SpNeRF: Memory Efficient Sparse Volumetric Neural Rendering Accelerator for Edge Devices

Yipu Zhang[1], Jiawei Liang[1], Jian Peng[1], Jiang Xu[2], Wei Zhang[1,*]
[1]*Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology*
[2]*Microelectronics Thrust, The Hong Kong University of Science and Technology (GZ)*
{yzhangqg, jliangbr, jpengai}@connect.ust.hk, jiang.xu@hkust-gz.edu.cn, wei.zhang@ust.hk

***Abstract*—Neural rendering has gained prominence for its high-quality output, which is crucial for AR/VR applications. However, its large voxel grid data size and irregular access patterns challenge real-time processing on edge devices. While previous works have focused on improving data locality, they have not adequately addressed the issue of large voxel grid sizes, which necessitate frequent off-chip memory access and substantial on-chip memory.**

This paper introduces SpNeRF, a software-hardware co-design solution tailored for sparse volumetric neural rendering. We first identify memory-bound rendering inefficiencies and analyze the inherent sparsity in the voxel grid data of neural rendering. To enhance efficiency, we propose novel preprocessing and online decoding steps, reducing the memory size for voxel grid. The preprocessing step employs hash mapping to support irregular data access while maintaining a minimal memory size. The online decoding step enables efficient on-chip sparse voxel grid processing, incorporating bitmap masking to mitigate PSNR loss caused by hash collisions. To further optimize performance, we design a dedicated hardware architecture supporting our sparse voxel grid processing technique. Experimental results demonstrate that SpNeRF achieves an average 21.07× reduction in memory size while maintaining comparable PSNR levels. When benchmarked against Jetson XNX, Jetson ONX, RT-NeRF.Edge and NeuRex.Edge, our design achieves speedups of 95.1×, 63.5×, 1.5× and 10.3×, and improves energy efficiency by 625.6×, 529.1×, 4×, and 4.4×, respectively.

***Index Terms*—Neural Rendering, Software-Hardware Co-Design, ASIC**

I. Introduction

Neural Radiance Field (NeRF) [1] represents a novel and promising approach for 3D scene rendering, particularly attractive for Augmented and Virtual Reality (AR/VR) applications due to its high rendering quality and relatively high rendering speed on high-end GPUs. While state-of-the-art (SOTA) NeRF algorithms [2] achieve real-time performance on high-end GPUs, they still struggle to meet the required processing speed on edge computing platforms, such as Jetson Xavier NX (XNX) [3] and Jetson Orin NX (ONX) [4]. The irregular memory access pattern introduced by multi-resolution hash encoding in [2] has been identified as a major efficiency bottleneck in both rendering and training.

Several hardware-software co-design methods [5, 6, 7, 8] have been proposed to address rendering and training challenges in NeRF. Among these, Instant-3D [5] and Instant-NeRF [7] are ASIC accelerators focusing on improving data locality during NeRF training. Two latest ASIC neural rendering accelerators, NeuRex [6] and RT-NeRF [8], exemplify different approaches to tackling rendering efficiency. NeuRex enhances data locality through restricted hashing, dividing large hash tables into subtables, but fails to address the large data size issue. RT-NeRF leverages sparsity in TensoRF [9] weight matrices using hybrid encoding. However, its performance is constrained by additional matrix-vector multiplications and limited exploration of voxel grid sparsity.

These approaches leave the large memory size problem unresolved, leading to **frequent off-chip memory access** and **large on-chip memory requirements**, significantly impeding existing works' overall performance. Recent algorithm research has explored model compression techniques, such as VQRF [10], which aims to minimize memory size by identifying redundancies in voxel grid data and pruning less important points. While this approach shows promise in reducing model size, it introduces **new challenges** when deployed on edge computing platforms: **First**, the original VQRF consumes considerable memory during the rendering process. This is because VQRF adopts a method of restoring the full voxel grid from compressed data before processing, which leads to frequent off-chip memory access and increased latency, posing challenges in both power consumption and performance. To reduce the memory size, we propose a hash mapping based preprocessing, reducing the memory size significantly. **Second**, the irregularity introduced by ray sampling in VQRF complicates the process of locating voxel grid positions and fetching non-zero data from compressed encoded data. Existing encoding methods for Sparse Matrix Multiplication (SpMM) are not suitable for the irregular data access patterns in VQRF. To overcome this challenge, we integrate an online decoding step into our processing flow. **Third**, current edge computing platforms lack effective support for our proposed preprocessing and online decoding steps. To mitigate this issue, we design a dedicated hardware architecture optimized for these operations, ensuring peak performance.

