SoC-Integrated DNN Acceleration: A Novel SRAM-Based Cache Architecture for Enhanced Machine Learning Workloads

ABSTRACT

This study introduces a novel approach to deep neural network (DNN) acceleration in System on Chips (SoCs), re-envisioning the role of Last-Level Cache (LLC). We leverage the distributed LLC in Intel's IceLake architecture, integrating tailored vector Multiply-Accumulate (MAC) Units within SRAM sub-arrays. This integration, exploiting the intrinsic properties of the SRAM, significantly outperforms traditional dedicated hardware accelerators in terms of area and performance. Our design, featuring combinational multipliers and optimized data distribution, achieves a notable 16× speedup in convolutional neural network (CNN) inference and approximately 300× in fully connected networks, while reducing energy consumption by more than 30× and 370× for CNN and fully connected layers, respectively. These results are compared to state-of-the-art dedicated accelerators of similar area. Crucially, these improvements are realized without sacrificing accuracy, as evidenced by comprehensive system and software level analyses across various neural networks, including AlexNet, ResNet, BERT, DLRM, and LSTM. This performance makes our approach a persuasive alternative for future SoC designs.

KEYWORDS

In-Memory-Computing, Processing-in-Memory, DNN acceleration

ACM Reference Format:

. Year. SoC-Integrated DNN Acceleration: A Novel SRAM-Based Cache Architecture for Enhanced Machine Learning Workloads. In *Proceedings of DAC (CONF'YY).* ACM, New York, NY, USA, 6 pages. https://doi.org/DOI

1 INTRODUCTION

Deep Neural Networks (DNNs) have become pivotal in various fields, excelling in tasks like image recognition, language processing, and recommendation systems. However, the intensive computational demands of DNNs pose significant challenges to general-purpose architectures, such as x86, particularly in terms of memory bandwidth and power efficiency. This has led to the development of specialized DNN hardware accelerators within System on Chips (SoCs), as seen in Exynos [22], Snapdragon [7], and Apple M1/M2 [8]. However, they are still hindered by memory wall and power wall [2]. In the quest to overcome these challenges, an emerging research area is the use of SRAM in SoCs for compute acceleration, particularly in light of the memory wall constraints. Techniques like bit-serial [23] and bit-parallel [10] computation have been explored for this purpose [13]. However, these approaches often only utilize a single or a few SRAM subarrays, and a comprehensive solution that integrates an entire cache architecture for ML compute acceleration within an SoC remains unexplored [13]. This paper fills this gap by proposing a novel SRAM-based DNN accelerator that utilizes the cache infrastructure of SoCs [18, 19].

The existing SRAM-based acceleration methods face several challenges, including: (1) Limited capacity of individual SRAM subarrays for caching neural network inputs and weights. (2) Significant changes in area and energy footprints when adapting SRAM cells for computation. (3) The requirement for both computation operands to be present within the same subarray. (4) Lack of methods for data migration across cache banks for accumulating partial products. (5) Traditional cache designs typically activate only a limited number of subarrays at a time, necessitating a paradigm shift for acceleration that aligns with system-level power, thermal design power (TDP), and area constraints.

Addressing these challenges, our contributions in this paper include: (1) Analysis of data dependency and reuse patterns in standard DNN workloads, highlighting the necessity for multi-subarray computation. (2) Introduction of efficient peripheral circuits that maintain the integrity of SRAM bit cells while minimizing read-write operations, and enabling a balanced pipeline operation in sync with SRAM timing. (3) Implementation of hardware-software co-design strategies to optimize compute operations without sacrificing computational accuracy. (4) Ensuring that the design adheres to system-level constraints regarding power and capacity. (5) Further, the proposed design allows hardware acceleration with minimal design change to cache - allowing ease of integration

Furthermore, we compare our proposed architecture with Simba accelerator from Nvidia (as a stand in for dedicated IP, with very high energy efficiency and throughput) underscoring the system-level benefits of repurposing SRAM for cache acceleration over dedicating separate IP for DNN acceleration. We observed a ∼**16**× and ∼**300**× **speedup** for CNNs and fully connected neural networks, respectively, for **ISO area cost** comparison with dedicated state-of-the-art accelerator. Further, the energy efficiency for inference also increased more than 30 folds.

