# Drift: Leveraging Distribution-based Dynamic Precision Quantization for Efficient Deep Neural Network Acceleration

Lian Liu[1,2,3,4], Zhaohui Xu[1,5,6], Yintao He[2,3], Ying Wang[1,2,3,4], Huawei Li[2,3], Xiaowei Li[2,3,4], Yinhe Han[1,2,3]

[1] *CICS, Institute of Computing Technology, Chinese Academy of Sciences*
[2] *SKLP, Institute of Computing Technology, Chinese Academy of Sciences*
[3] *University of Chinese Academy of Sciences* [4] *Zhongguancun National Laboratory*
[5] *School of Information Science and Technology, ShanghaiTech University*
[6] *Shanghai Innovation Center for Processor Technologies*

## ABSTRACT

Quantization is one of the most hardware-efficient ways to reduce inference costs for deep neural network (DNN) models. Nevertheless, with the continuous increase of DNN model sizes (240× in two years) and the emergence of large language models, existing static quantization methods fail to utilize the sparsity and redundancy of models sufficiently. Motivated by the pervasive dynamism in data tensors across DNN models, we propose a dynamic precision quantization algorithm to further reduce computational costs beyond statically quantized DNN models. Furthermore, we find that existing precision-flexible accelerators cannot support the DNN models with dynamic precision. To this end, we design a novel accelerator, Drift, and achieve online scheduling to efficiently support dynamic precision execution. We conduct experiments with various DNN models, including CNN-based and Transformer-based models. Evaluation results show that Drift achieves 2.85× speedup and 3.12× energy saving compared to existing precision-flexible accelerators with statically quantized models.

**ACM Reference Format:**
Lian Liu[1,2,3,4], Zhaohui Xu[1,5,6], Yintao He[2,3], Ying Wang[1,2,3,4], Huawei Li[2,3], Xiaowei Li[2,3,4], Yinhe Han[1,2,3], [1] *CICS, Institute of Computing Technology, Chinese Academy of Sciences*, [2] *SKLP, Institute of Computing Technology, Chinese Academy of Sciences*, [3] *University of Chinese Academy of Sciences* [4] *Zhongguancun National Laboratory*, [5] *School of Information Science and Technology, ShanghaiTech University*, [6] *Shanghai Innovation Center for Processor Technologies* . 2024. Drift: Leveraging Distribution-based Dynamic Precision Quantization for Efficient Deep Neural Network Acceleration. 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/10.1145/3649329.3655986

## 1 INTRODUCTION

Deep Neural Networks (DNNs) have revolutionized various fields, including computer vision and natural language processing. However, the continued growth in model size poses challenges for deploying DNNs on resource-constrained devices, such as smartphones and embedded systems [8, 13]. One promising approach to address these challenges is quantization, which compresses high-precision data types into low-precision integers without significantly compromising their accuracy.

Existing works [7, 10, 15, 31] mainly adopt static quantization methods, which determine the precision (bit-width) of activations and weights prior to inference. Nevertheless, as prior works [2, 24] indicate, large discrepancies exist among different regions of data tensors for DNN models, particularly the heavily researched large language models (LLMs) [1, 28]. For example, in computer vision tasks, the pixels representing the object of interest are typically more important than the background pixels. However, the discordance between the dynamics in the data tensor and existing static quantization methods precludes fully exploiting model sparsity and redundancy to further boost execution efficiency.

Consequently, recent works have explored dynamic precision quantization for DNNs. Precision Gating [30] initially performs low-precision computation and then supplements higher precision based on the output to sustain accuracy. DRQ [24] proposes a method that dynamically selects the precision (4-bit or 8-bit) for computation based on the average value of different regions in the activation tensor for image classification tasks. DTQAtten [27] selects different precisions for each token in the attention-based NLP models based on their difference in tolerance to noise. Nonetheless, existing dynamic precision quantization approaches have two drawbacks. (i) Existing dynamic quantization algorithms are designed for specific scenarios and lack generality to support diverse DNN models. For instance, directly applying DRQ, a dynamic quantization method designed for CNN-based image classification models, to the BERT-based NLP model incurs over 12% accuracy drop (Section 5.2). (ii) Existing precision-flexible accelerators designed for static quantization cannot support the DNN models with dynamic precision. For example, a typical precision-flexible accelerator, BitFusion [22], which is designed to support various low-precision computations, is utilized along with systolic array-based architecture. However, the systolic array-based architecture causes data flow stalls under dynamic precision computations, precluding hardware benefiting from low-precision computations (Section 2.3).

