INSPIRE: Accelerating Deep Neural Networks via Hardware-friendly Index-Pair Encoding

Fangxin Liu[1,2,†], Ning Yang[1,2,†], Zhiyan Song[2], Zongwu Wang[1,2], Haomin Li[1,2], Shiyuan Huang[1,2], Zhuoran Song[1], Songwen Pei[3] and Li Jiang[1,2]
1. Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
2. Shanghai Qi Zhi Institute 3. University of Shanghai for Science and Technology

ABSTRACT

Deep Neural Network (DNN) inference consumes significant computing resources and development efforts due to the growing model size. Quantization is a promising technique to reduce the computation and memory cost of DNNs. Most existing quantization methods rely on fixed-point integers or floating-point types, which require more bits to maintain model accuracy. In contrast, variable-length quantization, which combines high precision for values with significant magnitudes (i.e., outliers) and low precision for normal values, offers algorithmic advantages but introduces significant hardware overhead due to variable-length encoding and decoding. Also, existing quantization methods are less effective for both (dynamic) activations and (static) weights due to the presence of outliers.

In this work, we propose INSPIRE, an algorithm/architecture co-designed solution that employs an Index-Pair (INP) quantization and handles outliers globally with low hardware overheads and high performance gains. The key insight of INSPIRE lies in identifying typical features associated with important values, encoding them as indexes, and precomputing corresponding results for efficient storage in lookup table. During inference, the results of inputs with paired index can be directly retrieved from the table, which eliminates the need for any computational overhead. Furthermore, we design a unified processing element architecture for INSPIRE and highlight its seamless integration with existing DNN accelerators. As a result, INSPIRE-based accelerator surpasses the state-of-the-art quantization accelerators with a remarkable 9.31× speedup and 81.3% energy reduction, respectively, while maintaining superior model accuracy.

ACM Reference Format:
Fangxin Liu[1,2,†], Ning Yang[1,2,†], Zhiyan Song[2], Zongwu Wang[1,2], Haomin Li[1,2], Shiyuan Huang[1,2], Zhuoran Song[1], Songwen Pei[3] and Li Jiang[1,2]. 2024. INSPIRE: Accelerating Deep Neural Networks via Hardware-friendly Index-Pair Encoding. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655896

† These authors contributed equally.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655896

1 INTRODUCTION

Deep Neural Networks (DNNs) have achieved remarkable success in fields like computer vision and natural language processing [5, 23]. However, such performance boost has been achieved at the cost of extremely energy-intensive DNN models due to the increasingly larger model size [9]. For instance, state-of-the-art DNNs like GPT-3 may require up to GBs (Giga Bytes) for model size and 10^2 GFLOPs (Giga Floating Point Operations) for inference computation [2].

Therefore, deployment of such DNNs remains a challenge, requiring two crucial supports. The first one is specialized hardware accelerations for DNN inference, such as Eyeriss [3], BitFusion [19] and TPU [11]. The second is model compression techniques, which explore the opportunity of algorithm and hardware co-design to achieve better trade-offs between accuracy and hardware overhead. Model quantization is one of the most hardware-efficient compression techniques to reduce inference costs for large DNN models [6]. It represents model parameters with fewer bits to simplify the implementations and accelerate the inference execution speed in a hardware-friendly manner. Also, it is supported in GPU tensor core [15] and mainstream accelerator architectures like systolic array [3, 11].

However, existing quantization schemes are less effective when applied to DNNs that consider both activations and weights, especially for larger models. This is because the values in DNN tensors have a non-uniform distribution and non-uniform importance [13, 24]. Model performance is particularly sensitive to only a very small fraction of outliers, particularly in dynamic activations, which are far more significant than normal values [8, 12, 14]. Simply clipping outliers could result in a significant reduction in model accuracy, while including these outliers directly in the quantization range could lead to a cumulative increase in the rounding error. Consequently, a common practice is to use larger bitwidths (e.g., 8-bit or more) to quantize DNN models, such as activations.