To summarize, this work makes the following contributions:

- We propose SpNeRF, a software-hardware co-design framework leveraging sparsity in neural rendering to overcome the memory-bound bottleneck faced by previous works.
- We introduce a preprocessing step utilizing hash mapping to enhance memory mapping efficiency for sparse voxel

* Corresponding author.

978-3-9826741-0-0/DATE25/© 2025 EDAA

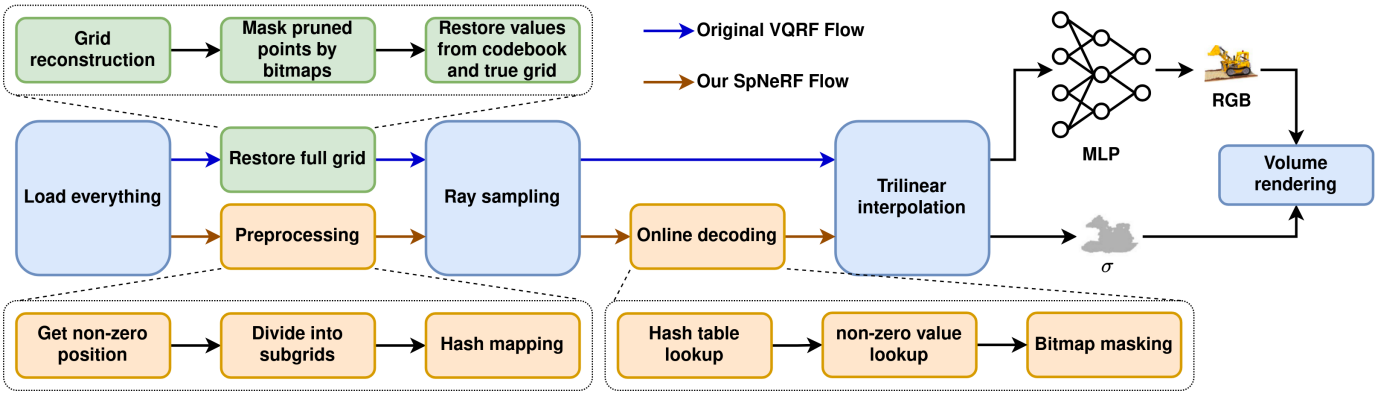


Fig. 1: Orginal VQRF flow and our SpNeRF flow

grids, minimizing the memory size for the voxel grid and simplifying non-zero data location in compressed structures during irregular memory access.

- We develop an online decoding step incorporating hash table lookup, non-zero value lookup, and bitmap masking, mitigating accuracy loss caused by hash collisions.
- We design a dedicated hardware architecture integrating a Sparse Grid Processing Unit (SGPU) and MLP Unit to efficiently support our SpNeRF algorithm, boosting sparse volumetric neural rendering efficiency.

II. Background and Motivation

A. Vanilla NeRF and VQRF

Vanilla NeRF [1] has revolutionized novel view synthesis by integrating computer graphics techniques with deep learning. As a pioneering work, NeRF introduced the use of deep neural networks to represent and render 3D scenes from a sparse set of input views. However, the original NeRF implementation relies on computationally expensive multi-layer perceptrons (MLPs) with a large number of parameters. Consequently, the rendering process can be time-consuming, often requiring hours or even days to generate a single novel view. Therefore, the volumetric method is proposed to boost training and rendering efficiency [11]. It estimated the color features and density of sampling points by interpolating the data stored in the voxel grid.

VQRF [10], as shown in Fig. 1, identifies the redundancy in voxel grid data and proposes voxel pruning and vector quantization to minimize the memory size. Also, VQRF has a relatively smaller MLP (only 3 layers with channel sizes of 128, 128, 3). However, it cannot fully take advantage of the sparsity brought by the voxel pruning since it requires restoring the full voxel grid using the pruned voxel grid points. Therefore, it leads to low efficiency when rendering on edge computing platforms, such as XNX and ONX.