To accommodate diverse types of DNNs (e.g. CNN-based, BERT-based, and even large language models), we eschew model-specific analysis and capitalize on the ubiquitous data distribution patterns in DNN models for dynamic precision quantization. Furthermore, we propose two novel metrics to evaluate the representation capacity of data (Section 3.2). Leveraging these metrics, we devise a dynamic precision quantization algorithm that dynamically selects precision settings by judging the representation capacity of low-precision format. Our algorithm tunes precision to maximize low-bit computations while maintaining accuracy. Need to mention that,

*DAC '24, June 23–27, 2024, San Francisco, CA, USA*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655986

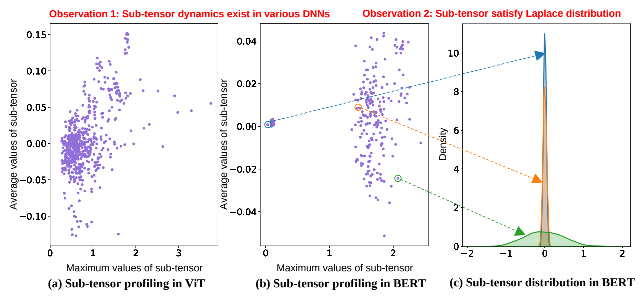


**Figure 1: The sub-tensor dynamics and distribution in DNNs.**

our proposed algorithm utilizes existing hardware resources and does not introduce additional computational or area overheads.

To efficiently support our proposed dynamic precision quantization algorithm, we further design a novel hardware architecture, Drift. Compared with existing accelerators, Drift provides flexible connections among processing elements (PEs) to simultaneously support different systolic arrays on one chip. Drift avoids data flow stalls by steering different precision computations into separate systolic arrays. Furthermore, the dynamic characteristic of our quantization method precludes knowing the computation ratio of each precision combination before runtime. To address this, we propose an online scheduling strategy that loads balances across systolic arrays to maximize overall performance.

In general, we present an algorithm-architecture co-design with dynamic precision quantization, namely Drift. We make the following contributions in this paper.

- We propose a dynamic precision quantization algorithm for diverse types of DNN models, including CNN-based and Transformer-based models. The algorithm leverages our proposed metrics to evaluate the representation capability and select appropriate precision settings for different data.
- We design a dataflow flexible novel hardware to benefit from our proposed dynamic precision quantization algorithm. This architecture can distribute different precision computations onto different systolic arrays to avoid data flow stalls. We further propose an online scheduling strategy to balance the computation in different precision settings.
- We implement and demonstrate the effectiveness of the proposed design on popular DNN models. On one hand, our proposed algorithm can efficiently support various DNN models, even including LLMs. On the other hand, evaluation on Drift achieves 2.85× and 1.64× speedup on average over the BitFusion and DRQ, respectively.

## 2 BACKGROUND & MOTIVATION

### 2.1 Sub-tensor Dynamics & Distribution

In DNN models, activation and weight data are typically represented as tensors. A sub-tensor refers to a subset of the elements in a tensor. For example, a patch in the activation tensor of a ViT [6] model or a token in the activation tensor of a BERT [3] model can be considered as a sub-tensor. As existing works [24, 27] indicate, different sub-tensors within a tensor often exhibit distinct data characteristics. However, little has been conducted detailed analysis on the distribution characteristics of different sub-tensors within one tensor. As a consequence, we profile the sub-tensor distributions at the patch and token granularity for ViT and BERT models. As shown in Figure 1, we can reveal two observations:

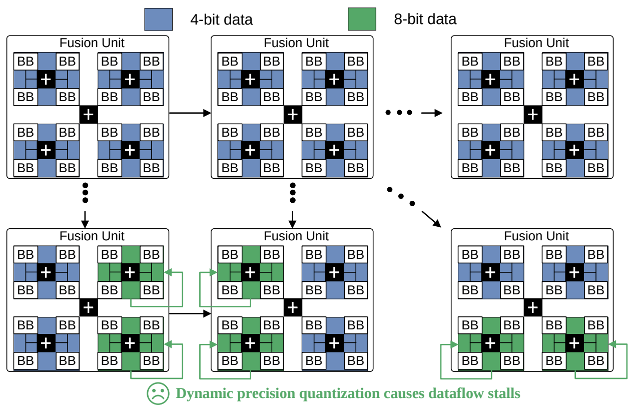


**Figure 2: Existing precision-flexible accelerator cannot support DNN inference with dynamic precision quantization.**

1. Sub-tensor dynamics are prevalent across diverse DNN models. As illustrated in Figure 1, sub-tensors within the same tensor exhibit vastly different value ranges and variances. For example, in the activation tensor of a ViT model activation tensor (Figure 1a), the maximum value of some sub-tensors is nearly 0 while others exceed 3. Using the same precision to represent both sub-tensors is clearly unnecessary in this case. Therefore, fully exploiting sub-tensor dynamics enables aggressively reducing precision for computations.

