2024 34th International Conference on Field-Programmable Logic and Applications (FPL) | 979-8-3315-3007-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/FPL64840.2024.00044

SDA: Low-Bit Stable Diffusion Acceleration on Edge FPGAs

Geng Yang[1,3], Yanyue Xie[2], Zhong Jia Xue[3], Sung-En Chang[2], Yanyu Li[2], Peiyan Dong[2],
Jie Lei[4], Weiying Xie[1], Yanzhi Wang[2], Xue Lin[2], Zhenman Fang[3]
[1]*Xidian University,* [2]*Northeastern University,* [3]*Simon Fraser University,* [4]*University of Technology Sydney*
Email: gengyang@stu.xidian.edu.cn, zhenman@sfu.ca

***Abstract*—This paper introduces SDA, the first effort to adapt the expensive stable diffusion (SD) model for edge FPGA deployment. First, we apply quantization-aware training to quantize its weights to 4-bit and activations to 8-bit ($W4A8$) with a negligible accuracy loss. Based on that, we propose a high-performance hybrid systolic array (hybridSA) architecture that natively executes convolution and attention operators across varying quantization bit-widths (e.g., $W4A8$ and all 8-bit QK^TV in attention). To improve computational efficiency, hybridSA integrates diverse DSP packing techniques into hybrid weight-stationary and output-stationary dataflows that are optimized for convolution and attention. It also supports flexible dataflow transitions to address the distinct demands of its output sequence by subsequent nonlinear operators. Moreover, we observe that nonlinear operators become the new performance bottleneck after the acceleration of convolution and attention, and offload them onto the FPGA as well. To reduce the latency of each nonlinear operator, we pipeline its own execution at a fine granularity. To minimize the resource utilization of nonlinear operators, we carefully balance their execution with hybridSA in a coarse-grained pipeline. Experimental results demonstrate that our low-bit ($W4A8$) SDA accelerator on the embedded AMD-Xilinx ZCU102 FPGA achieves a speedup of 97.3× (which takes about 2.1 minutes for one SD inference), compared to the original SD-v1.5 model on the ARM Cortex-A53 CPU (which takes about 3.5 hours for one SD inference). Our SDA project is open sourced here: https://github.com/Michaela1224/SDA_code.**

I. Introduction

In the past few years, diffusion models have demonstrated impressive quality improvement in various image generation tasks over generative adversarial networks [1]–[6]. Among them, the recent stable diffusion (SD) is a latent text-to-image diffusion model that can generate photo-realistic images from textual inputs [7]. However, the high quality of generated images comes at the cost of substantial computational and memory demands, which poses great challenges for deployment on resource-constrained edge devices.

i) **Large model size and computation cost.** SD mainly consists of variational autoencoder (VAE) [8], [9], text encoder [10], and UNet [11]. UNet is the major bottleneck (more than 98% of the total execution time) and has about 800 million parameters (3.1 GB in FP32). Moreover, the denoising UNet demands many iterative forwarding steps to ensure generative quality, e.g., 50 steps in SD-v1.5 [7], and each step requires about 730 GOPs (giga-operations). When running on an embedded ARM Cortex-A53 CPU, this UNet takes about 3.5 hours, which makes it impractical for deployment.

ii) **Complex model structure.** Shown in Fig. 1, in addition to the Transformer blocks [12], which comprises multi-head attention (including both self-attention and cross-attention) and feed-forward networks, the SD UNet also integrates convolution-based ResNet blocks [13]. These two blocks, at various downsample and upsample scales, are nearly evenly distributed within the SD UNet, which constrains the performance gains achievable by existing accelerators specialized for either the convolution or the attention operator [14]–[16].

iii) **Variety of nonlinear operators.** SD integrates a variety of widely used nonlinear operators, including LayerNorm (LNorm) [12], GroupNorm (GNorm) [17], SoftMax [12], GeGLU [18], and SiLU [19], as shown in Fig. 1. These nonlinear operators would become the new bottleneck after the main convolution and attention operators are accelerated. Moreover, they also pose new challenges for dataflow design of the main convolution and attention accelerator, as they require specific sequences of output streams from the main accelerator to facilitate the execution of nonlinear operators.

To address the above challenges, at the algorithm level, we first apply quantization-aware training for the SD model with 4-bit weight and 8-bit activation ($W4A8$) quantization, to reduce the model parameter size by 8× (from 3.1 GB to 389 MB) with comparable FID and CLIP scores.

At the hardware level, we design a low-bit stable diffusion accelerator (SDA) to efficiently execute SD inference on edge FPGAs. To meet the requirements of various main operators (*i.e.*, convolution and matrix multiplication inside attention) and their output sequences for nonlinear operators, we propose a high-performance hybrid systolic array (hybridSA) architecture. First, hybridSA supports native execution of both convolution and attention with different quantization bit-widths (e.g., $W4A8$ and all 8-bit QK^TV in attention). Second, to improve the computation efficiency in hybridSA, we employ two 4-bit DSP packing methods for convolution and matrix multiplication operators to allow them to share a single DSP (DSP48E2). We achieve an average DSP efficiency of 4 ops/DSP and 2 ops/DSP for convolution and matrix multiplication (4-bit multiply by 8-bit). Third, hybridSA supports flexible switching between weight-stationary and output-stationary dataflows to generate output sequences friendly for the processing of subsequent nonlinear operators.

Since nonlinear operators become the new performance bottleneck after the main operators are accelerated on our hybridSA, we accelerate them on the FPGA as well and carefully dataflow their execution with the hybridSA. First, to reduce the on-chip memory usage, we design a shared tile buffer used by all nonlinear operators to communicate with hybridSA. Second, to reduce the latency of each nonlinear operator, we pipeline its own execution in a fine granularity; note that pipeline has a negligible resource overhead. Third, to minimize the resource utilization by nonlinear operators while not hurting the overall performance, we dataflow their execution with hybridSA in a coarse-grained pipeline, and minimize the parallelism degree inside the nonlinear operators

979-8-3315-3007-5/24/$31.00 ©2024 IEEE
DOI 10.1109/FPL64840.2024.00044

to merely match the hybridSA speed so that their execution latency can be hidden (overlapped).

We collect our experiment results for running the contemporary SD-v1.5 model [20] on the embedded AMD-Xilinx ZCU102 ARM-FPGA system-on-chip (SoC). Compared to the original floating-point SD UNet running on the ARM CPU, our SDA design on the FPGA, with 4-bit weight and 8-bit activation, achieves a speedup of 97.3×, with comparable FID and CLIP scores. Compared to the ARM CPU, SDA achieves an average speedup of 126.5× and 77.2× for different ResNet and Transformer blocks, respectively.

