© 2024 Muhammad Husnain Mubarik

HARDWARE ACCELERATION OF NEURAL GRAPHICS

BY

MUHAMMAD HUSNAIN MUBARIK

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Doctoral Committee:

Professor Rakesh Kumar, Chair
Professor Deming Chen
Assistant Professor Saurabh Gupta
Dr. Ravi Iyer, Intel Labs
Associate Professor Yingyan (Celine) Lin

ABSTRACT

Neural graphics is a rapidly evolving domain which uses neural networks to replace entire or parts of rendering pipeline. There are two broad classes of neural graphics: 1) Non-generative neural graphics where conventional DNNs are used for rendering. 2) Generative neural graphics where generative models such as latent diffusion models are employed. This dissertation studies the performance characteristic of both non-generative as well as generative neural graphics pipelines on modern hardware platforms, and asks the question: Does neural graphics need hardware support? The research is segmented into three primary areas: non-generative neural graphics via Neural Representations (NRs), generative neural graphics via Latent Diffusion Models (LDMs), and performance enhancements through integrating neural graphics with relatively more mature neural super-resolution techniques.

Firstly, we study non-generative neural graphics by focusing on Neural Representations (NRs) which form the backbone of various applications like Neural Radiance and Density Fields (NeRF), Neural Signed Distance Functions (NSDF), Neural Volume Rendering (NVR), and Gigapixel Image Approximation (GIA). We identify significant performance gaps when rendering 4K resolution frames at 60 FPS on current GPUs, with a shortfall ranging from 1.51× to 55.50×. To bridge this gap, we propose the Neural Graphics Processing Cluster (NGPC), an architecture that accelerates input encoding and multi-layer perceptron kernels, achieving up to 58.36× improvement in application-level performance.

Secondly, in the realm of generative neural graphics, we explore Latent Diffusion Models (LDMs) which have surpassed all the other generative processes in high-fidelity image generation domain. The computational demands of LDMs are immense, particularly at high resolutions. We performed in-depth breakdown analysis of the primitive compute kernels of the latent diffusion models and used our kernel-level analysis to design a scalable ar-

chitecture – Latent Diffusion Model Processing Unit (LDPU). Our estimates show that LDPU increasing the energy efficiency by up to 572× compared to traditional GPU baselines.

Lastly, the dissertation combines neural graphics with neural super-resolution techniques to further enhance the output quality and computational efficiency of neural rendering processes. By employing these techniques, we can render at lower resolutions and upscale the outputs to achieve desired higher resolutions without the computational cost typically associated with high-resolution rendering.

This comprehensive study not only underscores the necessity of specialized hardware for neural graphics but also sets a precedent for future innovations in this domain, potentially revolutionizing applications in virtual reality (VR), augmented reality (AR), mixed reality (XR), and beyond.



To my advisor, Professor Rakesh Kumar, for his patience, invaluable guidance, and unwavering support. Your insights transformed me from a novice in computer architecture to a PhD candidate at a prestigious institution. I am deeply grateful for your belief in me, even when I doubted myself.

To my parents, brother, and sisters, thank you for shaping who I am and anchoring me with your selfless support and belief, even when my goals seemed distant.

To my wife, your love, understanding, and companionship have been my strength throughout this demanding journey. I am forever grateful for your unwavering support.

I dedicate this work to each of you with immense gratitude, for making this achievement possible.

ACKNOWLEDGMENTS

I would like to express my profound gratitude to a number of people and institutions whose support has been indispensable throughout the course of this research.

First and foremost, I extend my deepest appreciation to my advisor, Prof. Rakesh Kumar, for his invaluable guidance, patience, and continuous support. His insights and mentorship have been instrumental in shaping not only this research but also my development as a scholar and a professional.

I am grateful to my colleagues at PASSAT for their collaboration and encouragement. Special thanks go to Matthew Tomei, Nathaniel Bleier, Andrew Timothy Smith, Adam Auten, Ramakrishna Kanungo, Srijan Chakraborty, and Shreyas Kishore, whose contributions and discussions have significantly enriched this work.

