accuracy of the timeline-based accuracy performance model.

Predictor	Model	$\pm 10\%$ Accuracy	Size
FLOPs-based [34]	ResNet-18	18.4%	8B
	ResNet-101	12.8%	
	ViT-B16	11.5%	
	ViT-L16	8.2%	
nn-Meter [80]	ResNet-18	91.5%	~856MB
	ResNet-101	88.1%	
	ViT-B16	87.2%	
	ViT-L16	85.9%	
Timeline-based (Ours)	ResNet-18	86.2%	32B
	ResNet-101	84.6%	
	ViT-B16	82.1%	
	ViT-L16	80.8%	

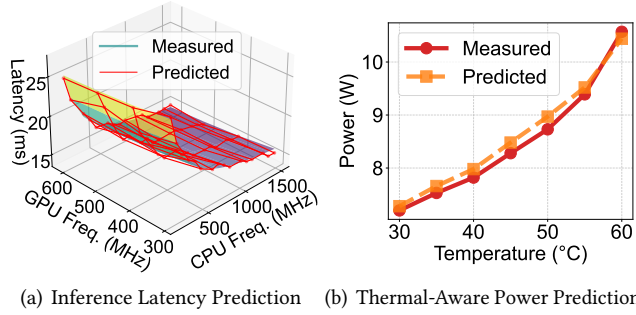


Figure 8. An example of offline modeling for ResNet-18: (a) latency prediction under different CPU/GPU frequency combinations and (b) thermal-aware power prediction.

leakage. Overall, our models provide the necessary deterministic foundation for proactive DVFS scaling, effectively bypassing the unpredictability and cold-start overheads inherent in traditional black-box learning methods.

3.3 Runtime Graph Partitioner

While the offline modeler provides optimal configurations for individual operators, applying these settings "as-is" at runtime is likely inefficient due to the non-negligible hardware switching latency (T_{switch}) associated with voltage regulation. To reconcile the conflict between fine-grained optimization and switching overhead, the runtime graph partitioner dynamically transforms the fine-grained operator graph into a sequence of execution-friendly super-blocks.

Greedy merging and super-block construction. The partitioner operates on the DNN directed acyclic graph by employing a greedy merging algorithm that traverses the topologically sorted sequence of operators. The core logic dictates that adjacent operators should be fused into a single super-block if they share similar sparsity characteristics and thus enjoy compatible optimal frequencies, or if the execution time of an individual operator is insufficient to amortize the cost of a frequency switch.

Algorithm 1 traverses the topologically sorted computation graph to construct super-blocks greedily. It initializes the first block with the leading operator (*line 1*) and tentatively adds the next operator v_{next} in the DNN sequence (*line 3*). For each candidate merge, the partitioner predicts the candidate's optimal frequency (*line 4*) and calculates the current block's execution time (*line 5*) to evaluate homogeneity and amortization within a single conditional check (*line 7*).

If two consecutive blocks are considered homogenous, that is, $f_{next} \approx f_{curr}$, the algorithm checks if the optimal frequency of the candidate operator f_{opt}^{next} falls within a tolerance threshold of the current block's frequency f_{opt}^{curr} . If the frequencies diverge significantly, the partitioner creates a block boundary (*line 11*). However, this decision is overridden by the amortization constraint (i.e., $T_{est} < N \times T_{switch}$),

which ensures that the candidate is merged (*line 8*) if the block's duration is insufficient to justify a frequency switch. If both conditions fail, the current block is finalized (*line 11*) and a new block is initiated (*line 12~13*).

Algorithm 1 Sparsity-aware Greedy Graph Partitioning

Input: $G(V, E)$: DNN computation graph sorted topologically
Input: M_{perf}, M_{power} : Offline performance and power models
Input: T_{switch} : Hardware frequency switching latency
Input: N : Amortization granularity factor
Output: $\mathcal{S}$: Sequence of (super-block, f_{opt}) tuples

```

1:  $\mathcal{S} \leftarrow \emptyset, Block_{curr} \leftarrow \{v_1\}, f_{curr} \leftarrow M_{perf}.predict(v_1)$ 
2: for  $i = 2$  to  $|V|$  do
3:    $v_{next} \leftarrow V[i]$ 
4:    $f_{next} \leftarrow M_{perf}.predict(v_{next})$ 
5:    $T_{est} \leftarrow M_{perf}.calc\_time(Block_{curr}, f_{curr})$ 
6:   // Check amortization and similarity constraints
7:   if  $T_{est} < N \times T_{switch}$  or  $f_{next} \approx f_{curr}$  then
8:      $Block_{curr} \leftarrow Block_{curr} \cup \{v_{next}\}$ 
9:      $f_{curr} \leftarrow \max(f_{curr}, f_{next})$  // Conservative merge
10:  else
11:     $\mathcal{S}.append((Block_{curr}, f_{curr}))$ 
12:     $Block_{curr} \leftarrow \{v_{next}\}$ 
13:     $f_{curr} \leftarrow f_{next}$ 
14:  end if
15: end for
16:  $\mathcal{S}.append((Block_{curr}, f_{curr}))$ 
17: return  $\mathcal{S}$ 

```

Latency amortization constraint. The partitioner enforces a strict latency amortization constraint to ensure that the energy savings from a DVFS switching latency are not negated by the performance penalty of the switch itself. A super-block is finalized only if its cumulative estimated execution time is such that:

$$T_{block} > N \times T_{switch} \quad (3)$$

where N is a tunable granularity factor that dictates the aggressiveness of the aggregation. If the accumulated duration of the current block is less than $N \times T_{switch}$, the partitioner is forced to merge the separate operator, effectively absorbing that into the larger block.

We analyze the reduction in scaling granularity and the corresponding mitigation of switching overhead across diverse architectures. Figure 9 shows that the partitioner effectively consolidates fine-grained operators into computationally substantial super-blocks to satisfy the latency amortization constraint. ResNet-18 is trimmed from 21 operators to just 2 super-blocks, while the deeper ResNet-101 is reduced from 105 operators to 16. Similarly, the partitioner aggregates the 38 and 74 operators of ViT-B16 and ViT-L16 into 8 and 12 super-blocks, respectively.

