A Neural Rendering Coprocessor With Optimized Ray Representation and Marching

Zhechen Yuan, *Student Member, IEEE*, Binzhe Yuan, Chaolin Rao, Yiren Zhu, Yunxiang He, Pingqiang Zhou, *Member, IEEE*, Jingyi Yu, *Fellow, IEEE*, and Xin Lou, *Senior Member, IEEE*

***Abstract*—Neural rendering, a transformative approach for 3-D scene reconstruction and rendering, has advanced rapidly in recent years. This article introduces an energy-efficient neural rendering coprocessor that implements the popular and widely used instant neural graphics primitive (Instant-NGP) algorithm. In particular, we address the challenges of limited resources for deploying Instant-NGP on edge by proposing a dedicated architecture, which incorporates three main innovations: 1) we optimize occupancy grid queries in the ray marching module by partitioning the grid and decoupling the query process from sampling point generation, which improves both efficiency and memory usage; 2) we introduce a bilinked list-based ray switching strategy, which ensures continuous pipeline utilization to overcome the inefficiencies caused by sequential processing; and 3) we optimize the hash encoding process by incorporating quantization-aware training (QAT), enabling the hash table to fit into on-chip memory, thereby improving performance on resource-constrained devices. To demonstrate the effectiveness of our architecture, we design and fabricate a proof-of-concept chip using 40-nm CMOS technology and develop a testing system to evaluate its performance. Measurement results validate the advantages of the proposed design, showing that our chip achieves superior energy efficiency compared to both server and edge graphics processing units (GPUs), as well as other state-of-the-art neural rendering chip designs.**

***Index Terms*—Coprocessor, energy-efficient, instant neural graphics primitive (Instant-NGP), neural rendering, ray marching.**

I. Introduction

RENDERING realistic scenes in virtual environments has long been a significant challenge across various applications, including video games, augmented reality (AR), virtual reality (VR), and other graphics-related fields. For these applications, rendering has to be performed in real time to ensure a smooth and immersive user experience. Generally, there are two widely used rendering techniques, i.e., the rasterization pipeline and the ray tracing pipeline [1]. The rasterization graphics pipeline is the cornerstone of real-time rendering, which is widely used in many applications such as video gaming and interactive simulations. This technique involves decomposing objects into primitives, usually triangles, which are then rasterized into fragments before applying color. As rasterization processes each triangle independently and in parallel, it is highly efficient and well-suited for hardware acceleration, making it ideal for applications that require high frame rates and responsiveness. However, such a method fails to achieve photorealistic rendering quality. Ray tracing, on the other hand, offers a more realistic rendering approach by simulating the way light interacts with objects in a scene. Unlike rasterization that starts from the object's perspective, ray tracing begins from the camera's viewpoint. Rays are cast from the camera into the scene, tracing their paths as they encounter surfaces. This method can accurately render complex optical effects such as reflections and refractions by tracing secondary rays as they bounce between objects. Although ray tracing offers high visual fidelity, it is computationally intensive. As a result, real-time applications have been limited in their use of ray tracing, especially in edge devices.

Deep learning has recently emerged as a transformative force in computer graphics, particularly in 3-D object modeling and scene rendering [2], [3]. With the advent of neural radiance fields (NeRFs) [4], neural volume rendering (NVR) has rapidly evolved into a prominent method, leveraging deep neural networks (DNNs) to reconstruct 3-D scenes and provide photorealistic rendering in an efficient manner. By encoding scenes and objects within the weights of a DNN, NVR methods implicitly map input coordinates to certain values such as color or radiance. Compared to traditional explicit 3-D representations such as polygon meshes, voxels, or point clouds, implicit neural scene representations are capable of capturing the details of complex surfaces and shapes in a more compact manner. However, due to its enormous computational demands that require a large number of sampling points and extensive multilayer perceptron (MLP) evaluations, the original NeRF algorithm could only achieve a rendering speed of 0.03 frames per second (FPS) at 800×800 resolution on a high-end V100 GPU [4].

To accelerate NeRF and its variants, various algorithmic-level optimizations have been developed [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. NSVF [5] employs a sparse voxel octree structure to bypass empty regions of a scene, significantly speeding up the rendering of new views and achieving a more than tenfold increase in inference speed compared to vanilla NeRF. DeRF [7] spatially decomposes the scene and allocates smaller networks to each decomposed space to

Received 1 March 2025; revised 1 May 2025; accepted 16 May 2025. This work was supported in part by Shanghai Fundamental Research Program under Grant 24JD1402300, in part by the Central Guided Local Science and Technology Foundation of China under Grant YDZX20223100001001, and in part by the National Key Research and Development Program of China under Grant 2023YFB4404000. *(Corresponding author: Xin Lou.)*

Zhechen Yuan, Binzhe Yuan, Yiren Zhu, Yunxiang He, Pingqiang Zhou, and Jingyi Yu are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China.

Chaolin Rao and Xin Lou are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China, and also with GGU Technology Ltd., Shanghai 201210, China (e-mail: louxin@shanghaitech.edu.cn).

Digital Object Identifier 10.1109/TVLSI.2025.3572959

reduce computational load. A similar idea is applied in KiloNeRF [8] to accelerate rendering by replacing a large MLP with thousands of smaller networks, achieving a 1000-fold increase in rendering speed. FastNeRF [9] uses GPU caches to store the relationship between sampling point locations and the radiance field, substituting network inference computations for retrieval. PlenOctrees [10] incorporates an octree structure to store the radiance field to avoid neural network evaluation and achieves up to 150 FPS at 800×800 resolution. While these methods have succeeded in accelerating NeRF rendering, they depend heavily on the substantial cache of existing GPUs for swift neural rendering or require extensive storage space to save parameters or data structures. Instant neural graphics primitive (Instant-NGP) [15] employs hash encoding for efficient neural rendering, requiring only 28-MB storage while achieving higher PSNR and real-time inference. In addition to these optimization efforts targeting NeRF's performance, several studies have integrated explicit 3-D representations with neural networks to achieve superior rendering quality and speed. However, the storage requirements of these explicit 3-D model representations remain significant. Taking the most representative Point-NeRF [16] as an example, Point-NeRF employs point cloud data as input and utilizes an MLP to predict the color and density of each point while maintaining the same rendering pipeline as NeRF. However, the parameters for a single scene (including both the point cloud and network weights) can reach up to 72 MB, much larger than Instant-NGP's. Moreover, its rendering speed and quality are both inferior to Instant-NGP.

In addition to algorithmic optimizations, dedicated NVR accelerators have also been proposed to solve the rendering efficiency bottleneck. As a pioneering work, ICARUS [17] introduced the first NeRF accelerator based on the original NeRF algorithm. However, the massive number of sampling point computations and the large size of the MLP network hinder it from achieving real-time rendering. As a follow-up work, MetaVRain [18], [19] proposed improvements by exploiting frame-to-frame correlations and storing intermediate MLP network computations from one frame to reuse in the next, significantly reducing the computational load. Although MetaVRain accelerates rendering speed by utilizing information between frames, as the rendering resolution increases, a large amount of intermediate network data needs to be stored. These data cannot be fully stored in on-chip SRAM and must be frequently exchanged with double data rate (DDR) memory, resulting in significant performance overhead. In addition, when the viewpoint changes rapidly, it becomes difficult to effectively utilize information between frames, leading to a further decline in performance.

The proposed coprocessor exhibits even more pronounced advantages in terms of energy efficiency. Compared to the Jetson Xavier NX [20], our design consumes $264.14\times$ less power. In addition, it uses $15.46\times$ less power than MetaVRain, a significant reduction that underscores the efficiency of our architecture for power-sensitive applications. This indicates that our accelerator not only achieves high performance but does so with minimal energy requirements. In this article, we first deploy Instant-NGP, on both desktop and edge GPUs to identify the bottlenecks affecting its rendering speed. Our analysis revealed that the two most time-consuming stages are: 1) the ray marching module, responsible for generating sample points and 2) the hash encoding module, which encodes the information of these sample points. We investigate the root causes of these bottlenecks and propose corresponding solutions, which can be summarized as follows.

1) The serial querying of the occupancy grid for each sampling point causes inefficiency. To address this, we partition the grid and use multistep ray marching, decoupling the grid query from point generation. This reduces memory overhead and improves query efficiency.
2) The ray marching, encoding, and network computation modules execute sequentially, and when a ray early terminates, it stalls the pipeline, reducing overall running efficiency. We propose a batch processing mechanism along with a bilinked list-based ray switching strategy to keep the pipeline active continuously, improving throughput.
3) Hash encoding is time-consuming, especially on devices with limited on-chip memory such as Jetson Xavier NX. We incorporate quantization-aware training (QAT), enabling the hash table to fit into on-chip memory, thereby minimizing off-chip memory accesses and improving performance.