B. Encoding Method

Various encoding methods have been proposed for SpMM to enhance memory and computation efficiency [12]. However, current approaches have limitations. The COO format requires storing all coordinates, which brings an extra 630 KB of memory usage for each scene on average in our experiments. CSR and CSC formats necessitate row-wise or column-wise data storage, respectively. While COO offers simple implementation but high memory overhead, CSR provides efficient row-wise access at the cost of poor column-wise performance, and CSC excels in column-wise operations but struggles with row-wise access. These encoding methods often result in excessive memory consumption and numerous lookups during irregular data access. Consequently, these factors lead to frequent off-chip memory accesses and the large on-chip memory demand, both of which significantly impact performance. A detailed analysis of these performance implications will be presented in Section II-C.

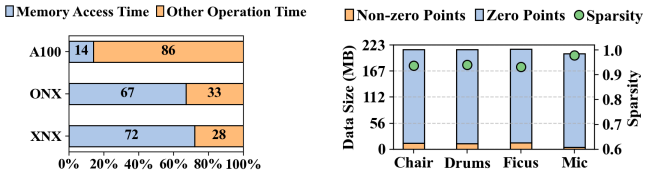
C. Profiling VQRF on GPUs

To identify VQRF bottlenecks, we profile the neural rendering process across various computing platforms and characterize sparse data in different datasets. We utilize one high-end computing platform, NVIDIA A100 (A100), and two edge computing platforms, ONX, and XNX, whose specifications are summarized in Table I. Evaluating VQRF on Synthetic-NeRF [1] datasets for each GPU, we present the runtime breakdown in Fig. 2(a). Profiling results reveal that edge computing platforms spend the most time accessing memory, while the A100 allocates minimal time to this task. We also examine voxel grid data redundancy by separating non-zero and zero components and calculate sparsity, as illustrated in Fig. 2(b).

TABLE I: A summary of profiling computing platforms

Spec.	A100 [13]	ONX [4]	XNX [3]
Tech.	7 nm	8 nm	16 nm
Power	400 W	25 W	20 W
DRAM	5120-bit 40 GB HBM2 1555 GB/s	128-bit 16 GB LPDDR5 102.4 GB/s	128-bit 16 GB LPDDR4 59.7 GB/s
GPU L2 cache	40 MB	4 MB	512 KB
FP32	19.5 TFLOPS	1.9 TFLOPS	885 GFLOPS
FP16	78 TFLOPS	3.8 TFLOPS	1.69 TFLOPS

Our profiling results yield two key observations: **First**, the rendering process on edge devices is predominantly memory bandwidth-bound. The proportion of time spent on memory access in edge computing platforms is $4.79\times \sim 5.14\times$ higher than in high-end computing platforms, due to the smaller



(a) Time distribution (b) Voxel grid data sparsity

Fig. 2: Profiling result for runtime on GPUs and sparsity on datasets

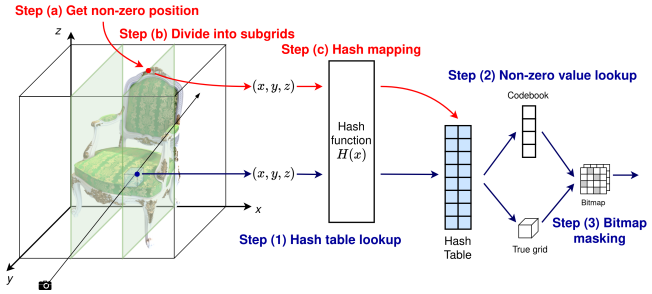


Fig. 3: An illustration of preprocessing and online decoding flow

L2 cache size and relatively low DRAM bandwidth of edge devices. **Second**, there is substantial redundancy in voxel grid data. As illustrated in Fig. 2(b), non-zero points occupy only $2.01\% \sim 6.48\%$ of total voxel grid data, indicating significant potential for leveraging sparsity to enhance efficiency. These findings underscore the need for an on-chip processing flow to mitigate expensive off-chip memory access and alleviate the memory bandwidth bottleneck.

III. Algorithm Design

In this section, we introduce the SpNeRF algorithm design, which enables efficient on-chip processing by fully exploiting sparsity to address the memory-bound problem. Our method replaces the conventional restore step with a preprocessing step and incorporates an online decoding step, which is shown in Fig. 1, significantly reducing memory bandwidth requirements and enabling efficient on-chip computation.

A. Hash Mapping Based Preprocessing

The preprocessing step encompasses three stages, as shown in the red line in Fig. 3. Initially, we identify the non-zero points in the sparse voxel grid and extract their position coordinates (x, y, z) into a vector $\boldsymbol{p} = [x, y, z]^T$. We then aggregate all these position vectors into a set $P_{nz} = \{\boldsymbol{p}_i | i = 1, 2, ..., N\}$, where N is the total number of non-zero points. This step preserves the spatial information and saves it for further voxel grid partition and hash mapping.

