An Energy-Efficient Accelerator for Medical Image Reconstruction From Implicit Neural Representation

Chaolin Rao, *Graduate Student Member, IEEE*, Qing Wu, Pingqiang Zhou, *Member, IEEE*, Jingyi Yu, *Fellow, IEEE*, Yuyao Zhang, *Member, IEEE*, and Xin Lou, *Senior Member, IEEE*

***Abstract*— This work presents an energy-efficient accelerator for medical image reconstruction from implicit neural representation (INR). The accelerator implements an INR-based algorithm to deliver high-quality medical image reconstruction with arbitrary resolution from a compact implicit format. In particular, we propose a dedicated hardware architecture based on an optimized computation flow for the INR-based reconstruction algorithm, which co-designs data reuse and computation load. The proposed architecture takes in the coordinate of the intersection of three scans and outputs all the voxel intensities, minimizing the data movement between on-chip and off-chip. To validate the proposed accelerator, we build a proof-of-concept prototype demonstration system using field programmable gate array (FPGA). We also map our design to 40nm CMOS technology to measure the performance of the proposed accelerator. The implementation results show that, running at 400MHz, the proposed accelerator is capable of processing medical images with 256 × 256 resolution in real-time at 26.3 frames per second (FPS), with a power consumption of only 795 mW. Comparison results show that the performance, as well as the energy efficiency of the proposed accelerator, outperforms the central processing unit (CPU)-based and graphic processing unit (GPU)-based implementations.**

***Index Terms*— Medical image reconstruction, implicit neural representation (INR), accelerator, energy-efficient.**

I. Introduction

THE transmission and display of medical images via portable terminals have always been a challenging problem. For presenting human anatomical structures, medical images are typically reconstructed and stored in three-dimension (3D) such as Computed Tomography (CT), or two-dimensional stacks such as Magnetic Resonance Image (MRI) and Positron Emission Tomography (PET). With the fast development of imaging hardware and reconstruction technique, higher resolution of medical images is achieved and support more effective diagnosis of physicians, while high-quality image also requires more transmission and display resources. Data compression is the generalized term for representing data in a compact form. The existing common medical image compression standard [1], [2] is based on discrete voxel representation due to the digital computation and storage system, which results in a severe trade-off problem between data accuracy and storage resources. The heavy burden of data dimension also requires more powerful hardware for 3D image display, storage and transmission. For example, most radiology departments are equipped with workstations for transferring, receiving and displaying patient medical images, as well as computer clusters with powerful storage devices for medical record maintenance. Advanced image compression and reconstruction techniques that decouple image size and data fidelity are therefore more efficient solutions than the previous works [3], [4]. With a high compression rate, the transmission is possibly to be completed wireless and received using portable terminals like tablets or even cellphones, which will highly expand application scenarios of medical images.

Recently, implicit neural representation (INR) [5], [6], [7], [8], [9] has been proposed to parametrize various signals continuously. Instead of voxel-based explicit representation, INR models an image as a continuous function that maps image spatial coordinates to the corresponding pixel intensities. Specially, the function is approximated by a fully-connected neural network, i.e., a multilayer perceptron (MLP), whose input is the spatial coordinate and output is the intensity at that location (Note that MLPs have some major advantages over convolutional neural networks (CNNs) for neural parameterization and reconstruction of signals [10], such that most of the recent works use MLP instead of CNN [5], [6], [7], [8], [9]). Finally, the MLP is optimized using stochastic descent algorithms to minimize the predicted error. Once the optimization is converged, the image is implicitly stored in the weights of the network. INR has two significant advantages: (i) The model size of an MLP is usually much smaller than the corresponding image size. For example, in this work, a 15-layers MLP with Fourier feature mapping (FFM) [7] is used to parametrize 3D MR images of $512\times512\times512$ size. The size of the original 3D MR image is 256 MB, while the size of the well-trained MLP weights is only 4.2 MB (represented in FP32). Moreover, as in most of the CNN accelerators [11], [12], [13], we can further compress the data by quantizing the algorithm; (ii) Benefiting from the continuous coordinate system, the well-trained

Manuscript received 7 May 2022; revised 24 October 2022 and 30 November 2022; accepted 16 December 2022. Date of publication 30 December 2022; date of current version 31 March 2023. This work was supported in part by the Shanghai Rising-Star Program under Grant 21QC1401400 and in part by the Central Guided Local Science and Technology Foundation of China under Grant YDZX20223100001001. This article was recommended by Associate Editor M. Martina. *(Chaolin Rao and Qing Wu contributed equally to this work). (Corresponding authors: Yuyao Zhang; Xin Lou.)*

The authors are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China (e-mail: raochl@shanghaitech.edu.cn; wuqing@shanghaitech.edu.cn; zhoupq@shanghaitech.edu.cn; yujingyi@shanghaitech.edu.cn; zhangyy8@shanghaitech.edu.cn; louxin@shanghaitech.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3231863.

Digital Object Identifier 10.1109/TCSI.2022.3231863

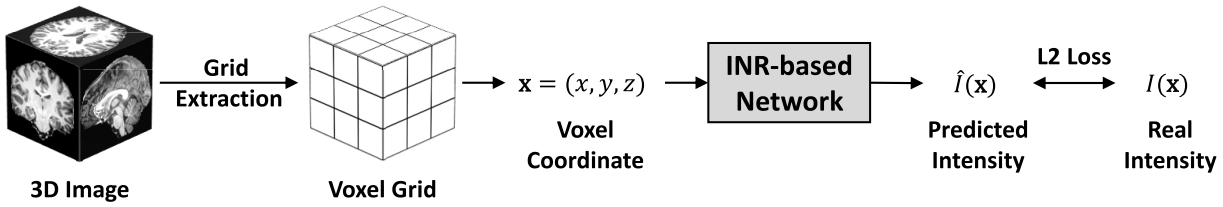


Fig. 1. Pipeline of INR-based algorithm for parameterizing 3D medical images.

model can reconstruct the compressed image response with an arbitrary resolution [14], [15], [16], [17]. Therefore, the reconstructed image resolution is theoretically not limited by the required storage resources.

To address the trade-off problem between data accuracy and storage & transmission resources, we propose an INR-based method for parametrizing medical images. he proposed algorithm differs from conventional compression methods in problem modeling. The most widely used image compression methods, e.g., PCA and DCT, map an image onto an energy-concentrated space guided by a set of orthogonal bases. For INR, we directly represent the original image using an implicit function approximated by the network, which decouples the relationship between image size and compression ratio. In particular, the proposed INR-based algorithm leverages a coordinate-based network with FFM to fit the raw image data. The parametric representation is continuous and implicit, which facilitates data compression and arbitrary resolution reconstruction. However, there are two limitations for practical deployment: (i) The proposed method is computationally demanding for medical image reconstruction. This is because it needs to query every voxel coordinate to produce the corresponding image intensity, leading to tremendous amount of network evaluation. For example, we have to conduct one million queries of the network for reconstructing a 3D image of $100 \times 100 \times 100$ size. (ii) The INR-based model in this work is MLP-based, meaning that many existing dedicated neural network accelerators are not feasible for it.

Recent works [18], [19], [20], [21], [22] on neural network accelerator mainly focus on CNN, which is widely used in many applications. Most of these works optimize data reuse [12], [18], [23] and computation flow [20], [21] for the convolutional operations. Moreover, although there are plenty of works for the long-standing matrix-matrix multiplication problem [24], [25], [26], [27], [28], most of them take advantage of the sparsity to optimize the computation. For example, SpArch [24] optimizes the data locality for both input and output matrix of sparse matrix multiplication. Subhankar [25] exploits a reconfigurable memory hierarchy to efficiently cache the sparse matrices. While in this work, the INR-based algorithm is MLP-based, and the model has to be queried many times to reconstruct one image, leading to a very different computation pattern. Besides, the sparsity of weight is very low (less than 2% based on our evaluation). Therefore, the aforementioned designs are not directly applicable to the reconstruction task in this work. Moreover, in the INR-based method, the input voxel coordinates need to be generated on the fly (to save memory access) and go through an FFM operation to enhance the learning of high-frequency information, which need extra computations.

Therefore, in this work, we further propose a dedicated accelerator for the execution of the INR-based medical image reconstruction algorithm. In particular, we propose an optimized computation flow by co-designing data reuse and computation load, aiming at balancing memory access and throughput. Based on that, a corresponding hardware architecture, which takes in the coordinate of the intersection point of three scans and outputs all the voxel intensities, is further proposed. To validate the proposed design, we build a prototype demonstration system using a field programmable gate array (FPGA). We also map our design to 40nm CMOS technology to measure the performance of the proposed accelerator. The implementation results show that, running at 400MHz, the proposed accelerator is able to process medical images with 256×256 resolution at 26.3 frames per second (FPS), with a power consumption of 795 mW. Comparison results show the proposed accelerator outperforms the central processing unit (CPU)-based and graphic processing unit (GPU)-based implementations.

The rest of the paper is organized as follows. Section II describes the INR-based medical image reconstruction algorithm. Section III analyzes the computation in the INR-based algorithm and proposes an optimized computation flow. Section IV presents the architecture of the proposed accelerator and Section V discusses the corresponding FPGA demonstration system and the application-specific integrated circuit (ASIC) implementation using standard CMOS technology. The conclusions are drawn in Section VI.