2 RELATED WORK

2.1 In-Memory Computing (IMC)

In-Memory Computing (IMC) has emerged as a transformative approach, with developments spanning charge-based and resistance-based memories. In the charge-based realm, SRAM, DRAM, and Flash are the mainstays, whereas RRAM, PCM, and STT-MRAM dominate the resistance-based category. However, only SRAM and DRAM have achieved mass manufacturing success, attributed to their process maturity and endurance [10, 13, 23].

DRAM-based accelerators like SkHynix's Newton and AiM GDDR6 [6, 11], and UpMem's DRAM accelerator [15], showcase the potential of DRAM in IMC. Yet, these designs exhibit higher energy costs and slower performance compared to conventional silicon-based

Table 1: ResNet data volume and volume of related words compared to operands

Layer	Input Words	Weight Words	Input word to Weight Relation	Weight word to Input Relation
Conv2_1x1	200704	4096	64	3136
Conv2_3x3	200704	36864	576	3136
Conv3_1x1	100352	16384	128	784
Conv3_3x3	100352	147456	1152	784
Conv4_1x1	50176	65536	256	196
Conv4_3x3	50176	589824	2304	196
Conv5_1x1	25088	262144	512	49
Conv5_3x3	25088	2359296	4608	49

methods. Additionally, DRAM's scalability is hampered by power constraints and manufacturing intricacies, often resulting in vendor-specific solutions. Importantly, DRAM-based designs do not align with our goal of enhancing SoCs with cache architecture tailored for DNN acceleration.

2.2 SRAM-based Computation

SRAM-based computation, incorporating both digital and analog methods, presents a different paradigm [13]. Analog methods activate multiple SRAM word lines simultaneously, exploiting intrinsic charge properties of SRAM arrays for computation. This output is then converted to digital form via an Analog-to-Digital Converter (ADC). Despite its innovative nature, the analog approach suffers from limited computational precision and high overheads in terms of area, energy, and latency due to the ADC.

Digital methods, on the other hand, offer better scalability and precision, aligning more closely with modern DNN operations. A notable example is the DNN+NeuroSim framework, which models SRAM-based IMC at the subarray level [19]. However, these methods are typically limited to subarray computations and do not fully address the data volume needs of inputs and weights in DNNs. Our work seeks to expand on these digital methodologies, overcoming the limitations of subarray-focused designs.

2.3 Accelerator Designs

Among various accelerator designs, Simba [25] stands out as a robust and efficient model [2]. Simba's architecture, featuring multiplier vector units sharing an input buffer, facilitates multiple multiplications per input read, enhancing computational efficiency. Despite the advancements in accelerator design through techniques like compression and quantization [1], our focus is on benchmarking against Simba's realized design due to its exemplary performance and energy efficiency. We aim to demonstrate the advantages of our SRAM-based DNN accelerator in comparison to Simba, highlighting the improvements in energy efficiency and computational effectiveness.

3 MOTIVATION AND PROBLEM DEFINITION

3.1 Deep Neural Network Workload Analysis

Deep Neural Networks (DNNs), particularly convolutional neural networks (CNNs), consist of multiple layers with each requiring the interaction of numerous input and weight terms. The optimal computational model would co-locate operands involved in the same computation. Table 1 illustrates the significant disparity in input and weight word volumes for various ResNet layers, underscoring the challenges in data co-location. Given typical SRAM array dimensions, data for each layer spans numerous subarrays, raising concerns about data redundancy due to repeated computations.

3.2 Intra-subarray Computational Limitations

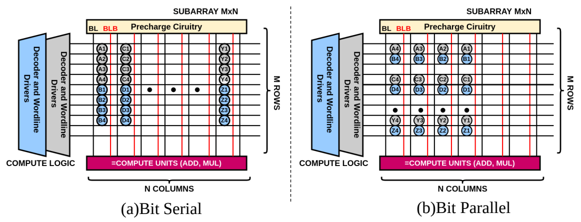

Figure 1: (a) Bit Serial and (b) Bit parallel capable IMC subarray from [18]

Fig. 1 demonstrates a subarray capable of bit-serial and bit-parallel computations. The bit-serial approach requires multiple cycles for an 8-bit multiplication, increasing energy consumption and latency. Similarly, bit-parallel computation, despite being single-cycle, also poses challenges in terms of energy efficiency.