Subsequently, we partition the identified non-zero points into K subgrids based on their x coordinate values. This partitioning is defined as $S_k = \{\boldsymbol{p}_i | \lfloor x_i/w \rfloor = k, \boldsymbol{p}_i \in P_{nz}\}$, where $k \in 0, 1, ..., K-1$, w is the width of each subgrid, and x_i is the x-coordinate of $\boldsymbol{p}_i$.

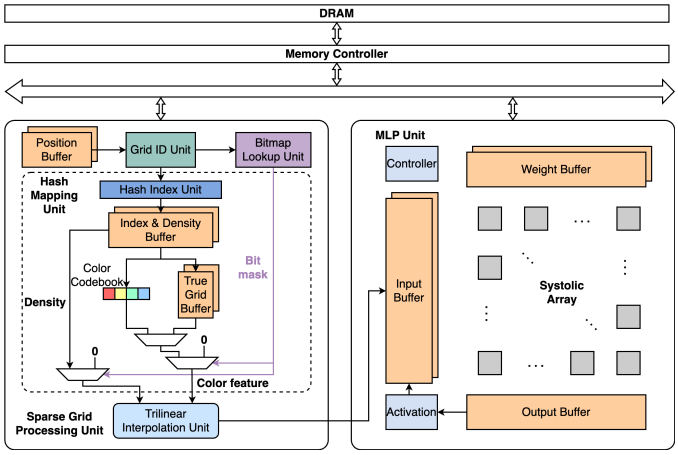


Fig. 4: SpNeRF accelerator architecture

Consequently, to efficiently support the irregular data access patterns inherent in neural rendering, we map each subgrid S_k into a separate hash table H_k using the following function from [2]:

$$h(\boldsymbol{p_i}) = (x_i\pi_1 \oplus y_i\pi_2 \oplus z_i\pi_3) \mod T \tag{1}$$

where T is the number of entries per hash table level, $\pi_1 = 1$, $\pi_2 = 2654435761$, and $\pi_3 = 805459861$. Note that each hash table entry stores the index for non-zero value lookup in online decoding. This final mapping process enables rapid lookup during the rendering process and eliminates the need for storing coordinates, thus solving the memory-bound problem and optimizing overall performance.

B. Online Sparse Voxel Grid Decoding

We incorporate an online decoding step between ray sampling and interpolation to support efficient data retrieval, as illustrated by the blue line in Fig. 3. This online decoding process unfolds as follows: Initially, for each sample point, we retrieve its position and calculate the corresponding hash index using (1). Subsequently, we utilize this hash index to fetch the lookup index for non-zero values from the hash table. Following this, we employ the retrieved 18-bit index to locate non-zero values in both the codebook and true voxel grid. To streamline this process, we implement a unified 18-bit addressing scheme for both the codebook and true voxel grid. Finally, we implement a bitmap masking technique to mitigate errors primarily caused by hash collisions. This approach utilizes a bitmap that stores a single bit for each voxel grid point, indicating whether the point is zero (0) or non-zero (1). By representing all voxel grid points' status in just 1 bit each, the bitmap provides a memory-efficient method to track non-zero values. During the decoding process, we consult this bitmap to effectively mask all erroneous values resulting from hash collisions, setting them to zero. Our observations indicate that hash collisions are the dominant source of errors in this process. Therefore, the bitmap masking step is crucial for maintaining accuracy in our decoding procedure. The detailed result for bitmap masking will be analyzed in Section V-B.