II. INR-Based Algorithm for Parameterizing Medical Images

A. Algorithm Overview

In our previous work [15], an implicit neural representation-based self-supervised learning model (IREM) is proposed to reconstruct arbitrary high-resolution (HR) isotropic 3D MR images from multiple anisotropic low-resolution (LR) MR scans with thick slices. Specifically, we modeled the observed LR image $I_{\mathbf{lr}}$ and the desired HR image $I_{\mathbf{hr}}$ by the same continuous implicit voxel function defined as below:

$$I = \mathcal{F}_\theta(\mathbf{x}) \tag{1}$$

where $\mathbf{x} = (x, y, z)$ is any 3D voxel spatial coordinate and I is the voxel intensity at the position $\mathbf{x}$ in the image. The LR

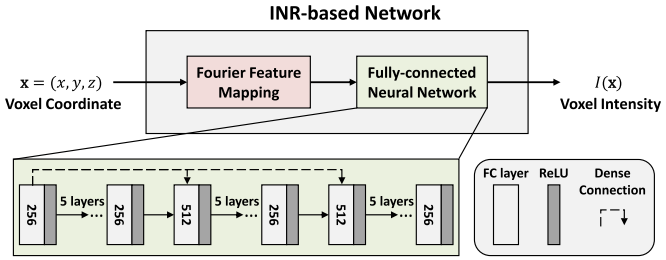


Fig. 2. Architecture of INR-based reconstruction algorithm.

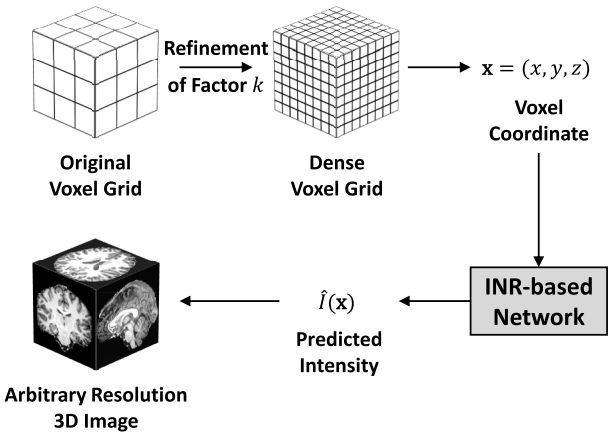


Fig. 3. The workflow of arbitrary-resolution medical image reconstruction by the well-trained INR-based network.

and HR images are considered as the explicit representation of the implicit function $\mathcal{F}_\theta(\mathbf{x})$ with different sampling rates. Then, the key idea is to approximate the function $\mathcal{F}_\theta(\mathbf{x})$ from the LR image by a fully-connected neural network with FFM. Once the continuous function is well estimated, the HR image of an arbitrary resolution can be recovered.

In this work, we extend IREM to propose an INR-based algorithm for parametrizing 3D medical images and the pipeline illustrated in Fig. 1. In particular, we first build a voxel grid according to the spatial information (origin, direction, and voxel size) of the 3D image. Then, the INR-based network $\mathcal{F}_\theta$ (θ denotes the trainable parameters of the network) takes each voxel coordinate $\mathbf{x} = (x, y, z)$ as input and predicts the corresponding voxel intensity $\hat{I}(\mathbf{x}) = \mathcal{F}_\theta(\mathbf{x})$. Finally, we optimize the network by minimizing the L2 loss function between the predicted voxel intensity $\hat{I}(\mathbf{x})$ and the real voxel intensity $I(\mathbf{x})$. Mathematically, the L2 loss is defined as below:

$$\mathcal{L}_{\mathrm{L2}}(\theta) = \sum_{i=1}^{N} \left(\hat{I}(\mathbf{x}_i) - I(\mathbf{x}_i)\right)^2 \tag{2}$$

where N is the mini-batch size during the model training (in our experiments $N = 25000$). We use Adam optimizer to optimize the INR-based network. The learning rate starts from 10^{-3} and decays by a factor of 0.5 for every 200 training epochs. The number of total training epochs is set to 1000.

B. Network Architecture

Fig. 2 demonstrates the architecture of the proposed INR-based network, which consists of a spatial coordinate encoding module, i.e., FFM, and a fully-connected neural network. The input and output of the network are voxel coordinate $\mathbf{x}$ and voxel intensity $I(\mathbf{x})$, respectively.

1) Fourier Feature Mapping: For image analysis, the high-frequency components generally correspond to the regions with complex details, while the low-frequency components correspond to the regions with smooth image structures. It is well-recognized that both low and high frequencies are important for medical image reconstruction. Hornik et al. [29] theoretically prove that a simple MLP can approximate any complex function. However, Rehanman et al. [30] demonstrate that neural network tends to learn lower-frequency components of function practically. A recent study [5] showed that encoding positions input from low-dimensional space into high-dimensional space can significantly improve the network's learning ability for high-frequency information. Subsequently, [7] proposed FFM, an extension of the positional encoding, to address the problem of limited performance for frequencies that are not axis-aligned. In our INR-based network, we use FFM to map the 3D input voxel coordinate $\mathbf{x} \in \mathbb{R}^3$ into the high-dimensional feature vector $\mathbf{v} \in \mathbb{R}^{2L}$ before the fully-connected neural network. Let $\gamma(\cdot)$ denote the FFM, which is defined as below:

$$\mathbf{v} = \gamma(\mathbf{x}) = [\cos 2\pi \mathbf{B}\mathbf{x}, \sin 2\pi \mathbf{B}\mathbf{x}]^{\mathrm{T}} \tag{3}$$

Here $\mathbf{B} \in \mathbb{R}^{L\times 3}$ is a real matrix (also called Fourier feature matrix), where each element is sampled from the Gaussian distribution $\mathcal{N}(0, 1)$. In this work, the dimension L is set to 128, empirically.

2) Fully-Connected Neural Network: The fully-connected neural network used in this work mainly consists of 15 fully-connected layers. Each fully-connected layer is followed by a ReLU activation layer. In order to ease the training of the network, we employ two dense connections that concatenate the input of the full-connected neural network to the output of the 5th ReLU activation layer and 10th ReLu activation layer, respectively. And except that the 6th and 11th fully-connected layers have 512 neurons, all other layers have 256 neurons.

C. Arbitrary-Resolution Image Reconstruction

After model optimization, INR-based network can be considered as the approximation of the implicit voxel function $\mathcal{F}_\theta$, which means that the 3D image is implicitly parameterized in the parameters of the fully-connected neural network. The workflow of arbitrary-resolution 3D image reconstruction by the well-trained INR-based network is shown in Fig. 3. We first conduct refinement of factor k on the original voxel grid to build a dense voxel grid (the factor k is an arbitrary integer or float number). Then, each voxel coordinate $\mathbf{x} = (x, y, z)$ from the dense voxel grid is fed into the network $\mathcal{F}_\theta$ to estimate the corresponding voxel intensity $\hat{I}(\mathbf{x})$. As a result, the arbitrary resolution 3D image can be reconstructed.

III. Design Analysis

In this section, we first analyze the computations in the INR-based medical image reconstruction algorithm, and propose a computation flow to balance data reuse and computation load. After that, the idea of coordinate generation is further proposed to minimize the data movement between off-chip to on-chip.

Algorithm 1 INR-Based Medical Image Reconstruction

Input: Voxel coordinates $\mathbf{x}(x, y, z)$, Fourier feature matrix $\mathbf{B}$
Output: Voxel Intensities $\boldsymbol{I}$

1: **for** $i \leftarrow 1$ to $N_{\mathbf{x}}$ **do**
2: $\quad \gamma(\mathbf{x}_i) = [sin(\mathbf{x}_i \cdot \mathbf{B}), cos(\mathbf{x}_i \cdot \mathbf{B})]^T$
3: $\quad O_{i,0} \leftarrow \gamma(\mathbf{x}_i)$
4: $\quad$ **for** $l \leftarrow 1$ to N_{layer} **do**
5: $\quad\quad$ **for** $h \leftarrow 1$ to $N_{out,l}$ **do**
6: $\quad\quad\quad$ **for** $w \leftarrow 1$ to $N_{in,l}$ **do**
7: $\quad\quad\quad\quad Psum_{i,l,h} += \mathbf{W}_{l,h,w} * O_{i,l-1,w}$
8: $\quad\quad\quad$ **end for**
9: $\quad\quad\quad$ **if** $l == N_{layer}$ **then**
10: $\quad\quad\quad\quad O_{i,l,h} = Psum_{i,l,h} + B_{l,h}$
11: $\quad\quad\quad$ **else**
12: $\quad\quad\quad\quad O_{i,l,h} = ReLU(Psum_{i,l,h} + B_{l,h})$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad\quad I_i \leftarrow O_{i,N_{layer}}$
16: $\quad$ **end for**
17: **end for**
18: **return** $\boldsymbol{I}$

A. Computation Flow Analysis

The INR-based algorithm in this work employs a 15-layer fully-connected neural network to extract the voxel intensities based on the input voxel coordinates (after FFM). Algorithm 1 describes the computation of a reconstruction task with a total number of $N_{\mathbf{X}}$ voxels. Note that N_{layer} is the number of layers in the network (15 in this work), $N_{in,l}$ and $N_{out,l}$ are the number of input neurons and output neurons of the lth layer, respectively. In this paper, we use neuron and activation to denote the feature between different layers. There are some key observations from Algorithm 1. First and foremost, the network has to be evaluated for each voxel to extract the corresponding intensity I_i. For illustration, to render three scans, i.e., Coronal scan, Axial scan and Sagittal scan, of an MRI image with a resolution of 256×256, the network has to be queried $256 \times 256 \times 3.196608$ times. This is very different from other neural network-based tasks such as image classification, where the network is evaluated only once. In addition, the model used in this work is fully-connected, instead of convolution-based. Therefore, in this work, a dedicated computation flow along with the corresponding hardware architecture is proposed. In our design, two loop optimization techniques, loop unrolling and loop tiling, are utilized together with data reuse schemes to efficiently map the algorithm to the proposed reconstruction accelerator.