In an effort to construct faster DNN models, many researchers have explored outlier-aware algorithm-and-accelerator co-designs that focus on quantization [8, 21, 24]. These designs incorporate hardware designs that can efficiently represent normal and outlier values with different precision. However, they are not compatible with existing DNN accelerators due to their variable-length encoding and unaligned memory access, requiring costly and complex hardware designs to support.

Previous designs often prioritized the reduction of bitwidths to simplify quantization, but this led to difficulties in accurately representing activation values [12]. Outliers would stretch the quantization range, causing most values to be compressed into just a few effective bits [24]. However, our paper presents a novel approach utilizing centroids to represent parameter distributions, rather than

quantization intervals [9]. Through this method, we create a global quantization process that can adapt to the varying importance of different values.

To optimize the performance and efficiency of deep neural networks, we propose a pioneering approach that leverages centroids and their associated indexes for aligned memory access and low-bit computation. By using indexes to record centroids in the codebook, we can maintain efficient quantization while only requiring fixed-size, low-precision int data types in the network. Furthermore, we present a novel index-pair processing element (IP-PE) design that can accommodate weights and activations based on indexes, not values, which is well-suited for lookup tables (LUTs). By simplifying PEs to support index-pair matching and utilizing a unified formatting scheme to compute on fixed-size, low-precision indexes, our approach delivers remarkable reductions in computation cost for inference. We make the following contributions in this paper:

- We propose INSPIRE as a novel approach for efficient encoding of activations and weights, offering a distinctive inference paradigm based on index representation data type.
- We propose the efficient architectural implementation and integration of INSPIRE quantization, specifically designed to handle cases where operators involve indexes rather than values.
- We evaluate the effectiveness of INSPIRE by comparing it with existing quantization accelerators. The results demonstrate the superiority of INSPIRE, achieving up to a 81.3% reduction in energy consumption and a 9.31× speedup while maintaining the similar accuracy as full-precision models.

2 BACKGROUND AND MOTIVATION

2.1 Outlier Matters in Quantization

We visually illustrate the significance of DNN outliers in Figure 1, using ResNet-50 as a representative CNN model and BERT for the attention-based model. The outliers in DNNs are ≈ 100× larger than most activation values [12, 24]. During quantization, these large outliers dominate the maximum magnitude measurement, resulting in low effective quantization levels for normal values and, consequently, substantial quantization errors. A notable observation is the significant difference in outlier magnitudes between attention-based and CNN models. Attention-based models exhibit outliers that are an order of magnitude larger than those in CNNs. While retraining algorithms can restore accuracy in CNN models even when outliers are clipped under ultra-low-bit precision (e.g., 4-bit), this proves to be more challenging for attention-based models due to their significantly larger outliers [8, 18].

2.2 Model Quantization

The research focus on outliers has spurred the development of outlier-aware quantization techniques [8, 24]. These methods typically utilize low-precision 4-bit integers for small, frequently occurring values and high-precision 32/16-bit floats or integers for infrequent, large values. For example, GOBO [25] and OLAccel [21] utilize a coordinate list to pinpoint outlier locations, allowing representation with high precision while compressing normal values with fewer bits. In BiScaled-DNN [10], all values share the same bit-width but differ in scale factors for normal values and outliers.

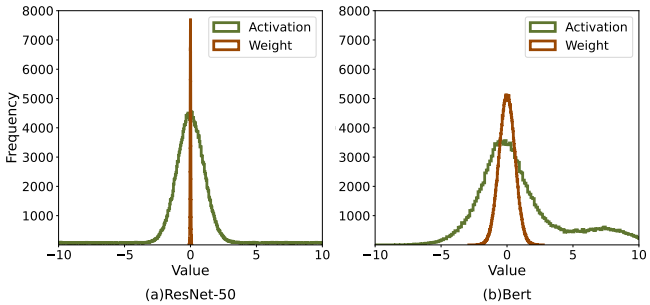

Figure 1: Distribution of activation/weight values in a single layer for the (a) ResNet-50 and (b) BERT model.

DRQ [20] employs a bitmap and uses mixed precision (4-bit and 8-bit) for data storage. ANT [8] uses mixed data types to better fit distributions, yet introducing complex logic for decoding the data type. However, these quantization techniques exploit variable-length data encoding, which leads to the non-alignment in the memory sub-system or complex hardware logic.