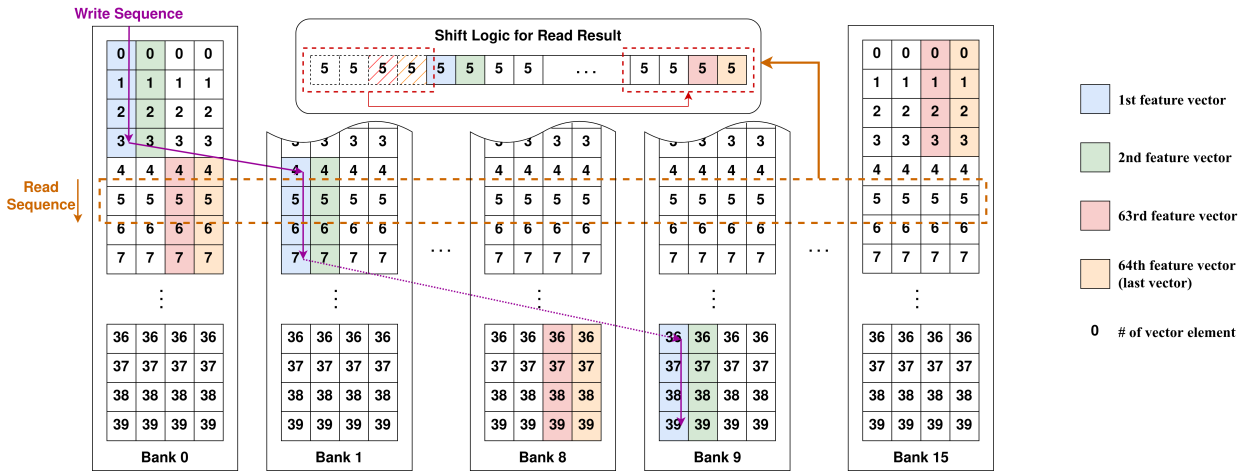


Fig. 5: Our proposed block-circulant storage format for input buffer

IV. Hardware Architecture Design

A. Overview

In this section, we present the SpNeRF architecture as illustrated in Fig. 4, designed to support our algorithm. The SpNeRF accelerator comprises two primary modules: the Sparse Grid Processing Unit (SGPU) and the MLP Unit. The SGPU is specifically tailored to execute the series of lookup operations required during online sparse voxel grid decoding. Complementing the SGPU, the MLP Unit is implemented as an output-stationary systolic array, a design commonly employed in conventional DNN accelerators. Our design method aligns with [6], extending a standard DNN accelerator with a specialized unit (in our case, the SGPU) to efficiently support neural rendering tasks. The overall dataflow of our system is initiated at the position buffer. Upon retrieval of a position, it is forwarded to the Hash Mapping Unit, which performs a lookup operation to locate the corresponding voxel grid data. The Trilinear Interpolation Unit subsequently processes this voxel grid data to compute the interpolation result. The resulting output is then concatenated with the view direction vector and stored in the input buffer of the MLP unit. The on-chip computing is in FP16 while the true voxel grid data is saved in INT8 format on off-chip memory to save the memory usage and reduce the communication overhead. To ensure high throughput, the entire design is fully pipelined. Furthermore, all buffers in the system are double-buffered, enabling simultaneous data fetching and processing.

B. Sparse Grid Processing Unit

The Sparse Grid Processing Unit (SGPU) is designed to support our proposed online sparse voxel grid decoding flow. It comprises four key components:

Grid ID Unit (GID). The GID computes ceiling and round results for each point position to locate the corresponding voxel grid vertex. It also calculates the weight by FP16 multipliers and subtractors for trilinear interpolation using the following equation introduced by [6]:

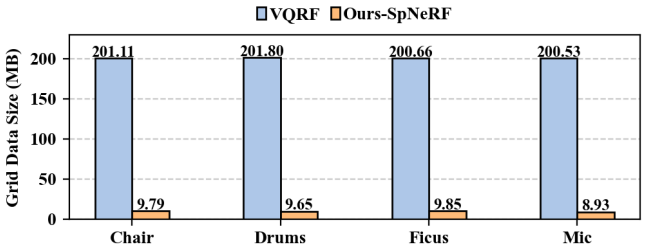
$$w = (1 - |x_p - x_g|) \cdot (1 - |y_p - y_g|) \cdot (1 - |z_p - z_g|) \quad (2)$$

where (x_p, y_p, z_p) represents the sample point position and (x_g, y_g, z_g) denotes the voxel grid vertex position.

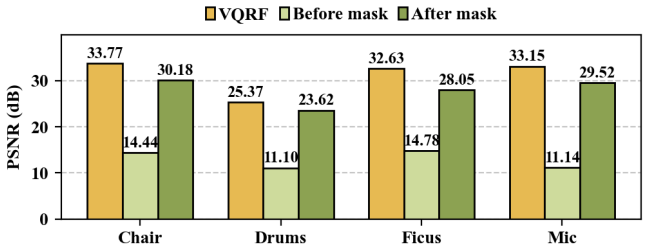
Bitmap Lookup Unit (BLU). The BLU stores the bit mask for the current subgrid. All bit masks are stored sequentially in a contiguous memory space, enabling efficient location of bits using voxel grid vertex positions as memory addresses and selection signals. The BLU lookup results are used to mask incorrect non-zero values caused by hash collisions in the Hash Mapping Unit.