The article is organized as follows. Section II benchmarks NeRF against Instant-NGP, highlighting the latter's advantages and analyzing its hardware bottlenecks across platforms, which motivates our solutions. Sections III–V elaborate on three targeted optimization strategies. Section VI presents the unified accelerator architecture. Section VII validates the design through silicon measurements and proposes a Multi-MAC variant, achieving higher throughput and energy efficiency. Section VIII concludes with research implications.

II. Background and Bottleneck Analysis

A. NeRF and Instant-NGP

Fig. 1(a) illustrates the rendering process in the original NeRF algorithm. For a ray corresponding to the rendering pixel, 64 sample points are uniformly sampled first, known as coarse sampling. The coordinate information (x, y, z) and directional information (θ, ϕ) of these sampled points are encoded to high-dimensional vectors through a positional encoding. The encoded vectors are subsequently processed by an MLP network to compute the density and color of each sample point. Following the coarse sampling, the region with high-density sample points is selected, and 128 additional points are further sampled within this region, known as fine sampling. These finely sampled points undergo the same process as the coarsely sampled points to compute their corresponding density and color. Finally, the volume rendering module integrates the density and color values of all 192 sample points (coarse and fine) to generate the final pixel color.

Fig. 1(b) illustrates the rendering pipeline of Instant-NGP. The improvements over the original NeRF can be summarized in the following two key aspects.

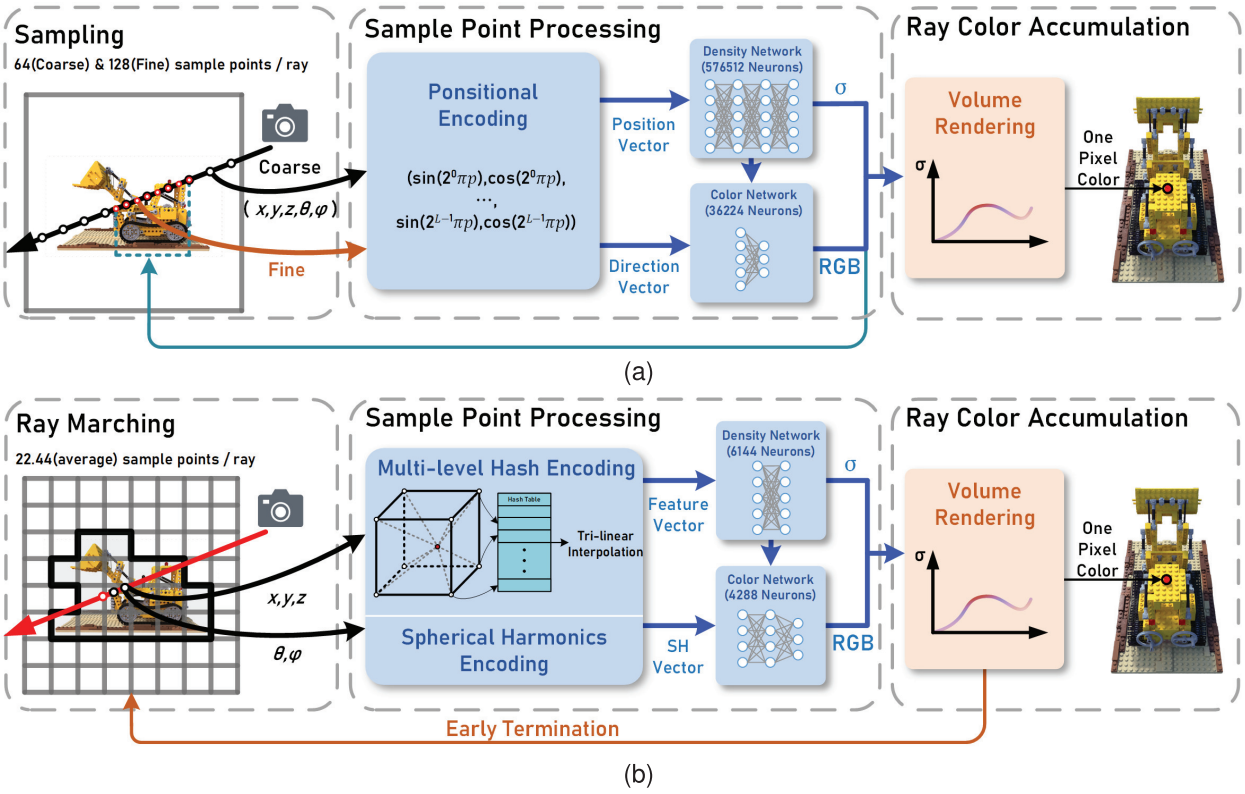


Fig. 1. Comparison between the NeRF and Instant-NGP rendering pipeline. (a) NeRF: coarse and fine uniform sampling, positional encoding, and a large MLP network. (b) Instant-NGP: specialized sampling combined with an occupancy grid, multiresolution hash encoding, and a smaller MLP network.

1) *Ray Marching With Early Termination:* Instant-NGP trains a binary occupancy grid for efficient point sampling. For a ray that traverses the binary occupancy grid, it generates sample points only within valid grids labeled by 1 s. In addition, when computing the RGB value of a pixel through volume rendering, the ray marching process terminates early if the ray's opacity reaches a predefined threshold that is close to 1, indicating that subsequent sample points contribute little to the ray's color. These two techniques in Instant-NGP significantly reduce the computational overhead associated with non-contributory sample points.
2) *Hash Encoding With Smaller MLP:* Instant-NGP replaces the traditional positional encoding used in NeRF with a multiresolution hash table-based encoding scheme. First, the eight vertex coordinates of the sampling point's grid cell are transformed through a hash function into eight address indices. These indices are then used to retrieve eight corresponding values from the hash table. Finally, these values undergo trilinear interpolation to generate the network's input features. This encoding efficiently maps spatial coordinates to feature vectors stored in a hash table. The hash table allows the model to store and retrieve high-frequency details without requiring the MLP to learn these details implicitly. As a result, the MLP only needs to focus on blending and interpreting these features, allowing it to be much smaller and more efficient. This innovation makes the model more efficient without sacrificing quality.

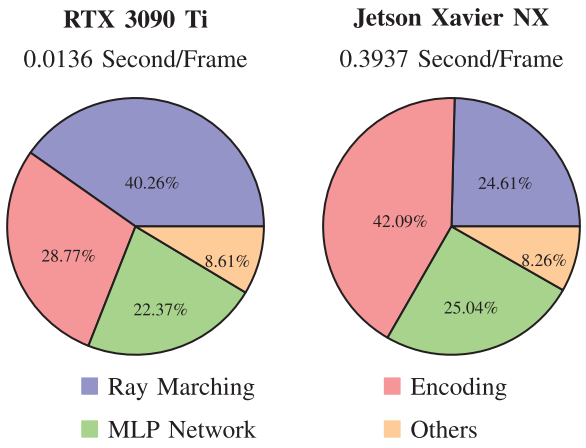


Fig. 2. Time consumption breakdown for Instant-NGP rendering on RTX 3090Ti and Jetson Xavier NX with total time values for both platforms.

B. Bottleneck Analysis

To identify the bottlenecks in Instant-NGP, we first deploy the algorithm on a desktop GPU, the NVIDIA RTX 3090 Ti. The time consumption breakdown is illustrated in Fig. 2. As shown, the ray marching and hash encoding modules account for approximately 70% of the total time consumption. We also deploy the same algorithm on an NVIDIA edge GPU, the Jetson Xavier NX, where the time consumption of the hash encoding module increases to 42.09%. These bottlenecks can be attributed to the following three main factors.

1) Occupancy Grid Queries: In ray marching, each time a ray enters a new occupancy grid, the grid value is queried to determine the validity of the sampling point. The tight coupling between sampling point generation and occupancy grid queries creates a dependence, forcing each grid query to finish before the next sampling point can be generated. In addition, traversing empty regions causes further delays and degrades throughput. The core issue lies in the need to query the occupancy grid for every sampling point.

2) Hash Encoding: As illustrated in Fig. 2, the hash encoding module accounts for a significant portion of total processing time. In particular, the time spent on the hash encoding exceeds nearly half of the total on edge devices. For further investigation, we measure the L1 and L2 cache hit rates. Our findings reveal that the Jetson Xavier NX, constrained by its limited on-chip SRAM resources, heavily depends on interactions with external DDR memory, resulting in significant delays. Specifically, the L1 cache hit rate on the Jetson Xavier NX is 64.05%, and the L2 cache hit rate is 32.98%. In comparison, the RTX3090Ti achieves L1 and L2 cache hit rates of 83.81% and 71.89%, respectively. This observation is further corroborated by the experiments in [21], which also emphasize the substantial performance impact of frequent DDR accesses on devices with limited on-chip memory resources.