2.3 Motivation

Weight distribution is generally uniform and flat, making it easy to quantize. Previous studies have demonstrated that quantizing LLM weights with 8-bit or even 4-bit precision does not compromise accuracy, aligning with our observation [8, 12, 14, 24]. However, the presence of outliers in DNNs poses challenges for activation quantization. Consequently, the conventional approaches involve adopting larger bitwidths, such as 8-bit or 16-bit, for activation quantization.

To this end, we explore an alternative approach inspired by the observation that clustering DNN values and learning centroids adaptively could be advantageous. However, this method introduces the need for additional indexes to record centroid locations. In response, our paper introduces a new inference paradigm based on these indexes, departing from the conventional quantization architecture that relies on bit-width. This paradigm replaces traditional computation operators with table lookup. With the preset quantization patterns, we precompute centroid results and store them offline in tables. During inference, matched centroids with inputs can be efficiently retrieved from the table, offering a more adaptive and efficient approach.

3 INSPIRE ALGORITHM

In this section, we introduce INSPIRE, our adaptive encoding approach that efficiently leverages intra-layer adaptive opportunities to globally fit the distributions of activations and weights. We propose a novel encoding scheme leveraging centroids to accommodate varying value significance within a layer. Additionally, we propose a dynamic framework that adapts to each layer's distribution by selecting appropriate centroids. Consequently, INSPIRE ensures aligned memory accesses and efficient low-bit computation, delivering significant benefits with minimal hardware overhead.

3.1 Algorithm Overview

Our approach to compressing model parameters deviates from conventional methods like sparsity or quantization. Rather than aiming for compression to specific bitwidths or maintaining a high sparsity

rate, we employ clustering to identify centroids that adapt to the importance of different parameter values. This strategy enables efficient processing of both outlier and normal values. Our objective is to identify the key factors (i.e., centroids) in the parameter distributions of DNN to minimize model loss.

For a given activation or weight matrix, the k-means clustering method, based on a similarity measure, proves to be the most effective for compression. We select initial C clustering centers and iteratively update them by classifying data into different clusters until minimal error variation is achieved. However, the direct application of the k-means clustering method faces challenges. The clustering effect depends on initialized centers and is sensitive to outliers. Moreover, while a smaller C can enhance compression, k-means, being an unsupervised learning method, cannot guarantee compressed accuracy.

To overcome these challenges, we introduce optimizations. First, we uniformly choose the initial C values over a range. Since uniform quantization is already effective, starting clustering from uniformly quantized values yields better results. Furthermore, we employ the Gap statistic method [22] to determine the initial C value, ensuring a balanced trade-off between compression quality and efficiency:

$$Gap(C) = E(\log D_C) - \log D_C \tag{1}$$

where $E(\log D_C)$ is obtained through the following process: we uniformly generate a number of random data points equal to the original dataset size within a defined range. k-means clustering is then performed on this random sample to obtain D_C. This process is repeated multiple times to obtain several $\log D_C$ values, and their average approximates $E(\log D_C)$. The initial number of centroids, determined by maximizing Gap(C), is obtained through this process.

Subsequently, based on the initial conditions, we employ k-means clustering to process weights from the pre-trained model and activations from random training samples to determine centroids. We then conduct inference to test the compressed model and obtain accuracy. Iteratively adjusting C based on observed accuracy losses, we identify the optimal activation C_A and weights C_W clusters.

3.2 Encoding and Indexing

After determining the centroids offline, we proceed with the compression and computation using these centroids. The centroid computation is conducted offline, utilizing pre-trained weight parameters and predetermined activations obtained from the training set. For a value x, we determine its compression result by finding the centroid with the smallest L1 norm absolute difference:

$$Compressed(x) = \min_i |C_i - x| \tag{2}$$

For static weight parameters, we precompute the calculations offline to optimize efficiency. On the other hand, for dynamic input activations, we perform online calculations to determine their compressed values, leveraging the pre-determined centroids and the L1-norm metric. To further enhance performance, we order the centroids from smallest to largest and initiate a binary search from the middle item. This binary search strategy effectively reduces the computation complexity from C to $\log C$. Moreover, the activation encoding process can be implemented in a pipeline, ensuring no increase in time cost.