This reduction in granularity translates directly into a drastic decrease in cumulative DVFS switching latency. Compared to operator-level DVFS scaling, SparseDVFS reduces the total switching latency by 7.0× for ResNet-18 and up to 8.5× for ViT-B16. These results reveal that by enforcing

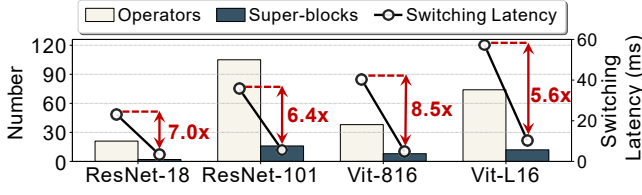


Figure 9. Impact of runtime graph partitioner on operator aggregation granularity and DVFS switching latency. We set aggregation factor $N = 5$.

$T_{block} > N \times T_{switch}$, SparseDVFS successfully amortizes the non-negligible DVFS switching latency. Nonetheless, we measure the effect of the configuration parameters determining the operation of the graph partitioner in Section 5.4.

3.4 Co-governor

This component aims to execute the schedule generated by the partitioner. To mitigate the antagonistic effect observed across heterogeneous edge AI platforms, as discussed in Section 2.3, the co-governor implements a frequency-unified scaling engine (FUSE) strategy. Instead of reacting to load, our co-governor proactively sets the frequency vector based on the known characteristics of the incoming super-block. **CPU/GPU/memory coordination.** SparseDVFS implements a proactive race-to-submit policy by treating CPU, GPU, and memory frequencies as a coupled vector. By intercepting CUDA streams and TensorRT execution queues, the co-governor identifies super-block boundaries and immediately locks the CPU to a high-performance state to accelerate pre-fill tasks like data preprocessing and kernel launches. This prevents the GPU starvation typically caused by reactive governors downclocking the CPU during GPU-intensive phases. Once the kernel is offloaded, the CPU frequency is allowed to drop, effectively masking hardware wake-up latencies and maximizing system-wide energy efficiency without negative effects on throughput.

As for memory, for sparse operators, SparseDVFS co-governor builds on the insight from the Roofline model that performance is memory-bound. In the presence of sparse blocks, the governor locks the memory controller to a high frequency to maximize bandwidth, while simultaneously lowering the GPU core frequency. This configuration saves GPU power without hurting performance, as the GPU was stalling on memory anyway. In the case of dense blocks, the governor scales the memory controller down to the minimum level required to support the GPU's compute throughput, reducing the power draw of the memory interface.

Pipeline-based look-ahead mechanism. To mitigate the impact of DVFS switching latency, the co-governor utilizes a pipeline-based look-ahead mechanism. Conventional DVFS

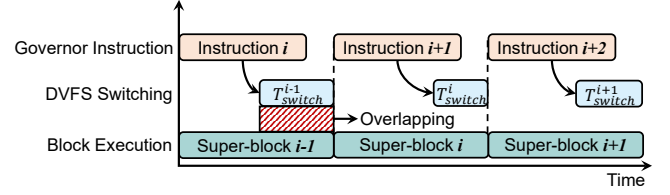


Figure 10. Pipeline-based Look-Ahead DVFS.

Table 4. Comparison of DVFS switching latency with conventional DVFS (serial execution) and pipeline-based look-ahead. We set the aggregation factor as $N = 5$.

Model	Default DVFS	Look-Ahead DVFS
ResNet-18	7.23ms	0.12ms
ResNet-101	10.81ms	1.45ms
ViT-B16	5.44ms	0.72ms
ViT-L16	7.92ms	1.08ms

governors typically execute scaling commands serially, causing significant execution stalls as the system waits for voltage and frequency levels to stabilize. As shown in Figure 10, SparseDVFS overcomes this bottleneck by employing a pipelined command queue that submits frequency scaling instructions for the subsequent super-block to hardware registers before the current super-block completes its execution. These asynchronous instructions mask the non-negligible DVFS switching latency, without inducing pipeline bubbles.

The benefits of this mechanism are quantified in Table 4. For instance, in ResNet-101, conventional serial DVFS switching incurs a cumulative latency of 10.81ms, which SparseDVFS drastically reduces to just 1.45ms. Similarly, for ViT-L16, the switching overhead is reduced from 7.92ms to 1.08ms. By overlapping the physical switching period with active computation, SparseDVFS ensures that the energy efficiency gains achieved through fine-grained frequency modulation are not negated by system-level performance penalties.

Synergy of blocking and Look-ahead. The look-ahead mechanism relies fundamentally on the structural foundation provided by graph blocking. If look-ahead instruction queuing is applied alone without super-block partitioning, microsecond-level overlapping operators issue high-frequency, back-to-back hardware register writes. This rapid succession saturates hardware control registers, causing instruction queue overflows and stalling the hardware pipeline.

Conversely, when combined with the graph partitioner, the look-ahead mechanism achieves complete synergy. Graph blocking creates execution windows long enough ($T_{block} > N \times T_{switch}$) to absorb asynchronous frequency transition commands, allowing physical switching overheads to be fully overlapped with active computation. As quantified in Table 3, this co-design eliminates the baseline serialization penalty (5.44 ~ 10.81 ms), overhead to just 0.12 ~ 1.45 ms

Table 5. Detailed configuration of four DNN models.

Model	Op. Count	FLOPs (G)	Size (MB)
ResNet-18	21	1.82	11.69
ResNet-101	105	7.87	44.55
ViT-B16	38	11.29	58.07
ViT-L16	74	39.86	203.36

and fully unlocking the theoretical energy savings of fine-grained DVFS.

4 Implementation

SparseDVFS sits between the deep learning framework and the Linux kernel frequency governors. The offline modeler and the runtime graph partitioner are implemented in Python, using PyTorch 2.1 for computation graph extraction and execution time prediction. The co-governor is implemented in C++ to minimize overhead during inference. The system is deployed on an NVIDIA Jetson Orin Nano edge device (JetPack 6.0 with Linux Kernel 5.15). This edge platform features an Ampere architecture GPU and an ARM Cortex CPU, sharing unified 8GB DRAM.