Hash Mapping Unit (HMU). The HMU is the core module of the SGPU, facilitating the hash table lookup crucial for our online sparse voxel grid decoding. It processes the voxel grid vertex results from the GID and computes the hash index using Equation (1). Once the hash index is determined, the color feature index and density are fetched from the hash table stored in the Index and Density Buffer. Our unified 18-bit addressing scheme differentiates color codebook and true voxel grid buffer lookups by comparing the index value. For a color codebook size of 4096×12, lookup requests with color feature index values below 4096 are served by the color codebook, while others are directed to the true voxel grid buffer. The final output is filtered by the bit mask from BLU.

Trilinear Interpolation Unit (TIU). The TIU plays a crucial role in processing and interpolating color features. Initially, it converts the original color features from the true voxel grid buffer, stored in INT8 format, to FP16 format by multiplying the lookup results with the scale factor. Meanwhile, the density data and the color feature vector from the codebook will be directly sent to later computations. Following this conversion, the TIU multiplies each transformed color feature with its corresponding voxel grid vertex weight, as computed by the voxel grid ID Unit. Finally, it accumulates the weighted color features from all eight surrounding voxel grid vertices to produce the final interpolation result. This process can be represented by the equation $C_{interp} = \sum_{i=1}^{8} w_i \cdot (s \cdot C_i)$, where C_{interp} is the interpolation result, w_i are the weights for each voxel grid vertex, s is the scale factor for de-quantization, and C_i color feature vector in corresponding voxel grid vertex.



(a) Memory size reduction



(b) PSNR result

Fig. 6: Algorithm evaluation result for our proposed SpNeRF

C. MLP Unit

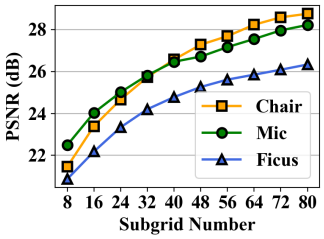
The MLP Unit consists of an output-stationary systolic array, activation unit, controller unit, and data buffers. It computes a 3-layer MLP with channel sizes of 128, 128, and 3, respectively. To enhance efficiency, we implement batch processing with a batch size of 64. Addressing the challenge of inconsistent vector sizes between the previous unit's output and the systolic array's input dimension, we propose a block-circulant storage format. This optimized memory pattern reduces both memory overhead and read time.

As illustrated in Fig. 5, the block-circulant storage format interleaves the input 39×1 vector across banks 0 to 9. Each input vector is segmented into 10 blocks, with each block containing four consecutive elements. Elements within the same block are stored successively, while adjacent blocks are stored in neighboring banks with a block offset of 4. The write sequence depicted in Fig. 5 demonstrates the process of writing the first vector into the input buffer. To ensure divisibility by 4, we pad the last element with 0. The reading sequence follows the vector element number in ascending order, with each read result undergoing a block shift to maintain the correct sequence of the first vector. As shown in Fig. 5, the current reading vector passes through shift logic, which repositions the first block from bank 0 to ensure that elements of the first vector are correctly aligned with the first row of the systolic array.

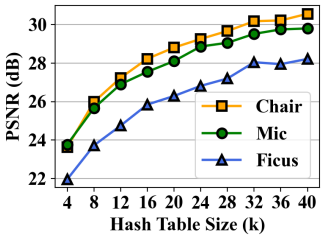
V. Evaluation

A. Evaluation Setup

Implementation. For Algorithm Evaluation, we implement our proposed SpNeRF algorithm based on VQRF [10] in PyTorch [14]. For Hardware Evaluation, we implement our accelerator in Verilog and synthesize our design using Synopsys Design Compiler based on TSMC 28nm CMOS Technology to obtain the power and area metrics. The operating clock frequency for our design is 1 GHz. On-chip SRAMs are generated by the provided memory compiler with the same technology. To evaluate the overall performance of our SpNeRF architecture, we develop a cycle-level simulator verified against our RTL design. The DRAM timing and power characteristics are obtained using Ramulator [15] with the configuration of LPDDR4-3200, which provides a bandwidth of 59.7 GB/s.