In summary, this paper makes the following contributions:

1. The first high-performance low-bit (W4A8) stable diffusion accelerator on edge FPGAs with a negligible accuracy loss.
2. A hybrid systolic array architecture that supports native execution of convolution and attention with different quantization bit-widths, flexible dataflow switching to facilitate nonlinear processing, and effective DSP packing.
3. Resource-efficient nonlinear units implementation and efficient integration with hybridSA via two-level pipelining.
4. A comprehensive evaluation and analysis of SDA.

II. Stable Diffusion and Low-Bit Quantization

A. Overview of Stable Diffusion Model

We select state-of-the-art stable diffusion v1.5 (SD-v1.5) [20] as the foundational model for our exploration in the text-to-image domain. The SD architecture is composed of three primary components: text encoder, UNet, and VAE decoder. Specifically, the denoising UNet is the most computationally intensive component in SD and requires numerous iterative forwarding steps to maintain high generative quality. For example, in SD-v1.5, the total number of denoising timesteps required for one inference is 50, which takes about 3.5 hours to run on an embedded ARM Cortex-A53 CPU. Therefore, our primary goal is to accelerate UNet for deployment on edge devices, especially on low power edge FPGAs.

Fig. 1 presents an overview of the UNet architecture in SD-v1.5. The core of UNet is its encoder-decoder structure. The encoder progressively downsamples the input, capturing information at different scales and abstracting the high-level features. The decoder then upsamples this information, reconstructing the image details. First, the UNet contains three Cross Attention Downsample Blocks (CADB) of different sizes. Immediately followed by a Downsample Block (DB), a Cross Attention Middle Block (CAMB), and an Upsample Block (UB). Finally, the UNet also contains three Cross Attention Upsample Blocks (CAUB) of different sizes. Each block within the architecture is further composed of either Transformer blocks, ResNet blocks, or a combination of both.

The ResNet block is composed of convolution operators (CONV3/CONV1), a fully connected layer (1D FC), GroupNorm (GNorm), and SiLU operators. The more complex Transformer block comprises convolution (CONV1) operators, attention operators, GNorm, LayerNorm(LNorm), and GeGLU [18] operators. Table I lists key variables and their explanations relating to the UNet structure.

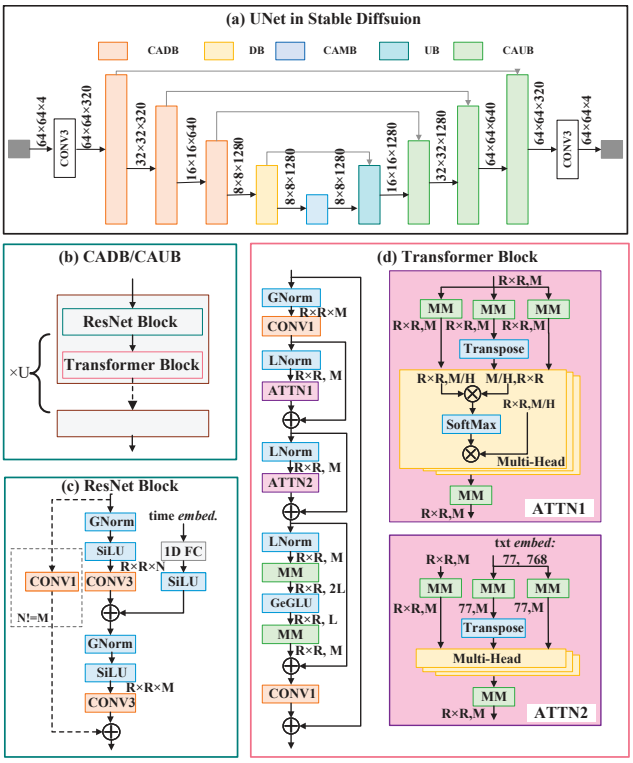

Fig. 1: UNet structure in SD. Variables explained in Table I.

TABLE I: Variables in UNet (Fig. 1) and their explanation

Operator	Variable	Explanation
Convolution (ResNet/Transformer)	R	Row/col size
	N/M	Number of input/output channels
	G	Number of channels per group for GNorm
Attention (Transformer)	R×R	Row size of input matrix in attention
	M	Col size of input matrix in attention
	H	Number of heads in attention
	L	Col size of output matrix for GeGLU

B. Our Low-Bit Quantization for Stable Diffusion

To reduce computational demands and model size of the SD model, quantization is considered.

1) Prior Quantization Studies and Limitations: Quantization is categorized into two types: post-training quantization (PTQ) and quantization-aware training (QAT). While previous work [21]–[23] applied PTQ to diffusion model quantization, PTQ often led to accuracy losses and seldom yielded stable diffusion results. On the other hand, QAT, as employed by TDQ [24] for diffusion model quantization, similarly did not demonstrate stable diffusion outcomes. A recent approach, EfficientDM [25], implemented QAT on SD quantization, achieving an accuracy comparable to those of floating-point models. However, EfficientDM was limited to weight quantization, maintaining floating-point activation in SD. These prior works [21]–[25] noted a key challenge in SD model quantization: the activation value range varies during the denoising process, rendering a single scaling factor inadequate for covering the activation range in each denoising step.

TABLE II: FID and CLIP score of our low-bit SD-v1.5

Method	Weight Width	Activation Width	FID ↓	CLIP ↑	GOPs	Param (MB)
SD-v1.5	32-bit	32-bit	26.63	0.3055	729.4	3,115
SD-v1.5 (QAT)	4-bit	8-bit	26.71	0.3051	854.8*	389

*We transform the 8-bit by 8-bit multiplication in attention to two 4-bit by 8-bit multiplications.

2) Activation Scale-Aware W4A8 Quantization: To address the prior limitation and enhance the accuracy of the quantized SD model, we utilized QAT and applied distinct scaling factors at each denoising step. Please note that in selecting the bit-width, we found that for weight representation, we can reduce it to 4-bit without degrading accuracy. However, activation is more sensitive than weight, and the value range varies during the iterative denoising process. Therefore, we need to use 8-bit representation for activation and apply distinct scaling factors at each denoising step to preserve accuracy.

We perform zero-shot evaluation on MS COCO dataset with 6000 randomly sampled prompts, and set classifier-free guidance scale as 7.5 to generate the images. We applied the Fréchet inception distance (FID) score to evaluate the quality of generated images. A lower FID score suggests that the distributions of generated images are more similar to those of real images, indicating better quality and higher similarity. Additionally, we used the contrastive language–image pre-training (CLIP) score to assess how well the content of the generated image aligns with the text prompt's description. A higher CLIP score denotes a better match, implying closer alignment between the image and the textual description.