Unlike existing DVFS governors that operate entirely within the OS kernel space, SparseDVFS operates in userspace by intercepting execution streams. Specifically, we detect upcoming operator sequences by intercepting CUDA streams [49] and TensorRT execution queues [51]. During runtime, the co-governor applies the FUSE strategy by directly manipulating the device’s frequency scaling interfaces. We utilized the **jetson_stats** [2] to monitor real-time power consumption across the CPU, GPU, and memory rails. To bypass the conventional governors (e.g., **Linux schedutil** [17] for CPUs and **simple_ondemand** [56] for GPUs), SparseDVFS takes direct control of the clock frequencies via the **jetson_clocks** provided by NVIDIA **nvpmode** [50] and **Linux sysfs** nodes.

5 Evaluation

5.1 Experiment Setup

Models and datasets. As shown in Table 5, we use CNN-based (e.g., ResNet-18 and ResNet-101 [20]) and Transformer-based (e.g., ViT-B16 and ViT-L16 [12]) architectures to demonstrate the use of SparseDVFS with different workloads. Models are pre-trained on ImageNet-2012 [35] and converted to ONNX [54] before being input to SparseDVFS.

Frequency configurations. To comprehensively sweep the optimization space, we utilize the full range of accessible frequencies on the Jetson Orin Nano. For CPU governor, there are 20 distinct frequency levels ranging from 115MHz to 1510MHz. For GPU governor, there exist 5 distinct frequency levels: 306/408/510/612/624MHz.

Baselines. We compare SparseDVFS against four power management strategies:

- **Default DVFS:** The standard Linux **schedutil** governor for the CPU and NVIDIA’s **simple_ondemand** governor for the GPU, which is our default baseline.
- **nvpmode (MAX-N) [50]:** A static maximum performance mode provided by NVIDIA JetPack that locks all clocks to their highest frequencies.
- **GearDVFS [40]:** A deep reinforcement learning (DRL)-based governor optimized for concurrent workloads that operates at a coarse model-level granularity.
- **Ascend-DVFS [70]:** A state-of-the-art fine-grained operator-level DVFS governor employing a genetic algorithm for frequency scaling.

Metrics. We evaluate SparseDVFS using three primary metrics that characterize the efficiency-performance-reliability trade-offs for edge inference.

- **Energy-per-inference:** It measures the energy consumed for a single forward pass, which is critical for edge devices operating under strict power envelopes. We obtain it by sampling real-time power draw across the CPU, GPU, and memory rails via the **jetson_stats** [2] and integrating the power over the operator execution timeline.
- **End-to-end latency:** It captures real-time performance, ensuring compliance with the strict timing requirements of edge AI applications. Latency is measured using a hardware timer to account for both active computation and the potential stalls induced by frequency switching.
- **Cost-Gain Ratio [10]:** It is a normalized metric to evaluate the net utility of DVFS scaling. It is defined as the ratio of latency penalty incurred to the energy savings achieved. This is essential for assessing whether the cost of hardware switching latency is effectively amortized by the gain in energy efficiency.

We also monitor thermal stability and frame-rate jitter during sustained workloads. This is important for passively-cooled edge hardware to ensure that the system does not trigger severe hardware throttling that could compromise inference.

5.2 Results

Energy and power. SparseDVFS provides great energy benefits. As shown in Figure 11, the power profile of SparseDVFS is significantly lower and more stable than the baselines. While **nvpmode** consistently draws peak power (15W) and **GearDVFS** fluctuates in the 10-12W range, SparseDVFS dynamically modulates power between 7W and 10W depending on the operator sparsity. By reducing frequency during high-sparsity phases (e.g., ReLU-heavy blocks), it drastically cuts the energy area under the curve.

Compared to Default DVFS, Figure 12 demonstrates that SparseDVFS achieves an average energy efficiency gain of 78.17%. Differently, **nvpmode** achieves a 46.86% gain, mostly due to reduced total execution time, **GearDVFS** achieves a

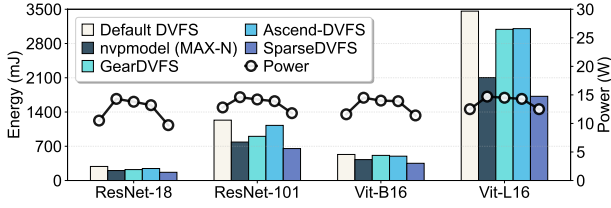


Figure 11. Energy and power profile comparison across different DNN models.

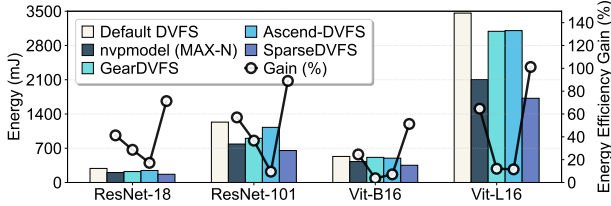


Figure 12. Normalized energy efficiency gain relative to the Default DVFS baseline.

20.25% gain, and Ascend-DVFS achieves a 11.33% gain. These results highlight the benefits of the sparsity-aware block-level approach over coarser model-level strategies.

End-to-end latency. As shown in Figure 13, SparseDVFS demonstrates the ability to maintain competitive latency while aggressively saving power. Among the systems we test, nvpmode (MAX-N) achieves the lowest latency by running all clocks at maximum. Nonetheless, SparseDVFS incurs a modest latency penalty of approximately 12.8% compared to MAX-N, while consistently outperforming the Default DVFS governors and matching the latency profile of GearDVFS.

Crucially, the switching latency reveals that SparseDVFS successfully overcomes the transition bottlenecks prevalent in other fine-grained methods. While Ascend-DVFS suffers from prohibitive switching overheads that escalate with model depth (e.g., reaching over 30ms for ViT-L16), SparseDVFS reduces these costs by up to 8.5 $\times$. By consolidating operators into super-blocks and employing a pipeline-based look-ahead mechanism, the cumulative switching penalty is suppressed to approximately 1ms for most models. This demonstrates that our strategy effectively masks hardware transition periods, ensuring that fine-grained energy modulation does not degenerate into a performance bottleneck.

Cost-gain analysis. To evaluate the trade-off between energy gain and latency performance, we utilize the a concept of Cost-Gain Ratio [10] (lower is better), by measuring the latency cost incurred for every unit of energy gain. Figure 14 shows that SparseDVFS achieves a far superior Cost-Gain Ratio, compared with the baselines. In contrast, GearDVFS and Ascend-DVFS incur higher latency penalties for their

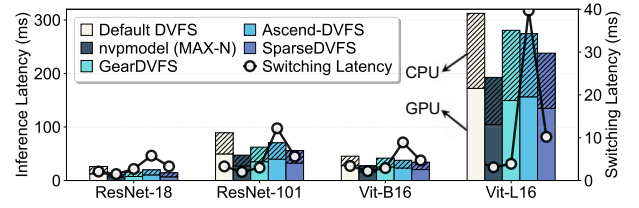


Figure 13. Comparison of end-to-end inference latency across different DNN models.