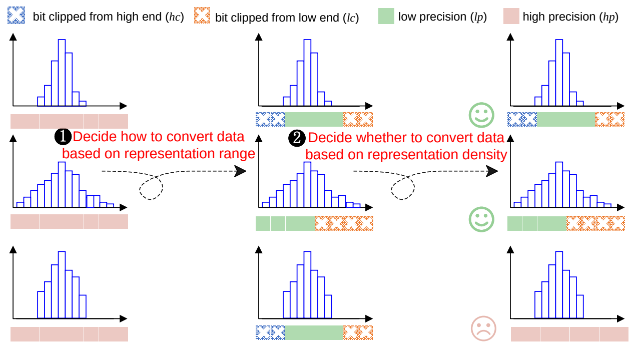
2. Despite exhibiting distinct value ranges and variances, we find that Laplace distributions well-approximate nearly all sub-tensors. As shown in Figure 1b-c, we sample three sub-tensors with distinct data characteristics from a BERT activation tensor and plot their distributions. Remarkably, despite differences, all sub-tensors roughly conform to Laplace distributions with zero mean. Our observations reveal this pattern persists across models. This ubiquity of Laplace-distributed sub-tensors enables us to design a generally applicable dynamic precision algorithm by exploiting their mathematical properties.

### 2.2 Dynamic Precision Quantization

With the exploration of sub-tensor dynamics in DNN inference, some works on dynamic precision quantization have been proposed. Precision Gating [30] proposes a per-value quantization method to determine each activation data into two precision settings (3-bit and 5-bit) dynamically and then retrain the model to recover its accuracy. However, the model retraining and per-value gating incur intolerable hardware costs [2, 11]. DRQ [24] exploits input feature map dynamics in image classification, where sparse sensitive areas govern model accuracy. Therefore, high-precision (8-bit) only needs to be used for these sensitive regions, while other parts use low-precision (4-bit). However, the proposed dynamic precision selection criteria cannot be migrated well to other types of DNN models (Section 5.2), and the region size needs to be carefully selected. DTQAtten [27] selects different precision settings for each token in the attention of NLP models based on their difference in tolerance to noise, which cannot transferred to computer vision models. Due to requiring costly retraining [30] or designing for specialized models [24, 27], prevailing dynamic quantization techniques have poor transferability to diverse types of DNN models.

### 2.3 Precision-Flexible Accelerator

Quantized DNN models, particularly mixed-precision models, have motivated the design of precision-flexible accelerators [12, 17, 20, 24].



bit clipped from high end (hc)
bit clipped from low end (lc)
low precision (lp)
high precision (hp)
❶Decide how to convert data based on representation range
❷Decide whether to convert data based on representation density

**Figure 3: The proposed algorithm utilizes representation range and density to decide whether and how to convert high-precision sub-tensor to low-precision.**

BitFusion [22] is a typical precision-flexible accelerator. As shown in Figure 2, it consists of a systolic array with flexible precision support formed by 2-bit BitBricks (BB) as basic blocks. Before runtime, multiple BitBricks can be spatially fused into a processing element (PE) to support different precision. For example, fusing 4 BitBricks together can form a 4-bit PE, as the colored part shows. However, dynamic precision quantization requires determining the precision of each sub-tensor at runtime, preventing pre-configuring appropriate precisions for sub-tensors. As illustrated in Figure 2, in order to support dynamic precision, a PE requires multiple cycles to compute high-precision data, resulting in data flow stalls. As a result, existing precision-flexible accelerators cannot support dynamic quantization algorithms.

## 3 DRIFT ALGORITHM DESIGN

As elucidated in Section 2.1, the prevalent sub-tensor dynamics motivate increasingly adopting dynamic quantization to reduce computational precisions. Specifically, dynamic precision quantization converts high-precision integers from initial quantization to lower precision values to further decrease computation costs. In this section, we first discuss the preliminary concept of dynamic precision quantization (Section 3.1), then present the metrics of representation capability (Section 3.2), and finally utilize them to achieve novel dynamic precision selection algorithm (Section 3.3).

### 3.1 Preliminary

**Quantization** compresses high-precision values into low-precision integers. For example, the process that quantizes 32-bit floating point values into 8-bit integers can be expressed as:

$$\overline{X}^{INT8} = \lceil \frac{X^{FP32}}{\Delta} \rfloor, \quad \Delta = \frac{max(|X|)}{2^{N-1}-1} \tag{1}$$

where $X$ is the floating-point tensor, $\overline{X}$ is the quantized integer tensor, $\Delta$ is the quantization scaling factor, $\lceil \cdot \rfloor$ is the rounding function, and $N$ is the number of bits (8 in our case).