I acknowledge the support and resources provided by Intel, which have significantly contributed to this research. I am thankful to Rama Harihara, Ravi Iyer, Nilesh Jain, Selvakumar Panneer, Vui Seng Chua, Tobias Zirr, Anton Kaplanyan, and Tanay Karnik for their technical advice and assistance.

I am also thankful to Prof. Mehdi B. Tahoori and his team at KIT, including Farhan Rasheed and Dennis Weller, for their pivotal contributions to my initial publications during the early years of my PhD. Their guidance helped set the direction for my research.

I am also deeply grateful to my family for their unwavering love and encouragement. To my father, Ali Raza, my mother, Asima, my brother, Muhammad Ehtisham Mubarik, and my sisters, Taiba and Laiba — your constant support and belief in my abilities have been my foundation. To my wife, Alishbah, thank you for your endless patience, understanding, and love, which have made all the difference on this journey.

My gratitude extends to my friends Shaik Althaf V. Shajihan, Usman Ahmed Syed, Shoaib A. Goraya, and Vikram Sharma Mailthody, who pro-

vided moral support and companionship during this challenging journey. Your camaraderie has been a source of strength and joy amidst the rigors of academic pursuit.

This dissertation would not have been possible without the support and belief of all these individuals and institutions. Thank you all for your contributions to my academic journey.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATIONS xiv

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 APPLICATIONS OF NEURAL GRAPHICS 4
2.1 Introduction 4
2.2 Applications of Non-Generative Neural Graphics 4
2.2.1 Computer Graphics 4
2.2.2 Medical Imaging 6
2.2.3 Autonomous Vehicles 6
2.2.4 Astronomy 7
2.2.5 Robotics 7
2.2.6 Industrial Metaverse 8
2.3 Potential Applications of Generative Neural Graphics 8
2.3.1 Image Synthesis and Editing from Scratch 8
2.3.2 Image Super-Resolution 9
2.3.3 Inpainting and Restoration 9
2.3.4 Video Generation 9
2.3.5 Frame Interpolation 10
2.3.6 Video Super-Resolution 10
2.3.7 Text-to-Image Generation 11
2.3.8 Scene Generation 11
2.3.9 3D Object Generation 11
2.3.10 Temporal Data Modeling 12
2.3.11 Other Applications 12
2.4 Summary and Discussion 12

CHAPTER 3 HARDWARE ACCELERATION OF NEURAL REPRESENTATIONS (NON-GENERATIVE NEURAL GRAPHICS) . 14
3.1 Introduction 14
3.2 Neural Graphics: An Overview 16

3.2.1 Input Encoding . . . 19
3.3 Does Neural Graphics Need Hardware Support? . . . 24
3.4 Understanding Performance of Neural Graphics Applications . 27
3.5 Neural Fields Processor: An Architecture for Neural Graphics 29
3.6 Evaluation and Results . . . 32
3.7 Summary and Discussion . . . 36

CHAPTER 4 HARDWARE ACCELERATION OF LATENT DIFFUSION MODELS (GENERATIVE NEURAL GRAPHICS) . . . 46
4.1 Introduction . . . 46
4.2 Diffusion Models: An Overview . . . 48
4.3 Do Latent Diffusion Models Need Hardware Support? . . . 52
4.3.1 Attention Mechanism . . . 55
4.4 Performance Diagnosis of Latent Diffusion Models . . . 57
4.4.1 Quantization and Sparsity in Latent Diffusion Models 60
4.5 LDPU: A Scalable Architecture for Latent Diffusion Models . 61
4.5.1 Microarchitecture of Diffusion Processing Element (DPE) . . . 66
4.5.2 Modes of Operation of Diffusion Processing Element (DPE) . . . 67
4.6 Evaluation and Results . . . 68
4.7 Summary and Discussion . . . 72

CHAPTER 5 IMPROVING NEURAL RENDERING LATENCY WITH SUPER-RESOLUTION . . . 77
5.1 Introduction . . . 77
5.2 Motivation . . . 79
5.3 Open-source, state-of-the-art super-resolution algorithms . . . 81
5.4 Combining Super Resolution with Neural Graphics . . . 87
5.5 Related Work . . . 89
5.6 Summary and Discussion . . . 92