(a) Hash table size = 16 k



(b) Subgrid number = 64

Fig. 7: (a) PSNR vs. different subgrid number and (b) PSNR vs. hash table size

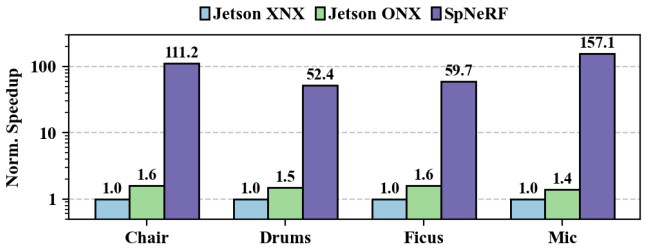
Datasets & Baseline. To evaluate the performance of the proposed SpNeRF, we conduct experiments on the Synthetic-NeRF [1] dataset. For the hardware evaluation baseline, we select the edge computing platforms and edge accelerators as our baselines. For edge computing platforms, we compare our design with Jetson Xavier NX 16 GB and Jetson Orin NX 16 GB (commonly used edge GPUs). For edge accelerators, we compare our design with RT-NeRF.edge and NeuRex.edge (both are dedicated ASIC accelerators for neural rendering).

B. Algorithm Evaluation

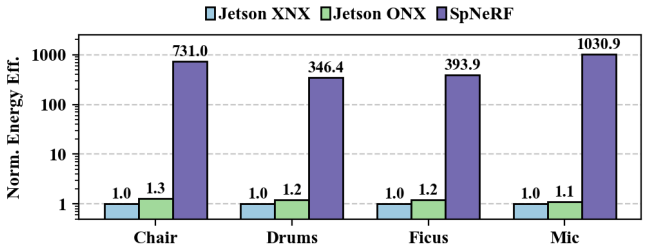
Subgrid Number & Hash table size. Fig. 7 illustrates the relationship among PSNR, subgrid number, and hash table size. The results reveal that PSNR increases rapidly initially but slow down beyond certain values for both subgrid number and hash table size. Based on this analysis, our design adopts a subgrid number of 64 and a hash table size of 32 k, as larger values yield only marginal improvements in PSNR.

Memory Size Reduction. Fig. 6(a) compares the voxel grid data size of our proposed SpNeRF with the original VQRF. SpNeRF achieves an average memory size reduction of $21.07\times$ compared to VQRF. This significant reduction in memory footprint is primarily attributed to the online sparse voxel grid decoding technique, which eliminates the necessity of restoring the full voxel grid data.

PSNR Performance. Fig. 6(b) illustrates the PSNR results for VQRF, SpNeRF before bitmap masking, and SpNeRF after bitmap masking. Higher PSNR values indicate better image quality. The results demonstrate that our SpNeRF, with bitmap masking applied, maintains PSNR levels comparable to VQRF while simultaneously achieving substantial memory

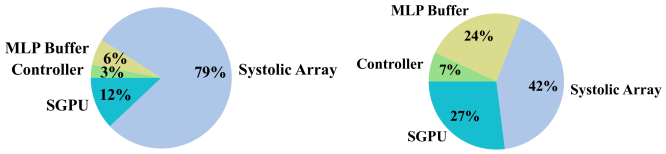


(a) Normalized speedup compared with XNX and ONX



(b) Normalized energy efficiency compared with XNX and ONX

Fig. 8: Normalized speedup and energy efficiency compared with edge computing platforms



(a) Area breakdown (b) Power breakdown

Fig. 9: Area and Power of SpNeRF

reduction. This indicates that the proposed method effectively preserves image quality despite the significant decrease in memory usage.

C. Hardware Evaluation

Performance. Fig. 8 shows the normalized speedup and energy efficiency compared with edge devices and Table II gives the comparison between our design and previous works. The results indicate that SpNeRF achieves great advances in both speedup and energy efficiency. In comparison, SpNeRF achieves $52.4\times \sim 157.1\times$ speedup compared with XNX and $34.9\times \sim 112.2\times$ speedup compared with ONX. This improvement is attributed to hash mapping adopted in preprocessing and online decoding, eliminating the need for frequent off-chip memory access.

Area. Fig. 9(a) illustrates the area breakdown of our SpNeRF accelerator, while Table II presents the total area and area efficiency of our design. In the area breakdown, the MLP buffer accounts for 58 KB SRAM (comprising input buffer, output buffer, and weight buffer), and the SGPU contains 571 KB SRAM. Notably, in our design, on-chip SRAM occupies only a small fraction of the overall area. This contrasts with other designs, where on-chip SRAM typically dominates the chip area. The results demonstrate that the memory reduction achieved through online sparse voxel grid decoding effectively minimizes on-chip SRAM size, leading to superior area efficiency of $2.67\times$ to $3.04\times$ compared to previous works.