After the initial quantization of floating point data to integers, dynamic precision quantization will further convert high-precision integers to low-precision values for suitable sub-tensors. As the middle column shown in Figure 3, there are multiple choices to convert a high-precision integer to a low-precision. It can clip $hc$ bits from the high end and $lc$ bits from the low end to convert an $hp$-bit value to an $lp$-bit one. For example, there are five choices to convert an 8-bit integer to 4-bit. All the variables above satisfy the following constraints:

$$\begin{aligned} &hp = hc + lp + lc \\ &hp, lp, hc, lc \geq 0 \end{aligned} \tag{2}$$

Consequently, a dynamic quantization algorithm needs to not only ascertain whether to convert high-precision values to low-precision, but also elect the appropriate conversion strategy.

### 3.2 Representation Capability

To uniformly evaluate different precision conversion choices, we propose the concept of representation capability. Specifically, we propose two novel metrics, **representation range** (RR) and **representation density** (RD), to evaluate the representation capability.

Considering the quantization process in Equation (1), for the quantized tensor, its representation range is $(2^{N-1}-1)\cdot\Delta = max(|X|)$, and its representation density is $\Delta$. In fact, the representation range determines the maximum range that the quantized value can represent, while the representation density indicates the quantization error caused during the rounding process.

When we convert an $hp$-bit sub-tensor $Y_{hp}$ to $lp$-bit $Y_{lp}$ by clipping $hc$-bit from the high end and $lc$-bit from the low end, its representation capability can be characterized as:

$$\begin{aligned} Y_{lp} - RR: &\quad \frac{2^{hp-1}-1}{2^{hc}} \cdot \Delta \\ Y_{lp} - RD: &\quad 2^{lc} \cdot \Delta \end{aligned} \tag{3}$$

Therefore, we can utilize the representation capability to characterize whether the lower-precision sub-tensor adequately conveys the original data.

### 3.3 Dynamic Precision Selection

Based on our observation presented in Section 2.1, we assume that every sub-tensor $Y$ follows a Laplace distribution with zero mean. Fortunately, the Laplace distribution with zero mean exhibits a useful property: taking the absolute value of data drawn from this distribution yields an exponential distribution.

$$|Y| \sim Exponential(b^{-1}) \tag{4}$$

By using the maximum likelihood estimation for the exponential distribution, $b$ can be represented by $avg(|Y|)$, while the variance of $Y$ is $var(Y) = 2 \cdot avg(|Y|)^2$.

Consequently, we can use the maximum and average value of sub-tensor $Y$ to measure its numerical range and variance, and further utilize the representation ability to evaluate whether a low-precision value can effectively represent the sub-tensor. In general, our dynamic precision selection process is illustrated in Figure 3.

We divide the entire process into two steps. First, we use the evaluation of the representation range to determine how to convert high-precision sub-tensor to low-precision (Figure 3❶). Specifically, we need to decide the bit that can be clipped from the high end and the low end. Based on the relation in Equation (2), when $hc$ is decided during inference, the corresponding data conversion strategy is fixed. Since the representation range of the low-precision sub-tensor $Y_{lp}$ must exceed its maximum value, we can determine the value of $hc$ based on this criteria.

$$\begin{aligned} &Y_{lp} - RR \geq max(|Y|) \\ \Rightarrow \quad &hc = \lfloor log_2(\frac{2^{hp-1}-1\cdot\Delta}{max(|Y|)}) \rfloor \end{aligned} \tag{5}$$

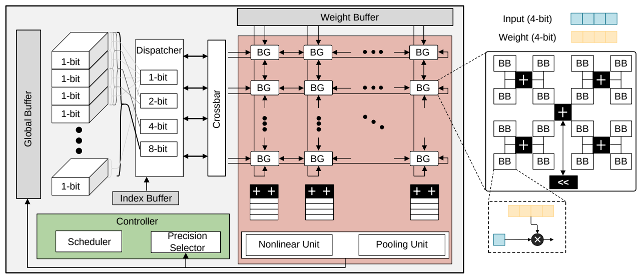


Figure 4: The architecture overview of Drift.

As exemplified in the second row of Figure 3, due to the wide distribution range, we cannot clip bit from the high end but only clip bit from the low end ($hc = 0, lc = 4$).