After determining the centroids, we observe an opportunity to simplify the computation of the activation-weight matrix. Given that centroid values are predetermined and limited, we can precompute the multiplication results of compressed activations and weights, storing them in a lookup table. This eliminates the need for multiplication during actual operations, as the results can be directly read. Taking advantage of this optimization, we convert the encoding results from centroid values to their corresponding indexes. Through the construction of index-pairs for activation and weight indexes, a lookup table is formed, facilitating direct index usage for data transmission and result retrieval. INSPIRE, our compression scheme, follows a two-phase process outlined in Algorithm 1: 1) In phase 1, we utilize the pre-trained weight parameters and a subset of the training data to calculate and determine the number of clusters C_a and C_w along with the corresponding clustering centroids. We encode indexes in the order of their numerical values $[0, 1, \ldots, C-1]$. 2) In phase 2, we convert the parameters directly to the corresponding indexes through binary search. We perform searches directly through the indexes to obtain the calculation results and carry out accumulation and output.

4 INSPIRE ARCHITECTURE

In this section, we describe the incorporation of INSPIRE into the systolic array architecture. The unique encoding introduced by INSPIRE presents challenges in designing the processing element (PE), as it now works with indexes rather than values for computations. To this end, we introduce the IP-PE architecture, which replaces the multiply-accumulate (MAC) operation among values featuring data types with a table lookup based on indexes. Additionally, we present the hardware encoder tailored for the aforementioned index-pair encoding.

4.1 Architecture Overview

Figure 2 provides an overview of the INSPIRE architecture, consisting of multiple PE pages. Each PE page holds two buffers for storing the indexes of encoded activations and weights. The weights buffer contains preprocessed weight data entered into the PE array in advance. INSPIRE adopts the IP-PE design, based on index-pair matching, instead of the conventional PE design. The results are obtained through search, pass through the accumulation unit, and enter the encoder. The encoder transforms the results into indexes, which are then outputted through the output buffer.

The INSPIRE architecture includes a controller, an accumulation unit, and associated peripheral circuitry. As INSPIRE encodes based on indexes, there is no need for a decoder in the architecture, resulting in a significant reduction in computational load. The detailed description of IP-PE and the encoder highlights how INSPIRE optimally leverages the benefits of neural network compression.

4.2 IP-PE Design

As previously discussed, with the offline encoding of activations and weights completed, matrix computations now leverage a LUT instead of the traditional PE. The search operation, driven by the weight-activation index pair, efficiently retrieves the corresponding computation results, as depicted in Figure 2(c). Furthermore, given that the computation involves a fixed weight pattern (one

Algorithm 1 INSPIRE Encoding Algorithm

Input: Pre-trained model M, Training dataset D, Online input activation X, Accuracy loss constraint $\mathcal{L}$ (by user).
Output: Indexed model M_q and a codebook T.

{Phase-1: Get the centroids.}
$C_A, C_W \leftarrow \text{GapStatistic}(M, D)$
▷ Get initial number of centroids.
while $(\mathcal{L}_q \geq \mathcal{L})$ **do**
  for layer $l := 1$ to L **do**
    $V_A^l, V_W^l \leftarrow \text{Clustering}(C_A, C_W, M, D)$
  end for
  $Q_W \leftarrow \text{Compress}(C_W, M)$
  ▷ Get the compressed model.
  $\mathcal{L}_q \leftarrow \text{Evaluate}(Q_W, V_A)$
  $C_A, C_W \leftarrow \text{Update}(\mathcal{L}_q, \mathcal{L})$
  ▷ Increase or decrease C depending on the accuracy loss.
end while
$I_A, I_W \leftarrow \text{Indexing}(C_A, C_W, V_A, V_W)$
▷ After find the optimal centroids, do indexing on activation and weight.
$M_q \leftarrow \text{Indexing}(I_W, Q_W)$
▷ Convert the compressed model to indexed model.
$T \leftarrow \text{Compute}(I_A, I_W, V_A, V_W)$
▷ Make the index pair codebook.
{Phase-2: Fast online encoding and calculation}
while Have input activation $x \in X$ **do**
  $i_x \leftarrow \text{BinaryIndex}(x, V_A)$
  ▷ Convert to index by binary search among activation centroids.
  $r_x \leftarrow \text{Lookup}(i_x, i_w, T)$
  ▷ Get calculation results by index pair matching.
end while
$R \leftarrow \text{Accumulate}(r)$
▷ Accumulate and output.
return Indexed model M_q and a look-up table T.