3.3 Access Energy Implications

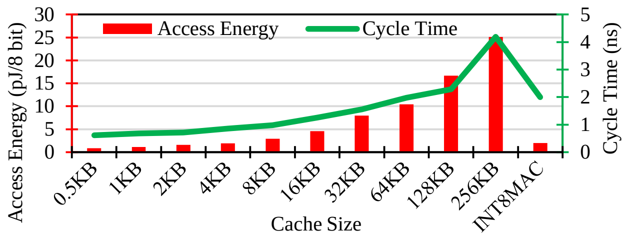

Figure 2: SRAM cache access energy and cycle time for different capacities

Fig. 2 compares the access energy of SRAM caches of various sizes against their cycle times. Notably, access energy increases significantly with capacity, often exceeding the computational energy of standard multipliers. This analysis points to the necessity of minimizing access operations, favoring single-cycle combinational computations over multi-cycle processes.

3.4 Problem Definition

Our analysis underscores the limitations of existing intra-subarray SRAM-based computation methods. Moreover, in many SoCs, the Last Level Cache (LLC) operates independently of the compute pipeline of dedicated DNN hardware acceleration IPs. This observation presents a unique opportunity: if we can modify these LLCs to perform DNN acceleration — potentially with an area overhead similar to that of dedicated hardware accelerator IPs — we could achieve higher throughput without compromising energy efficiency. This approach would leverage the existing cache infrastructure, repurposing it for enhanced computational capabilities, thus providing an innovative solution to the challenges posed by current DNN processing methods.

Table 2: Quantized multipliers that can be implemented at the MAC unit of DNN accelerators, as shown in [3]

Multiplier	Abbrv.	Type	Bit Width	Bit Arrangment	Latency (ps)	Energy (pJ)	Area (μm^2)	LeNet Acc.	ResNet Acc.
IEEE 754 Multiplier(32,8)	FP32	Floating Point	32	(N,es)	1299	22.5	13830.84	98.49%	82.21%
IEEE 754 Multiplier(16,5)	FP16	Floating Point	16	(N,es)	979	4.55	3095.28	98.49%	82.02%
Bfloat16 Multiplier(16,8)	BF16	Floating Point	16	(N,es)	832	2.75	2148.12	98.52%	79.19%
Array Multiplier	AM16	Integer	16	-	1780	8.81	6015.24	98.55%	53.18%
Approximate Fixed Posit Multiplier(10,4)	AFPOS10	Approximate	8	(N,es)	602	0.607	959.04	98.62%	80.73%
Approximate Fixed Posit Multiplier(8,2)	AFPOS8	Approximate	6	(N,es)	563	0.463	776.88	98.62%	80.73%
Approximate Fixed Posit Multiplier(6,2)	AFPOS6	Approximate	4	(N,es)	321	0.0794	285.12	98.14%	60.11%

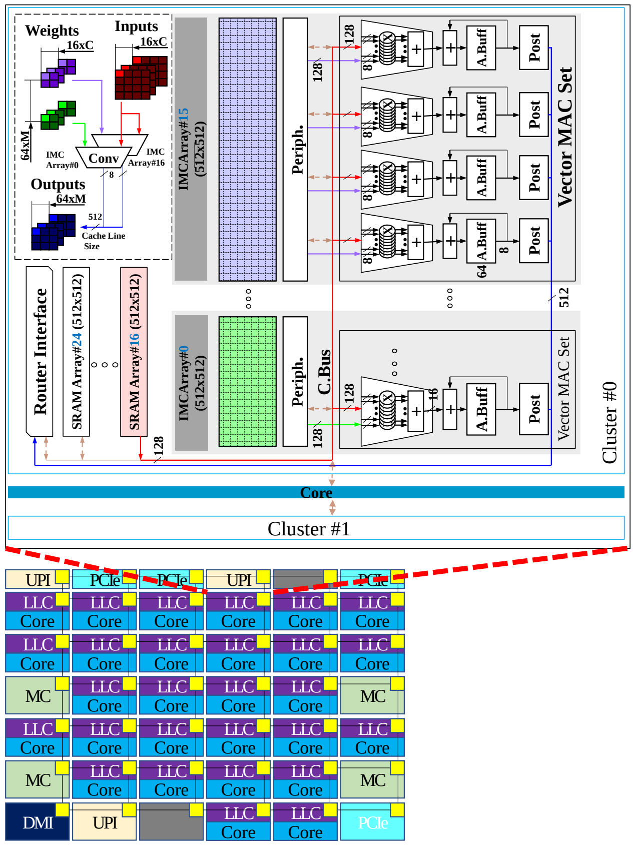

Figure 3: Top:Proposed inter-sub array design, consisting of two clusters at each LLC of a core. Bottom:Distributed LLC of Icelake SP [16]