3) Sequential Processing: In the original implementation, the ray marching, encoding, and network computation modules operate sequentially. Each sampling point must be fully processed before the ray advances to query the occupancy grid. This approach introduces inefficiencies, as it leaves other modules idle during processing, resulting in suboptimal utilization of system resources and diminished overall performance.

C. Proposed Solutions

To address the bottlenecks mentioned above, we propose the following three solutions, each targeting a specific problem.

1) *Multistep Advance-Based Ray Marching With Optimized Occupancy Grid Queries:* In the ray marching module, a major bottleneck is the time required to access the occupancy grid for each sampling point to verify its validity. To address this, we partition the occupancy grid based on spatial structure, enabling adjacent lookup requests to be completed within the same cycle. In addition, by employing a **multistep advanced-based ray marching strategy** for a single ray within one cycle, we achieve both efficient and rapid ray traversal. Moreover, we decouple the occupancy grid query process from sampling point generation. The entire scene is quickly traversed to query the occupancy grid, compressing the results into a representation of **nonempty interval** areas. This representation records only the start and end points of valid areas, significantly reducing memory overhead compared to storing the coordinates of all sampling points. The compressed data are then integrated with a doubly linked list-based ray state management system, dynamically generating sampling points for active rays within a single cycle, ensuring full utilization of the sampling point processing pipeline.

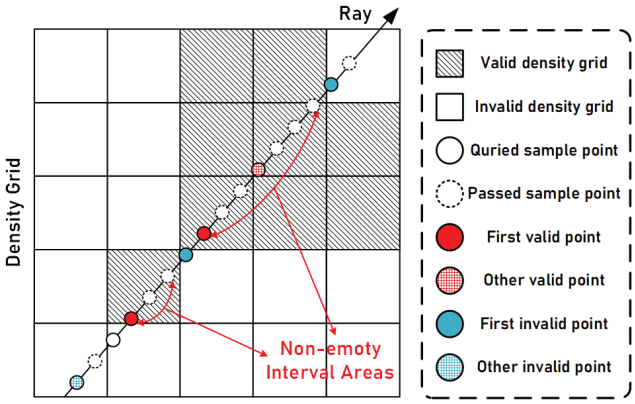


Fig. 3. Core concept of the proposed multistep ray marching, enabling multistep ray processing within a single cycle. Only the first sample point that enters the grid generates a lookup request, and the results are divided into four categories.

2) *Optimized Hash Encoding With Limited On-Chip Memory:* To mitigate the hash encoding bottleneck, particularly on resource-constrained edge devices, we reduce the upper limit of the hash table to fit entirely within the on-chip SRAM. This eliminates the need for frequent external DDR memory accesses and significantly reduces latency. We also applied QAT to lower the bit width of the hash table. This further reduces storage requirements, allowing the hash table to operate efficiently within the available SRAM without significantly affecting rendering quality.

3) *Batch Processing With Bilinked List-Based Ray Switching Strategy:* To address the inefficiencies caused by sequential ray marching, we introduce batch processing with a bilinked list-based ray switching strategy. Instead of processing a single ray continuously, rays are processed in batches, generating sampling points in turn. Once a sampling point is passed to subsequent modules for processing, the ray marching module switches to another ray to step forward. This approach ensures that the pipeline remains active, even when some rays terminate early. In addition, we use a bilinked list-based ray switching strategy to dynamically manage active rays, storing the indices of the previous and next active rays for efficient updates. This design enables seamless pipelined execution and minimizes idle time in the pipeline.

By implementing the above solutions, we achieve substantial improvements in pipeline efficiency, memory utilization, and overall system performance while maintaining high rendering quality.

III. Multistep Advance-Based Ray Marching With Multiblock Occupancy Grid

Fig. 3 illustrates the core design of the ray marching module, enabling multistep ray processing within a single cycle. A query request is triggered only when the first sample point enters a new density grid, while subsequent points are skipped. Sample points are categorized into four types based on query

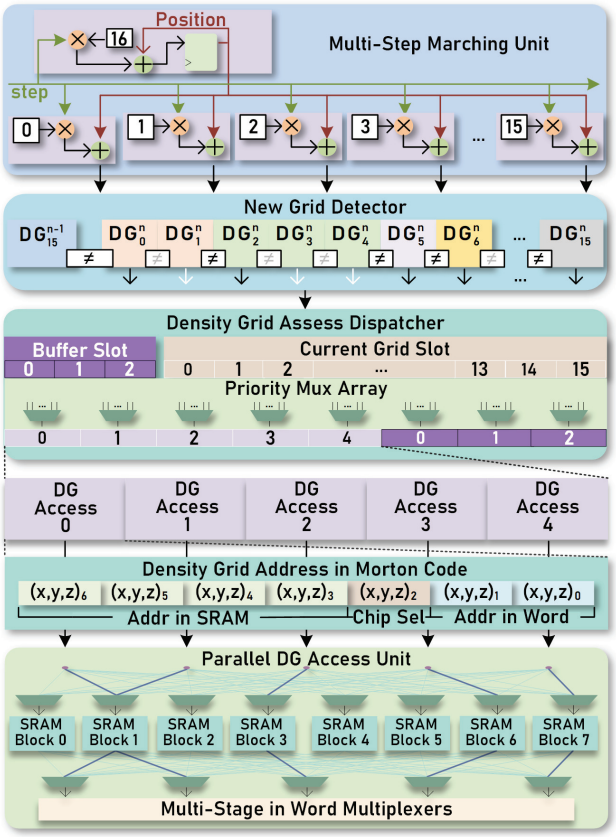


Fig. 4. Architecture of the proposed multistep ray marching module, along with the implementation details for density grid access requests' generation.

validity, with only the "first valid point" (start) and the point before the "first invalid point" (end) being recorded and stored.

A. Multistep Ray Marching

Fig. 4 shows multistep ray marching implementation and density grid access request generation. This unit computes multiple adjacent sample points in parallel using multiply-add units, with the starting point updated next cycle. Ray lengths are computed alongside sample coordinates to check if a point exits the axis-aligned bounding box (AABB), terminating computation if so. In our implementation, we choose to process 16 steps in parallel to match the sampling point generation speed of the ray stepping module with the processing speed of the subsequent sample point processing module. This configuration ensures that the system operates efficiently without overloading the hardware. Increasing the number of parallel steps beyond 16 results in minimal performance gains and only leads to increased hardware consumption.

After computing coordinates of adjacent sample points, bit shifting and truncation derive their spatial occupancy grid indices, which query grid memory to check valid samples. To avoid redundancy, queries are triggered only when adjacent indices differ, ensuring a single access per cell. This reduces unnecessary memory access and boosts ray tracing efficiency.

In the proposed multistep ray marching, the number of access requests generated per cycle may vary. To manage this variability, we introduce a density grid assess dispatcher,

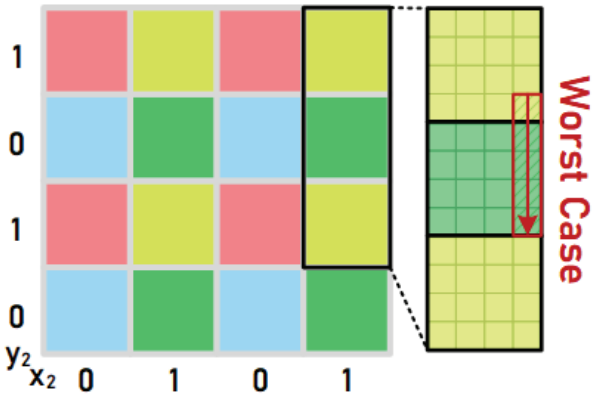


Fig. 5. Specialized multiblock density grid bitmap in 2-D space and a corresponding diagram illustrating maximum queries handled in the worst case scenario.

as shown in Fig. 4, to improve the utilization of subsequent occupancy grid access units and minimize pipeline stalls. The dispatcher collects valid query indices from adjacent sampling points and forward as many queries as possible to the back-end pipeline. If the number of queries exceeds the back end's processing capacity, surplus requests are temporarily stored and combined with new requests in the next cycle. If the total number of requests in a cycle, including those temporarily stored in the buffer, exceeds 8, the pipeline will stall for one cycle. However, this scenario occurs with a probability of only 0.003% when processing 16 steps each cycle.