We compare the quantitative accuracy of our quantized model with the baseline floating-point model in Table II. Specifically, we quantize the model weight to 4 bits and activation to 8 bits (W4A8). Shown in Table II, the low-bit SD model achieves comparable FID and CLIP scores to the original floating-point model while compressing the parameter size by 8 times. It is important to note that the lower-bit model exhibits higher GOPs due to our conversion of 8-bit activation multiplied by 8-bit activation (i.e., $Q \times K^T$ and $QK^T \times V$) in the attention of Transformer block into twice as many operations of 4-bit by 8-bit for uniform representation.

III. Low-Bit Stable Diffusion Accelerator Design

A. Overall SDA Architecture and Design Novelties

Based on our quantized low-bit SD model in Section II, we design a high-performance stable diffusion accelerator (SDA) on the embedded AMD-Xilinx ARM-FPGA SoC, whose overall architecture is shown in Fig. 2. Its key component is the SD core on the FPGA, which accelerates all the operators in SD UNet on the FPGA to avoid the performance limitation imposed by Amdahl's law. The SD core includes: 1) a hybrid systolic array (hybridSA) to accelerate the ***main operators***, convolution in all blocks and matrix multiplication in attention blocks, with different quantization bit-widths (e.g., $W4A8$ and all 8-bit QK^TV in attention), 2) a special function unit (SFU) to accelerate all nonlinear operators, plus the linear shortcut add and transpose operators (we denote all these operators in SFU as ***nonlinear operators*** for the simplicity of writing),

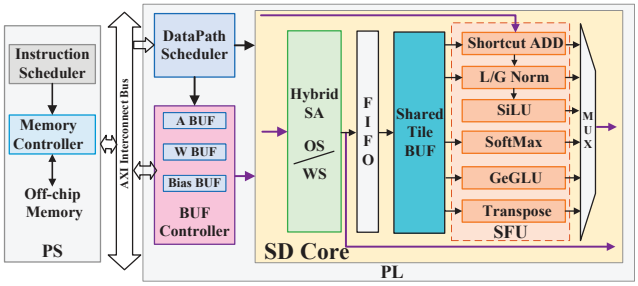

Fig. 2: Overall architecture of SDA.

and 3) a shared tile buffer that is effectively shared by all nonlinear operators to communicate with hybridSA. Note that hybridSA takes 4-bit weights and 8-bit activations as inputs, and we decompose one 8-bit by 8-bit multiplication into two 4-bit by 8-bit multiplications to be computed by hybridSA. SFU takes 16-bit fixed-point data as inputs. The quantization and dequantization units that convert data precisions between hybridSA and SFU are omitted in Fig. 2.

To execute different combinations of main operators and nonlinear operators in the SD model, the ARM CPU sends scheduling instructions to configure the datapath scheduler on the FPGA via the AXI bus that connects to the shared off-chip memory. The datapath scheduler subsequently directs the SD core to execute using the corresponding dataflow mode of hybridSA and nonlinear operators in the SFU. At the same time, the datapath scheduler also instructs the buffer controller to manage the corresponding on-chip buffer reads and writes.

Novel hybridSA design. HybridSA exploits a hybrid systolic array design with flexible switching between output stationary (OS) and weight stationary (WS) dataflows to support native execution of various main operators in convolution and attention with different quantization bit-widths. It also generates output sequences friendly for the dataflow processing of subsequent nonlinear operators. Moreover, hybridSA employs two efficient DSP packing techniques inside each PE (processing element) tailored for each dataflow mode to improve the DSP utilization; on average, each DSP can pack 4 W4A8 operations in convolution and 2 W4A8 operations in matrix multiplication.

Novel resource-efficient nonlinear units (SFU) design and integration. In SDA, we accelerate all nonlinear operators inside the SFU and carefully dataflow their execution with hybridSA. Each nonlinear operator exploits the fine-grained pipeline to reduce its latency. All nonlinear operators share the same tile buffer to dataflow with hybridSA, so as to reduce on-chip memory usage. Moreover, to minimize the resource utilization by nonlinear operators, the parallelism degree inside nonlinear operators is minimized to merely match the hybridSA speed in the coarse-grained pipeline.

B. HybirdSA for Low-bit Convolution and Attention

1) Design Challenges: The design of the hybridSA architecture needs to meet the following requirements:

#1 Native support for various main operators with different bit-widths: Our analysis indicates that convolution

operators and matrix multiplication operators in the SD UNet contribute approximately equally to the total model GOPs. Moreover, despite adopting W4A8 quantization for the SD model, attention modules (ATTN1 and ATTN2 in Fig. 1) in the Transformer block still require 8-bit by 8-bit multiplications (e.g., $Q \times K^T$ and $QK^T \times V$), which account for 29.5% of the total GOPs in the low-bit SD model.

This requires a unified architecture to efficiently support the native execution of those main operators. While prior studies [14], [16], [26]–[28] have extensively explored systolic array (SA) architectures to support the execution of convolution or matrix multiplication, they only support one of them in native hardware execution and pay extra overhead to convert the other operator into the one with native hardware support.

#2 High computation efficiency with effective DSP packing: To improve the computation efficiency for low-bit matrix multiplication (MM) and convolution, it is natural to expect that the design of each processing element (PE) can maximize the DSP utilization through appropriate low-bit DSP packing methods. While a DSP can easily pack two (and at most two) W4A8 multiplications in MM [29], recent studies [30]–[32] demonstrate that for W4A4 convolution, a DSP can pack up to 6 multiplications and 2 additions (i.e., 8 W4A4 operations in total). Therefore, our goal is to pack 4 W4A8 operations for convolution and 2 W4A8 operations for MM, which poses further challenges to integrate two different DSP packing mechanisms into one unified hybridSA design.

#3 Flexible dataflow switching to generate friendly output sequences for subsequent nonlinear units: Shown in Fig. 1, after the MM operator, SoftMax, LNorm, and GeGLU require the hybridSA to output a complete row of data ($\mathbb{Z}^{1\times R\times R}$ for Eq. (1), $\mathbb{Z}^{1\times M}$ for Eq. (2), and $\mathbb{Z}^{1\times 2L}$ for Eq. (4)) as quickly as possible; note that we implement the CONV1 operator as MM as well. This prefers an output-stationary (OS) dataflow design for hybridSA, so that hybridSA can directly stream its outputs onto those nonlinear operators instead of writing and then reading the off-chip memory. On the other hand, following convolution (CONV3) in the ResNet block, GNorm requires the hybridSA to output data for the first G channels ($\mathbb{Z}^{R\times R\times G}$ for Eq. (2)) as soon as possible. This prefers a weight-stationary (WS) dataflow design for hybridSA. As a result, hybridSA needs to support the native execution and flexible switching of MM-OS and CONV-WS dataflows.