column at a time), the LUT design of the IP-PE is further optimized. Sequentially fixing the weights in the LUT ensures that only results corresponding to different activations with this weight are retained. This optimization significantly reduces the number of LUT entries recorded in a single IP-PE from $C_W \times C_A$ to C_A, resulting in a noteworthy improvement in search speed when searching multiple IP-PEs simultaneously. This approach also markedly reduces the area occupied by a single IP-PE, similar to the bucket method based on the weight centroids, as illustrated in Figure 2(b). Even if the platform lacks support for such a flexible configuration, the IP-PE can maintain $C_W \times C_A$ table entries, given the small number of clusters and to avoid becoming a bottleneck in the DNN computation.

4.3 Encoder Design

We design the encoder, as depicted in Figure 3. A binary discriminator tree with $\log C$ layers is employed to encode C centroids, where each node is equipped with a comparator. In such a tree, leaf nodes save the centroid values and non-leaf nodes save average number of their child nodes. The comparator compares the input

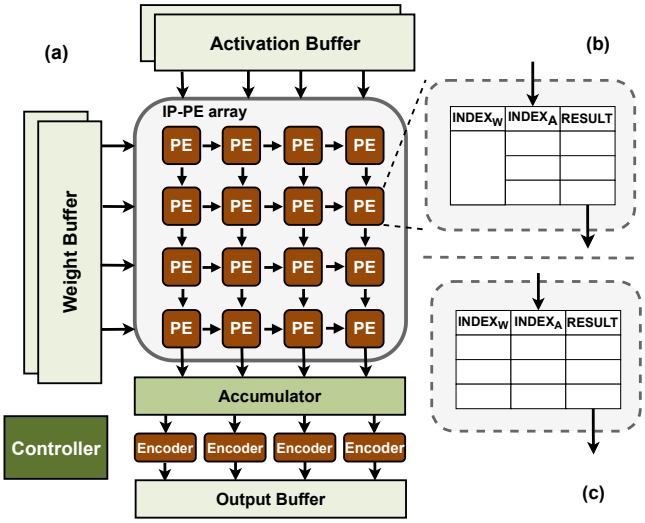

Figure 2: (a) Overview of INSPIRE architecture. (b) The IP-PE design of with LUT entries $1 \times C_A$. (c) The IP-PE design of with LUT entries $C_W \times C_A$

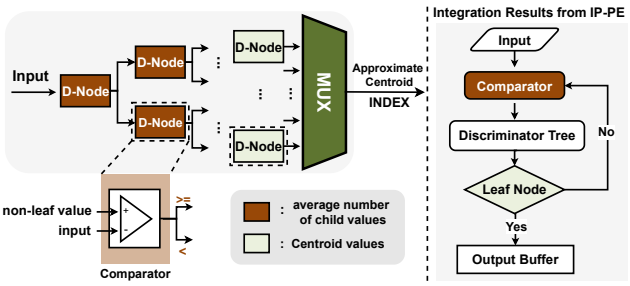

Figure 3: Left: The design of INSPIRE encoder. Right: The computation process of encoding.

with the current node value, the result of which decides which branch input data will flow into. This process will not finish until the input reaches the leaf node storing the centroid whose index will be recorded as the encoding result. The overall computation process iterates $\log C$ times through the nodes. In the pipelined computation process, the comparison steps do not become the bottleneck. Only one node of a layer is activated each cycle, allowing the rest of the nodes to remain inactive in the computation, thereby maximizing energy consumption savings.