To enable parallel processing of multiple adjacent queries, we use Morton encoding to store the spatial density grid bitmap in multiply blocks. The transformed address is divided into three segments: the low segment identifies a specific bit within a word in the SRAM, the middle segment selects a block across multiple SRAMs, and the high segment indexes different words within a single SRAM. Fig. 5 shows the specialized multiblock density grid bitmap in 2-D space and a corresponding diagram illustrating the maximum queries handled in the worst case scenario. Blocks of the same color are stored within the same SRAM, with each color block containing 64 density grid bits. In the worst case scenario, a ray progresses along one axis, beginning with a query at the edge of one color block, followed by four queries in an adjacent block. If a sixth block query is needed, it would conflict with the first block's query. To avoid such conflicts, we instantiated five density grid access units in hardware, enabling conflict-free density grid queries each cycle. This organization ensures that the target data for adjacent queries up to a certain number, i.e., either reside within the same word of a specific SRAM or span different SRAMs, effectively avoiding bank conflicts within the memory.

Since the density grid serves as a filter for sampling points, a coarser-grained density grid would lead to an increased number of sampled points, thereby slowing down rendering speed, while affecting final rendering quality slightly. Table I evaluates the impact of different density grid sizes on PSNR, the number of sampled points, and memory consumption. The results indicate that while larger grid densities improve PSNR

TABLE I

PSNR, THE NUMBER OF SAMPLED POINTS, AND MEMORY CONSUMPTION WITH DIFFERENT DENSITY GRID SIZES

Density Grid Size	32^3	64^3	128^3	256^3
PSNR (dB)	31.03	33.32	33.89	34.01
Number of Sample Points (x10^7)	1.53	0.82	0.47	0.34
Density Grid Memory Cost (MB)	0.004	0.032	0.25	2.0

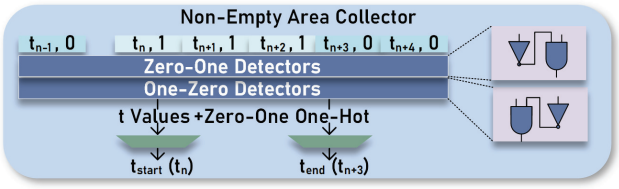


Fig. 6. Detailed architecture of the NEAC, which facilitates querying the density grid and retrieving sample point data.

and reduce computation time, they require exponentially more memory. At size 256^3 (versus 128^3), the PSNR/computation benefits diminish, while memory reaches 2 MB. Therefore, considering the tradeoff among computational cost, memory consumption, and PSNR, we adopt the same size as in the original Instant-NGP: 128^3.

B. Nonempty Interval Ray Presentation

Direct sample point generation and processing cause delays as density grid queries disrupt the continuous generation of sampling points, causing downstream stalls. Storing all points incurs high overhead, especially for terminated noncontributing points. Our solution records only interval start/end areas, ensuring fast regeneration while saving storage. This strategy seamlessly integrates with batch-processing sample point processing module, enabling efficient sequential-to-parallel ray processing transition.

As illustrated in Fig. 6, the detailed architecture of the nonempty area collector (NEAC) implements the proposed nonempty interval ray representation for querying the density grid and retrieving sampling point data. The results of empty and nonempty queries form a binary sequence, where the nonempty interval collector identifies edges in the sequence and represents the nonempty intervals using the ray lengths of their start and end points. After completing the density grid queries, it is essential to store information about valid sample points along each ray for subsequent encoding and neural network computations. To optimize memory usage, the spatial range representing the distribution of valid sample points along each ray is stored, rather than the coordinates of individual sample points. Specifically, the starting and ending ray lengths of these intervals are used to indicate the spatial range of valid sample points. Since the number of intervals per ray may vary, a dynamic storage approach is adopted. Instead of allocating fixed-size storage for each ray, interval data are efficiently organized in memory, with a dedicated memory block assigned to record the start and end addresses of the interval data for each ray.

C. Filtered Occupied Grid

Fig. 7(a) illustrates the density grid of the LEGO dataset [4], which is utilized for ray marching and provides a partial representation of the scene's 3-D features. Notably, fog-like regions appear around the scene. Sample points within these regions, after processing through the density network, yield very low densities (below 0.01), thereby contributing negligibly to the final RGB values of the pixels.

The presence of these fog-like areas can cause rays passing through them to be erroneously identified as valid, increasing the computation of ineffective sample points. For rays traversing these regions before reaching actual scene elements, this leads to the premature accumulation of a substantial number of ineffective sample points.

To address this issue, a preprocessing step is applied to the density grid to eliminate fog-like areas. Grid cells with values below a fixed threshold are filtered out, as illustrated in Fig. 7(b). This reduction has minimal impact on the final rendering quality, resulting in only a 0.01 decrease in PSNR. By filtering these grids, the number of valid ray evaluations is reduced by 3.8%, and sample point computations are reduced by 1.79%, thereby enhancing overall rendering efficiency without compromising image quality.

With the filtering of the density grid, it is no longer necessary for rays to traverse the entire original bounding box, which would consume considerable time stepping through empty, noninformative regions. To optimize this process, we define a new AABB based on the valid regions within the filtered density grid. As illustrated in Fig. 7(c), the AABB allows most rays to traverse only within the relevant region, from T_s' to T_e', rather than across the full bounding box from T_s to T_e. Rays that do not intersect with the AABB can be identified as invalid, thus avoiding unnecessary computation. This approach accelerates the ray marching module by an average of 117.4%.

IV. BATCH PROCESSING WITH A BILINKED LIST-BASED RAY SWITCHING STRATEGY

To improve sequential ray marching efficiency, we implement batch processing using nonempty interval ray representation. It processes rays in batches, sequentially generating all sampling points by accessing nonempty interval memory. After sending a point for processing, the module instantly switches to the next active ray.

To manage active rays within a batch, we implement a bilinked list-based ray switching strategy. This structure records the status of all rays in the batch, storing the indices of the previous and next active rays to enable efficient updates. This dynamic management ensures that terminated rays are quickly removed from the active list, allowing the pipeline to remain active even when some rays finish early.

By combining the nonempty interval representation and the bilinked list-based management, our design achieves seamless pipelined execution, minimizing idle time in the pipeline and ensuring efficient utilization of computational resources. This approach enables consistent and rapid generation of sampling points while maintaining the flexibility needed for dynamic ray processing.

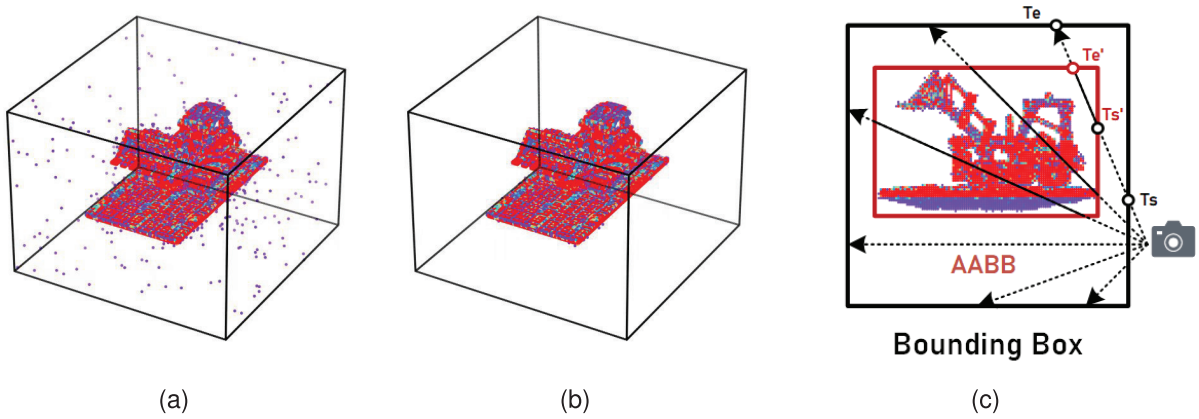


Fig. 7. Illustration of the proposed density grid filtering utilizing the LEGO dataset. (a) Raw density grid. (b) Filtered density grid. (c) New AABB box with decreased ray marching range.