CHAPTER 6 DISCUSSION AND FUTURE WORK . . . 94
6.1 Integration of Neural Graphics with Conventional Rendering Pipelines . . . 95
6.2 Bringing Systolic Arrays Closer to Graphics Pipelines . . . 95
6.3 Unified Hardware Architecture for Generative and Non-Generative Neural Graphics . . . 96
6.4 Tiny Neural Network Architectures for Resource-Constrained Environments . . . 96
6.5 Simulation and Emulation Tools for Neural Graphics . . . 96

REFERENCES . . . 97

LIST OF TABLES

3.1 The parameters of the four neural graphics applications for different input encoding schemes. 29
3.2 The percentage utilization of the GPU compute and memory resources, for the input encoding and the MLP kernels for all four neural graphics applications and for different input encodings. 30
3.3 The input/output bandwidth and the data access time for our NGPC architecture. 36

4.1 UNet configuration for latent diffusion model [1]. 54
4.2 The number of ResNet and Transformer blocks in latent diffusion model and the percentage time spent on those blocks for VGA resolution. 59
4.3 The percentage of static sparsity, dynamic sparsity, special weight values and special activation values in quantized and fine-tuned latent diffusion models. 62
4.4 The synthesized area of the DPE and the power estimates for various operating modes the DPE. We scaled the area and power numbers to 7nm technology node using often used scaling formulas [2]. 71

5.1 The number of pixels in the x and y dimensions for the selected image resolutions. 81
5.2 The percentage of time spent on the Transformer and the ResNet blocks for different image resolutions. 84
5.3 Model size and performance comparison of the state-of-the-art open-source super-resolution frameworks for 2× up-sampling. 87
5.4 Quality comparison of the state-of-the-art open-source super-resolution frameworks for 2×2 up-sampling. 88

LIST OF FIGURES

2.1 Potential Applications of Neural Graphics. 5

3.1 Rendering pipeline: Conventional Computer Graphics [3] VS Neural Graphics [4]. 17
3.2 Inherent benefits of Neural Graphics over Conventional Computer Graphics. 20
3.3 Different stages of a neural graphics pipeline. 20
3.4 Structure of the four neural graphics applications under study. . . 21
3.5 Kernel level performance breakdown analysis of neural graphics applications: The number of cycles consumed by the different kernels of the neural graphics applications (as percentage of total application cycles). (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*. 38
3.6 Illustration of *Multi resolution grid encoding*. 39
3.7 Scheduling of a typical neural graphics application on GPU. . . . 39
3.8 The operation level breakdown of the of input encoding kernels for different input encoding types. The five most expensive operations in terms of number of cycles spent. MRHG : *multi resolution hashgrid* , MRDG : *multi resolution densegrid* , LRDG : *low resolution densegrid* 40
3.9 Architecture of the Neural Fields Processor (NFP) 40
3.10 a) Interaction of the NGPC with GPU. b) Programming model for NGPC. c) Pseudocode for programming NGPC. 41
3.11 Block diagram of the emulator. 41
3.12 The performance of the neural graphics applications on our proposed architecture for the scaling factor of 8, 16, 32, and 64. NGPC-8 has 8 NFP units, NGPC-16 has 16 NFP units and, so on. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*. 42

3.13 The performance improvement of the input encoding kernels and the multi-layer perceptron kernels individually, on our architecture, for the scaling factors of 8, 16, 32, and 64. For all four neural graphics applications and different input encoding types. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*, where dotted lines labeled as mlp_imp_TA are the performance improvements of MLP engine modeled with timeloop and accelergy. 43
3.14 The number of pixels that can be rendered for a given FPS target with and without neural graphics processing cluster. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*. 44
3.15 The area and Power of neural graphics processing cluster , normalized with respect to the area and power of Nvidia RTX 3090 die area and power. 45