4.4 Index-based Computation Support

The INSPIRE encoding significantly reduces the number of parameters that requires storage through the utilization of clustering methods, thereby minimizing overhead. To further mitigate transmission and storage costs, we convert compressed values to indexes, replacing high-precision values with low-precision indexes in the computation. Additionally, instead of resource-intensive multiplications, we opt for a fast index-pair table lookup to reduce computational and latency overhead.

Simultaneously, our proposed INSPIRE-based encoding supports the systolic array architecture by introducing the innovative IP-PE with a LUT and optimized encoder for efficient result searching.

Table 1: Accuracy results of evaluated model and dataset.

Type	CNN-based			Attetion-based	
Model	VGG16	ResNet18	ResNet50	ViT	BERT
Dataset	ImageNet				SST-2
FP-32 Acc.(%)	71.59	69.76	76.15	84.19	90.45
INSPIRE Acc.(%)	71.03	68.91	75.55	83.33	89.95
$\#C_A$	12	16	16	54	64
$\#C_W$	6	5	9	32	54

This design enhances pipelining and parallelization, maximizing acceleration through co-design of algorithms and architectures.

5 EXPERIMENTS

5.1 Experimental Methodology

Benchmark. To validate that INSPIRE can effectively accelerate DNNs, we evaluate both CNN and attention-based models, including computer vision and natural language processing tasks. For computer vision, we use VGG-16, ResNet-18, ResNet-50, and the attetion-based model ViT[7] on the ImageNet dataset [5]. For natural language processing, we evaluate BERT-Base with eight datasets from the GLUE dataset suite [23]. We exploit pre-trained network models from TorchVision and Huggingface Model Zoo, and their validation accuracy with FP32 as baseline.

Baselines. We implement the INSPIRE encoding framework in PyTorch [17]. We evaluate five baselines compared against INSPIRE, including: 1) Eyeriss [3], a spatial energy-efficient dataflow architecture employing coarse-grained INT16 quantization throughout the network, serving as the baseline for standard NN accuracy; 2) BitFusion [19], an accelerator with a composable MAC unit that can change quantization at the layer granularity, maintaining accuracy with coarse-grained INT16 quantization throughout the network; 3) OLAccel [16], a state-of-the-art accelerator based on outlier-aware low-precision computation; 4) AdaFloat [21], requiring an 8-bit floating-point data type to maintain the original model accuracy; 5) ANT [8], a hardware-friendly quantization accelerator combining power-of-two data types and INT types for low bit-width. For models that are not available in their paper, we reproduce their experiments and report the results.

Accelerator Implementation. We implement the INSPIRE IP-PE and encoder described in Section 4 with the Verilog RTL. Meanwhile, we use the 28 *nm* TSMC technology library and Synopsys Design Compiler [4] to study the area and energy of those components that we designed. In addition, for a fair comparison, we use the same global buffer capacity (2 MB) and memory bandwidth for all these accelerators and use CACTI [1] to estimate it that can satisfy our design goals. We use DeepScaleTool to scale all designs to the 28 *nm* process for the iso-area comparison. To evaluate the performance of our proposed INSPIRE architecture, we develop a cycle-accurate simulator to simulate the PE array and output accumulation together with the encoder. The INSPIRE architecture can be scaled to a larger number of PEs under the same area budget.

5.2 Experimental Results

Accuracy Results. We apply the INSPIRE coding algorithm, as described in Section 3, to compress various pre-trained models. The results in Table 1 reveal that activations and weights can be effectively clustered to just a dozen or even a few centroids, equivalent to achieving compression to 3-4 bits. Notably, this compression retains a similar level of accuracy to the original models. Specifically, INSPIRE employs 12 centroids for activation and 6 centroids for weight in VGG16, demonstrating impressive compression performance. Similarly, for attention-based models, INSPIRE effectively limits clustering to a relatively low level.

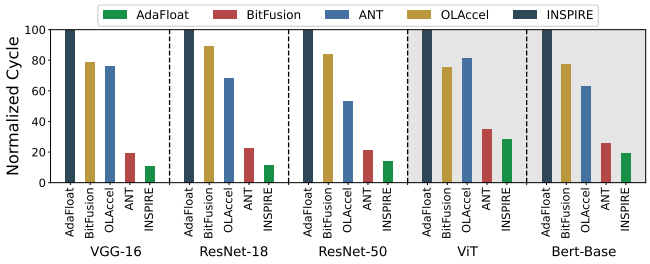

Figure 4: Performance comparison for different networks.