1) Loop Unrolling: Loop unrolling is a technique of replicating the computational circuits with a number of copies that are able to execute in parallel. From Algorithm 1 we find that, except for the loop of the network layer (line 4), in which the computation of next layer is related to the previous one, there is no other data dependency. Therefore, the corresponding loops can be unrolled, leading to:

- *Voxel-level unrolling*. For the computation of a certain layer, we can process in parallel the matrix-vector multiplication between the feature vectors corresponding to multiple voxels and the weights of that layer.
- *Height loop unrolling*. The multiplication between the same neuron with multiple weights that corresponds to different output neurons can be processed concurrently.
- *Width loop unrolling*. The computation between weights and input neurons that corresponds to the same output neurons can be processed in parallel.

Apart from loop unrolling (which is important for throughput), memory access optimization is also critical for computational efficiency, since data movement can be time and power consuming. Therefore, loop unrolling needs to be co-designed with memory access to balance the computation load and data reuse.

Fig. 4 illustrates two kinds of data reuse schemes that can be applied to reduce the amount of memory access during the execution of the INR-based algorithm. By *voxel-level unrolling*, the computation of each layer can be modeled as a matrix multiplication operation between a weight matrix and an input neuron matrix consisting of N_{pixel} vectors. In *Weight Reuse*, each weight in W, for example, A in Fig. 4(a), is multiplied with N_{pixel} neurons in the same coordinate of the input neuron vectors before the update of W, while in *Input Reuse*, each input neuron, e.g., B in Fig. 4(b), is reused across N_{out} weights at the same column of the weight matrix, corresponding to N_{out} output neurons.

2) Loop Tiling: Loop tiling, which splits the overall computation into multiple parts, is often used in network computation [18], [31] to fit the given amount of computational resources and on-chip memories. Although the amount of neurons in each layer (256) is far less than that of CNN models such as ResNet [32], directly storing all the inter-layer features on-chip will consume 96 MB ($196608 \times 256 \times 16$ bits) storage if voxel loops are unrolled, occupying a significant amount of chip area. Moreover, the size of weight matrix of most layers is 256×256, meaning that using a large matrix multiplication block for an entire layer demands very high bandwidth ($256 \times 256 \times 8$ bits $\times$ 400 MHz $\approx$ 262 GB/s) for weight memory and consumes a lot of hardware resources ($256 \times 256 = 65536$ multipliers). Therefore, it is more economical to tile the overall computation in both matrix multiplication level and voxel level. As shown in Fig. 5, at the matrix multiplication level, we partition a large matrix multiplication into a number of smaller parts, each with a size of $T_o \times T_i$, for example, the $\{W_1, W_2, \ldots\}$ in the weight matrix, to balance the memory bandwidth for weights and computation throughput. At the voxel level, we tile them into a number of batches, each with B_i inputs. Combining the

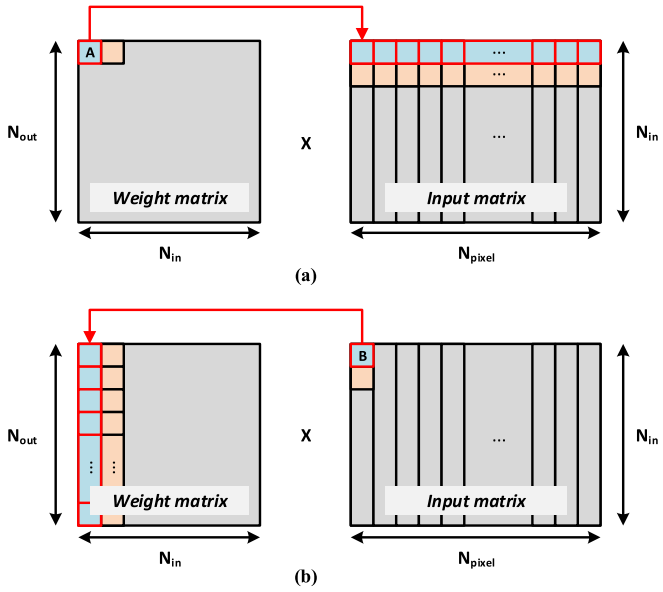


Fig. 4. Illustration of possible data reuse schemes for the INR-based algorithm. (a) Weight (A) reuse across different input vectors. (b) Input neuron (B) reuse among different weights of the same column.

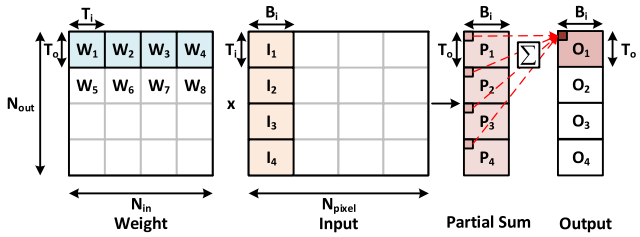


Fig. 5. The proposed computation flow. At first, the total input with N_{pixel} vectors is tiled into several parts, each with B_i vectors. Then the weight matrix is split into a set of submatrices $\{W_1, W_2, \ldots\}$. Correspondingly, the input is split into $\{I_1, I_2, \ldots\}$. During the execution, the multiplication between W_1 and I_1 is first operated and the psum matrix P_1 is generated. Other parts P_2, P_3, P_4 are operated in the same manner. By summing up these four psum matrices, a part of the output matrix O_1 is generated. The overall data flow for the PE array will be explained in Fig. 8.

tiling of matrix multiplication level and voxel level, the input matrix can be divided into several sub-matrices each with $T_i \times B_i$ elements, as $\{I_1, I_2, \ldots\}$ shown in Fig. 5. During the computation, one batch of voxels is loaded into the accelerator for the computation of the entire network. With a proper choice of B_i (128 in this work), we can save all the inter-layer neurons on-chip with significantly less on-chip memory (192 KB), minimizing data communication with off-chip memory.

3) Overall Computation Flow: Based on the above considerations on loop unrolling, data reuse and loop tiling, Fig. 5 illustrates the computation flow of the proposed INR-based reconstruction accelerator. In particular, for a weight matrix with $N_{out} \times N_{in}$ elements, we first split it into a set of submatrices $\{W_1, W_2, , W_3, \ldots\}$, with each submatrix containing $T_o \times T_i$ elements. Correspondingly, each input vector is also divided into N_{in}/T_i parts, with each part having T_i neurons. Suppose we tile all the voxels into a number of batches, each with B_i vectors. We can group T_i neurons of B_i vectors to form sub input matrices $\{I_1, I_2, I_3 \ldots\}$. During the execution, we first

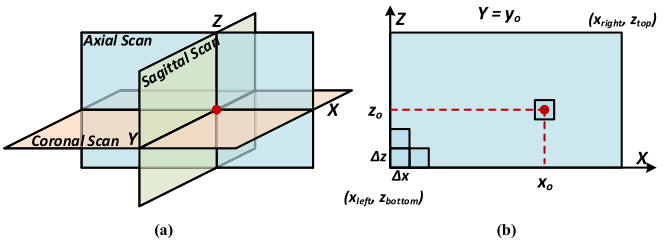


Fig. 6. Illustration of coordinates generation. (a) Three scans are perpendicular to one of the three coordinate axes. (b) Illustration of the coordinate values on Axial scan.

compute the multiplication between W_1 and I_1 to generate a partial sum (Psum) matrix P_1. The system proceeds to the multiplications of the following sub-weight matrices with the corresponding sub-input matrices to generate the remaining Psum matrices P_2, P_3, P_4. By summing up four parts of Psum matrices, we can finally get the submatrix O_1 of the output neuron. Following this flow, the other three output matrices O_2, O_3, O_4 can be generated, which are further used in the computation of the next layer. Note that in this work, all the experimental results are generated based on the setting of $T_i = T_o = 64$ and $B_i = 128$. This setting can be modified to meet different specifications.

B. Coordinates Generation

The original inputs of the algorithm as shown in Algorithm 1 include 3D spatial coordinates of all the voxels. For the reconstruction of three scans with 256×256 pixels, we need to feed $256 \times 256 \times 3 = 196608$ sets of coordinates into the system. However, for medical image reconstruction applications shown in Fig. 6, the coordinates of all three interested planes can be determined by the corresponding intersection point, since these planes are perpendicular to one of the three coordinate axes. For example, as shown in Fig. 6(b), the Axial scan is perpendicular to the Y-axis, such that the voxels on this plane have the same y coordinate value. Therefore, given the boundaries of the 3D volume of the model, as well as the image resolution to be rendered, we can input the coordinates of the intersection point (x_o, y_o, z_o) and calculate all other remaining coordinates within the accelerator. For example, the coordinates of voxels of the Axial scan plane in Fig. 6(b) can be generated as

$$\begin{aligned} x_i &= x_{left} + cnt_1 \times \Delta x \\ y_i &= y_o \\ z_i &= z_{bottom} + cnt_2 \times \Delta z \end{aligned} \tag{4}$$

where x_{left} and z_{bottom} are the x value of left most voxel and the z value of bottom-most voxel, respectively, Δx and Δz are the resolution in the X-axis and Z-axis, respectively, and cnt_1 and cnt_2 are the number of pixels in the X-axis and Z-axis, respectively. Therefore, instead of inputting all the coordinate sets for three scans, we build an on-chip coordinate generator according to (4) to compute all the coordinate sets on-chip. As a result, the inputs are reduced to the boundaries, the resolution of the reconstructing image and the coordinate