Then, the algorithm needs to determine whether the conversion strategy can satisfy the representation density (Figure 3❷). The representation density is related to the variance of data tensor [16]. When the data tensor has a large variance, a larger scaling factor can be tolerated for quantization. However, when the variance is small, a smaller scaling factor must be used to maintain model accuracy. Therefore, we need to ensure the ratio of the variance of $Y$ to the representation density is greater than a threshold $\delta$.

$$\frac{var(Y)}{Y_{lp} - RD} = \frac{2 \cdot avg(|Y|)^2}{2^{lc} \cdot \Delta} \geq \delta \tag{6}$$

As illustrated in Figure 3, due to a small variance, the sub-tensor in the third row fails to satisfy the representation density constraint, resulting in the selection of high-precision 8-bit for computation while the other sub-tensors can use 4-bit.

Here, the threshold value $\delta$ is a hyperparameter determined before inference. In this paper, we use the Hessian-aware strategy [5, 23] to select the appropriate threshold. The Hessian-aware strategy can quickly identify the minimum threshold with negligible impact on model accuracy. This allows our algorithm to select low-precision sub-tensors as much as possible. After finishing the selection steps above, we can acquire the specific precision selections, as well as their corresponding precision conversion choices for each sub-tensor, as illustrated in the last column of Figure 3.

## 4 DRIFT ARCHITECTURE DESIGN

Although the proposed dynamic precision quantization algorithm reduces computational complexity, the data flow stalls preclude the existing precision-flexible accelerator benefiting from dynamic precision. Therefore, we design a novel accelerator, Drift, to fully exploit the reduced computations in the dynamic precision quantization algorithm and improve the efficiency of model inference. In this section, we first present the overview of our design Drift accelerator(Section 4.1). Then, we explain how dataflow splitting is utilized to avoid stalls in Drift (Section 4.2). We further propose an online scheduling strategy to balance the computational efficiency of different precisions (Section 4.3).

### 4.1 Architecture Overview

The architecture overview of Drift is presented in Figure 4. It contains the following major components.

**Computing engine.** The computing engine in Drift is composed of an array of BitGroups (BGs) as the basic building units, while each BG contains a $4 \times 4$ BitBricks (BB) array, which is similar to [20]. Each BB can support a 1-bit and 4-bit multiplication between input

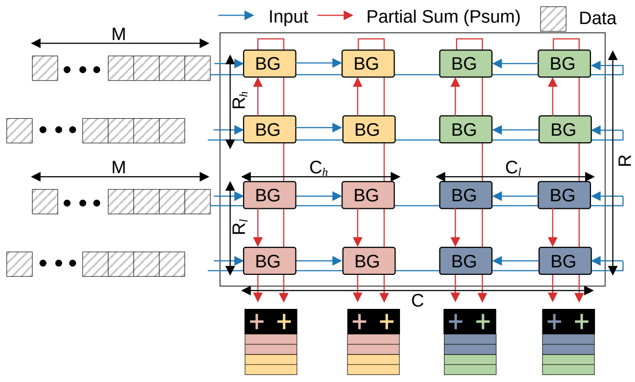


Figure 5: The hardware resource in Drift can be allocated to support different systolic arrays.

and weight. This enables Drift to support more flexible precision (e.g., 3-bit and 5-bit in [30]). Unlike existing accelerators, each BG establishes bidirectional connections with its surrounding BGs. In addition, the computing engine also contains a nonlinear unit and pooling unit to support activation and pooling functions in DNN models. Meanwhile, the pooling unit is also used to execute the proposed dynamic precision quantization algorithm.

**Memory hierarchy.** Drift mainly contains three on-chip buffers. The global buffer controls the input and output of the DNN model, sending input activations to the computing engine and receiving output data from it. The weight buffer stores the quantized weights and supplies them to the computing engine. Additionally, to support dynamic precision computation, Drift contains an index buffer that tracks the precision of data at specific positions. This serves as a reference for the dispatcher to control access to the activation data.

**Controller.** The controller in Drift contains two components: the precision selector and the scheduler. The precision selector executes our proposed dynamic precision algorithm based on the results from the pooling unit, and records the execution results. It primarily consists of a comparator to perform the relevant comparisons, and a lookup table to record the precision selection results. The scheduler utilizes the results from the precision selector to dynamically adjust the data flow direction between BGs at run-time. The specific scheduling method will be explained in Section 4.3.

### 4.2 Architecture for Dataflow Splitting

As demonstrated in Section 2.3, supporting different precision computations in a single systolic array causes dataflow stalls. Our insight is that we can map different precision computations to different systolic arrays with their own data flow. For this purpose, we implement bi-directional connections between BGs to provide more flexible interconnections. As shown in Figure 5, by configuring the data flow direction among BGs, we can split Drift into four different systolic arrays, allowing each systolic array to independently support computations of one specific precision. For example, we can allow the systolic array in the top left corner to support high-precision matrix multiplication (GEMM) between activations and weights, while letting the systolic array in the bottom left corner support operations between low-precision activations and high-precision weights.