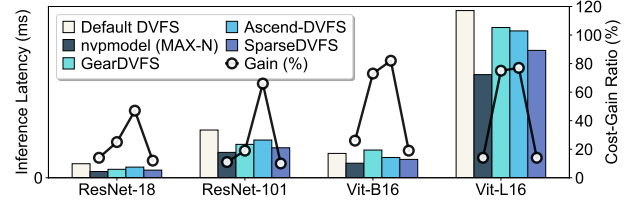


Figure 14. Comparison of Cost-Gain Ratio across different DVFS strategies.

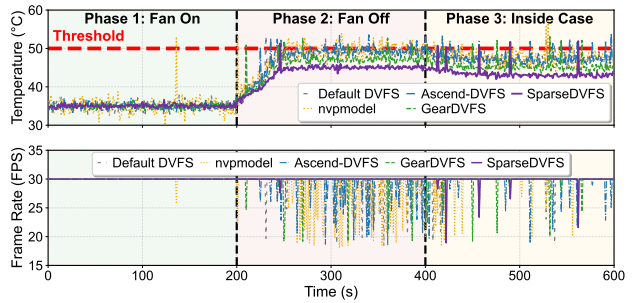


Figure 15. Thermal stability and frame rate jitter analysis of NVIDIA Jetson Orin Nano under sustained load (ViT-B16).

respective energy savings. Instead, nvpmode achieves a performance close to SparseDVFS, but running at thermal limits.

Thermal throttling and stability. We evaluate the thermal throttling of SparseDVFS for stability analysis. As shown in Figure 15, nvpmode and Default DVFS quickly reach the thermal throttle temperature, leading to severe frame rate jitter. SparseDVFS, due to its lower average power draw, maintains a stable temperature trajectory, delaying the onset of thermal throttling significantly. The jitter level for SparseDVFS is the lowest among all systems we test, providing evidence of its use on passively-cooled edge devices.

5.3 Ablation Study

We conduct an ablation study on ViT-B16 using the DOTA-v1.0 and VisDrone 2019 datasets to validate the co-governor

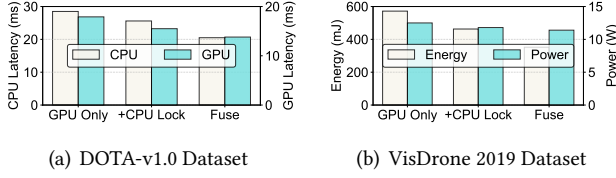


Figure 16. Ablation studies of each component of the co-governor on (a) DOTA-v1.0 and (b) VisDrone 2019 datasets, respectively. We use ViT-B16 as test load.

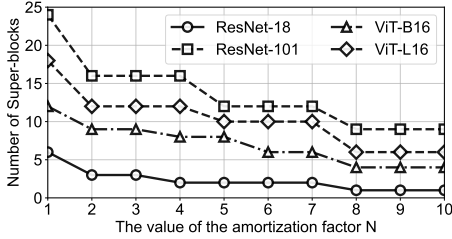


Figure 17. Impact of the amortization factor N on super-block aggregation granularity.

components. As shown in Figure 16, *GPU Only* scaling suffers from the CPU-GPU antagonistic effect, where independent CPU frequency drops during GPU-intensive phases induce wake-up latency spikes, the *+CPU Lock* configuration mitigates GPU starvation by maintaining the CPU in a high-performance state during critical kernel launch windows. Ultimately, the complete FUSE strategy that treats CPU, GPU, and memory frequencies as a coupled vector, achieves optimal energy efficiency by carefully synchronizing transitions and minimizing total energy consumption across diverse edge workloads.

5.4 Operator Aggregation Granularity

Number of super-blocks. The amortization factor N regulates the trade-off between scaling granularity and DVFS switching latency. Figure 17 shows a consistent inverse correlation between N and super-block count. For layer-dense models like ResNet-101, small values for N capture fine-grained sparsity but incur excessive switching risk. Increasing N to 10 effectively consolidates operators into computationally significant units. Conversely, ViT-L16 exhibits a more gradual aggregation curve. Because its Transformer blocks possess substantial execution times, the amortization constraint (i.e., $T_{block} > N \times T_{switch}$) is easily satisfied even at lower N values.

Inference latency and switching. Figure 18 illustrates how N balances inference latency and switching overhead. At $N = 1$, switching latency dominates the timeline, particularly for ResNet-101. Although ViT-L16’s bigger operators

buffer against switching costs, the Look-Ahead mechanism remains essential for masking transitions between its massive MLP and Attention blocks. Notably, an excessively high N induces a frequency lag effect, where SparseDVFS misses downclocking opportunities by merging distinct and dense phases into a single block.

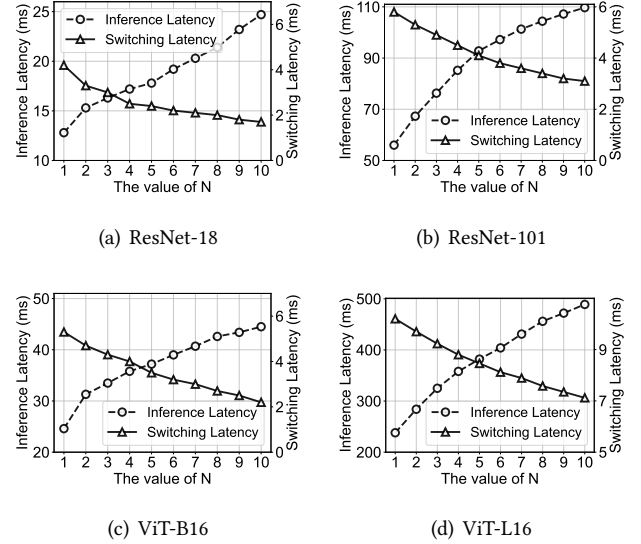


Figure 18. Trade-off between inference latency and switching overhead relative to N .