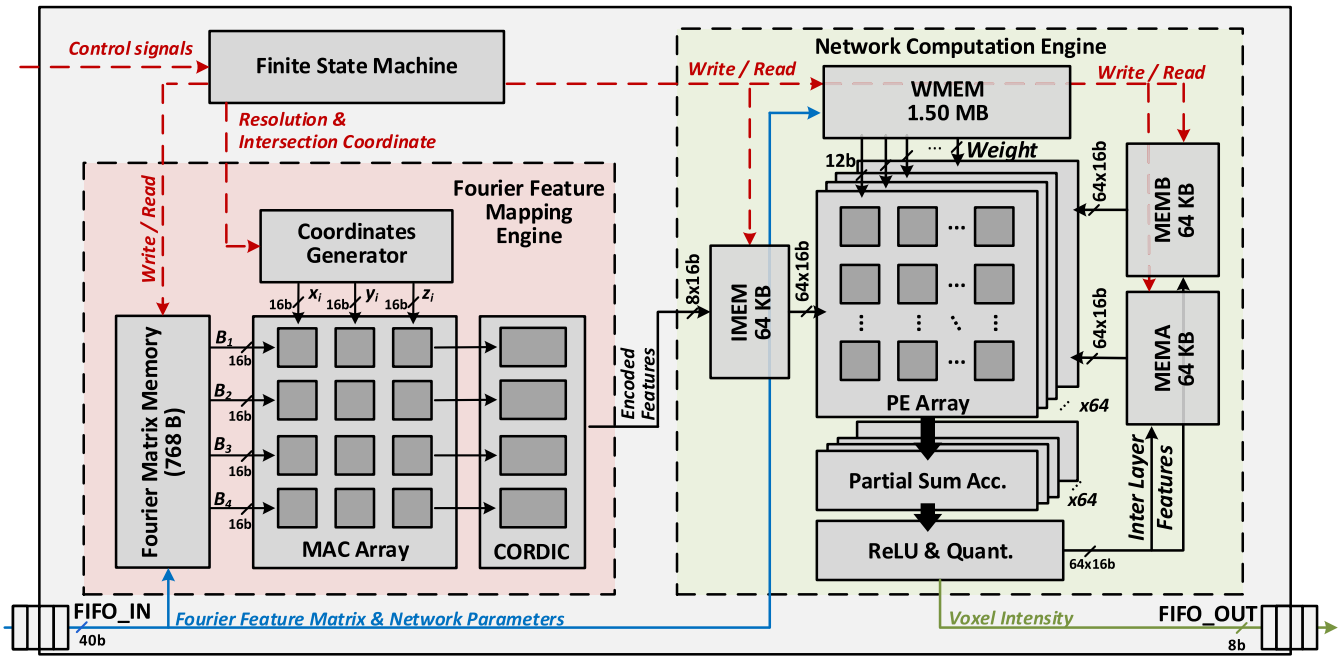


Fig. 7. Overall architecture of the proposed accelerator.

of the intersection point, leading to a significant amount of data transfer reduction compared to the original algorithm.

IV. Architecture of the Proposed Accelerator

A. Overall Architecture

Fig. 7 demonstrates the overall architecture of the proposed accelerator. It mainly consists of two parts, namely, the FFM engine and the network computation engine. The FFM engine is a dedicated module to implement the mapping from input coordinates to high-dimensional encoded features, while the network computation engine is responsible for generating the final voxel intensities through network evaluations. A finite state machine (FSM) is employed as a top controller to control the execution of the algorithm according to the external control signals. In particular, the FSM accepts and decodes two kinds of control signals, i.e., parameter loading and configuration setting & reconstruction start. Two first-in-first-out (FIFO) modules connect the accelerator to the external memories, where FIFO_IN is used to load the Fourier feature matrix and network parameters into the accelerator and FIFO_OUT is responsible for writing out the generated voxel intensities.

B. Network Computation Engine

As shown in Fig. 7, the network computation engine consists of 64 parallel processing element (PE) arrays, 64 partial sum accumulation (PSA) blocks, a ReLU & Quantization unit and four memory blocks. In our design, a weight memory block with a size of 1.5 MB is used to store all the weights of the network during the execution of the algorithm. In addition, there are three neuron memories, i.e., one input memory (IMEM) for saving the input neuron of the first layer, and two inter-layer neuron memories (MEMA and MEMB) for buffering the neurons of internal layers. Note that MEMA and MEMB are designed to work in an exchangeable manner, i.e., if MEMA is used as the output neuron memory of the current layer, it will be the input memory for the next layer, and vice versa.

Fig. 8 explains the detailed data flow in the PE arrays. For each PE array, the same input neuron x is shared by 64 PEs to multiply with 64 weights from the same column of the submatrix in Fig. 5, generating 64 products. During the computation, 64×64 weights stay in the PE array for $B_i = 128$ cycles. In each cycle, a new input vector with 64 neurons are loaded into 64 PE arrays to multiply with these weights for different Psum vectors. Moreover, there are two weight buffers in each PE as illustrated in Fig. 8(b). One of them controlled by W_sft is used to save the next part of weight matrix with a shift register manner while the other one controlled by W_set is used for the current product computation.

The products from PE arrays are transferred to the PSA blocks for accumulation to generate the activations. There are two kinds of Psum accumulation as shown in Fig. 9, i.e., adder tree accumulation and tile accumulation. In each cycle, 64 PE arrays generate 64 product vectors, with each vector having 64 products as shown in the top left of Fig. 9. When summing up the products at the same horizontal position, i.e. the same row, we generate a partial sum vector corresponding to the matrix-vector multiplication between sub weight matrix and input vector, for example the first column of the partial sum matrix P_1. The adder tree then sums up 64 products in three stages. Each stage contains adders with four input data. Tile accumulation sums up the Psums from the same position of the Psum matrix to generate the final result as shown in the bottom left of Fig. 9. Each tile accumulation is responsible for the computation of one row of the output matrix O_1. Note

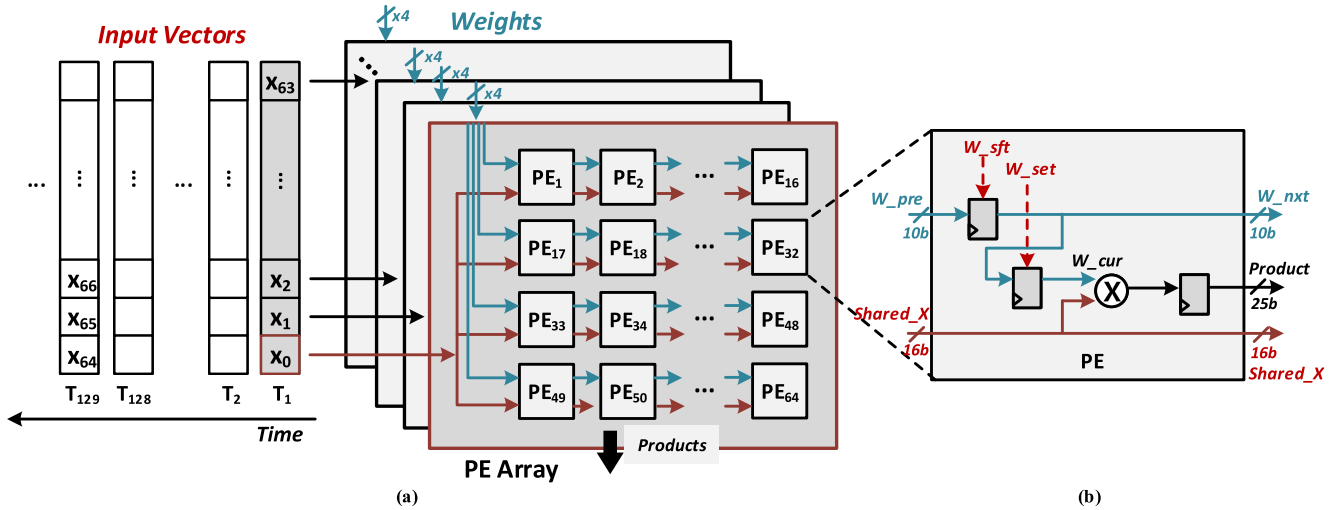


Fig. 8. Data flow in the PE arrays. (a) The input vectors are loaded into corresponding PE arrays in a sequential manner while the same input neuron is shared by 64 PEs. (b) The detailed structure of a PE, including two weight buffers and one multiplication unit.

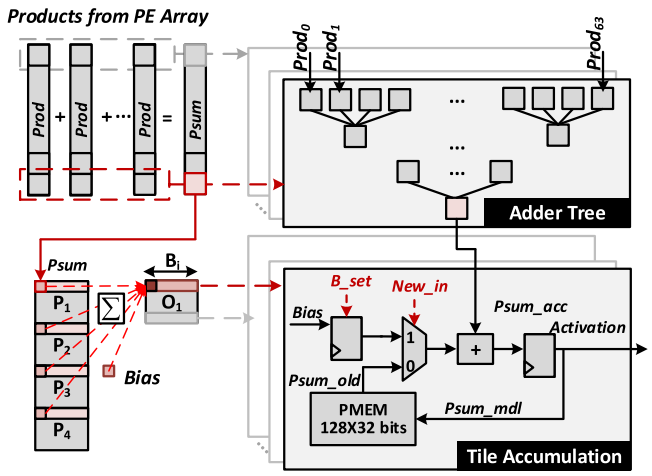


Fig. 9. Structure of the partial sum accumulation block.

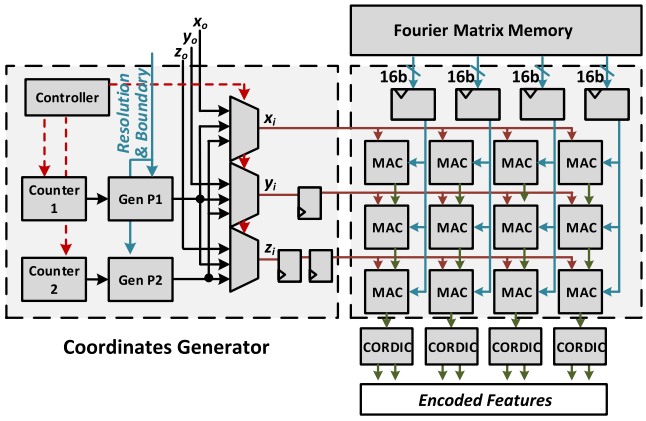


Fig. 10. Structure of the Fourier feature mapping block.

that a tile accumulation block contains a 128×32 bits memory (PMEM) for saving 128 Psums that correspond to 128 input vectors. Moreover, there is a bias buffer to store the bias for the current output activation. When a new Psum comes out from adder tree accumulation, we add it with the bias and save it into the PMEM. After 128 cycles, the Psum from adder tree accumulation will be added to the Psum read from the PMEM instead of the bias.