2) HybridSA with MM-OS and CONV-WS Dataflows: To meet the above design requirements, we design our hybridSA architecture as shown in Fig. 3. It has $X \times Y$ processing elements (PEs), which natively support the execution of MM-OS and CONV-WS dataflows to maximize the sharing of scarce on-chip resources. Each PE takes a 4-bit weight and 8-bit activation (W4A8) as inputs; and the 8-bit by 8-bit multiplication is decomposed into two 4-bit by 8-bit multiplications with an extra shift and addition. Inside each PE, it further decomposes the 8-bit activation into two 4-bit segments, so that it can apply the more effective 4-bit (W4A4) DSP packing [30], [31] to improve the computation efficiency.

MM-OS dataflow: For the matrix multiplication (MM) inside

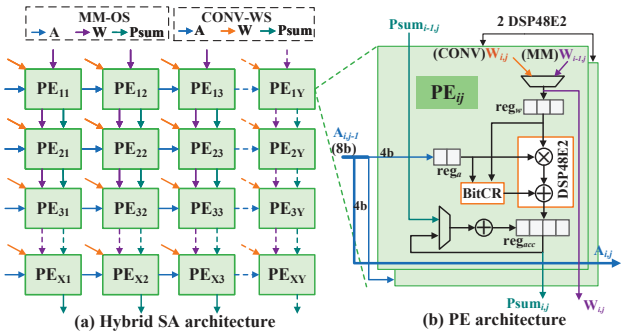

Fig. 3: Overview of $X \times Y$ size hybridSA design supporting hybrid output-stationary dataflow for matrix-multiplication (MM-OS) and weight-stationary dataflow for convolution (CONV-WS), and its PE design with 4-bit DSP-packing.

the Transformer blocks, hybridSA operates in the MM-OS mode to generate friendly output sequences for its subsequent SoftMax, LNorm, and GeGLU nonlinear operators. As shown in Fig. 3(a), 8-bit activations (A) and 4-bit weights (W) are propagated rightwards and downwards, respectively. Output results are generated internally in each PE and then pushed downwards through the array. As depicted in Fig. 4(a), hybridSA with MM-OS dataflow first generates a tile of output data along the rows of the matrix ($\mathbb{Z}^{X\times M}$) to satisfy the data consumption of the subsequent nonlinear units. Note that convolutions with a kernel size of 1 (CONV1), in the ResNet block and the Transformer block, are processed as MM to operate in the OS-based dataflow.

Shown in Fig. 3(b), each PE receives two 8-bit activations A at a time and splits each into a pair of 4-bit activations. The first two aligned 4-bit activations from two 8-bit activations are then fed into one DSP48E2, while the remaining pair of 4-bit activations are directed to another DSP48E2. Each DSP48E2 performs four multiplications [29] along with two 4-bit weights, as shown in Fig. 4(c). A cost-effective bit-width correction (BitCR in Fig. 3(b)) circuit is added to correct the sign-bit contamination in the packed multiplications [33]. In summary, one DSP performs four W4A4 multiplications, or two W4A8 multiplications in MM-OS dataflow mode.

CONV-WS dataflow: To compute convolution operators and to output friendly sequences for the subsequent GNorm nonlinear operator, hybridSA executes in CONV-WS mode. As depicted in Fig. 3(a), CONV-WS dataflow shares the activation supply logic in that of MM-OS dataflow, but each PE individually owns weights and passes partial results (Psum) downwards. Fig. 5(a) shows that hybridSA can sequentially generate a block of output data along the channels ($\mathbb{Z}^{R\times R\times G}$) required for executing the subsequent GNorm nonlinear operator. Within each PE, the same DSP48E2 can perform six 4-bit multiplications and two 4-bit additions [30]–[32] in CONV-WS mode, as shown in Fig. 5(c). That is, each DSP can perform four W4A8 operations (three multiplications and one addition) in CONV-WS dataflow mode.

MM-OS and CONV-WS dataflow switching: Lastly, shown in Fig. 4(b) and Fig. 5(b), we reorganize the output data layout

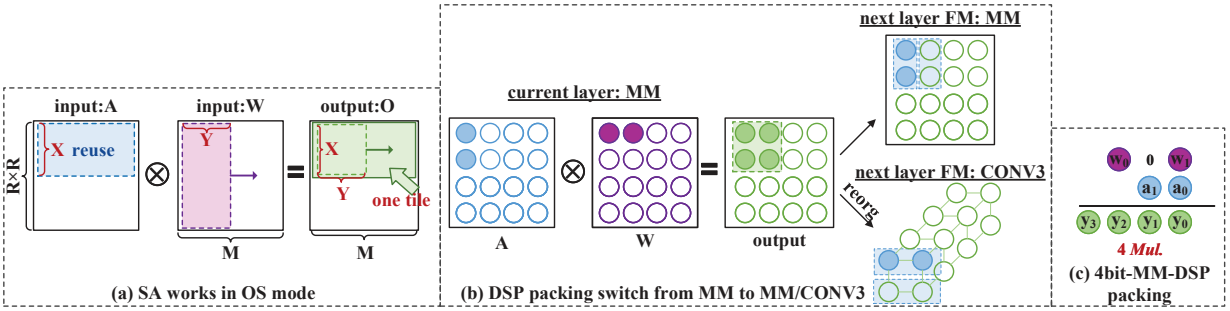

Fig. 4: Workflow and DSP packing method when hybridSA works in output-stationary (OS) dataflow mode for matrix multiplication (MM) in low-bit attention operators. The systolic array (SA) has $X \times Y$ PEs.