Note that the average 12.8% latency penalty reported in Section 5.2 is not a rigid performance degradation, but rather a flexible, user-tunable constraint governed directly by the amortization factor N . The existence of this tunable knob allows users to dynamically regulate the trade-off based on specific deployment requirements. For strict real-time or time-sensitive tasks where latency are non-negotiable, operators can configure a lower amortization threshold (e.g., $N = 2$). Conversely, energy-constrained IoT or energy-harvesting edge nodes can specify a higher N (e.g., $N = 5 \sim 8$) to maximize power efficiency.

Power and energy. As shown in Figure 19, the energy-per-inference quota displays a characteristic U-shaped curve relative to N . At low N values, the system suffers from high energy consumed during the frequent high-voltage transitions between blocks. Conversely, at very high N values, the lack of granularity prevents the co-governor from identifying short-lived sparse windows, forcing the GPU to remain in high-power states even during memory-bound phases. For the computationally heavy ViT-L16, the energy savings are most pronounced at moderate N settings.

6 Related Work

Performance modeling for edge inference. Existing studies like nn-Meter [80] and nn-stretch [71] rely on kernel-level

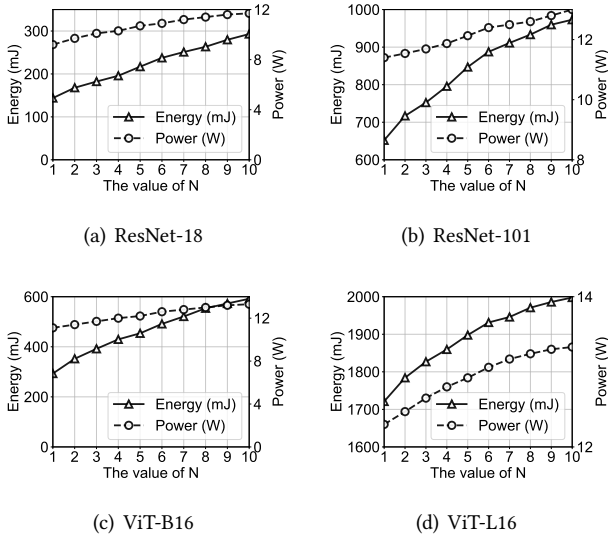


Figure 19. Impact of the aggregation factor N on system power and energy consumption.

profiling, while EdgeFM [77], Im-Meter [69], and nnPerf [7] focus on model-level latency modeling (i.e., the aggregate end-to-end inference time measured from a black-box perspective). While computationally efficient, this coarse-grained approach ignores intra-inference computational fluctuations and hardware transition overheads occurring between individual operators. Unlike these black-box models or memory-centric simulations like Symphony [58] and Spatula [14], SparseDVFS implements a white-box and sparsity-aware model. It maps operator-level sparsity to optimal frequency triplets with deterministic and lightweight lookup complexity suited for low-latency edge execution.

DVFS technology for AI accelerators. Frequency scaling evolved from coarse model-level adaptation (e.g., zTT [30] and GearDVFS [40]) to prohibitive layer- or operator-level scaling (e.g., Ascend-DVFS [70] and CLONE [66]). Existing work also includes techniques to eliminate component interference, like FUSE [81] and collaborative inference methods like DVFO [84] and E4 [83]. While previous studies focus on high-overhead operator-level scaling or low-precision model-level governors, SparseDVFS aggregates operators into super-blocks and employs a look-ahead mechanism, amortizing the non-negligible DVFS switching latency.

Sparse optimization for DNN models. Exploiting model sparsity and low-precision representation is a primary lever for reducing computational and memory intensity of DNN models. Algorithmic efforts like nmSPARSE [39], Sparse VILA [29], and FlashAttention-2 [9] focus on designing specialized sparse kernels or hardware primitives like SparseCore [27]. While these works focus on designing specialized

sparse kernels, SparseDVFS treats operator sparsity as a dynamic control signal for DVFS. It bridges the gap between algorithmic sparsity and the physical power states of commercial hardware, enabling unified frequency modulation that saves energy without sacrificing model performance.

7 Discussion

Non-DNN workloads. The physics-based modeling and graph partitioner of SparseDVFS are fundamentally workload-agnostic. The framework models workloads as general data-flow directed acyclic graphs (DAGs). Consequently, non-DNN pipelines structured as execution DAGs (e.g., image processing or sensor fusion) can directly adopt SparseDVFS to achieve proactive, bandwidth-aware frequency scaling.

Sparsity patterns. SparseDVFS accommodates diverse sparsity patterns [39] by incorporating the storage sparsity coefficient S_{mem} . Unlike structured sparsity [73, 78, 86] (i.e., prune according to regular patterns to form a predictable sparse structure), unstructured sparsity [16, 26, 74, 85] (i.e., random pruning to form arbitrary sparsity patterns) induces random memory accesses that degrade cache efficiency, driving $S_{mem} \rightarrow 0$ even under high computational sparsity S_{comp} . Our offline modeler explicitly captures this cache degradation, prompting the co-governor to lock the memory controller to peak frequency for bandwidth while downclocking GPU cores to save dynamic power.

Memory access awareness. SparseDVFS relies on the compute reduction aspect of sparsity. However, memory access overhead, specifically the latency of DRAM row activation, remains a critical factor. We may integrate real-time memory controller performance counters [32, 38, 44, 67, 76] into the runtime partitioner. This would allow the system to dynamically adjust parameters based on bus contention, rather than solely on static profiling.

Hardware adaptation. SparseDVFS naturally scales to multi-GPU servers (e.g., NVIDIA DGX) running large-scale workloads like LLMs [5, 23]. During memory-bound LLM decode phases [3, 4, 42], SparseDVFS can coordinate multi-GPU core and memory subsystems by downclocking compute units during sparse tensor retrievals to minimize idle power draw and delay thermal throttling under strict latency budgets. Furthermore, transfer learning [47, 57, 79, 87] can map learned V/F configurations across heterogeneous edge hardware (e.g., from GPU to TPU) without exhaustive profiling.

8 Conclusion

We presented SparseDVFS, a block-level DVFS technique for energy efficiency edge inference. By elevating operator sparsity to a first-class metric, SparseDVFS bridges the gap between operators and the power states of commercial edge hardware. The block-level aggregation strategy in SparseDVFS successfully captures intra-inference computation, while mitigating the non-negligible DVFS switching latency.