4 PROPOSED ARCHITECTURE

Our proposed architecture, integrating compute capabilities into the Last Level Cache (LLC), is depicted in Fig. 3. This design innovatively upgrades each of the 28 LLCs, equipping them with specialized computational functionality. A subset of the SRAM sub-arrays within each LLC is enhanced with specialized Vector MAC units, as detailed in Section 4.2. Furthermore, we have restructured the connectivity within these LLCs to facilitate an efficient data flow during DNN inference, a process thoroughly outlined in Section 4.3. This strategic modification of the LLC architecture represents a significant step forward in leveraging existing cache structures for advanced DNN processing.

4.1 Existing Cache Structure

The Intel Icelake SP architecture, as shown in Fig. 3, is characterized by its distributed cache structure [16]. Each core is equipped with a Last Level Cache (LLC) of 1.5MB, summing up to a total capacity of 42MB for the architecture under consideration. This distributed caching system presents a unique opportunity for DNN acceleration, as it allows for efficient data handling and potential parallelism.

The structure inherently involves a trade-off: while cache access within the 1.5MB LLC is straightforward, any requirement beyond this capacity necessitates communication via the Network-on-chip. This aspect introduces both opportunities and challenges in the context of DNN workload management.

Considering the size of sub-arrays (32KB, or 512x512), the architecture can support 48 such sub-arrays working in concert. This capability is particularly advantageous for accommodating the data volumes required by most DNN layers. In cases where layers exceed the capacity of a single LLC block, the architecture's ability to partition data across multiple blocks can be harnessed. Such partitioning is beneficial for layers with high weight reuse at initial stages and for managing output channels in final layers where weight reuse is comparatively lower.

Our proposed arch allows use of unaltered foundry-optimized SRAM cells which leads to compact, energy-efficient design.

4.2 Approximate Posit Number System and Vector Multiplier Design

In advancing DNN accelerator designs, our study has focused on the trade-offs between accuracy and resource utilization among various number systems, as illustrated in Table 2 from [3]. This analysis, particularly targeting ResNet models, revealed that certain number systems could achieve accuracies comparable to the baseline FP32 accuracy without necessitating network retraining.

In our work, we took the initiative to modify the PyTorch framework to facilitate neural network training with various number systems and workloads. This exploration was a key part of our contribution, revealing that the quantization limits can indeed be extended beyond conventional boundaries. Notably, we found that training with the AFPOS6 system achieves accuracies remarkably close to the FP32 standard, with deviations within a narrow margin of ±1%. This finding, a result of our targeted research efforts, underscores the potential of AFPOS [3, 4] in efficient neural network quantization.

Futher, A pivotal finding from this analysis is the efficacy of the Approximate Posit Number system, which attains the accuracy of the IEEE 754 standard (32-bit floating-point format) using an 8-bit data word, with and without retraining. This efficiency in bit-width significantly reduces the area and energy requirements for multiplier circuits. Consequently, for our proposed architecture, we selected the AFPOS10 (Approximate Fixed Posit Multiplier with 10-bit encoding) using 8-bit data. This choice balances accuracy and efficiency, catering to neural networks trained on specific number systems and those trained on FP32.

The implementation of the AFPOS10 is showcased in our proposed vector multiplier design, depicted in Fig. 4. This design integrates 16 AFPOS10 multipliers with adders in a tree-like structure, enabling efficient Multiply-Accumulate (MAC) operations with 16 + 16 + 1 operands. Through extensive synthesis under various constraints, we observed that minor adjustments in cycle time can notably reduce the multiplier's area. Hence, we adopted the AFPOS10 vector MAC operating at its highest frequency to align with the SRAM sub-array's read-write delay, ensuring a balanced pipeline and optimized computation efficiency, as illustrated in Fig. 2.