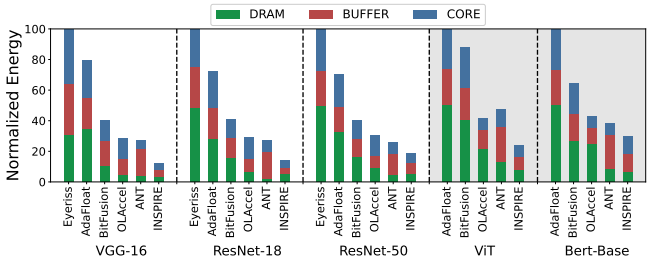

Figure 5: Energy comparison for different networks.

On one hand, the INSPIRE encoding substantially reduces the storage demand for neural network weights, transitioning from FP-based values to INT-based indexes ($\leq$ 5-bit). On the other hand, although compressing activations poses a more intricate challenge, INSPIRE adeptly lessens the number of activated centroids to represent activations. These findings highlight the efficacy and scalability of the proposed INSPIRE.

Performance. Figure 4 illustrates the normalized execution cycles of INSPIRE and other methods on different networks. From the plot, INSPIRE exhibits a significant advantage in terms of performance improvement. In comparison, INSPIRE achieves up to 9.31×, 7.95×, 7.06×, and 1.97× speedup values over AdaFloat, BitFusion, OLAccel, and ANT. Meanwhile, the performance improvement of INSPIRE is more pronounced on DNNs with more concentrated clustering and demonstrates excellent performance on attention-based models. Specifically, INSPIRE can achieve up to an 89.2% performance improvement on VGG-16. This substantial improvement is attributed to the index pair encoding algorithm in INSPIRE, allowing the replacement of the 4-bit or 8-bit PE multiplication with IP-PE, an efficient lookup table design. The IP-PE further saves the area of a single PE, enabling the deployment of more IP-PEs within the same area budget, resulting in better speedup.

Energy Figure 5 compares the normalized energy consumption of INSPIRE and other works. The energy consumption is decomposed into DRAM, global buffer, and processing core (Core). For ResNet-50, INSPIRE consumes up to 81.3% less energy than Eyeriss and up to 27.7% less than ANT. For Bert, INSPIRE consumes

Table 2: The configuration and area breakdown of INSPIRE and other works under 28 nm process.

Architecture	Core		
	Component	Number	Area(mm^2)
INSPIRE	Encoder(12.44 μm^2)	128	0.165
	IP-PE(9.95μm^2)	16,384	
ANT	Decoder(4.9μm^2)	128	0.327
	4-bit PE(79.57 μm^2)	4,096	
BitFusion	4-bit PE	4,096	0.326
OLAccel	4-bit & 8-bit PE	1,152	0.309
AdaFloat	8-bit PE	896	0.327
Eyeriss	16-bit PE	168	0.309

69.8%, 53.4%, 30.1%, and 21.8% less energy than AdaFloat, BitFusion, OLAccel, and ANT, respectively.

This energy advantage primarily comes from optimizing IP-PE by substituting the costly multiplication operation with encoder subtraction and table lookup, thereby significantly reducing energy consumption. Additionally, INSPIRE stores only indexes in the buffer, offering it a substantial advantage over other schemes.

Area. Table 2 shows the area breakdown of INSPIRE under 28 nm process, with other accelerators scaled to the same process for comparison. The results indicate that the encoder just occupy a small fraction of the area, about 0.96% of the overall device. And the encoding strategy enables low hardware overhead through the implementation of Table Lookup in IP-PE. Overall, with a similar area budget, INSPIRE accommodates a larger number of IP-PE. The small area overhead of our INSPIRE directly benefits from the carefully designed index pair encoding and computation based on the table lookup.