Acknowledgment

We thank our shepherd and the anonymous reviewers for their constructive and insightful feedback. This work was partly supported by the Swedish Foundation for Strategic Research (SSF).

References

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016). doi:10.48550/arXiv.1607.06450
- [2] Raffaello Bonghi. 2026. jetson-stats: Interactive system-monitoring and control utility for NVIDIA Jetson family. https://github.com/rbonghi/jetson_stats Accessed: 2026-03-01.
- [3] Hao Mark Chen, Zhiwen Mo, Guanxi Lu, Shuang Liang, Lingxiao Ma, Wayne Luk, and Hongxiang Fan. 2026. Fasttts: Accelerating test-time scaling for edge LLM reasoning. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 732–748. doi:10.1145/3779212.3790161
- [4] Wenyang Chen, Chengzhi Lu, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2026. High Throughput and Low Latency LLM Serving via Adaptive KV Caching. In *Proceedings of the 21st European Conference on Computer Systems*. 56–71. doi:10.1145/3767295.3803570
- [5] Yukang Chen, Weihao Cui, Han Zhao, Ziyi Xu, Xiaoze Fan, Xusheng Chen, Yangjie Zhou, Shixuan Sun, Bingsheng He, and Quan Chen. 2026. Towards High-Goodput LLM Serving with Prefill-decode Multiplexing. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2030–2047. doi:10.1145/3779212.3790236
- [6] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving heterogeneous machine learning models on {Multi-GPU} servers with {Spatio-Temporal} sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 199–216. <https://www.usenix.org/conference/atc22/presentation/choi-seungbeom>
- [7] Haolin Chu, Xiaolong Zheng, Liang Liu, and Huadong Ma. 2023. nnPerf: Demystifying DNN runtime inference latency on mobile platforms. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*. 125–137. doi:10.1145/3625687.3625797
- [8] Weihao Cui, Han Zhao, Quan Chen, Hao Wei, Zirui Li, Deze Zeng, Chao Li, and Minyi Guo. 2022. {DVABatch}: Diversity-aware {Multi-Entry}{Multi-Exit} batching for efficient processing of {DNN} services on {GPUs}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 183–198. <https://www.usenix.org/conference/atc22/presentation/cui>
- [9] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023). doi:10.48550/arXiv.2307.08691
- [10] Kemal Davaslioglu and Ender Ayanoglu. 2014. Quantifying potential energy efficiency gain in green cellular wireless networks. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2065–2091. doi:10.1109/COMST.2014.2322951
- [11] Nan Ding and Samuel Williams. 2019. An instruction roofline model for gpus. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 7–18. doi:10.1109/PMBS49563.2019.00007
- [12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=YicbFdNTTy>
- [13] Hongxiang Fan, Stylianos I Venieris, Alexandros Kouris, and Nicholas Lane. 2023. Sparse-dysta: Sparsity-aware dynamic and static scheduling for sparse multi-dnn workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 353–366. doi:10.1145/3613424.3614263
- [14] Axel Feldmann and Daniel Sanchez. 2023. Spatula: A hardware accelerator for sparse matrix factorization. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 91–104. doi:10.1145/3613424.3623783
- [15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 315–323. <https://proceedings.mlr.press/v15/glorot11a.html>
- [16] Ashish Gondimalla, Mithuna Thottethodi, and TN Vijaykumar. 2023. Eureka: Efficient tensor cores for one-sided unstructured sparsity in dnn inference. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 324–337. doi:10.1145/3613424.3614312
- [17] Redha Gouicem, Damien Carver, Jean-Pierre Lozi, Julien Sopena, Baptiste Lepercq, Willy Zwaenepoel, Nicolas Palix, Julia Lawall, and Gilles Muller. 2020. Fewer Cores, More Hertz: Leveraging {High-Frequency} Cores in the {OS} Scheduler for Improved Application Performance. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 435–448. <https://www.usenix.org/conference/atc20/presentation/gouicern>
- [18] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU power modeling for multi-domain voltage-frequency scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 789–800. doi:10.1109/HPCA.2018.00072
- [19] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale preemption for concurrent {GPU-accelerated}{DNN} inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 539–558. <https://www.usenix.org/conference/osdi22/presentation/han>
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778. doi:10.1109/CVPR.2016.90
- [21] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International conference on software engineering*. 523–534. doi:10.1145/2884781.2884791
- [22] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016). doi:10.48550/arXiv.1606.08415
- [23] Mert Hidayetoglu, Aurick Qiao, Michael Wyatt, Jeff Rasley, Yuxiong He, and Samyam Rajbhandari. 2026. Shift parallelism: Low-latency, high-throughput llm inference for dynamic workloads. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1749–1763. doi:10.1145/3779212.3790219
- [24] Mark Hill and Vijay Janapa Reddi. 2019. Gables: A roofline model for mobile socs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 317–330. doi:10.1109/HPCA.2019.00047
- [25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017). doi:10.48550/arXiv.1704.04861
- [26] Geonhwa Jeong, Po-An Tsai, Abhimanyu R Bambhaniya, Stephen W Keckler, and Tushar Krishna. 2025. Enabling unstructured sparse acceleration on structured sparse accelerators. *Proceedings of Machine Learning and Systems* 7 (2025). https://proceedings.mlsys.org/paper_files/paper/2025/hash/