C. Fourier Feature Mapping Engine

The FFM operation mainly consists of two parts: (i) matrix-vector multiplication between the coordinates of voxels $\mathbf{x}(x, y, z)$ and Fourier feature matrix $\mathbf{B}$ and (ii) the triangle functions operates on the resultant inner products. The FFM engine in our design performs the FFM of the input coordinates with one multiplication-accumulation (MAC) array and one Coordinate Rotation Digital Computer (CORDIC) array as shown in Fig. 10. A MAC array, containing 12 MAC blocks, performs the matrix-vector multiplication (3) between the voxel coordinates $\mathbf{x}(x, y, z)$ and the Fourier feature matrix $\mathbf{B}$ in a pipelined manner. Four CORDIC modules perform the sine and cosine functions on the four parallel results from the MAC array and export the encoded features to the input neuron memory, i.e., IMEM, in the network computation module. A coordinate generation unit in FFM receives the reconstruction configuration from the top FSM and generates the coordinates (x_i, y_i, z_i) of all the voxels and sends them to the MAC array. Fourier feature matrix $\mathbf{B}$ with a size of 3×128 is saved in one 768B Fourier feature matrix memory.

Fig. 10 illustrates the data flow in the FFM engine. As shown in Fig. 10, in each cycle, the coordinate generator outputs one coordinate of the voxel (x_i, y_i, z_i) while Fourier feature matrix memory outputs 4×3 feature parameters to the 12 MAC in 3 cycles. Every four MACs on the same row of the MAC array read in the same axis, for example, x_i for the top four MACs. Along the vertical direction, three parameters of the same frequency (corresponding to a column in the Fourier feature matrix) are separately written into three MACs to multiply with three axes. Three registers buffering the y_i and z_i are used to pipeline the inner products of voxel

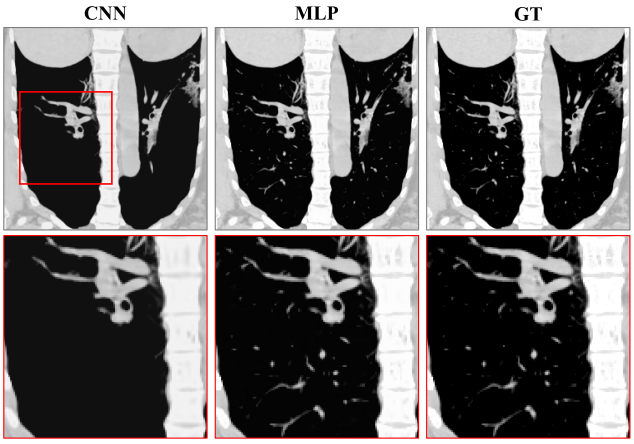


Fig. 11. Reconstruction results of a 15-layer CNN and a 9-layer MLP for parameterizing a CT image.

and frequency parameters. Following the MAC array, four CORDIC blocks separately operate on four inner products come from the bottom four MACs to generate the encoded features. These features are then written into the IMEM of the network computation engine.

Recall that in the network computation engine, we group up $B_i = 128$ inputs into one batch to compute the entire network. Correspondingly, the FFM engine needs to generate the needed 128×256 features. Since there are only 12 MACs in the FFM engine, we need to traverse the frequency F_{freq} and input coordinate to generate these encoded features. There are two kinds of implementations, which are (i) keeping the input coordinate in MAC and traversing the frequencies or (ii) traversing the input while keeping the frequency. As shown in the Axial scan in Fig. 6, if we traverse the input coordinate, for example, we loop the X-axis first, then the values of y_i and z_i remain the same during $Resolution_X$ times, where $Resolution_X$ is the number of pixels in X-axis. This is because these voxels hold the same y and z coordinate values. Therefore, we can keep the frequency in the MAC array and only generate the x values for $Resolution_X$ cycles. Since we keep the frequency in the MAC array while the y and z coordinate values stay the same in $Resolution_X$ cycles when traversing the coordinates, the products between $\{y, z\}$ and the corresponding frequencies will not change. Therefore, we can save the dynamic power of the multiplication of y and z.

V. Implementation Results

This section presents the implementation results to validate the algorithm and evaluate the performance of the proposed reconstruction accelerator. We conduct our experiments on two image modalities, including MRI from [15] and CT from [33]. Three T1-weighted (T1w) brain MR images from three healthy adults on a 7T MR scanner and one CT image from a patient with COVID-19 on a commercial multi-detector CT scanner.

A. CNN Vs. MLP for Parametrization and Reconstruction of Medical Images

Although CNNs can also be used to fit 2D natural images, MLPs have some major advantages in the parameterization and reconstruction of medical images. In this section, we conduct experiments to demonstrate these advantages. Specifically, we build a 15-layer CNN and 9-layer MLP to fit a CT image of 256×256 resolution. The FFM is employed in both implementations for a fair comparison.

TABLE I

Qualitative Comparison of Reconstruction Results Using CNN and MLP

Network	PSNR (dB)	Model Size (MB)	Training Time
15-layer CNN	28.08	36.00	40m 33s
9-layer MLP	47.13	2.51	13m 39s

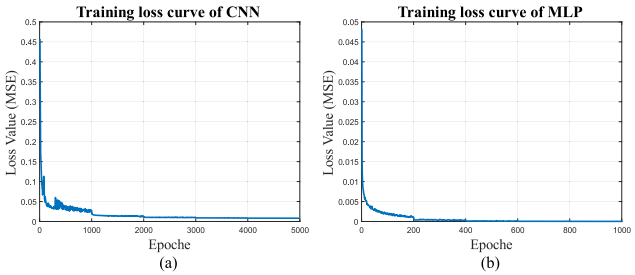


Fig. 12. Training loss curves of a 15-layer CNN (a) and a 9-layer MLP (b) for parameterizing a CT image.

Fig. 11 shows the reconstruction results of CNN and MLP implementation, along with the ground truth image. As we can see, the result of CNN is over-smooth, where some image details are lost as shown in the red box. While MLP reconstructs the image with many details, which is close to the ground truth. We also report the quantitative results in Table I, where MLP significantly outperforms CNN in terms of peak-signal-to-noise ratio (PSNR) (47.13 vs 28.08 dB). Since convolutional operations in CNNs impose great inductive bias toward low-frequency images, i.e., locality, CNNs hardly fit high-frequency signals accurately. While MLP does not include any explicit prior and thus can obtain better performance. Table I also shows MLP is superior to CNN in both model size and training time. Moreover, we demonstrate the training loss curves of the MLP and CNN in Fig. 12. Compared with CNN, the training of MLP is more stable and obtains a lower error. Therefore, we choose MLP in the proposed INR-based algorithm.

B. Quantization Results Analysis

Before discussing hardware implementation, we present the reconstruction results of the INR-based algorithm (to be executed on the proposed accelerator) under different quantization settings. In this work, linear quantization is used to transform a trained floating point model to a corresponding fixed point one. For reconstruction quality evaluation, the widely used PSNR and structural similarity (SSIM) are adopted in this work. For each model from four datasets (three MRI and one CT model), we randomly choose 300 slices as input to reconstruct the medical images and calculate the average PSNR and SSIM

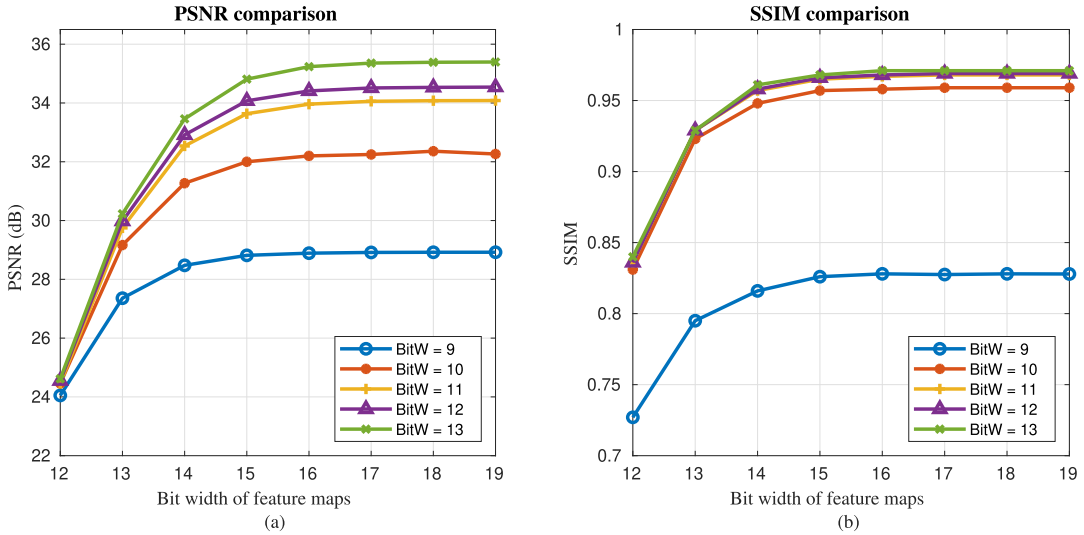


Fig. 13. (a) PSNR and (b) SSIM performance of reconstruction results under different quantization settings.

among these images according to the following definitions:

$$PSNR = 20 \cdot log_{10}(\frac{2^B - 1}{\sqrt{MSE}})$$
$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$
$$SSIM(I,K) = \frac{(2\mu_I\mu_K + c_1)(2\sigma_{IK} + c_2)}{(\mu_I^2 + \mu_k^2 + c_1)(\sigma_I^2 + \sigma_K^2 + c_2)} \quad (5)$$

Here I is the ground truth image and K is the reconstructed image, B represents the pixel depth of images, $m \times n$ is the resolution of the image. In addition, μ_I and μ_k are the pixel sample mean of I and K, respectively, σ^2 is the variance of each image, σ_{IK} is the covariance of I and K, c_1 and c_2 are two small constants to stabilize the division with weak denominator.