Table 3: Synthesis Result: Area, Energy, and latency values of Vector MAC (16 multipliers and adder tree)

Unit	Constraint	Area (um^2)	Timing (ps)	Power (with stimulus) (W)				
				Switching	Internal	Dynamic	Leakage	Total
AFPOS6	max Freq	2.38E+03	5.75E+02	2.11E-05	1.20E-04	1.41E-04	2.45E-07	1.41E-04
AFPOS8	max Freq	8.53E+03	9.56E+02	1.09E-03	1.45E-03	2.54E-03	8.81E-07	2.54E-03
AFPOS10	max Freq	1.55E+04	1.96E+03	1.22E-03	1.43E-03	2.65E-03	1.60E-06	2.65E-03

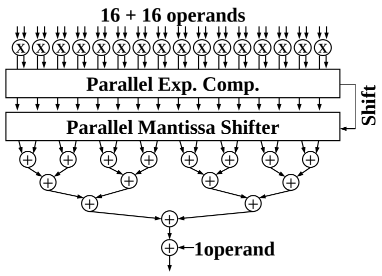

Figure 4: Approximate POSIT10 Vector MAC with the same cycle time as SRAM subarray cycle time, Refer Table 3

Moreover, Fig. 8 in Section 6.2.2 presents a comparative analysis of system-level amortized energy consumption for each MAC operation. This comparison underscores the energy efficiency of our AFPOS10-based architecture, particularly in contrast with designs like the FCL Simba. The Approx. Fixed Posit Regime used in our design emerges as notably energy-efficient compared to other multiplier types, highlighting its suitability for energy-conscious DNN acceleration in SoC cache architectures.

4.3 Inter-subarray Computation

Our proposed architecture employs a strategy that leverages multiple SRAM subarrays, designating specific subarrays for weights and inputs to facilitate efficient computation and data communication. As highlighted in Table 1, a single input word in a DNN layer, even one with a 1x1 filter size, is involved in multiple multiplications - a minimum of 64, corresponding to the multiplications across different output channels. This scenario necessitates distributing filters associated with various output channels across multiple subarrays, while a designated subarray provides a common input to all. Further, the work itself evaluate for NN including AlexNet, LeNet, MobileNet, ResNet, VGG, BERT, DLRM and LSTM.

To enable effective data sharing and computation, we propose the instantiation of 8-bit combinational multipliers at the periphery of each SRAM subarray. Given subarrays of size 32KB, this setup allows for the integration of 64 such multipliers per subarray. When input words are read from an input SRAM subarray, they participate in multiple computations, generating a series of product vectors. This process could potentially lead to increased demands in terms of communication and data writing. To address this, we suggest attaching adder trees to the multipliers, following two primary criteria: **(1)** The input data for the multipliers must be additive to avoid the need for padding, which can be detrimental to efficiency. In most layers, barring the initial ones, the number of input channels typically meets or exceeds 64, aligning well with our design. **(2)** There needs to be a synchronization between the row read time of the subarrays and the compute latency of the multipliers and attached adder trees. At a 65nm process node, a 512x512 subarray can access data in approximately 2ns (as shown in Fig. 2). With an AFPOS10 multiplier, the latency is about 600ps. When 16 such multipliers are combined with an adder tree, the total compute time aligns with the 2ns read time, enabling efficient pipeline operation.

For each 32KB subarray, four output words are generated per cycle. Optimal data writing requires collecting a complete row's worth of data (64 words), necessitating the amalgamation of outputs from 16 subarrays before writing back to a subarray. This alignment is crucial since the outputs often become inputs in subsequent layers.

To ensure this alignment, and that each word written back represents different channels, we propose a balanced pipeline that produces a row of partial products per cycle. Additionally, an accumulation buffer is employed to aggregate these partial products, sized appropriately to minimize the frequency of weight subarray reads. Based on our analysis, this buffer size is set to accommodate the worst-case scenario where each weight word is involved in at least 49 multiplications, for most CNNs, as seen in Table 1.

Finally, the architecture faces the challenge of effectively segmenting subarrays for storing weights and inputs/outputs. Our analysis suggests that the most voluminous input sizes require seven subarrays, with an additional seven needed for output storage post computation. Accordingly, allocating 16 subarrays for input/output and 32 for weights appears to be a pragmatic approach, especially at the LLC level where this division can be structured into two clusters. Each subarray designated for weights is equipped with an AFPOS10 multiplier and an adder tree, forming the core of our proposed inter-subarray computation design.

5 METHODOLOGY

Our experimental design is methodically structured, focusing on three critical components:

1. Hardware Modeling: We utilize a suite of tools for detailed hardware modeling. **Timeloop** [17] is employed for modeling the DNN accelerator, capturing nuances from various memory hierarchies like DRAM, SRAM, and registers, as well as the spatial organization of MAC units. **Accelergy** [24] estimates the area and energy of DNN accelerator sub-components, providing energy metrics on a per-action basis. **Cacti** and **Aladdin** [14, 20] assess the area and power requirements of SRAMs and MAC units. We select the 65 nm technology node for our evaluations, sourcing area, delay, and energy metrics for computational blocks and multipliers from **Cadence Genus**.