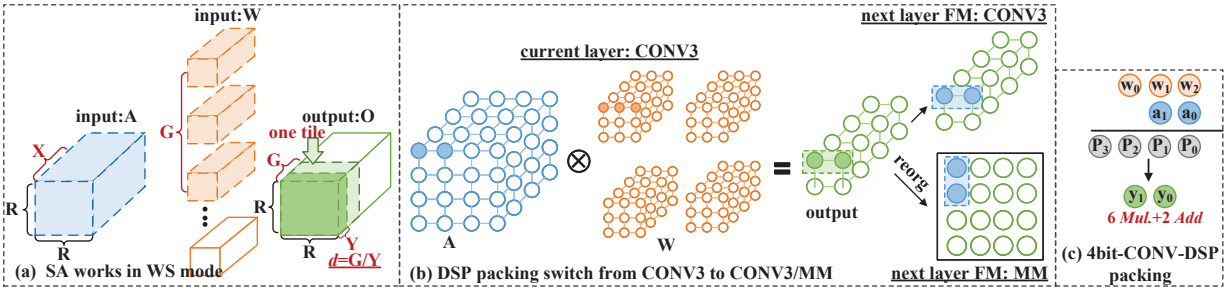

Fig. 5: Workflow and DSP packing method when hybridSA works in weight-stationary (WS) mode for low-bit convolution. The systolic array (SA) has $X \times Y$ PEs.

from the packed DSP computations on the fly, so that we can ensure consistent data packing when transitioning between MM-OS and CONV-WS dataflows with different DSP packing techniques, without any additional conversion overhead.

C. Efficient Nonlinear Units and Integration with HybridSA

1) Overall Design Principles: The complex SD UNet model structure, shown in Fig. 1, incorporates a variety of nonlinear operators, such as SoftMax, LNorm, GNorm, SiLU, GeGLU, and linear shortcut add and transpose. As will be presented in Fig. 11 in Section IV-C, after the acceleration of main operators, these nonlinear operators become the new performance bottleneck and need to be offloaded onto the FPGA as well. A straightforward implementation of all operators—with the same parallelism degree as the main SA design (e.g., [14] did it this way for SoftMax) and a separate communication channel between hybridSA for each operator—will soon consume all the FPGA resources while for the majority of the time, the nonlinear units remain idle.

To reduce the on-chip memory usage, we design a shared tile buffer to be shared by all nonlinear units to communicate with hybridSA. To minimize the resource utilization by nonlinear operators while not hurting the overall performance, we employ a two-level pipeline design. First, we pipeline the execution inside each nonlinear operator in a fine granularity. Second, we design a coarse-grained pipeline between the nonlinear operators and hybridSA to hide their execution latency; the parallelism degree inside nonlinear operators can be minimized to merely match the hybridSA speed.

Since the support of shortcut add and transpose units are relatively straightforward given the shared tile buffer, next we describe more implementation details for other nonlinear units.

2) SoftMax Unit: The SoftMax operator [12] in the ATTN1 and ATTN2 of the Transformer block is calculated as:

$$SoftMax(x_i) = e^{x_i - x_{\max}} / \sum_{j=1}^{R \times R} e^{x_j - x_{\max}} \quad (1)$$

where $x_i \in \mathbb{R}^{1 \times R \times R}$. SoftMax necessitates three sequential row-wise data accesses, involving the identification of the maximum value (MAX), exponentiation and summation (SUB-EXP-ACC), and the normalization division (DIV).

Fig. 6(a) illustrates the detailed hardware structure of the SoftMax unit, composed of a coarse-grained tile-level pipeline and a fine-grained row-level pipeline. In the coarse-grained tile-level pipeline, double tile buffers are inserted between the hybridSA and SoftMax unit. The tile buffer collects and stores row-wise tile data generated by hybridSA, and facilitates the rate transition from the high-speed hybridSA (stage 1) to the low-speed SoftMax unit (stage 2).

In stage 1, the maximum value (MAX) is also calculated while storing tile data. Since SoftMax's input data are from the matrix multiplication result of all 8-bit QK^TV, the two 4b×8b outputs from hybridSA need to be left-shifted and added (ADD in stage 1). For stage 2, double row buffers are introduced to pipeline exponentiation summation (SUB-EXP-ACC in stage 2-1) and normalization division (DIV in stage 2-2).

Fig. 7 depicts the scheduling view between hybridSA and SoftMax unit. Row-level pipeline implements time overlap of exponentiation and summation, and normalization division; while tile-level pipeline overlaps the entire SoftMax computation with hybridSA calculation (operating in MM-OS mode). Consequently, the parallelism setting for the SoftMax unit only needs to ensure that the computation is completed within the

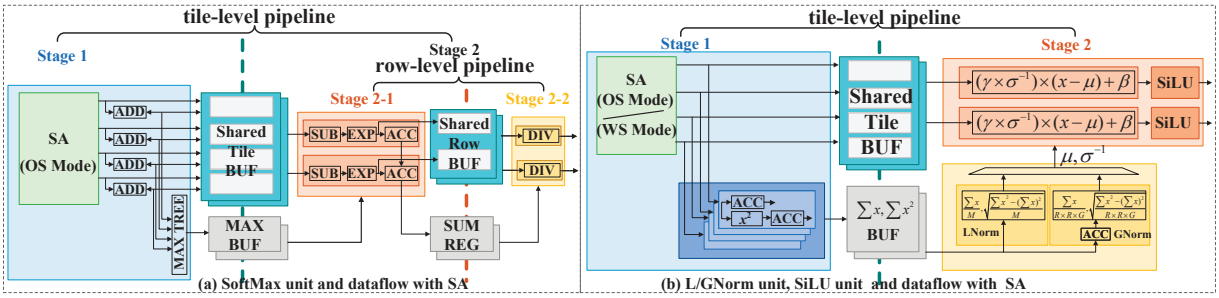

Fig. 6: Hardware structure of SoftMax unit (a), L/GNorm unit and SiLU unit (b), and their dataflow with hybridSA.

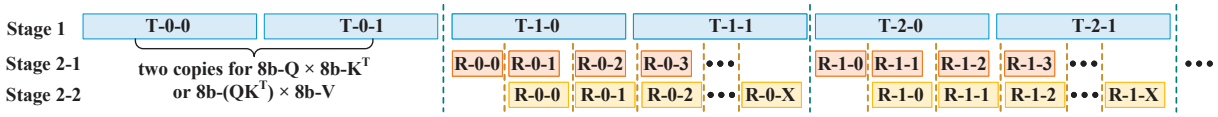

Fig. 7: Scheduling view between MM-OS-mode hybridSA and SoftMax unit.

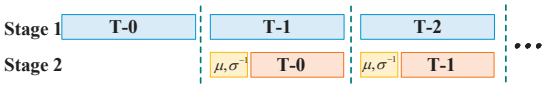

Fig. 8: Scheduling view between hybridSA and L/GNorm unit.