Fig. 13 shows the average PSNR and SSIM between the medical images rendered by the quantized models and the ground truth images under different quantization settings, i.e., different weight and feature map bit width combinations. As we can see, for both PSNR and SSIM, there is a significant improvement when the bit width of weights increased from 9 to 10. While for the cases of weight bit width larger than 10, the rate of improvements becomes smaller, especially for SSIM. As we can see from Fig. 13(b), the SSIM curves for weight bit width=10,11,12 and 13 are very close to one another, and the improvements from weight=11 to 13 are negligible. On the other hand, the reconstruction quality has a great improvement when the bit width of the feature maps changes from 12 to 13. As the increase of feature map bit width, the slope of the curves decreases. As we can see, all the curves are almost flat when the bit width is larger than 16, meaning that the benefit of further increasing the bit width of feature maps is very small.

According to the above quantization analysis, it is found that there are multiple settings to meet a given PSRN/SSIM specification. These settings lead to different hardware resource utilization, especially the size of on-chip memory. In particular, if the bit width of weights increments by 1 bit, the total size of on-chip memory will increment by 128 KB since all the weights are stored on-chip in our design. On the other hand, 1-bit increment in all the feature maps leads to an increment of total on-chip memory by only 12 KB since we used only three neuron memory (IMEM, MEMA and MEMB) to save the feature maps. Therefore, from the on-chip memory perspective, it is preferable to increase the bit width of feature maps instead of weights to meet a given PSRN/SSIM specification.

According to the results presented in existing literature, the PSNR and SSIM performance for similar reconstruction tasks such as super-resolution of medical images [34], [35], [36], [37] range from 28 to 35 and 0.70 to 0.94, respectively, where the typical values are around 30 and 0.9, respectively. Moreover, according to the consultation with clinical doctors, medical images with PSNR larger than 30 are enough for most of the practical usages. Therefore, in the remaining of experiments, we use a quantization setting of 10 bits for weights and 16 bits for feature maps, translating to a PSNR index of 31.95 dB and an SSIM index of 0.949. But it is worth noting that the proposed architecture is flexible enough to be modified to other quantization settings.

C. Visual Comparison of Reconstruction Results

To visualize the reconstruction results, Fig. 14 shows the comparison of different reconstruction results, including the images generated by the quantized INR model (executed on the proposed accelerator) and the original INR-based reconstruction algorithm, as well as the ground truth. In this comparison, we present the reconstruction results of three sets of MRI datasets (first three columns) and one set of CT dataset (last column). In order to compare the details, we zoom one part of the rendered image (indicated by the red bounding box) and display it on the bottom right of each result. Note that the CT dataset in the last column presents the COVID-19 lung images where the highlighted parts are caused by infection. As we can see, the reconstruction results of the

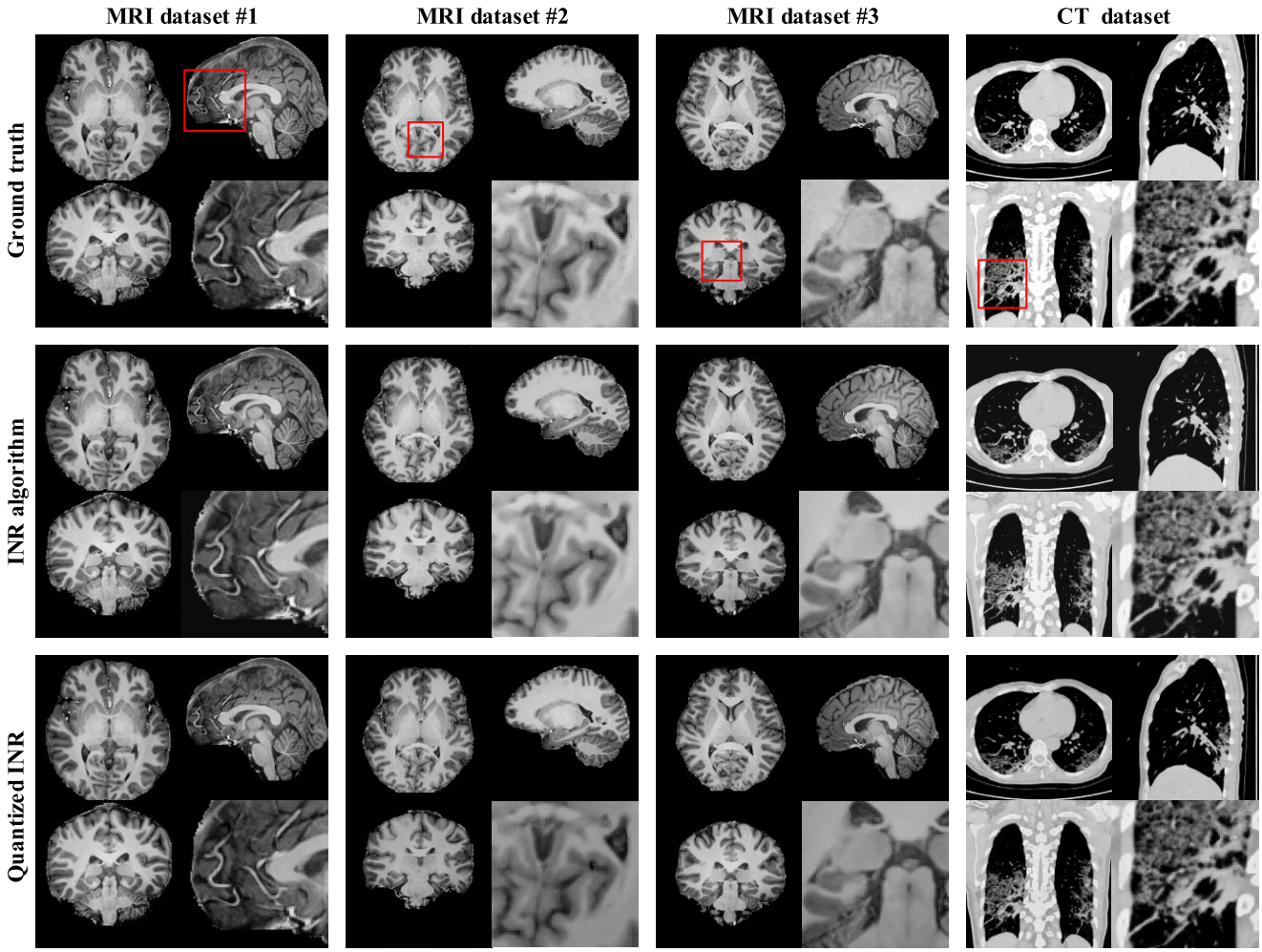


Fig. 14. Visual comparison of reconstruction results (generated by the original INR-based reconstruction algorithm and quantized INR model) with ground truth. Each result contains three scans and one enlarge part at the right bottom to show the details. The first three columns are MRI datasets of brain scans and the last column is a CT dataset of COVID-19 lung images.

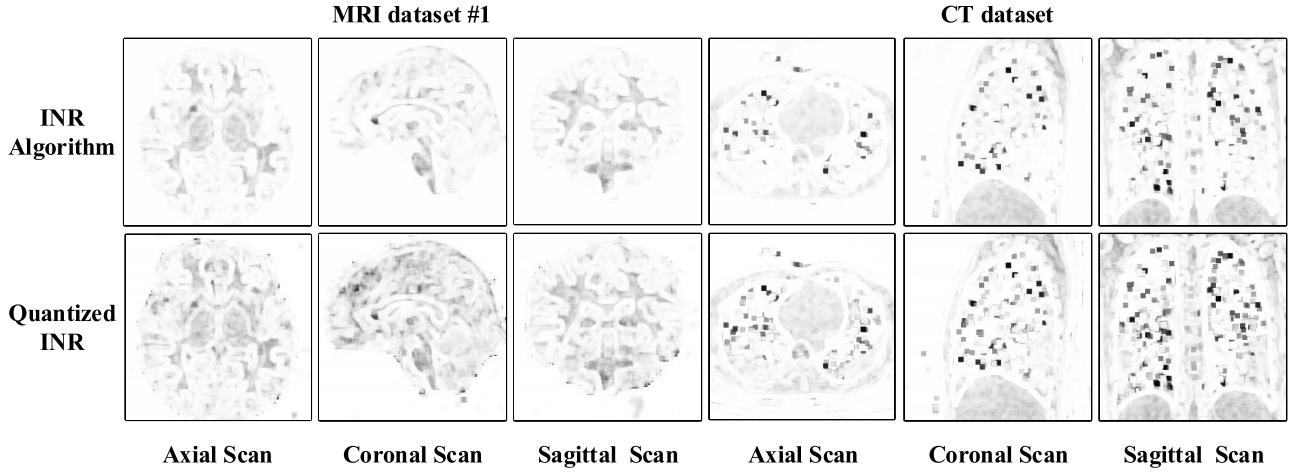


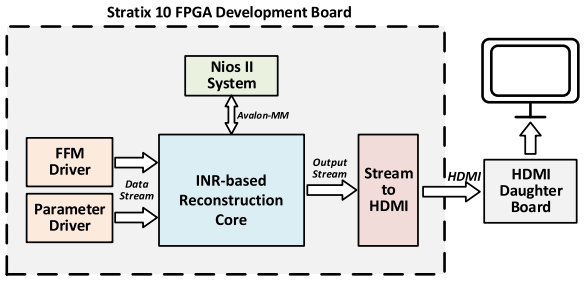
Fig. 15. The SSIM difference maps (with ground truth images) of the results generated by the proposed INR-based reconstruction algorithm and the corresponding quantized model. The first three difference maps are corresponding to the first MRI dataset in Fig. 14 and the last three difference maps are corresponding to the CT dataset in Fig. 14.

quantized model and the original model are visually very close to each other. Compared with the ground truth images, the results of the original INR algorithm and the quantized INR model are a little bit smoothed in some gray parts. But the boundaries between the dark and bright parts are clear for all the implementations, and there is no critical information loss.