Furthermore, the computing resources occupied by one systolic array can be dynamically reallocated. For example, as shown in Figure 5, we can reconfigure the data flow direction of the partial sum (psum) in the third row of BGs from downward to upward. This allows the systolic array for high-precision activation computation to have more computational resources.

## 4.3 Balanced Online Scheduling

Since the proportion of data with different precisions varies in each layer, we need to dynamically adjust the resource distribution in each systolic array to balance the latency of each part and reduce the idleness of hardware resources. To this end, we achieve balanced resource allocation via a greedy scheduler. This scheduler uses an analytical model to estimate the GEMM operation latency under different precision settings and systolic array sizes. The analytical model is extended from [21].

Assume that we need to map a GEMM operation onto our BG-based systolic array, where the three dimensions of GEMM are $M, K, N$, the precision for activation and weight are $pa$ and $pw$, as well as the size of the systolic array is $R \times C$. As indicated in Figure 5, Drift adopts a weight-stationary (WS) dataflow that weight data is preserved in each BG during inference. The computation process includes data preloading and execution, which can be expressed as:

$$T_{pre} = R, \quad T_{exe} = (M + R + C - 2)$$
$$T_{total} = (T_{pre} + T_{exe}) \lceil \frac{pa \cdot K}{4R} \rceil \lceil \frac{pw \cdot N}{16C} \rceil \tag{7}$$

In the preloading stage, weight data is preloaded following top-down behavior, costing $R$ cycles. In the execution stage, the systolic array needs extra $R + C - 2$ cycles to compute the last value. Since a BG contains $4 \times 4$ bitbricks, with each supporting a 1-bit and 4-bit multiplication, the required repetitions are $\lceil \frac{pa \cdot K}{4R} \rceil \lceil \frac{pw \cdot N}{16C} \rceil$.

As illustrated in Figure 5, Drift can be divided into four systolic arrays with different sizes to support GEMM with different precision settings. Here we use $h$ and $l$ to denote high-precision and low-precision data, respectively. Thus, the execution latency of the four systolic arrays can be expressed as $T_{hh}, T_{hl}, T_{lh}$ and $T_{ll}$. Our scheduling target is to minimize the maximum running latency across the four systolic arrays by selecting the appropriate sizes $R \times C$ for each systolic array:

$$\min_{R,C} \max \{ T_{hh}, T_{hl}, T_{lh}, T_{ll} \} \tag{8}$$

Since the precision selections for activations and weights are independent, we can greedily adjust R and C separately to find the optimal scheduling.

## 5 EVALUATION

## 5.1 Experimental Setup

**Algorithm setting.** To validate our dynamic precision quantization algorithm, we conduct experiments on various types of DNN models, including CNN-based [9] (ResNet18, ResNet50), ViT-based [6, 25] (ViT-B, DeiT-S) and BERT-based [3] models. We finetune the BERT model on three tasks from GLUE [18]. Furthermore, we validate our dynamic precision quantization algorithm on three typical large language models, GPT2-XL [19], BLOOM-7B1 [26], and OPT-6.7B [28], to demonstrate the effectiveness in the era of large pre-trained models, with the dataset Wikitext103 [18] and C4 [4]. We compare our algorithm with floating point models (FP32), static quantization models with 8-bit (INT8), and DRQ. For a fair comparison, we set 4-bit precision as our low-precision choices and use the same sub-tensor size as in DRQ. Need to note that, our proposed algorithm can also easily support other precision settings and sub-tensor sizes.

**Hardware setting.** We use RTL to implement our specialized design and synthesize using Synopsys DC on 40nm TSMC library

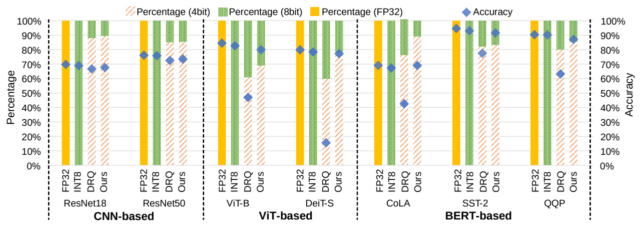


**Figure 6: Comparison of NN accuracy with different methods.**

at the frequency of 500 MHz. The DRAM access latency and power estimation are obtained from DRAMSim3 [14]. For system-level evaluation, we develop a cycle-accurate simulator based on [22], cross-verified with the RTL implementation. We compare Drift with three different accelerators: Eyeriss, BitFusion, and DRQ. Here, we use Eyeriss to execute FP32 models, and BitFusion to execute INT8 models. Meanwhile, DRQ adopts its variable-speed systolic array design to execute dynamic precision DNN models. Following the hardware configurations presented in [24], Eyeriss has 224 ($14 \times 16$) PEs in its computing engine, while other accelerators have 792 units (BitFusion unit or BitGroup) in their own computing engine.