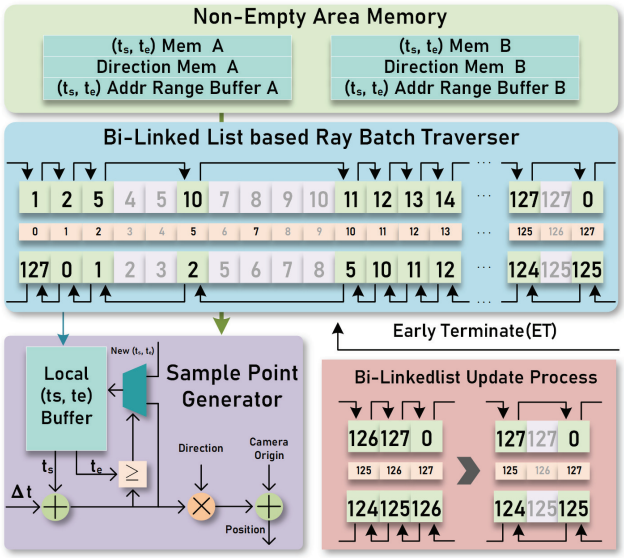


Fig. 8. Detailed architecture of the SPGU. The SPG employs a bilinked list to rapidly identify the next ray that has not been terminated early. It subsequently retrieves the corresponding t_s and t_e information from the nonempty memory to efficiently generate the next valid sampling point for that ray. In the bilinked list structure, the smaller number in the middle of each node represents the unique identifier of the current ray. The number at the top indicates the identifier of the next connected ray, while the bottom number indicates the identifier of the previously connected ray, establishing bidirectional references. In addition, black signifies that the ray is still alive, whereas gray signifies that the ray is terminated.

Fig. 8 shows the detailed architecture of the sample point generation unit (SPGU), which includes a 20-kB nonempty area memory (NEMEM), a bilinked list-based ray batch traverser, and a sample point generator (SPG). In this setup, each element in the traverser represents a ray and is connected to its preceding and succeeding active rays via the bilinked list. The SPG uses the bilinked list to determine which ray requires a sample point and accesses the 20-kB NEMEM. By utilizing the prestored camera coordinates, ray directions, and the real-time step distance, the system can directly compute the sample points within a single cycle. When the volume rendering unit (VRU) sends an early termination signal, the bilinked list is immediately updated to ensure that all subsequent sampling points are generated only from active rays. The bottom right part of Fig. 8 shows how the bilinked list is updated when Ray 126 is early terminated.

The combination of a bilinked list-based ray switching strategy and batch processing significantly enhances overall pipeline efficiency. Depending on the configuration of downstream computational resources, which influences the latency of processing sampling points, this approach reduces the number of invalid sampling point computations by 32.14%–44.12% compared to serial processing of individual rays.

V. Optimized Hash Encoding

In Instant-NGP, the hash table occupies a large portion of on-chip storage. When set to 2^{19} with a 16-bit half-precision floating-point number (FP16), a scene requires 25 MB, far exceeding the SRAM capacity of edge devices and resulting in frequent DDR interactions, which inevitably introduces latency. To address this, we reduced the hash table size to fit within the available SRAM, minimizing DDR access and latency. Quantization has proven to be an effective method for simplifying neural model computations [22], [23]. However, as the encoding unit for positional and directional data, the hash table size is crucial to rendering quality. Therefore, we applied QAT to reduce the hash table bit width, lowering storage requirements while preserving rendering quality. This section analyzes the impact of quantization and hash table size reduction to minimize on-chip SRAM usage while maintaining rendering quality.

To balance the size of on-chip hash storage, we perform experiments on different hash table sizes to examine their impact on the final rendering quality. The average PSNR versus hash size versus hash memory cost results are presented in Fig. 9. As we can see, the rendering quality measured in PSNR, as well as the memory cost, rises with the increase in the hash table size. The benefit of a larger hash table size slows down when it is larger than 2^{17}, while the table size increases exponentially. A hash table size of 2^{16} offers a balanced choice, while 2^{17} provides a better option if

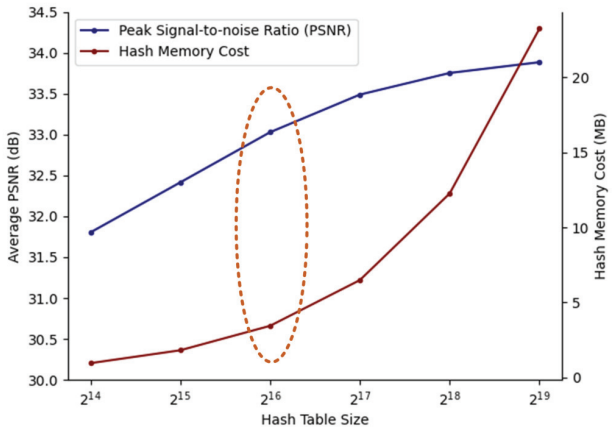


Fig. 9. Comparison of hash memory cost and rendering quality (PSNR) across different hash table sizes using FP16. The selected hash table size is highlighted by a red dashed bounding box.

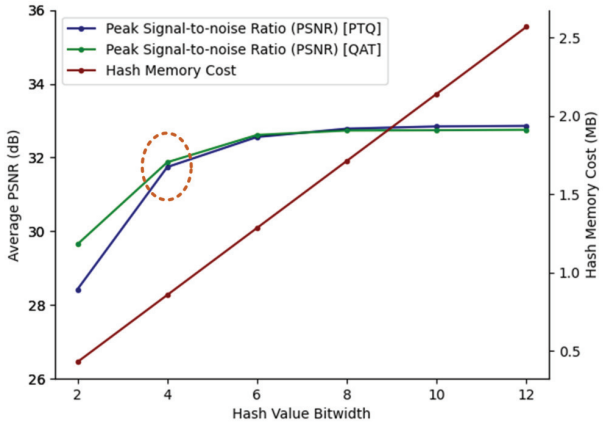


Fig. 10. Hash memory cost and rendering quality (PSNR) with varying hash value quantization bit widths using 2^{16} hash table size. The selected hash value quantization bit widths are highlighted by a red dashed bounding box.

additional silicon area is available for a larger table size. In our chip implementation, we employ a table of 2^{16} due to the limited chip size, while it is straightforward to extend to a larger table.

In addition, we apply quantization to the hash values to further reduce the table size. Fig. 10 presents the PSNR and memory cost for various quantization bit widths. Although hash storage requirements increase linearly with quantization bit width, PSNR gains slow down considerably after four bits, with minimal improvement beyond six bits. In our chip implementation, we select a 4-bit quantization for the hash values. However, notably, rendering quality can be further enhanced by increasing the hash table size and using more bits for quantization at the cost of silicon area.

To enhance reconstruction and rendering quality, we integrate QAT into our neural modeling process. Unlike post-training quantization (PTQ), QAT incorporates quantization directly within the training phase, allowing the model to adapt to reduced precision as it learns. Specifically, we simulate fixed-point computation by constraining the numerical range

Fig. 11. Rendering quality comparison of different quantization methods.

and precision of model parameters. The average PSNR results corresponding to different quantization bit widths are also presented in Fig. 10. As can be seen, the model using the QAT method shows an improvement in PSNR compared to the PTQ method when being quantized to low bits while maintaining the same hash table size and bit width. In addition, we evaluate and compare PSNR and model size across various quantization methods, as shown in Fig. 11. The QAT method renders more realistic details in the image, particularly in the shadow areas and the reflective surface areas, whereas the PTQ results exhibit noticeable rendering errors with incorrect pixel details in certain areas.

VI. Overall Chip Architecture

Based on the three strategies outlined above, we propose an Instant-NGP-based NVR coprocessor, which has been fabricated using 40-nm CMOS technology to verify the feasibility and real-world power consumption of the entire system. Fig. 12 illustrates the overall architecture of the proposed coprocessor, which includes a top controller, a ray marching unit (RMU), an SPGU, an encoder, an MLP engine, and a VRU. The end-to-end data flow is shown in Fig. 13.

The RMU consists of a density grid traverser (DGT), an NEAC, and a density grid memory (DMEM). This module is crucial for determining which parts of the scene require processing based on ray activity. The DGT performs ray marching through the density grid, using stepwise sampling to assess the validity of the ray. For valid rays, the NEAC identifies nonempty areas that are then stored in the NEMEM within the SPGU.

The SPGU generates positional and directional data for each sample point along valid rays. Fig. 14 illustrates the detailed structures of the encoder and the MLP engine. The encoder processes the coordinates and directional information of these sample points, with the high-dimensional encoded data passed to the MLP engine to compute density and color features. The encoder separately encodes the coordinate and directional data, enabling the MLP engine to utilize both to derive the required density and color features. The neural network employs a five-layer MLP architecture consisting of two density layers ($32 \rightarrow 64 \rightarrow 16$) and three color layers ($32 \rightarrow 64 \rightarrow 64 \rightarrow 3$). Due to process limitations and on-chip area constraints, the MLP engine deploys a small matrix computation resource: an MAC Array 0 consisting of 32×32 PEs and an MAC Array 1 consisting of 64×3 PEs.