Power. Fig. 9(b) depicts the power breakdown of our SpNeRF accelerator and Table II provides the total power and power efficiency of our design. The power breakdown reveals that the systolic array accounts for the dominant portion of overall power consumption in our design, contrasting with previous studies where on-chip SRAM was the primary power consumer. Our approach achieves a $4\times$ to $4.37\times$ improvement

TABLE II: Summary of comparisons between related work and our SpNeRF

Accelerator	RT-NeRF [8]	NeuRex [6]	SpNeRF (Ours)
SRAM (MB)	3.5	0.86	0.61
Area (mm^2)	18.85	1.31	7.7
Tech. (nm)	28	28	28
Power (W)	8	1.31	3
DRAM	LPDDR4-1600 17 GB/s	LPDDR4-3200 59.7 GB/s	LPDDR4-3200 59.7 GB/s
FPS	45	6.57*	**67.56**
Energy Eff. (FPS/W)	5.63	5.15	**22.52**
Area Eff. (FPS/mm^2)	2.38	2.09	**6.36**

* NeuRex only provides normalized speedup. Here we infer from Jetson XNX rendering speed.

over previous works. When comparing with XNX and ONX, as shown in Fig.8(b), SpNeRF achieves $346.4\times \sim 1030.9\times$ and $288.7\times \sim 937.2\times$ energy efficiency improvement, respectively. This advantage comes from two key factors: First, our design minimizes on-chip SRAM size, which has been observed to be a dominant contributor to power consumption in previous works. Second, our design fully leverages voxel grid sparsity, significantly reducing off-chip memory access, another major source of power consumption.

VI. Conclusion

This paper presents SpNeRF, a novel software-hardware co-design approach for facilitating memory efficiency in sparse volumetric neural rendering. We propose hash mapping-based preprocessing and online sparse voxel grid decoding techniques to fully leverage the sparsity in voxel grid data. To support our algorithmic innovations, we introduce a dedicated hardware architecture designed for efficient neural rendering. Experimental results demonstrate that our design significantly outperforms edge computing platforms and previous works in terms of speedup, energy efficiency, and area efficiency. These improvements make SpNeRF a promising solution for advancing the field of neural rendering, particularly in resource-constrained environments.

Acknowledgment

This work was partially supported by AI Chip Center for Emerging Smart Systems (ACCESS), Hong Kong SAR and Collaborative Research Fund (UGC CRF) C5032-23G.

REFERENCES

[1] 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 *European Conference on Computer Vision*, Springer, 2020, pp. 405–421.

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM transactions on graphics (TOG)*, vol. 41, no. 4, pp. 1–15, 2022.

[3] M. Ditty, A. Karandikar, and D. Reed, "Nvidia's xavier soc," in *Hot chips: a symposium on high performance chips*, 2018.

[4] M. Ditty, "Nvidia orin system-on-chip," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, IEEE Computer Society, 2022, pp. 1–17.

[5] S. Li *et al.*, "Instant-3d: Instant neural radiance field training towards on-device ar/vr 3d reconstruction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[6] J. Lee, K. Choi, J. Lee, S. Lee, J. Whangbo, and J. Sim, "Neurex: A case for neural rendering acceleration," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[7] Y. K. Zhao, S. Wu, J. Zhang, S. Li, C. Li, and Y. C. Lin, "Instant-nerf: Instant on-device neural radiance field training via algorithm-accelerator co-designed near-memory processing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2023, pp. 1–6.

[8] C. Li, S. Li, Y. Zhao, W. Zhu, and Y. Lin, "Rt-nerf: Real-time on-device neural radiance fields towards immersive ar/vr rendering," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[9] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," in *European conference on computer vision*, Springer, 2022, pp. 333–350.

[10] L. Li, Z. Shen, Z. Wang, L. Shen, and L. Bo, "Compressing volumetric radiance fields to 1 mb," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4222–4231.

[11] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 651–15 663, 2020.

[12] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[13] J. Choquette and W. Gandhi, "Nvidia a100 gpu: Performance & innovation for gpu computing," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, IEEE Computer Society, 2020, pp. 1–43.

[14] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[15] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.