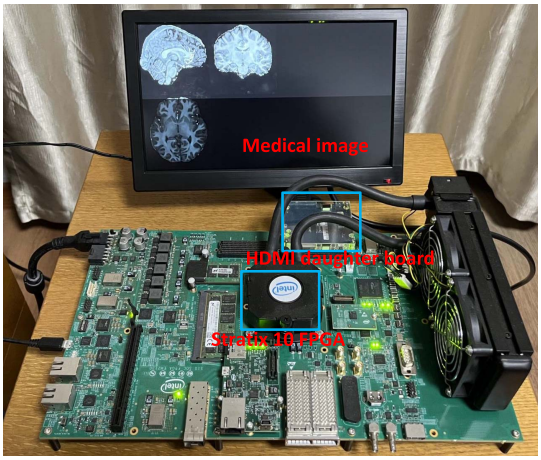
TABLE II
RESOURCES UTILIZATION OF THE FPGA IMPLEMENTATION

Resources	ALM	FF	M20K	DSP
Used	526208	275351	794	60
Available	933120	1866240	11721	5760
Utilization	56.4%	14.75%	6.77%	1.04%

ALM: Adaptive logics module. FF: Filp-flop. M20K: M20K memory block.



(a)



(b)

Fig. 16. Demonstration system of the proposed reconstruction accelerator implemented in Intel Stratix 10 FPGA SX SoC development board. (a) Block diagram of the demonstration system. (b) Photo of the setup system.

Moreover, as shown in the CT dataset, the infection parts are clear in the images generated by the original INR algorithm and quantized INR model.

To further visualize the difference between the ground truth images and the rendered results, we present the SSIM difference maps between the rendered results and the ground truth of the first MRI dataset and the CT dataset in Fig. 15. For SSIM difference, the brighter of the maps, the more similar of two images. As shown in the figure, most of the differences come from the gray area of the ground truth image, which is similar to that in the reconstruction results comparison in Fig. 14. Moreover, the SSIM difference maps of original INR and quantized INR models are visually very close to each other. Although extra errors are introduced by model quantization, the average SSIM of the quantized implementation can be as high as 0.94, indicating that the reconstruction results of the quantized model are very close to the ground truth.

D. FPGA Implementation

To validate the proposed accelerator, we code the proposed design in VerilogHDL and build a proof-of-concept prototype system using the Intel Stratix 10 FPGA. Fig. 16(a) shows the block diagram of the demonstration system, consisting of a reconstruction accelerator core, two memory drivers, one Nios II embedded processor and one stream to HDMI module. The two memory drivers include two SRAM blocks to store the Fourier feature matrix and the network parameters, which are streamed to the accelerator through AXI4-Stream bus. The Nios II embedded processor works as the overall system controller. It uses an Avalon-MM interface to send the control signals to the reconstruction accelerator and monitor the state of the system with the addressable registers in the accelerator core. The stream to HDMI module receives the data stream from the reconstruction accelerator and saves it into an internal SRAM block. An HDMI reading controller in this module then reads out the data and converts it into HDMI streams. With the FMC HDMI daughter board plugged into the Stratix 10 board, the HDMI stream is sent to the monitor for display. Fig. 16(b) shows a photo of the overall demonstration system, where the images on the monitor are three rendered scans of Medical images.

The demonstration system is implemented using Quartus prime Pro 21.0. Table II lists the resource utilization of the prototype reconstruction system. Running at a clock frequency of 170 MHz, the proposed design is able to render medical images with 128×128 resolution at a frame rate of 44.7 FPS. If we increase the image resolution to 256×256, the achievable frame rate is 11.2 FPS. In terms of power consumption (obtained from the post-implementation power report generated by Quartus with typical activity), the entire system dissipates 7.11 W, whereas the reconstruction accelerator consumes about 6.36 W. According to the report, the clock network of the system costs about 2.62 W, which accounts for about 36.8% of the total power, and the registers cost about 2.94 W, which is about 41.3% of the total power.

E. ASIC Implementation Results

To evaluate the performance of the proposed reconstruction accelerator, we also implement our design in 40 nm CMOS technology. The design is synthesized with Synopsys Design Compiler in topographical mode, where the placement information is also taken into consideration during the implementation. The implementation results show that the overall system costs about 1.72 MB SRAM, where 87.2% are used for storing the weights of the network. The proposed system, running at 400 MHz, achieves about 26.3 FPS for the reconstruction of medical images with 256×256 resolution. Furthermore, we use Synopsys PrimePower to analyze the power consumption of the generated netlist with actual switching activity information extracted by taking in real data during the simulation. Running at 400 MHz, the overall system consumes 795 mW of power.

To have a general comparison with existing neural network accelerators, we present the implementation results of the proposed design as well as some well-known accelerators in Table III. For a fair comparison, we normalize the throughput

TABLE III
GENERAL COMPARISONS WITH OTHER RELATED ACCELERATORS

	VLSI 2018 [26]	Cambricon X [28]	RNA [23]	AICAS 2021 [27]	This work
Process (nm)	14	65	65	40	40
Methodology	Measurement	Post layout	Post layout	Synthesis	Synthesis
Frequency (MHz)	1060	1000	NA	1600	400
Core Area (mm^2)	0.024	6.38	1.64	0.57	13.8
Bit Width (bits)	8	16	16	8	I-16, W-10
# MAC	16	256	16	8	4160
Power (mW)	47.7	954	133	93	795
Throughput [a](MAC/s)	1.08×10^6	6.55×10^7	NA	8.19×10^5	2.66×10^8
Throughput (GOP/s)	136	544	4.07	900	3328
Energy Efficiency [a](MAC/s/W)	3.38×10^7	6.87×10^7	NA	8.8×10^6	3.35×10^8
Energy Efficiency (TOPs/W)	2.9	0.57	0.03	9.7	4.186

[a] For MAC/s and MAC/s/W, we normalize the results into 1-bit multiplication (i.e. $MAC/s = \#MAC \times BI \times BW \times Frequency$, where BI and BW is the bit-width of feature maps and weights respectively) to address the problem of different computation precision.

TABLE IV
COMPARISONS OF DIFFERENT IMPLEMENTATIONS

Platform	CPU	GPU	FPGA	ASIC
	Intel i9-9940x	NVIDIA 2080Ti	Stratix 10	HLMC
Process	14 nm	12 nm	14 nm	40 nm
Clock Frequency	3.3 GHz	1.35 GHz	170 MHz	400 MHz
Throughput (FPS) [a]	1.379	5.58	11.2	26.3
Power (W)	-	58.5[b]	7.11	0.795
Energy Efficiency (mW/pixel)	-	0.16	9.68×10^{-3}	4.61×10^{-4}
Energy Efficiency (frames/W)	-	0.096	1.575	33.08

[a] The image resolution is 256×256.
[b] Obtained by the NVIDIA system management interface.

and energy efficiency using two different figure-of-merits, i.e., MAC/s & GOP/s and MAC/s/W & TOPs/W, respectively. As we can see, the proposed accelerator occupies a larger silicon area than these four existing accelerators, since we use more MACs to meet the requirement of real-time rendering. As a result, the throughput of the proposed design is also much larger. For energy efficiency, the design in [27] is superior to others in TOPs/W, while the proposed design is better in normalized MAC/s/W since the bit width of our work is larger.

To the best knowledge of the authors, this is the first dedicated accelerator for INR-based medical image reconstruction. The existing accelerator designs for CNNs and MLPs are not suitable for running the proposed INR-based algorithms since we have other modules such as coordinate generation and FMM. Therefore, we compare our ASIC design with CPU-based and GPU-based implementations, as well as our own FPGA implementation in Table IV. For performance evaluation, we randomly select 300 coordinate sets from the previous four models and render the corresponding images to calculate the throughput of CPU and GPU implementations. Both CPU-based and GPU-based implementations are under the PyTorch framework. Specifically, we use Intel i9-9940X CPU with 128GB memory to evaluate the CPU-based implementation (implemented using the PyTorch framework with AVX enabled). For the GPU version, we use one NVIDIA GeForce RTX 2080Ti with CUDA version of PyTorch to run the same task. Moreover, for GPU, we use the NVIDIA system management interface to monitor the average power consumption during the running of the algorithm.

In general, compared with CPU, GPU and FPGA implementations, the ASIC design achieves the highest performance (26.3 FPS) with much less power consumption, translating to a superior energy-efficiency performance. For CPU, it takes about 725 ms to render a single image, leading to a frame rate of 1.379 FPS. The GPU implementation achieves 5.58 FPS with a power consumption of about 58 W. The proposed ASIC implementation can process images with the same resolution at a frame rate of 26.3, and consumes only 0.795 W. As a result, the energy efficiency of the proposed ASIC design, measured by frames/W, is 345 times higher than GPU for executing the same medical image reconstruction tasks. For the FPGA implementation, the energy efficiency is about 16.4 times higher than that of the GPU implementation.

VI. CONCLUSION

In this work, we propose an INR-based algorithm for parametrizing 3D medical images to address the trade-off problem between data accuracy and storage & transmission resources.