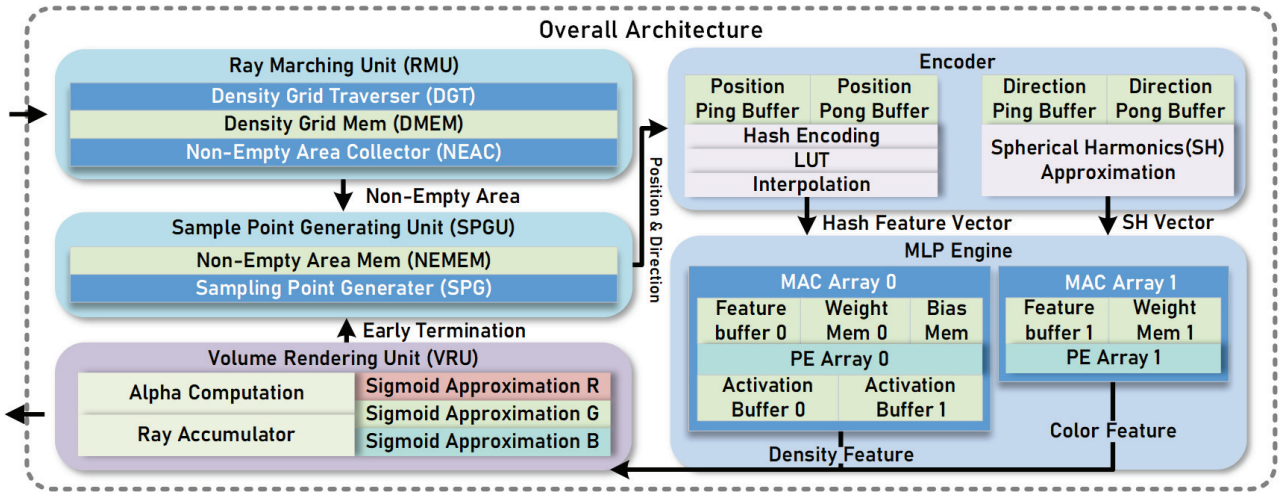


Fig. 12. Overall architecture of the proposed Instant-NGP-based NVR coprocessor.

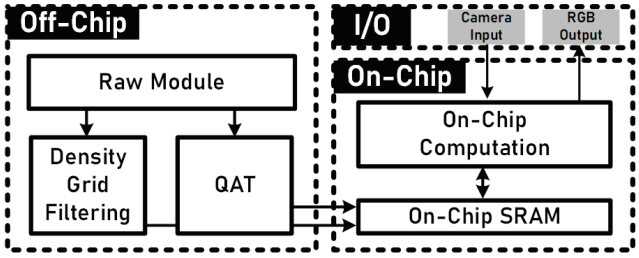


Fig. 13. System-level data flow: off-chip density grid filtering and QAT quantization with parameters stored in on-chip SRAM, taking camera data as input and outputting rendered RGB results.

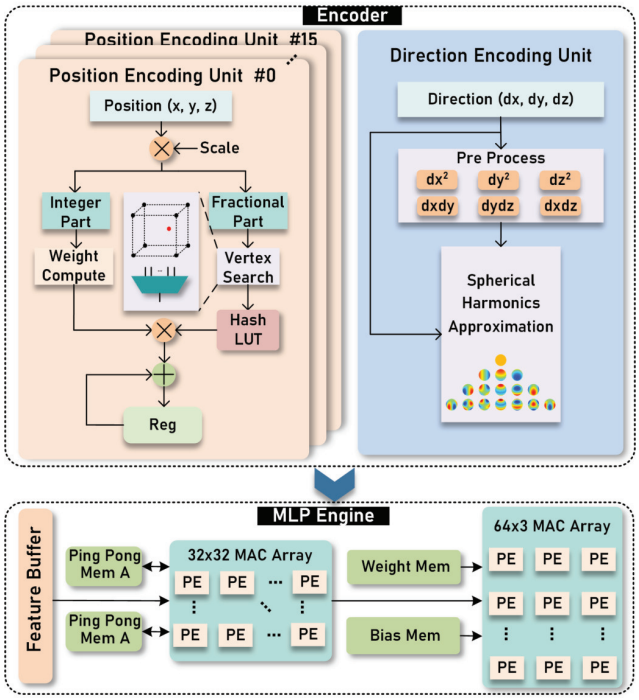


Fig. 14. Detailed structures of the encoder and the MLP engine.

Finally, the features from multiple sample points along a ray are sent to the VRU to compute the final red, green, blue, and alpha (RGBA) values for that ray. Once the RGBA values are generated, an early termination signal is sent back to the SPGU, halting further sample point generation for that ray.

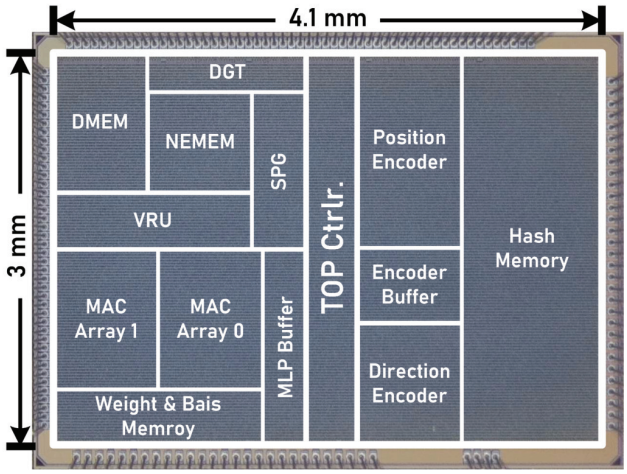


Fig. 15. Chip micrograph.

VII. Chip Measurement Results

In this section, we summarize the measurement results of the prototype chip and the comparison with state-of-the-art designs. We also present the rendering quality results both quantitatively, in terms of PSNR, and visually. Our prototype chip was fabricated using 40-nm CMOS technology, featuring a die area of 3×4.1 mm^2 with a total of 1.31-MB SRAM integrated. The micrograph of the chip is shown in Fig. 15.

A. Chip Measurement Platform Setup

To evaluate our chip, we developed a testing platform comprising a user-friendly desktop computer and an Altera HAN Pilot FPGA board with an FPGA mezzanine connector (FMC) interface, enabling efficient communication with the chip, as illustrated in Fig. 16. The computer connects to the FPGA board via a PCIe interface. The parameters of the

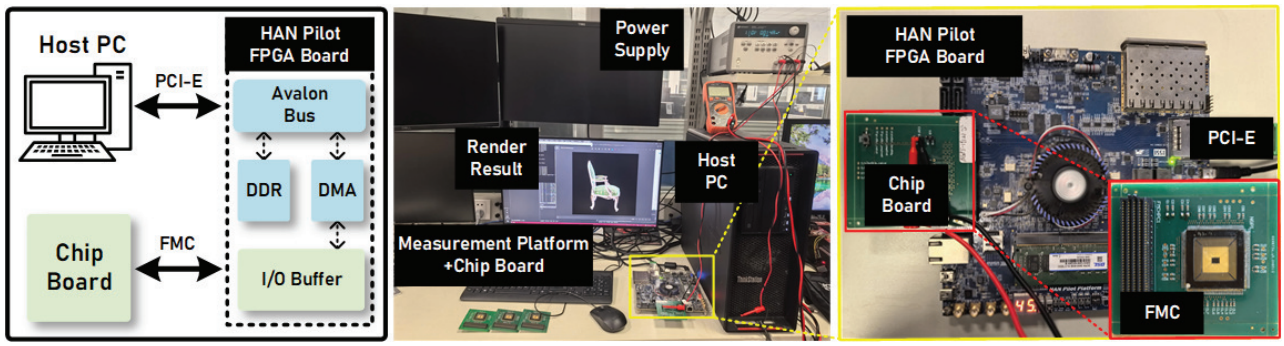


Fig. 16. Setup of the chip testing platform.

TABLE II
SPECIFICATIONS AND CHIP MEASUREMENT RESULTS

Technology	40nm CMOS
Die Area	12.3 mm^2
On-Chip SRAM	1.31 MB
Supply Voltage	0.9 V
Frequency	100 MHz
Power	78.3 mW
Throughput	2.6 Mpixels/S
Energy Effciency	30.5 nJ/Pixel

neural model (implicitly representing the 3-D scenes) and the viewing direction & position are updated from the computer to the FPGA. These data are further transmitted to the neural rendering chip through the FMC interface. The DDR memory within the FPGA board is used to temporarily store the data from the computer and our neural rendering chip. The system is controlled by the commands from the host computer, which are transmitted through the same link as the viewing direction & position. The rendering results are collected by the FPGA board and then sent to the host computer for further display and analysis.

B. Measurement and Comparison Results

Table II summarizes the performance of our chip. It was fabricated using 40-nm CMOS technology and occupies a core area of 12.3 mm^2 with 1.31-MB on-chip SRAM. Given the varying resolutions across different studies, we use pixels per second as the metric for throughput evaluation to ensure fair comparison. Based on throughput and power data collected from the testing platform, we calculate the energy consumption per rendered pixel. Operating at a clock frequency of 100 MHz, our chip consumes 78.3 mW of power, achieving a throughput of 2.6 Mpixels/s and an energy efficiency of 30.5 nJ/pixel.