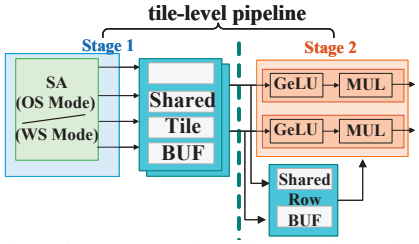

Fig. 9: Design of GeGLU unit and its dataflow with hybridSA.

time hybridSA generates two 4b×8b tiles, thereby reducing on-chip resource consumption.

3) L/GNorm Unit and SiLU Unit: LNorm and GNorm operators [17] share the following identical computations:

$$L/GNorm(x_i) = \gamma_i \times (x_i - \mu)/\sigma + \beta_i \quad (2)$$

where $x_i \in \mathbb{Z}^{1\times M}$ for LNorm, $x_i \in \mathbb{Z}^{R\times R\times G}$ for GNorm. They require three sequential accesses to input data to compute the mean (μ), standard deviation (σ), and normalization based on pre-trained affine transform parameters (γ and β).

Based on their common computational characteristics, we design a tile-level pipeline as depicted in Fig. 6(b), where LNorm and GNorm share most of the logic. We adopt integer quantized norm proposed in [34], which iteratively computes both the mean and mean-square value [35] (stage 1). Additionally, to accommodate different accumulation requirements for μ and σ in LNorm and GNorm, we added extra accumulation (ACC) for GNorm in stage 2. Specifically, for matrix tile data ($\mathbb{Z}^{X\times M}$) generated by hybridSA in MM-OS mode, LNorm directly obtains X means and variances without additional accumulation. For convolutional tile data ($\mathbb{Z}^{R\times R\times G}$) generated by hybridSA in CONV-WS mode, GNorm enables the accumulator to obtain one mean and variance.

Shown in Fig. 8, the parallelism setting of the L/GNorm unit only needs to ensure that the current tile calculation is completed before the next tile data arrives. In addition, the SiLU unit following GNorm in the ResNet block has the same parallelism factor with GNorm and executes hardware-friendly pixel-wise linear hard approximation calculations [36]:

$$SiLU(x_i) \approx x_i \times ReLU6(x_i + 3)/6 \quad (3)$$

4) GeGLU Unit: The GeGLU operator [18] in the Transformer block needs one row of data to compute as follows:

$$GeGLU(x_i) = x_i \times GeLU(x_{L+i}) \quad (4)$$

$$GeLU(x_{L+i}) \approx x_{L+i} \times ReLU6(1.702x_{L+i} + 3)/6 \quad (5)$$

where $x_i \in \mathbb{R}^{1\times L}$.

Shown in Fig. 9, GeGLU has a similar tile-level pipeline, where piece-wise linear hard approximation [36] for GeLU is executed in stage 2. The shared row buffer used in SoftMax (Fig. 6) is reused to store the first half of a row ($x_i \in \mathbb{R}^{1\times L}$) and feed data for multiplication (MUL) performed after GeLU.

IV. Evaluation

A. Experimental setup

1) Training Setup: We utilize the diffusers library as the foundation for our code and employed Quantization-Aware Training (QAT) on the SD-v1.5 model using publicly available datasets [37], [38]. In addition to 4-bit weight and 8-bit activation quantization, other bias parameters adopt a 16-bit fixed-point format. We document the quantitative outcomes in terms of FID and CLIP scores on the MS-COCO 2014 validation set [39] for zero-shot evaluation. The majority of our training utilized 16 nodes, each equipped with 8 NVIDIA A100 GPUs boasting either 40GB or 80GB of memory. We opted for the AdamW [40] optimizer with a weight decay setting of 0.01 and established a training batch size of 2,048.

2) Hardware Platform: Our proposed SDA is evaluated on the AMD-Xilinx-ZCU102 ARM-FPGA SoC board with a high-speed 4GB DDR4 SODIMM. This board integrates an ARM Cortex-A53 CPU and a ZU9EG FPGA, which has 2,520 DSPs, 912 BRAMs, and 274.1K LUTs. We utilize Vitis HLS and Vivado to implement our SDA design, which runs on board at a clock frequency of 250MHz. The resource utilization is collected from post place-and-route reports and the power is measured using a power meter. Our CPU baseline is the original SD model with FP32 precision running on the ARM CPU as it does not support low-bit precision.

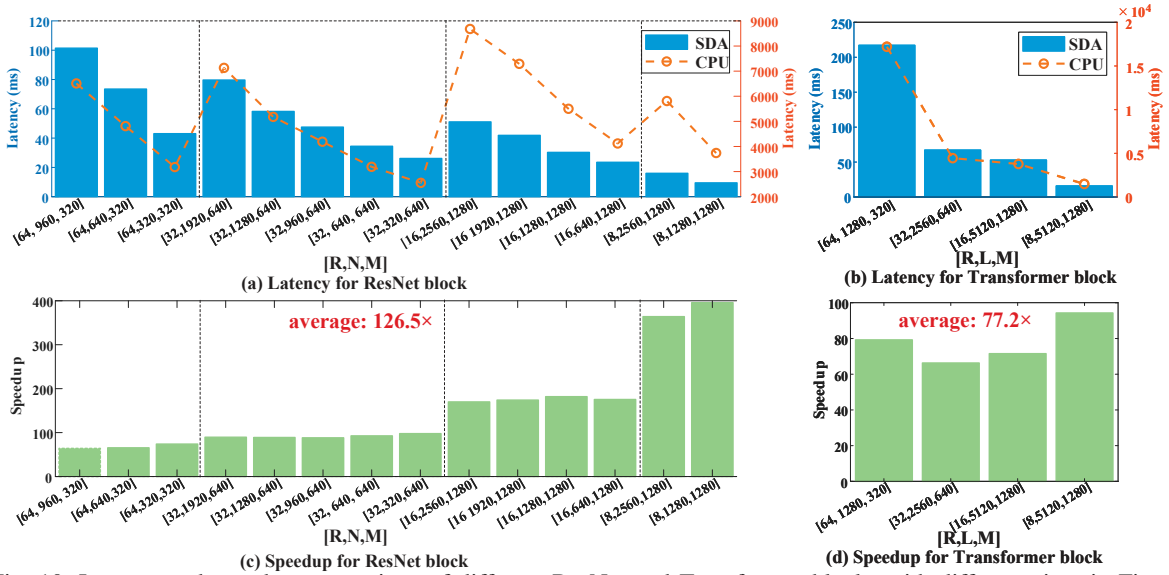

Fig. 10: Latency and speedup comparison of different ResNet and Transformer blocks with different sizes in Fig. 1.