- e2ec2530db26b54d0b3b060c1e4a1bda-Abstract-Conference.html
- [27] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th annual international symposium on computer architecture*. 1–14. doi:10.1145/3579371.3589350
- [28] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G Rogers, Tor M Aamodt, and Nikos Hardavell. 2021. AccelWattch: A power modeling framework for modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International symposium on microarchitecture*. 738–753. doi:10.1145/3466752.3480063
- [29] Samir Khaki, Junxian Guo, Jiaming Tang, Shang Yang, Yukang Chen, Konstantinos N Plataniotis, Yao Lu, Song Han, and Zhijian Liu. 2025. SparseVILA: Decoupling Visual Sparsity for Efficient VLM Inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 23784–23794. doi:10.1109/ICCV51701.2025.02208
- [30] Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. 2021. zTT: learning-based DVFS with zero thermal throttling for mobile devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 41–53. doi:10.1145/3458864.3468161
- [31] Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. 2022. zTT: Learning-based DVFS with zero thermal throttling for mobile devices. *GetMobile: Mobile Computing and Communications* 25, 4 (2022), 30–34. doi:10.1145/3529706.3529714
- [32] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. 2023. MoCA: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 828–841. doi:10.1109/HPCA56546.2023.10071035
- [33] Seah Kim, Hyoukjun Kwon, Jinook Song, Jihyuck Jo, Yu-Hsin Chen, Liangzhen Lai, and Vikas Chandra. 2023. Dream: A dynamic scheduler for dynamic real-time multi-model ml workloads. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 73–86. doi:10.1145/3623278.3624753
- [34] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. 2019. μ layer: Low latency on-device inference using co-operative single-layer acceleration and processor-friendly quantization. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–15. doi:10.1145/3302424.3303950
- [35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012). <https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>
- [36] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551. doi:10.1162/neco.1989.1.4.541
- [37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791
- [38] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 574–587. doi:10.1145/3575693.3578835
- [39] Bin Lin, Ningxin Zheng, Lei Wang, Shijie Cao, Lingxiao Ma, Quanlu Zhang, Yi Zhu, Ting Cao, Jilong Xue, Yuqing Yang, et al. 2023. Efficient gpu kernels for n: m-sparse weights in deep learning. *Proceedings of Machine Learning and Systems* 5 (2023), 513–525. https://proceedings.mlsys.org/paper_files/paper/2023/hash/a10deb4d5227a8ea307ea8ff3cb712f4-Abstract-mlsys2023.html
- [40] Chengdong Lin, Kun Wang, Zhenjiang Li, and Yu Pu. 2023. A workload-aware DVFS robust to concurrent tasks for mobile devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–16. doi:10.1145/3570361.3592524
- [41] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mccnet: Tiny deep learning on iot devices. *Advances in neural information processing systems* 33 (2020), 11711–11722. <https://proceedings.neurips.cc/paper/2020/hash/86c51678350f656dccc7f490a43946ee5-Abstract.html>
- [42] Zejia Lin, Hongxin Xu, Guanyi Chen, Zhiguang Chen, Yutong Lu, and Xianwei Zhang. 2026. Bullet: Boosting gpu utilization for llm serving via dynamic spatial-temporal orchestration. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 290–306. doi:10.1145/3779212.3790135
- [43] Neiwun Ling, Xuan Huang, Zhihe Zhao, Nan Guan, Zhenyu Yan, and Guoliang Xing. 2022. Blastnet: Exploiting duo-blocks for cross-processor real-time dnn inference. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. 91–105. doi:10.1145/3560905.3568520
- [44] Jinshu Liu, Hamid Hadian, Yuyue Wang, Daniel S Berger, Marie Nguyen, Xun Jian, Sam H Noh, and Huaicheng Li. 2025. Systematic cxl memory characterization and performance analysis at scale. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1203–1217. doi:10.1145/3676641.3715987
- [45] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. 2022. VELTAIR: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In *Proceedings of the 27th ACM international conference on architectural support for programming languages and operating systems*. 388–401. doi:10.1145/3503222.3507752
- [46] Pietro Mercati, Andrea Bartolini, Francesco Paterna, Tajana Simunic Rosing, and Luca Benini. 2014. A linux-governor based dynamic reliability manager for android mobile devices. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–4. doi:10.7873/DATE.2014.117
- [47] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? *Advances in neural information processing systems* 33 (2020), 512–523. https://proceedings.neurips.cc/paper_files/paper/2020/hash/0607f4c705595b911a4f3e7a127b44e0-Abstract.html
- [48] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: Timely inference serving for dynamic edge networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 277–290. doi:10.1109/RTSS55097.2022.00032
- [49] NVIDIA Corporation. 2026. NVIDIA CUDA C Programming Guide. <https://docs.nvidia.com/cuda/> Accessed: 2026-03-01.
- [50] NVIDIA Corporation. 2026. NVIDIA Jetson Linux Developer Guide: NVP-Model Command-Line Tool. <https://docs.nvidia.com/jetson/archives/> Accessed: 2026-03-01.
- [51] NVIDIA Corporation. 2026. NVIDIA TensorRT: Programmable Deep Learning Accelerator. <https://developer.nvidia.com/tensorrt> Accessed: 2026-03-01.
- [52] NVIDIA Jetson Modules. 2026. NVIDIA Jetson Modules for Edge AI and Robotics. <https://developer.nvidia.com/embedded/jetson-modules> Accessed: 2026-03-01.
- [53] Young H Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W Lee. 2021. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE,