To comprehensively evaluate our design, we selected a GPU implementation and dedicated neural rendering coprocessors for fair comparison. This includes Instant-NGP implementations on the edge platform NVIDIA Jetson Xavier NX and two state-of-the-art neural rendering chips, MetaVRain [18], [19] and NeuGPU [25], [26]. We select dedicated neural rendering coprocessors with available chip measurement data as reference to ensure a fair comparison. Table III summarizes the comparison results, which indicates that the proposed neural rendering coprocessor outperforms both the Jetson Xavier NX and MetaVRain in rendering throughput and energy efficiency. Specifically, our coprocessor achieves a 1.38× improvement in rendering throughput over the Jetson Xavier NX and a notable 3.16× speedup compared to MetaVRain. This substantial increase in rendering speed demonstrates our coprocessor's capacity to handle high-performance rendering tasks more efficiently than other edge solutions, making it well-suited for real-time applications.

The proposed coprocessor exhibits even more pronounced advantages in terms of energy efficiency. Compared to the Jetson Xavier NX, our design consumes 264.14× less power. In addition, it uses 15.46× less power than MetaVRain, a significant reduction that underscores the efficiency of our architecture for power-sensitive applications. This indicates that our accelerator not only achieves high performance but does so with minimal energy requirements. To facilitate fair comparisons with other neural rendering accelerators fabricated using more advanced 28-nm CMOS technology, we normalized our results from the 40-nm process to the 28-nm process using the scaling factors for technology nodes proposed in [28]. However, after normalization to the 28-nm process, although the energy required to render each pixel is close to that of the state-of-the-art NeuGPU, there is a small difference in energy per pixel and a big gap in throughput between our chip and the NeuGPU, which is due to the limitations in chip process and area during the tape-out process. In the tape-out version of the chip, the MLP engine only includes a single MAC Array 0, leading to insufficient computational resources and becoming a bottleneck, thus reducing throughput. On the other hand, other state-of-the-art NVR designs, MetaVRain, NeuGPU, and Hi-NeRF [27], are based on 28-nm processes and have sufficient on-chip area to deploy more computational resources. As accelerator architectures based on Instant-NGP, NeuGPU implements 16 S^3 cores, each containing 16 MAC trees (16 units per tree), while Hi-NeRF deploys 32 rendering cores with each core integrating 48 discrete MAC units and 32 MAC trees (eight units per tree). In contrast, our single-MAC system incorporates only one 32 × 32 MAC Array 0 and one 64 × 3 MAC Array 1.

TABLE III
COMPARISON OF PERFORMANCE ACROSS VARIOUS NEURAL RENDERING DESIGNS AND IMPLEMENTATIONS

	Jetson Xavier NX (NVIDIA) [20]	RT-NeRF [24]	MetaVRain [18], [19]	NeuGPU [25], [26]	Hi-NeRF [27]	This Work Single-Mac	This Work Multi-Mac
Technology (nm)	12	28	28	28	28	40	40
Model Type	Instant-NGP	RT-NeRF	NeRF	Instant-NGP	Instant-NGP	Instant-NGP	Instant-NGP
Frequency (MHz)	1100	1000	100	200	300	100	200
On-chip SRAM (MB)	10	3.5	2	2.06	2.5	1.31	1.33
Logic Area (mm^2)	350	18.85	20.25	20.25	15	12.30 / 5.74 (28nm)*	15.54 / 7.26 (28nm)*
Power (W)	15 (TDP)	8	0.310	0.728	1.9	0.078 / 0.052 (28nm)*	0.822 / 0.548 (28nm)*
Throughput (Pixel/Second)	1868800	28800000	659574	47040000	76800000	2570240	35484000
Energy Per Pixel (μJ/Pixel)	/**	0.278	0.470	0.0155	0.025	0.030 / 0.020 (28nm)*	0.023 / 0.0154 (28nm)*
PSNR (dB)	33.18	31.79	30.74	/	32.1	31.84	31.84

** Normalized by scaling factors across technology nodes from [28].*
*** Since Jetson Xavier NX is not consistently under full load and contain many power-consuming modules unrelated to neural rendering computations, we omit the energy-per-pixel metric of Jetson Xavier NX to ensure a fair comparison..*

To address the limitation, we expanded the MAC Array 0 in the system to ten units, ensuring that the MLP computation component no longer acts as the throughput bottleneck of the system. To align with increased compute power, we expanded ray marching steps to 32, density grid bits for each density grid block to 512, density grid accesses to 9, and the dispatcher's buffers to 7—ensuring single-cycle processing of 32 steps. Using the simulation data based on a 40-nm CMOS process, we simulate the throughput, power consumption, and area for this revised multi-MAC system version. In addition, we also normalized the chip test and simulation data to the 28-nm CMOS process by scaling factors across technology nodes. Here, we have also included additional state-of-the-art NVR accelerators for comparison with our design, which only include simulation results. The comparison results are also shown in Table III. As can be seen, although power consumption increases greatly, the multi-MAC version outperforms the single-mac version in both throughput and energy per pixel—achieving 13.81× higher throughput while maintaining only 0.77× energy per pixel. Compared to the state-of-the-art Hi-NeRF, which is also based on Instant-NGP, our multi-MAC version's throughput is slightly lower, mainly due to our lower frequency and on-chip area. However, in terms of energy per pixel, our performance, based on 40-nm data, reaches 92.4% of Hi-NeRF's efficiency. When normalized to the same 28-nm process, our energy consumption is only 61.7% of Hi-NeRFs. While maintaining nearly identical energy-per-pixel efficiency, our design achieves 46.2% of NeuGPU's throughput using only 35.85% of its hardware resources. The single-mac version, with its ultralow power consumption, suits power-sensitive but throughput-tolerant applications such as medical devices, whereas the multi-MAC version, excelling in throughput and energy efficiency, is ideal for mainstream edge computing scenarios.

Fig. 17 demonstrates the speedup effects of various proposed techniques on system performance. Both single-MAC and multi-MAC systems gain significant performance boosts from multistep ray marching, batch processing, and density grid filtering.

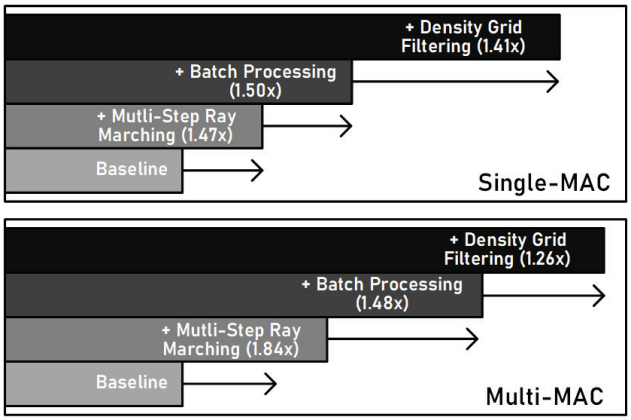


Fig. 17. Performance speedup analysis of proposed optimization techniques (multistep ray marching, batch processing, and density grid filtering) relative to the baseline (no optimizations) in both single-MAC and multi-MAC systems.

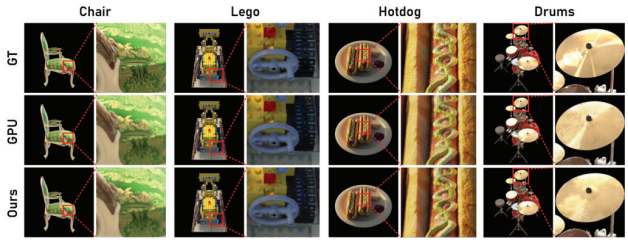


Fig. 18. Visualization and comparison of rendering quality.

In alignment with other studies on neural rendering coprocessor design, we evaluated our test chip using the commonly used NeRF synthesis dataset [4]. For each scene, we conducted tests on multiple viewpoints and computed the average PSNR value. Fig. 18 presents a visual comparison of the rendering results of our chip with that of the GPU results and the ground truth (GT), while Table IV provides the individual PSNR values and the averaged one across different scenes. Our rendering quality achieves an average PSNR of 31.84 compared to the GT. Furthermore, the visual comparison presented in

TABLE IV
PSNR RESULTS ACROSS DIFFERENT SCENES

Scene	Chair	Drums	Ficus	Hotdog
PSNR (dB)	32.88	26.03	31.70	35.92
Scene	Lego	Materials	Mic	Ship
PSNR (dB)	33.38	29.22	35.16	30.46
Average (dB)	31.84			

Fig. 18 illustrates that our chip produces exceptionally high rendering quality, with virtually no discernible differences from the results obtained using GPUs.