## 5.2 Algorithm Performance

Figure 6 presents the corresponding accuracy and percentage of different data types when adopting various quantization algorithms on diverse DNN models. Overall, our dynamic precision quantization algorithm achieves over 82.4% 4-bit data computation with only 1% accuracy loss compared to INT8 models. When executing BERT on the CoLA dataset, our algorithm even achieves better accuracy than the FP32 model (69.3% vs. 69.1%), demonstrating its effectiveness. As one can notice, although DRQ achieves similar accuracy to our algorithm on CNN-based image classification models, directly applying it to ViT and BERT models causes unacceptable accuracy drops (more than 12%). In contrast, our proposed algorithm is broadly applicable across different types of DNN models.

**Table 1: Evaluate our proposed algorithm on LLM models.**

| Method | GPT2-XL | | BLOOM-7B1 | | OPT-6.7B | |
|---|---|---|---|---|---|---|
| | Wiki | C4 | Wiki | C4 | Wiki | C4 |
| FP32 | 17.48 | 16.30 | 13.05 | 14.94 | 22.14 | 10.63 |
| INT8 | 18.29 | 17.35 | 14.04 | 16.18 | 22.34 | 10.73 |
| Ours | **18.12** | **17.15** | 15.44 | 18.27 | **21.86** | 11.12 |
| | 91.2% | 93.2% | 74.9% | 73.8% | 90.7% | 86.7% |

We further evaluate our proposed dynamic precision quantization algorithm on three representative LLMs, as shown in Table 1. The gray part in the last row indicates the percentage of low-precision (4-bit) data computation when using our algorithm. It can be noticed that using our proposed algorithm can achieve better performance (lower perplexity is better) than basic INT8 models for GPT2-XL, with over 90% of 4-bit data. The evaluation result further shows that our proposed algorithm outperforms the FP32 model to execute OPT-6.7B on the Wikitext dataset, which is quite impressive. Overall, our proposed algorithm can effectively support emerging LLMs with negligible performance degradation, which is necessary in the era of large pre-trained models.

## 5.3 Hardware Performance

As explained in Section 5.1, we compare the performance and energy against existing designs of Eyeriss, BitFusion, and DRQ. The performance and energy consumption of the Eyeriss design without model compression are normalized to 1.

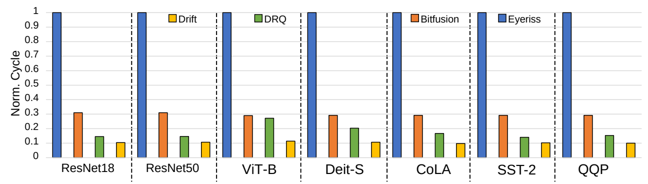


**Figure 7: Latency evaluation for different DNN accelerators.**

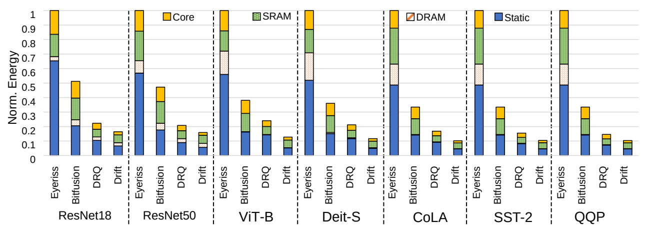


**Figure 8: Comparison of normalized energy and breakdown with different architecture designs.**

As shown in Figure 7, Drift has the most significant advantage in latency speedup. Generally, Drift can achieve on average 9.57× and 2.85× speedup when compared to Eyeriss and BitFusion respectively. As we can notice, despite DRQ and Drift having a similar proportion of low-precision computations, Drift achieves 1.64× speedup compared to DRQ. This is because DRQ struggles to address the dataflow stalls. Especially when the proportion of low-precision computation is reduced (e.g. when executing ViT-B), the frequent occurrence of high-precision computation renders DRQ nearly unable to reap performance benefits from dynamic precision quantization (only 1.07× speedup over BitFusion). In contrast, Drift can map computations of different precisions to separate systolic arrays to avoid data flow stalls during inference.