The parametric representation of the INR-based algorithm is continuous and implicit, which facilitates data compression and arbitrary resolution reconstruction. However, the process of image reconstruction from neural representations is computationally demanding. Therefore a dedicated hardware accelerator for the execution of the INR-based reconstruction algorithm is further proposed. In particular, we proposed an optimized computation flow by co-designing data reuse and computation load, aiming at balancing memory access and throughput. Based on that, a corresponding hardware architecture, which takes in the coordinate of the intersection point of three scans and outputs all the voxel intensities, is further proposed. To validate the proposed design, we build a prototype demonstration system on FPGA. We also map our design to 40nm CMOS technology to measure the performance of the proposed accelerator. The implementation results show that, running at 400MHz, the accelerator is able to process medical images with 256×256 resolution in real-time at a frame rate of 26.3 FPS, with a power consumption of only 795 mW. Comparison results show that the performance, as well as the energy efficiency of the proposed accelerator, outperforms the CPU-based and GPU-based implementations.

REFERENCES

[1] P. Mildenberger, M. Eichelberg, and E. Martin, "Introduction to the DICOM standard," *Eur. Radiol.*, vol. 12, no. 4, pp. 920–927, Apr. 2002.

[2] M. Mustra, K. Delac, and M. Grgic, "Overview of the DICOM standard," in *Proc. 50th Int. Symp. ELMAR*, vol. 1, Sep. 2008, pp. 39–44.

[3] K. A. Kotteri, A. E. Bell, and J. E. Carletta, "Multiplierless filter bank design: Structures that improve both hardware and image compression performance," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 6, pp. 776–780, Jun. 2006.

[4] H.-S. Kim, J. Lee, H. Kim, S. Kang, and W. Park, "A lossless color image compression architecture using a parallel Golomb-rice hardware CODEC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 11, pp. 1581–1587, Nov. 2011.

[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.

[6] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 7462–7473.

[7] M. Tancik et al., "Fourier features let networks learn high frequency functions in low dimensional domains," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 7537–7547.

[8] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 165–174.

[9] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4460–4470.

[10] A. Tewari et al., "Advances in neural rendering," *Comput. Graph. Forum*, vol. 41, no. 2, pp. 703–735, 2022.

[11] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[12] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Nov. 2017.

[13] J. Yang et al., "GQNA: Generic quantized DNN accelerator with weight-repetition-aware activation aggregating," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 10, pp. 4069–4082, Oct. 2022.

[14] Y. Chen, S. Liu, and X. Wang, "Learning continuous image representation with local implicit image function," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 8628–8638.

[15] Q. Wu et al., "IREM: High-resolution magnetic resonance image reconstruction via implicit neural representation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Interv.*, 2021, pp. 65–74.

[16] J. Tang, X. Chen, and G. Zeng, "Joint implicit image function for guided depth super-resolution," in *Proc. 29th ACM Int. Conf. Multimedia*, Oct. 2021, pp. 4390–4399.

[17] L. Chai, M. Gharbi, E. Shechtman, P. Isola, and R. Zhang, "Any-resolution training for high-resolution image synthesis," 2022, *arXiv:2204.07156*.

[18] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[19] T. Yuan, W. Liu, J. Han, and F. Lombardi, "High performance CNN accelerators based on hardware and algorithm co-optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 250–263, Jan. 2021.

[20] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An architecture to accelerate convolution in deep neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 65, no. 4, pp. 1349–1362, Apr. 2018.

[21] J. Jo, S. Kim, and I.-C. Park, "Energy-efficient convolution architecture based on rescheduled dataflow," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4196–4207, Dec. 2018.

[22] Y.-J. Lin and T. S. Chang, "Data and hardware efficient design for convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 5, pp. 1642–1651, May 2018.

[23] F. Tu, S. Yin, P. Ouyang, L. Liu, and S. Wei, "Reconfigurable architecture for neural approximation in multimedia computing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 3, pp. 892–906, Mar. 2019.

[24] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "SpArch: Efficient architecture for sparse matrix multiplication," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 261–274.

[25] S. Pal et al., "A 7.3 m output non-zeros/J sparse matrix–matrix multiplication accelerator using memory reconfiguration in 40 nm," in *Proc. Symp. VLSI Technol.*, Jun. 2019, pp. C150–C151.

[26] M. Anders et al., "2.9TOPS/W reconfigurable dense/sparse matrix-multiply accelerator with unified INT8/INTI6/FP16 datapath in 14NM tri-gate CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 39–40.

[27] W.-C. Lin, Y.-C. Chang, and J.-D. Huang, "An efficient and low-power MLP accelerator architecture supporting structured pruning, sparse activations and asymmetric quantization for edge computing," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–5.

[28] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[29] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Dec. 1989.

[30] N. Rahaman et al., "On the spectral bias of neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5301–5310.

[31] Z. Que et al., "Recurrent neural networks with column-wise matrix-vector multiplication on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 2, pp. 227–237, Feb. 2022.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[33] Q. Wu, R. Feng, H. Wei, J. Yu, and Y. Zhang, "Self-supervised coordinate projection network for sparse-view computed tomography," 2022, *arXiv:2209.05483*.

[34] C. You et al., "CT super-resolution GAN constrained by the identical, residual, and cycle learning ensemble (GAN-CIRCLE)," *IEEE Trans. Med. Imag.*, vol. 39, no. 1, pp. 188–203, Jan. 2019.

[35] C. Zhao, B. E. Dewey, D. L. Pham, P. A. Calabresi, D. S. Reich, and J. L. Prince, "SMORE: A self-supervised anti-aliasing and super-resolution algorithm for MRI using deep learning," *IEEE Trans. Med. Imag.*, vol. 40, no. 3, pp. 805–817, Mar. 2021.

[36] Y. Sui, O. Afacan, A. Gholipour, and S. K. Warfield, "Learning a gradient guidance for spatially isotropic MRI super-resolution reconstruction," in *Proc. Int. Conf. Medi. Imag. Comput. Comput.-Assist. Interv.*, 2020, pp. 136–146.

[37] V. Cherukuri, T. Guo, S. J. Schiff, and V. Monga, "Deep MR brain image super-resolution using spatio-structural priors," *IEEE Trans. Image Process.*, vol. 29, pp. 1368–1383, 2020.

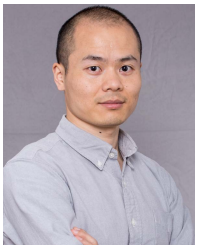
Chaolin Rao (Graduate Student Member, IEEE) received the B.Eng. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2016. He is currently pursuing the Ph.D. degree with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China. His current research interests include neural rendering and VLSI design for artificial intelligence.

Qing Wu received the B.Eng. degree in communications engineering from the China University of Geosciences, Wuhan, China, in 2020. He is currently pursuing the Ph.D. degree with ShanghaiTech University, Shanghai, China. His current research interests include medical image reconstruction and implicit neural representation.

Pingqiang Zhou (Member, IEEE) received the B.E. degree from the Nanjing University of Posts and Telecommunications, China, in 2005, the M.E. degree from Tsinghua University, Beijing, China, in 2007, and the Ph.D. degree from the University of Minnesota in 2012. He is currently an Associate Professor with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China. Prior to joining ShanghaiTech, he worked at the IBM T. J. Watson Research Center as a Research Intern in 2011 and the University of Minnesota as a Post-Doctoral Researcher from 2012 to 2013, respectively. He was with the University of California at Berkeley, Berkeley, as a Visiting Scholar, in 2015. His current research interests include the computer-aided design of VLSI circuits, computer architecture, and hardware security. He received the best paper nominations in ASP-DAC 2010 and CSTIC 2016. He has been serving on the technical program committees of many international conferences, such as DAC, ICCAD, and ASP-DAC. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM—II: EXPRESS BRIEFS.

Jingyi Yu (Fellow, IEEE) received the B.S. degree from Caltech in 2000 and the Ph.D. degree from MIT in 2005. He is currently the Vice Provost at ShanghaiTech University. Before joining ShanghaiTech, he was a Full Professor with the Department of Computer and Information Sciences, University of Delaware. His research interests span a range of topics in computer vision and computer graphics, especially on computational photography, nonconventional optics, and camera designs. He was a recipient of the NSF CAREER Award and the AFOSR YIP Award and has served as an Area Chair for many international conferences, including CVPR, ICCV, ECCV, IJCAI, and NeurIPS. He was the Program Chair of CVPR 2021 and will be the Program Chair of ICCV 2025. He has been an Associate Editor of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, the IEEE TRANSACTIONS ON IMAGE PROCESSING, and the *Computer Vision and Image Understanding* (Elsevier).

Yuyao Zhang (Member, IEEE) received the B.Eng. degree from the Department of Electronic Information Engineering, Harbin Engineering University, Harbin, China, in 2007, the M.Sc. degree from the Department of Electronic Information Engineering, Harbin Institute of Technology, Harbin, in 2010, and the Ph.D. degree from the Institute National des Sciences Appliquees de Lyon, Lyon, France, in 2014. She currently works with the School of Information Science and Technology, ShanghaiTech University, specializing in medical image processing.



Xin Lou (Senior Member, IEEE) received the B.Eng. degree in electronic information technology and instrumentation from Zhejiang University (ZJU), China, in 2010, the M.Sc. degree in system-on-chip design from the Royal Institute of Technology (KTH), Sweden, in 2012, and the Ph.D. degree in electrical and electronic engineering from Nanyang Technological University (NTU), Singapore, in 2016. Then, he joined VIRTUS, IC Design Centre of Excellence, NTU, as a Research Scientist. In March 2017, he joined the School of Information Science and Technology, ShanghaiTech University, Shanghai, China, as an Assistant Professor and established the VLSI Signal Processing Laboratory. His research interests include energy-efficient integrated circuits for machine vision and 3D graphics, and other VLSI digital signal processing systems.