VIII. CONCLUSION

In this work, we present an energy-efficient coprocessor designed for Instant-NGP, a state-of-the-art neural rendering method. Our design introduces three key innovations.

1) We optimize occupancy grid queries in ray marching by partitioning the grid and decoupling the query process from sampling point generation, improving both efficiency and memory utilization.
2) We propose a batch processing strategy with a bilinked list-based ray switching mechanism, ensuring continuous pipeline utilization and overcoming the inefficiencies of sequential processing.
3) We enhance hash encoding by fitting the hash table into on-chip SRAM and applying QAT, reducing memory access latency and improving performance on resource-constrained platforms. To validate our design, we implemented and fabricated a 40-nm CMOS prototype chip. We also expanded the computational resources and conducted simulations to enable fair comparisons with existing designs. Experimental results demonstrate significant improvements in energy efficiency over GPUs, as well as state-of-the-art neural rendering accelerators. These results highlight our accelerator's ability to deliver high-throughput, low-power neural rendering, paving the way for real-time, high-quality rendering on edge devices.

REFERENCES

[1] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proc. 11th Annu. Conf. Comput. Graph. Interact. Techn.*, 1984, pp. 137–145.

[2] L. Alzubaidi et al., "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, pp. 1–74, Mar. 2021.

[3] A. Tewari et al., "Advances in neural rendering," in *Proc. ACM SIGGRAPH Courses*. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1–320, doi: 10.1145/3450508.3464573.

[4] 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," 2020, *arXiv:2003.08934*.

[5] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," in *Proc. 34th Conf. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2020, pp. 15651–15663. [Online]. Available: https://proceedings.neurips.cc/paperfiles/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf

[6] D. B. Lindell, J. N. P. Martel, and G. Wetzstein, "AutoInt: Automatic integration for fast neural volume rendering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 14556–14565.

[7] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi, "DeRF: Decomposed radiance fields," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 14153–14161.

[8] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 14335–14345.

[9] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "FastNeRF: High-fidelity neural rendering at 200FPS," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 14346–14355.

[10] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "PlenOctrees for real-time rendering of neural radiance fields," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 5752–5761.

[11] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 5855–5864. [Online]. Available: https://api.semanticscholar.org/CorpusID

[12] T. Neff et al., "DONeRF: Towards real-time rendering of compact neural radiance fields using depth Oracle networks," *Comput. Graph. Forum*, vol. 40, no. 4, pp. 45–59, Jul. 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID

[13] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 5449–5459.

[14] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "TensoRF: Tensorial radiance fields," 2022, *arXiv:2203.09517*.

[15] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–15, Jul. 2022, doi: 10.1145/3528223.3530127.

[16] Q. Xu et al., "Point-NeRF: Point-based neural radiance fields," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2022, pp. 5438–5448.

[17] C. Rao et al., "ICARUS: A specialized architecture for neural radiance fields rendering," *ACM Trans. Graph.*, vol. 41, no. 6, pp. 1–14, Nov. 2022, doi: 10.1145/3550454.3555505.

[18] D. Han, J. Ryu, S. Kim, S. Kim, J. Park, and H.-J. Yoo, "MetaVRain: A mobile neural 3-D rendering processor with bundle-frame-familiarity-based NeRF acceleration and hybrid DNN computing," *IEEE J. Solid-State Circuits*, vol. 59, no. 1, pp. 65–78, Jan. 2024.

[19] D. Han, J. Ryu, S. Kim, S. Kim, and H.-J. Yoo, "2.7 MetaVRain: A 133 mW real-time hyper-realistic 3D-NeRF processor with 1D-2D hybrid-neural engines for metaverse on mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 50–52.

[20] NVIDIA.(2025). *Jetson Xavier NX*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/

[21] J. Lee, K. Choi, J. Lee, S. Lee, J. Whangbo, and J. Sim, "NeuRex: A case for neural rendering acceleration," in *Proc. 50th Annu. Int. Symp. Comput. Archit.* New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–13, doi: 10.1145/3579371.3589056.

[22] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," 2018, *arXiv:1802.05668*.

[23] J. Yang et al., "Quantization networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7300–7308.

[24] 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 *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 1–9.

[25] J. Ryu et al., "20.7 NeuGPU: A 18.5mJ/Iter neural-graphics processing unit for instant-modeling and real-time rendering with segmented-hashing architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2024, pp. 372–374.

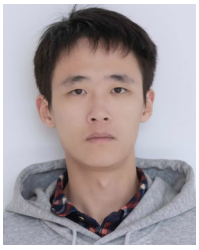
[26] J. Ryu et al., "NeuGPU: An energy-efficient neural graphics processing unit for instant modeling and real-time rendering on mobile devices," *IEEE J. Solid-State Circuits*, vol. 60, no. 1, pp. 99–111, Jan. 2025.

[27] L. Wu et al., "Hi-NeRF: A multicore NeRF accelerator with hierarchical empty space skipping for edge 3-D rendering," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 12, pp. 2315–2326, Dec. 2024.

[28] S. Sarangi and B. Baas, "DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

Zhechen Yuan (Student Member, IEEE) received the B.Eng. degree in electronics engineering from ShanghaiTech University, Shanghai, China, in 2021, where he is currently working toward the Ph.D. degree.

His research interests include neural rendering and energy-efficient VLSI design for computer graphics and deep learning.



Binzhe Yuan received the B.Eng. degree in electronic information engineering from ShanghaiTech University, Shanghai, China, in 2022, where he is currently working toward the M.Eng. degree.

His current research interests include high-performance computer arithmetic and VLSI design for neural rendering or other digital signal processing systems.

Chaolin Rao received the B.Eng. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2016, and the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2023.

Then, he joined the GGU Technology Company Ltd., Shanghai, China. His research interests include computer vision, energy-efficient VLSI circuits and systems, and smart vision circuits and systems.

Yiren Zhu received the B.Eng. degree in electronics science and technology from Shanghai University, Shanghai, China, in 2023. He is currently working toward the M.S. degree at the Electronics Science and Technology, ShanghaiTech University, Shanghai.

His research interests include computer vision accelerator design, especially software/hardware coverification.

Yunxiang He is currently working toward the B.Eng. degree at ShanghaiTech University, Shanghai, China.

His research interests include the architecture of custom accelerators based on vision, computer graphics, and deep learning.

Pingqiang Zhou (Member, IEEE) received the B.E. degree from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2005, the M.E. degree from Tsinghua University, Beijing, China, in 2007, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2012.

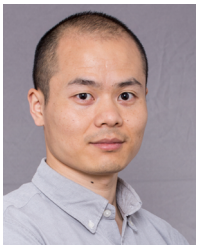
Prior to joining ShanghaiTech, Shanghai, China, he was a Research Intern with the IBM T. J. Watson Research Center in 2011 and a Postdoctoral Researcher with the University of Minnesota from 2012 to 2013. He was with the University of California at Berkeley, Berkeley, CA, USA, as a Visiting Scholar in 2015. He is currently an Associate Professor with the School of Information Science and Technology, ShanghaiTech University. His current research interests include the computer-aided design of VLSI circuits, computer architecture, and hardware security.

Prof. Zhou 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 and is an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM—II: EXPRESS BRIEFS.

Jingyi Yu (Fellow, IEEE) received the B.S. degree from Caltech, Pasadena, CA, USA, in 2000, and the Ph.D. degree from MIT, Cambridge, MA, USA, in 2005.

Before joining ShanghaiTech University, Shanghai, China, he was a Full Professor with the Department of Computer and Information Sciences, University of Delaware, Newark, DE, USA. He is currently the Vice Provost with ShanghaiTech University. His current research interests include computer vision and computer graphics, especially computational photography and nonconventional optics and camera designs.

Dr. Yu is a recipient of the NSF CAREER Award and the AFOSR YIP Award. He 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.



Xin Lou (Senior Member, IEEE) received the B.Eng. degree in electronic information technology and instrumentation from Zhejiang University (ZJU), Hangzhou, China, in 2010, the M.Sc. degree in system-on-chip design from the Royal Institute of Technology (KTH), Stockholm, 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 Center of Excellence at NTU as a Research Scientist. He is currently a tenured Associate Professor with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China. His research interests primarily focus on high-performance and energy-efficient integrated circuits and systems for vision and graphics processing.

Dr. Lou serves as an Associate Editor of IEEE TRANSACTIONS ON VLSI SYSTEM and was an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS in 2022 and 2023, respectively, and a Guest Editor of Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS in 2024.