- 584–597. doi:10.1109/HPCA51647.2021.00056
- [54] ONNX Project Developers. 2026. ONNX: Open Neural Network Exchange. <https://github.com/onnx/onnx> Accessed: 2026-03-01.
- [55] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2023. Gemel: Model merging for {Memory-Efficient}, {Real-Time} video analytics at the edge. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 973–994. <https://www.usenix.org/conference/nsdi23/presentation/padmanabhan>
- [56] Venkatesh Pallipadi and Alexey Starikovskiy. 2006. The ondemand governor. In *Proceedings of the linux symposium*, Vol. 2. 215–230. <https://www.kernel.org/doc/ols/2006/ols2006v2-pages-223-238.pdf>
- [57] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359. doi:10.1109/TKDE.2009.191
- [58] Michael Pellauer, Jason Clemons, Vignesh Balaji, Neal Crago, Aamer Jaleel, Donghyuk Lee, Mike O'Connor, Angshuman Parashar, Sean Treichler, Po-An Tsai, et al. 2023. Symphony: Orchestrating sparse and dense tensors with hierarchical heterogeneous processing. *ACM Transactions on Computer Systems* 41, 1–4 (2023), 1–30. doi:10.1145/3630007
- [59] Padmanabhan Pillai and Kang G Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*. 89–102. doi:10.1145/502034.502044
- [60] Enrico Reggiani, Cristóbal Ramírez Lazo, Roger Figueras Bagué, Adrián Cristal, Mauro Olivieri, and Osman Sabri Unsal. 2022. Bison-e: A light-weight and high-performance accelerator for narrow integer linear algebra computing on the edge. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 56–69. doi:10.1145/3503222.3507746
- [61] Joseph Rogers, Taha Soliman, and Magnus Jahre. 2024. AIO: An abstraction for performance analysis across diverse accelerator architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 487–500. doi:10.1109/ISCA59077.2024.00043
- [62] Shuyao Shi, Jiahe Cui, Zhehao Jiang, Zhenyu Yan, Guoliang Xing, Jianwei Niu, and Zhenchao Ouyang. 2022. VIPS: Real-time perception fusion for infrastructure-assisted autonomous driving. In *Proceedings of the 28th annual international conference on mobile computing and networking*. 133–146. doi:10.1145/3495243.3560539
- [63] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014). doi:10.48550/arXiv.1409.1556
- [64] Marco Siracusa, Lorenzo Di Tucci, Marco Rabozzi, Samuel Williams, Emanuele Del Sozzo, and Marco D Santambrogio. 2020. A cad-based methodology to optimize hls code via the roofline model. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9. doi:10.1145/3400302.3415730
- [65] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 590–606. doi:10.1145/3694715.3695964
- [66] Chunlin Tian, Xinpeng Qin, Kahou Tam, Li Li, Zijian Wang, Yuanzhe Zhao, Minglei Zhang, and Chengzhong Xu. 2025. {CLONE}: Customizing {LLMs} for Efficient {Latency-Aware} Inference at the Edge. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 563–585. <https://www.usenix.org/conference/atc25/presentation/tian>
- [67] Wen-Jer Tsai, Chun-Chang Lu, Ting-En Hsieh, Tao-Cheng Lu, Kuang-Chao Chen, Tuo-Hung Hou, and Chih-Yuan Lu. 2026. A Page-Buffer and Fail-Bit Counter-Based In-Memory Computing Platform for 3D NAND Flash Memory. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2026). doi:10.1109/JETCAS.2026.3659465
- [68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017). <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [69] Haoxin Wang, Xiaolong Tu, Hongyu Ke, Huirong Chai, Dawei Chen, and Kyungtae Han. 2025. Im-Meter: Unveiling Runtime Inference Latency for On-Device Language Models. In *Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing*. 1–17. doi:10.1145/3769102.3770614
- [70] Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, et al. 2025. Using analytical performance/power model and fine-grained dvfs to enhance ai accelerator energy efficiency. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 1118–1132. doi:10.1145/3669940.3707231
- [71] Jianyu Wei, Ting Cao, Shijie Cao, Shiqi Jiang, Shaowei Fu, Mao Yang, Yanyong Zhang, and Yunxin Liu. 2023. Nn-stretch: Automatic neural network branching for parallel inference on heterogeneous multi-processors. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 70–83. doi:10.1145/3581791.3596870
- [72] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. 2023. Adaptivenet: Post-deployment neural architecture adaptation for diverse edge environments. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–17. doi:10.1145/3570361.3592529
- [73] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems* 29 (2016). <https://proceedings.neurips.cc/paper/2016/hash/41bfd20a38bb1b0bec75acf0845530a7-Abstract.html>
- [74] Lucas Wilkinson, Kazem Cheshmi, and Maryam Mehri Dehnavi. 2023. Register tiling for unstructured sparsity in neural network inference. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1995–2020. doi:10.1145/3591302
- [75] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76. doi:10.1145/1498765.1498785
- [76] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. 2023. Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 374–389. doi:10.1109/HPCA56546.2023.10070999
- [77] Bufang Yang, Lixing He, Neiwen Ling, Zhenyu Yan, Guoliang Xing, Xian Shuai, Xiaozhe Ren, and Xin Jiang. 2023. Edgefm: Leveraging foundation model for open-set learning on the edge. In *Proceedings of the 21st ACM conference on embedded networked sensor systems*. 111–124. doi:10.1145/3625687.3625793
- [78] Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. 2025. Lserve: Efficient long-sequence llm serving with unified sparse attention. *Proceedings of Machine Learning and Systems* 7 (2025). https://proceedings.mlsys.org/paper_files/paper/2025/hash/cc8c6b9d89f7a898a29f58869b238e46-Abstract-Conference.html
- [79] Wei Ying, Yu Zhang, Junzhou Huang, and Qiang Yang. 2018. Transfer learning via learning to transfer. In *International conference on machine learning*. PMLR, 5085–5094. <https://proceedings.mlr.press/v80/wei18a.html>
- [80] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 81–93. doi:10.1145/3458864.

- 3467882
- [81] Zongpu Zhang, Pranab Dash, Y Charlie Hu, Qiang Xu, Jian Li, and Haibing Guan. 2025. Dissecting the Impact of Mobile DVFS Governors on LLM Inference Performance and Energy Efficiency. *arXiv preprint arXiv:2507.02135* (2025). doi:10.48550/arXiv.2507.02135
 - [82] Ziyang Zhang, Huan Li, Yang Zhao, Changyao Lin, and Jie Liu. 2023. Pos: An operator scheduling framework for multi-model inference on edge intelligent computing. In *Proceedings of the 22nd International Conference on Information Processing in Sensor Networks*. 40–52. doi:10.1145/3583120.3586953
 - [83] Ziyang Zhang, Yang Zhao, Ming-Ching Chang, Changyao Lin, and Jie Liu. 2025. E4: Energy-Efficient DNN Inference for Edge Video Analytics Via Early Exiting and DVFS. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 1165–1173. doi:10.1609/aaai.v39i1.32104
 - [84] Ziyang Zhang, Yang Zhao, Huan Li, Changyao Lin, and Jie Liu. 2024. DVFO: Learning-based DVFS for energy-efficient edge-cloud collaborative inference. *IEEE Transactions on Mobile Computing* 23, 10 (2024), 9042–9059. doi:10.1109/TMC.2024.3357218
 - [85] Tiandong Zhao, Shaoqiang Lu, Chen Wu, and Lei He. 2024. ChatOPU: An FPGA-based overlay processor for large language models with unstructured sparsity. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. 1–9. doi:10.1145/3676536.3676761
 - [86] Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, Guanyu Feng, Xin Lv, Xiao Chuanfu, Dahua Lin, and Chao Yang. 2025. Sampleattention: Near-lossless acceleration of long context llm inference with adaptive structured sparse attention. *Proceedings of Machine Learning and Systems* 7 (2025). https://proceedings.mlsys.org/paper_files/paper/2025/hash/2d04d97593c8c33d415337f408ed0e1b-Abstract-Conference.html
 - [87] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76. doi:10.1109/JPROC.2020.3004555