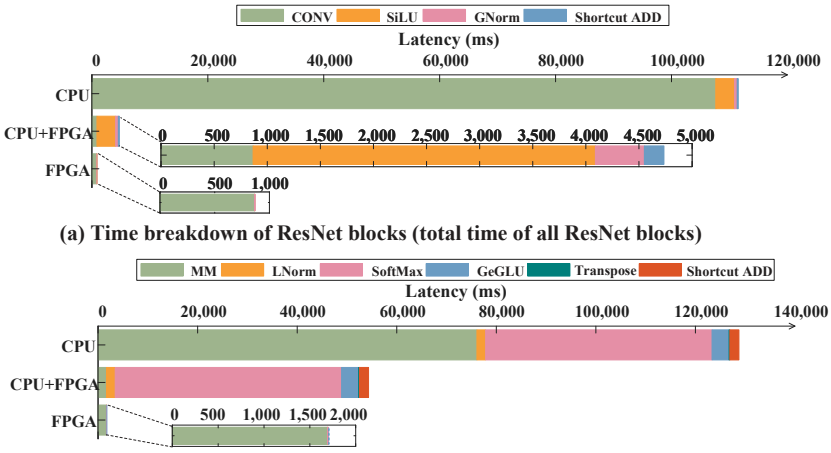

Fig. 11: Execution time breakdown before/after acceleration.

TABLE III: Performance and energy-efficiency comparison

Platform	Performance (iter/s)	Power (W)	Energy (iter/s/W)	GOPS
CPU	0.004	2.871	0.0014	3
SDA	0.389	9.977	0.039	332.5
GPU	0.386	7.5	0.052	281.5
	0.489	15	0.033	356.7

B. Overall Performance

Shown in Table III, compared to the embedded ARM CPU, our SDA achieves a 97.3× speedup (measured in SD UNet iterations processed per second) and a 27.9× energy efficiency improvement. The overall throughput of the SDA is 332.5 GOPS. Due to the adoption of different DSP packing methods, CONV3 and MM/CONV1 operators in the low-bit SD model achieve a throughput of 400.6 GOPS and 301.3 GOPS, respectively. Fig. 10 represents the speedup of ResNet and Transformer blocks of various sizes. Our SDA demonstrates an average speedup of 126.5× and 77.2× for ResNet and Transformer blocks over CPU, respectively.

In addition, we also run SD UNet on Jetson TX2 embedded GPU. At 7.5W, it achieves 0.386 iter/s and 0.052 iter/s/W. At 15W, it achieves 0.489 iter/s and 0.033 iter/s/W. Our SDA achieves 0.389 iter/s and 0.039 iter/s/W, which is comparable to the TX2 performance running at 7.5W and slightly outperforms TX2 in performance/watt running at 15W.

C. Performance Breakdown

Fig. 11 (a) and (b) break down the execution time for main and nonlinear operators in ResNet and Transformer blocks before and after acceleration. Specifically, when all operators are executed on the ARM CPU, CONV in the ResNet block and MM in the Transformer block occupy 96.5% and 59.1% of the overall time, becoming the primary latency bottlenecks. Subsequently, after accelerating these main operators with our proposed hybridSA, the time percentage of CONV and MM decreases to 18.3% and 3.1% of the updated overall time, respectively. At this point, nonlinear operators—especially

TABLE IV: Resource utilization of our SDA accelerator

Component	DSP	LUT	BRAM
HybridSA	485(19.2%)	140,397(51.2%)	333(36.5%)
Nonlinear Units	211(8.4%)	46,562(17.0%)	342.5(37.6%)
Other Logic	110(4.4%)	33,737(12.3%)	137.5(15.0%)
Total	806(32.0%)	220,696(80.5%)	813(89.1%)

SiLU and GNorm in the ResNet block, SoftMax and GeGLU in the Transformer block—executing on the CPU become the new bottleneck. Due to Amdahl's law, it does not help much even if we increase the spatial size of hybridSA to further reduce the latency of main operators. Instead, we should allocate hardware resources now to accelerate the nonlinear operators. Indeed, our SDA design eliminates the new bottleneck imposed by nonlinear operators, which are accelerated on the FPGA and dataflowed with our hybridSA to well hide their latencies.

D. Resource Utilization

Table IV illustrates the resource utilization of the proposed SDA. We implement the proposed hybridSA with a spatial size of 20×10, and configure the parallelism factor for SoftMax, L/GNorm, SiLU and GeGLU to 5. HybridSA consumes the majority of the resources as planned and the entire design is bottlenecked by LUT and BRAM usages. It only uses 32% of the DSP resources, due to the efficient DSP packing in hybridSA. Meanwhile, DSP packing comes at a cost of increased LUT usage, and it is nontrivial to further increase the spatial size of hybridSA. Moreover, as presented in Section IV-C, it is more important to allocate some hardware resources to accelerate the nonlinear operators (the new bottleneck after the main operator acceleration) and hide their execution latencies to achieve the best overall performance.

V. Related Work

A. FPGA-based Systolic Array Designs

The systolic array (SA) architecture, including output-stationary (OS), weight-stationary (WS), and input-stationary (IS) dataflows, is widely adopted to accelerate DNN inference, since its simple PE array design with local neighboring communication can be easily scaled up with a high clock frequency [14], [16], [26]–[28], [41]. Table V summarizes the differences between our SDA and prior SA studies.

First, existing SA architectures are designed and optimized only for a single main operator, either matrix multiplication or convolution, and incurs extra overhead to convert the other main operator into the one with native hardware support. In contrast, our SDA accelerator supports the native execution of both main operators in the same hybridSA hardware in the MM-OS and CONV-WS dataflow modes, respectively.

Second, most existing SA architectures only support a subset of nonlinear operators involved in CNN or Transformer-based models. In contrast, our SDA accelerator accelerates a variety of nonlinear operators, including SoftMax, L/GNorm, SiLU and GeGLU, and carefully dataflows their execution with hybridSA with minimal resource utilization.

Third, most existing SA architectures do not well support DSP packing, especially the latest 4-bit DSP packing for convolutions. In contrast, our SDA accelerator well integrates state-of-the-art DSP packing techniques [29]–[32] to improve computation efficiency and carefully tunes the dataflow switching to avoid data repacking overhead.