2. Evaluation Across Neural Network Models: The study spans a range of neural network architectures, from older models like AlexNet [9] and VGG [21], to newer architectures such as ResNet [5] and MobileNet [12]. This broad spectrum enables a comprehensive evaluation of the proposed architecture's versatility and effectiveness.

3. Optimal Workload Mapping: We prioritize optimal mapping of workloads over a fixed scheme, allowing for a more objective and flexible comparison between different accelerator designs.

5.1 Hardware Representation and Optimal Mapping

Using **Timeloop Mapper**, we identify all feasible mappings of workloads to our architecture, evaluating latency and energy for each via Timeloop and Accelergy models. The optimization targets minimal delay first, followed by minimal energy, focusing on the most efficient mapping.

For context, the foundational hardware, Simba, includes DDR4 DRAM memory, a reference design architecture, 25GBps DRAM bandwidth, and 46 pJ/bit access energy. Its global buffer is set at 64 KB, with PE weights, inputs, and outputs buffered at 65 KB, 16 KB, and 5 KB, respectively. Each of its 16 PEs houses 64 MAC units, operating at 1ns latency with Float16 word size.

Our approach scales Simba's PE count to match the area of our proposed architecture, culminating in an 8-instance Simba configuration (4×2) for maximum output parallelism.

5.2 Workload Selection

We focus on inference tasks from CNN architectures, starting with AlexNet, LeNet and VGG for baseline guidance, particularly due to VGG16's use of 3×3 filters. For deeper insights, we extend our evaluation to ResNet, MobileNet, BERT, DLRM and LSTM, as per the MLPerf inference suite [12]. In ResNet's case, we concentrate on distinct layers, projecting their results across ResNet derivatives.

5.3 Accuracy Evaluation

Baseline accuracies are determined using LeNet, VGG, and ResNets within the Pytorch framework, employing 32-bit floating-point data words. Datasets like MNIST (for LeNet), CIFAR-10 (for ResNet), and ImageNet (for VGG) support model training and accuracy measurements. We have modified Pytorch to redirect internal DNN MAC operations to corresponding software-based multiplier models, with each multiplier meticulously modeled at the event level for bit-level precision.

6 EXPERIMENTAL RESULTS

6.1 Constraints

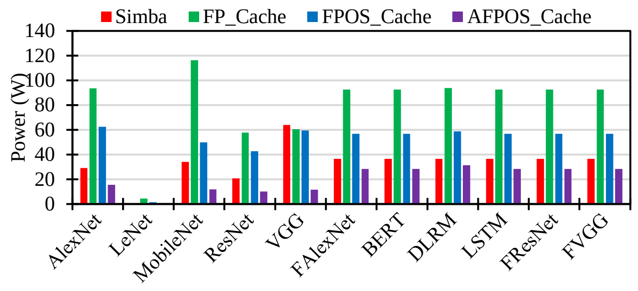

Figure 5: Power consumed during inference of DNNs

6.1.1 TDP (Thermal Design Power). Thermal Design Power (TDP) is a critical system-level constraint. The Skylake SP architecture, for instance, has a socket TDP of 270W but is rated at 165W. Fig. 5 shows the TDP for our proposed system with various multipliers. Our analysis considers only the cache compute energy, assuming the core remains largely inactive. The results indicate that high-density floating-point multipliers could create thermal hot-spots, while their quantization leads to accuracy degradation (see Table 2). Conversely, using AFPOS10 multipliers keeps the TDP well below the chip's threshold. Notably, Simba shows lower power consumption than our cache with FP16 multipliers due to fewer multipliers being used.

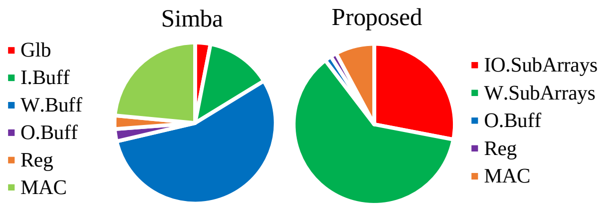

Figure 6: Area distribution in Simba and Proposed design. The proposed design introduces an 11% area overhead to the existing cache.