Additionally, we characterize the energy costs of different hardware architectures and perform a breakdown analysis of the overheads, including static energy and dynamic energy (DRAM, on-chip buffer, and core), as presented in Figure 8. Drift achieves averaged 8.11×, 3.12×, and 1.54× energy reduction over Eyeriss, BitFusion, and DRQ, respectively. The overall performance improvement and energy reduction come from the following contributions: (i) the computation of low-precision data; (ii) the efficient architecture design that reduces computing stalls; and (iii) the balanced online scheduling that achieves load balance among different systolic arrays. Although DRQ and Drift require loading and computing similar amounts of data, Drift fully utilizes hardware resources during execution due to the efficient process of dynamic precision data. Consequently, static energy constitutes a smaller fraction in Drift compared to DRQ (41.2% vs. 51.9%). This also enables Drift to attain 1.54× energy savings than DRQ. Importantly, for fair evaluation, we only use 4-bit precision as the low-precision data computations. In practice, as the BG design in Drift can efficiently support more flexible precision settings (e.g., 3-bit and 5-bit), even lower precisions could be utilized for further performance improvements [29].

## 6 CONCLUSION

In this work, we observed the dynamic nature of the sub-tensor during DNN inference and proposed a generally applicable dynamic precision quantization algorithm. Our algorithm greatly reduces the required precision with negligible accuracy loss. Furthermore, we found that existing precision-flexible accelerators cannot support the efficient execution of DNN models with dynamic precision. To this end, we proposed a novel architecture, Drift, to fully exploit the variable precisions computations during inference. In conclusion, by combining our algorithm with specialized architecture design, Drift achieves 2.85× higher performance compared to the prior design with negligible loss in model accuracy.

## ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (NSFC) under grant No. 62090024, 62222411, and 62025404. The corresponding author is Ying Wang.

## REFERENCES

[1] Tom Brown et al. 2020. Language models are few-shot learners. *NIPS* (2020).
[2] Tim Dettmers et al. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339* (2022).
[3] Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[4] Jesse Dodge et al. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758* (2021).
[5] Zhen Dong et al. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *ICCV*.
[6] Alexey Dosovitskiy et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
[7] Steven K Esser et al. 2019. Learned step size quantization. In *ICLR*.
[8] Amir Gholami et al. 2021. Ai and memory wall. *RiseLab Medium Post* (2021).
[9] Kaiming He et al. 2016. Deep residual learning for image recognition. In *CVPR*.
[10] Itay Hubara et al. 2016. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. arXiv:1609.07061
[11] Dongseok Im et al. 2023. Sibia: Signed Bit-slice Architecture for Dense DNN Acceleration with Slice-level Sparsity Exploitation. In *HPCA*.
[12] Patrick Judd et al. 2016. Stripes: Bit-serial deep neural network computing. In *MICRO*.
[13] Cangyuan Li, Ying Wang, Huawei Li, and Yinhe Han. 2023. APPEND: Rethinking ASIP Synthesis in the Era of AI. In *DAC*. IEEE, 1–6.
[14] Shang Li et al. 2020. DRAMsim3: a cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters* (2020).
[15] Lian Liu et al. 2023. An Automatic Neural Network Architecture-and-Quantization Joint Optimization Framework for Efficient Model Inference. *IEEE TCAD* (2023).
[16] Zhenhua Liu et al. 2021. Post-training quantization for vision transformer. *NIPS* (2021).
[17] Erjing Luo et al. 2023. DeepBurning-MixQ: An Open Source Mixed-Precision Neural Network Accelerator Design Framework for FPGAs. In *ICCAD*. IEEE, 1–9.
[18] Daniel W Otter et al. 2020. A survey of the usages of deep learning for natural language processing. *TNNLS* (2020).
[19] Alec Radford et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* (2019).
[20] Sungju Ryu et al. 2019. Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation. In *DAC*.
[21] Ananda Samajdar et al. 2020. A systematic methodology for characterizing scalability of dnn accelerators using scale-sim. In *ISPASS*.
[22] Hardik Sharma et al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ISCA*.
[23] Sheng Shen et al. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *AAAI*.
[24] Zhuoran Song et al. 2020. Drq: dynamic region-based quantization for deep neural network acceleration. In *ISCA*.
[25] Hugo Touvron et al. 2021. Training data-efficient image transformers & distillation through attention. In *ICML*.
[26] BigScience Workshop et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100* (2022).
[27] Tao Yang et al. 2022. DTQAtten: Leveraging dynamic token-based quantization for efficient attention architecture. In *DATE*.
[28] Susan Zhang et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).
[29] Wei Zhang et al. 2020. TernaryBERT: Distillation-aware Ultra-low Bit BERT. In *EMNLP*.
[30] Yichi Zhang et al. 2019. Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations. In *ICLR*.
[31] Xiandong Zhao et al. 2020. Linear symmetric quantization of neural networks for low-precision integer hardware. (2020).