Impact of Centroid Number. The number of centroids impacts not only the inference efficiency but also the model accuracy. Figure 6 presents accuracy and normalized energy for different centroid numbers in ResNet-50. Generally, a smaller number of centroids results in fewer entries in IP-PE, reducing searching costs but potentially degrading model accuracy. Additionally, it is observed that activation is more sensitive to the number of centroids compared to weight. Therefore, a higher number of centroids is required for activation to maintain accuracy. The energy consumption primarily depends on the bitwidth required for the centroid index, impacting both storage and encoding overhead. The trade-off involves choosing an optimal solution that balances accuracy and energy consumption.

6 CONCLUSION

This paper introduces INSPIRE, an algorithm/architecture co-design framework poised to facilitate efficient DNN inference. Central to this approach is the strategic utilization of the table lookup, simplifying the inference software and hardware design and decoupling with the table updates. By the centroid quantization technique for DNN, INSPIRE achieves comparable accuracy for complex tasks with much less resource cost. Experiments demonstrate that INSPIRE can yield satisfactory performance gains with trivial accuracy loss, making it a promising solution for the DNN deployment.

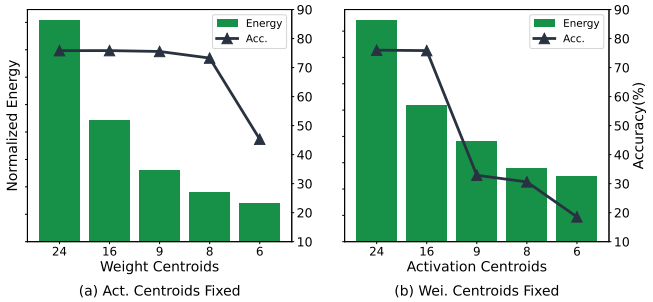

Figure 6: Accuracy and normalized energy with different number of centroids. The Centroids of both activation (a) and weight (b) are fixed to 16.

ACKNOWLEDGMENTS

We thank Huan Zhou for improving the figures. This work was partially supported by the National Natural Science Foundation of China (Grant No. 61975124, 62202288). Li Jiang is the corresponding author (ljiang_cs@sjtu.edu.cn).

REFERENCES

[1] Rajeev Balasubramonian et al. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *TACO* (2017).
[2] Tom Brown et al. 2020. Language models are few-shot learners. *NIPS* (2020).
[3] Yu-Hsin Chen et al. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *JETCAS* (2019).
[4] Synopsys Design Compiler. 2019. [Online]. Available: https://www.synopsys.com/support/training/rtlsynthesis/design-compiler-rtl-synthesis.html.
[5] Jia Deng et al. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.
[6] Lei Deng et al. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* (2020).
[7] Alexey Dosovitskiy et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[8] Cong Guo et al. 2022. Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization. In *MICRO*.
[9] Benoit Jacob et al. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*.
[10] Shubham Jain et al. 2019. BiScaled-DNN: Quantizing long-tailed datastructures with two scale factors for deep neural networks. In *DAC*.
[11] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *ISCA*.
[12] Sangil Jung et al. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *CVPR*.
[13] Fangxin Liu et al. 2021. Improving neural network efficiency via post-training quantization with adaptive floating-point. In *ICCV*.
[14] Fangxin Liu et al. 2024. SPARK: Scalable and Precision-Aware Acceleration of Neural Networks via Efficient Encoding. In *HPCA*.
[15] Nvidia. 2020. Ampere Architecture Whitepaper. https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.
[16] Eunhyeok Park et al. 2018. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*.
[17] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *NIPS* (2019).
[18] Jennifer Scott and Miroslav Tuuma. 2023. *Algorithms for sparse linear systems*. Springer Nature.
[19] Hardik Sharma et al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ISCA*.
[20] Zhuoran Song et al. 2020. Drq: dynamic region-based quantization for deep neural network acceleration. In *ISCA*.
[21] Thierry Tambe et al. 2020. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In *DAC*.
[22] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *J R STAT SOC B* (2001).
[23] Alex Wang et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv* (2018).
[24] Guangxuan Xiao et al. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*.
[25] Ali Hadi Zadeh et al. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *MICRO*.