6.1.2 Area Constraints. Fig. 6 compares the area distribution between our proposed architecture and an ISO area Simba instance, our baseline. In calculating the ISO area for Simba, we considered the cache's overhead. Adding AFPOS10 vector multipliers around the SRAM arrays increases the cache area by only 11%. In contrast, Simba allocates only a quarter of this overhead area for compute units, with the rest consumed by dedicated buffers.

6.2 Performance Metrics

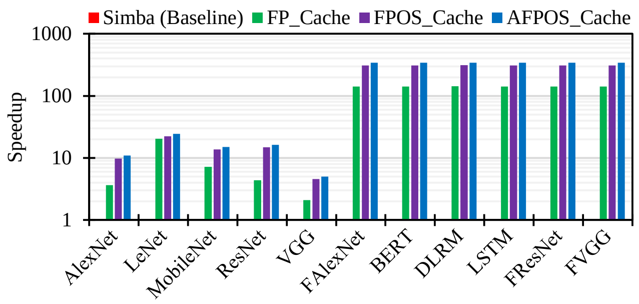

Figure 7: Speedup (Simba used as reference 1) of cache with different multipliers

6.2.1 Speedup. Fig. 7 illustrates the system-level speedup achieved by our architecture in processing various types of DNNs, compared to the ISO area Simba configuration. For convolutional neural networks (CNNs), our design achieves a significant speedup of 16×, a result of increasing the number of multipliers sevenfold within the same SoC area. Despite these multipliers being relatively slower, they are more efficient in terms of area and power usage. Additionally, distributing computations efficiently across 28 LLC blocks and the ability to keep layer data stationary within the caches significantly enhances utilization, as memory bandwidth is no longer a bottleneck. Thus, in the context of fully connected networks, our architecture shows an even more impressive speedup of approximately 300×.

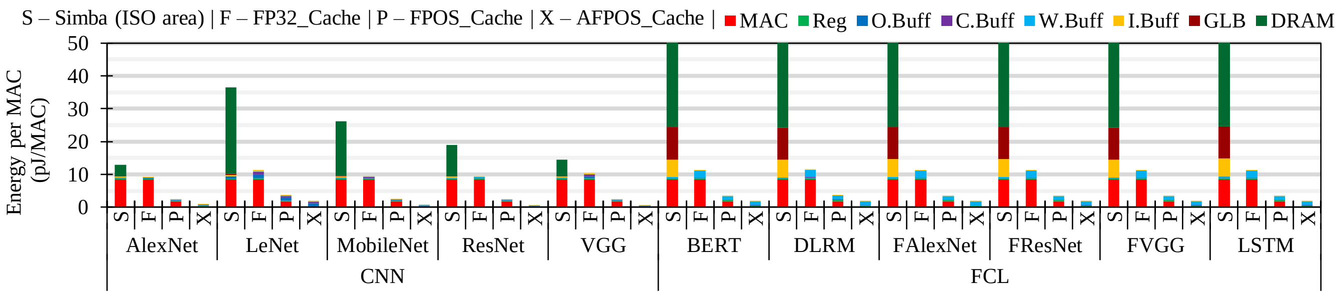

Figure 8: Comparison of amortized energy consumption per MAC operation. The graph shows energy usage for Fully Connected Layer (FCL) Simba with DRAM (truncated from ∼ 700pJ) and our architectures with FP16 (Float_Cache) and Fixed Posit Regime multipliers (FPOS_Cache), underscoring the energy efficiency of the proposed design.

6.2.2 Energy Consumption. Fig. 8 reveals our architecture's substantial energy efficiency in DNN processing. Key to this achievement is the integration of advanced AFPOS10 multipliers. Unlike the narrower multipliers in the Simba design, our wider multipliers necessitate fewer reads and writes, leading to significant energy savings at the local computation level.

The large cache in our system further enhances energy efficiency by reducing dependencies on main memory, which is typically more energy-consuming. This architectural feature allows us to store entire models within the cache, drastically cutting down memory access energy. Consequently, we observe an energy efficiency improvement of 30× for CNN layers and an impressive 370× for fully connected layers, compared to the Simba setup.

In addition to these hardware innovations, the incorporation of model quantization plays a vital role. By optimizing data representation, it significantly lowers the energy required for buffer operations. Moreover, the shift from frequent DRAM accesses to predominantly using the upgraded cache for storing weights and inputs contributes to the overall reduction in energy demand. These combined factors — efficient multipliers, large cache capacity, and model quantization — collectively establish our architecture as a highly energy-efficient solution for DNN acceleration in SoCs.