TABLE V: Comparison with prior FPGA-based SA studies

Prior Works	Dataflow		Suppoted Operators							Precison	DSP Packing
			Native Main Operator		Nonlinear Operator						
	OS	WS	CONV	MM	SoftMax	LNorm	GNorm	SiLU	GeGLU		
[26]	✔	✗	✗	✔	✔	✔	✗	✗	✗	W4A8	✔
[27]	✔	✗	✗	✔	✔	✔	✗	✗	✗	W8A8	✗
[28]	✔	✗	✔	✗	✗	✗	✗	✗	✗	W8A8 W16A16	✗
[14]	✔	✔	✗	✔	✔	✗	✗	✗	✗	-	✗
[16]	✔	✔	✔	✗	✗	✗	✗	✗	✗	W8A8	✔
SDA	✔	✔	✔	✔	✔	✔	✔	✔	✔	W4A8	✔

B. FPGA-based Accelerator for Large Language Models

More recently, a growing body of work has demonstrated the potential of FPGAs in accelerating the emerging large language model (LLM) inference [15], [26], [42]. The latest FlightLLM [15] proposed a complete mapping flow that integrates configurable sparse DSP chain, always-on-chip decoding, and length adaptive compilation for LLMs, which achieved 6× higher energy efficiency and 1.8× better cost efficiency against Nvidia V100 on the LLaMA-7B model. However, existing FPGA-based accelerators for LLMs mainly focus on accelerating language model inference based on Transformers, and all of them are deployed on cloud FPGAs. Different to previous work, our SDA is the first to explore the potential of edge FPGAs to accelerate large low-bit stable diffusion model and support both the attention-based Transformer blocks and the convolution-based ResNet blocks.

VI. Conclusion and Future Work

In this paper, we have presented SDA, the first efficient low-bit stable diffusion inference accelerator on edge FPGAs. Our SDA accelerator proposes a high-performance hybridSA architecture that supports native execution of convolution and attention blocks with different quantization bit-widths, with effective DSP packing techniques. Moreover, it implements various resource-efficient nonlinear units, and efficiently integrates them with hybridSA—which supports flexible dataflow switching to generate friendly outputs for subsequent nonlinear processing—to minimize their resource utilization while well hiding their execution latencies via two-level pipelining. Compared to the ARM CPU, our SDA achieves a speedup of 97.3× while maintaining comparable FID and CLIP scores on the modern SD-v1.5 model. In future, we will focus on more microarchitecture optimizations to achieve more competitive performance compared to embedded GPUs and extend SDA to support more applications and platforms.

Acknowledgements

This work was supported in part by NSERC Discovery Grant RGPIN-2019-04613, DGECR-2019-00120, Alliance Grant ALLRP-552042-2020; CFI John R. Evans Leaders Fund and BC Knowledge Development Fund; NSF Award CNS-1909172 and IIS-2310254.

REFERENCES

[1] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3836–3847.

[2] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8821–8831.

[3] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.

[4] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen, "GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models," in *Proceedings of the 39th International Conference on Machine Learning*, 2022, pp. 16 784–16 804.

[5] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet *et al.*, "Imagen video: High definition video generation with diffusion models," *arXiv preprint arXiv:2210.02303*, 2022.

[6] U. Singer, A. Polyak, T. Hayes, X. Yin, J. An, S. Zhang, Q. Hu, H. Yang, O. Ashual, O. Gafni, D. Parikh, S. Gupta, and Y. Taigman, "Make-a-video: Text-to-video generation without text-video data," in *International Conference on Learning Representations (ICLR)*, 2023.

[7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 684–10 695.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[9] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International conference on machine learning*. PMLR, 2014, pp. 1278–1286.

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 2015, pp. 234–241.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[14] Z. Wang, G. Wang, H. Jiang, N. Xu, and G. He, "Cosa: Co-operative systolic arrays for multi-head attention mechanism in neural network using hybrid data reuse and fusion methodologies," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[15] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, Y. Dai, J. Li, Z. Wang, R. Zhang, K. Wen, X. Ning, and Y. Wang, "Flightllm: Efficient large language model inference with a complete mapping flow on fpga," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024.

[16] X. Cai, Y. Wang, X. Ma, Y. Han, and L. Zhang, "Deepburning-seg: Generating dnn accelerators of segment-grained pipeline architecture," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1396–1413.

[17] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[18] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[19] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural networks*, vol. 107, pp. 3–11, 2018.

[20] Hugging Face, "Stable diffusion v1-5 model." [Online]. Available: https://huggingface.co/runwayml/stable-diffusion-v1-5

[21] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-training quantization on diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 1972–1981.

[22] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "Ptqd: Accurate post-training quantization for diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[23] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-diffusion: Quantizing diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 535–17 545.

[24] J. So, J. Lee, D. Ahn, H. Kim, and E. Park, "Temporal dynamic quantization for diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[25] Y. He, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "EfficientDM: Efficient quantization-aware fine-tuning of low-bit diffusion models," in *International Conference on Learning Representations (ICLR)*, 2024.

[26] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *arXiv preprint arXiv:2312.15159*, 2023.

[27] Y. Chen, T. Li, X. Chen, Z. Cai, and T. Su, "High-frequency systolic array-based transformer accelerator on field programmable gate arrays," *Electronics*, vol. 12, no. 4, p. 822, 2023.

[28] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2019.

[29] T. Han, T. Zhang, D. Li, G. Liu, L. Tian, D. Xie, and Y. S. Shan, "Convolutional neural network with int4 optimization on xilinx devices," *Xilinx White Paper, WP521*, 2020.

[30] J. Sommer, M. A. Özkan, O. Keszocze, and J. Teich, "Dsp-packing: Squeezing low-precision arithmetic into fpga dsp blocks," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 160–166.

[31] J. Zhang, M. Zhang, X. Cao, and G. Li, "Uint-packing: Multiply your dnn accelerator performance via unsigned integer dsp packing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[32] X. Liu, Y. Chen, P. Ganesh, J. Pan, J. Xiong, and D. Chen, "Hikonv: High throughput quantized convolution with novel bit-wise management and computation," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, p. 140–146.

[33] S. Grys, "Signed multiplication technique by means of unsigned multiply instruction," *Computers & Electrical Engineering*, vol. 37, no. 6, pp. 1212–1221, 2011.

[34] P. Dong, L. Lu, C. Wu, C. Lyu, G. Yuan, H. Tang, and Y. Wang, "Packqvit: Faster sub-8-bit vision transformers via full and packed quantization on the mobile," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[35] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[36] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[37] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 278–25 294, 2022.

[38] M. Byeon, B. Park, H. Kim, S. Lee, W. Baek, and S. Kim, "Coyo-700m: Image-text pair dataset," *Coyo-700m: Image-text pair dataset*, 2022.

[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[40] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[41] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[42] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 616–630.