7 CONCLUSION

In this study, we revolutionize SoC architectures for deep neural network (DNN) processing by exploiting SRAM subarrays in distributed caches. Our novel approach integrates an approximate posit number system and custom multipliers into the cache, complemented by an aligned adder tree for streamlined vector operations. This design not only facilitates efficient data sharing across SRAM subarrays via a shared bus system but also maintains balanced read/write operations essential for system efficiency and stability.

Contrary to potential concerns about performance trade-offs, our architecture achieves a significant 16× speedup in CNN processing within the same area footprint, coupled with over 30× improvement in energy efficiency, without compromising accuracy.

Our findings offer a compelling alternative to traditional dedicated IP solutions for DNN acceleration, showcasing the untapped potential of SRAM subarrays in SoC caches. This work sets a new precedent for efficient, powerful DNN processing in future SoC designs.

REFERENCES

[1] Giorgos Armeniakos et al. 2022. Hardware approximate techniques for deep neural network accelerators: A survey. *Comput. Surveys* 55, 4 (2022), 1–36.
[2] Tom Glint et al. 2023. Analysis of Conventional, Near-Memory, and In-Memory DNN Accelerators. In *2023 ISPASS*. IEEE, 349–351.
[3] Tom Glint et al. 2023. Hardware-Software Codesign of DNN Accelerators using Approximate Posit Multipliers. In *ASP-DAC*. https://doi.org/10.1145/3566097.3567866
[4] Varun Gohil et al. 2021. Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications. *IEEE TCAS II: Express Briefs* 68, 10 (2021), 3341–3345. https://doi.org/10.1109/TCSII.2021.3072217
[5] Kaiming He et al. 2016. Deep residual learning for image recognition. In *IEEE CVPR*. 770–778.
[6] Mingxuan He et al. 2020. Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning. In *2020 MICRO*. IEEE, 372–385.
[7] Andrey Ignatov et al. 2019. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 3617–3635.
[8] David Kasperek et al. 2023. Comparison of the Usability of Apple M2 and M1 Processors for Various Machine Learning Tasks. *Sensors* 23, 12 (2023), 5424.
[9] Alex Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).
[10] Kyeongho Lee et al. 2020. Bit Parallel 6T SRAM In-memory Computing with Reconfigurable Bit-Precision. In *2020 DAC*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218567
[11] Seongju Lee et al. 2022. A 1ynm 1.25 v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications. In *2022 ISSCC*, Vol. 65. IEEE, 1–3.
[12] Peter Mattson et al. 2019. MLPerf training benchmark. *arXiv preprint arXiv:1910.01500* (2019).
[13] Sparsh Mittal et al. 2021. A survey of SRAM-based in-memory computing techniques and applications. *Journal of Systems Architecture* 119 (2021), 102276.
[14] Naveen Muralimanohar et al. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.
[15] Geraldo F Oliveira et al. 2022. Accelerating Neural Network Inference with Processing-in-DRAM: From the Edge to the Cloud. *IEEE Micro* 42, 6 (2022), 25–38.
[16] Irma Esmer Papazian. 2020. New 3rd Gen Intel® Xeon® Scalable Processor (Codename: Ice Lake-SP).. In *Hot Chips Symposium*. 1–22.
[17] Angshuman Parashar et al. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *ISPASS*. IEEE, 304–315.
[18] Alok Parmar et al. 2022. An automated approach to compare bit serial and bit parallel in-memory computing for dnns. In *2022 ISCAS*. IEEE, 2948–2952.
[19] Xiaochen Peng et al. 2019. DNN+ NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEDM*. IEEE, 32–5.
[20] Yakun Sophia Shao et al. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ISCA*. IEEE, 97–108.
[21] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[22] Jinook Song et al. 2019. 7.1 An 11.5 TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC. In *2019 ISSCC*. IEEE, 130–132.
[23] Neelam Surana et al. 2020. Robust and High-Performance 12-T Interlocked SRAM for In-Memory Computing. In *2020 DATE*. 1323–1326. https://doi.org/10.23919/DATE48585.2020.9116361
[24] Yannan Nellie Wu et al. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *ICCAD*. IEEE, 1–8.
[25] Brian Zimmer et al et al. 2020. A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE JSSC* 55, 4 (2020), 920–932.