4.1 A schematic of diffusion model based rendering pipeline. [5] . 49
4.2 Forward and reverse diffusion chains in diffusion models, aka Denoising Diffusion Probabilistic Models (DDPMs). . . . 50
4.3 Different stages of latent diffusion model based rendering pipeline. 52
4.4 Architecture of the primitive block of latent diffusion model (UNet backbone). 53
4.5 The performance characteristics of stable diffusion running on a desktop-class GPU (RTX 3090, 24 GB). For the out-of-the-box PyTorch based stable diffusion implementation with vanilla attention mechanism. 53
4.6 The dataflow of the attention mechanism. 55
4.7 Profiling of FlashAttention based Stable diffusion running on a desktop-class CPU (Intel i9-10920X) and a server-class CPU (Intel Xeon Gold 6430L). The x-axis has different resolution levels. The bars show time spent in seconds and the line shows the number of pixels rendered. 58
4.8 Profiling of FlashAttention based Stable diffusion running on a desktop-class GPU (RTX3090, 24GB). The x-axis has different resolution levels. The bars show time spent in seconds and the line shows the number of pixels rendered. . 59
4.9 Breakdown of compute cycles spent on different stages of stable diffusion based rendering pipeline. On average 95% of the GPU compute cycles are spent on latent diffusion model (UNet backbone). 62

4.10 The operation level breakdown of the of UNet for different rendering resolutions. The five most expensive kernels in terms of number of cycles spent. Attention mechanism, element-wise operations such as normalization, exponent etc, explicit convolution kernels, fused gemm operations for NN layers (implicit convolution and MatMul for dense layers with activations), explicit gemm kernels. 63
4.11 The distribution of static sparsity, dynamic sparsity, special weight values, and special activation values as percentages across various layers of quantized and fine-tuned latent diffusion models at Int8 and Int4 quantization levels. . . 64
4.12 Architecture of the Latent Diffusion Processing Unit (LDPU) 65
4.13 Interaction of the LDPU architecture with GPU. b) Programming model for the LDPU. c) An example pseudocode for programming LDPU. 68
4.14 Different modes of operation supported by diffusion processing element (DPE). 68
4.15 The block diagram of the evaluation methodology. 71
4.16 Comparison of number of cycles estimated for systolic array modeled using Timeloop/Accelergy [6] and our custom analytical model. 72
4.17 The estimated performance improvement, energy efficiency and performance improvement / mm2 for Int8 variants of LDPU architecture, for different scaling factors. 73
4.18 The estimated performance improvement, energy efficiency and performance improvement / mm2 for Int4 variants of LDPU architecture, for different scaling factors. 74
4.19 Area and Power overhead of different variants of the LDPU architecture for different scaling factors. 75
4.20 The estimated annual energy consumption and corresponding CO2 emissions from generating images using diffusion models at various resolutions, both with and without LDPU, expressed in terms of the equivalent number of U.S. households' energy consumption per year and the equivalent annual CO2 footprint of a certain number of advanced cars. 76

5.1 In client-only scenario super resolution improves quality, frame-rate, and reduces frame latency. In cloud-SR-to-client scenario SR improves server density as more clients can be addressed. In cloud-to-client-SR super-resolution reduces strain on bandwidth requirements and also improves the server density. 79

5.2 Normalized pixel count and rendering time for Neural Representations (NRs) on an RTX 3090 (24GB memory), relative to VGA resolution. 82
5.3 Normalized pixel count and rendering time for Latent Diffusion Model on an intel i9-10920X, relative to VGA resolution. 82
5.4 Normalized pixel count and rendering time for Latent Diffusion Model on an intel Xeon Gold 6430L, relative to VGA resolution. 83
5.5 Normalized pixel count and rendering time for Latent Diffusion Model on an RTX3090 (24GB memory), relative to VGA resolution. 83
5.6 Model architecture of a typical super-resolution network, including a feature extractor, a convolutional long short-term memory (ConvLSTM) network and an unbalanced UNet for reconstruction. 87
5.7 Rendering time of neural representations for different image resolution levels with and without 2× super-resolution. . . 90
5.8 Rendering time of latent diffusion models for different image resolution levels with and without 2× super-resolution. . . 90

LIST OF ABBREVIATIONS

NeRF: Neural Radiance Fields
LDM: Latent Diffusion Models
NG: Neural Graphics
NRs: Neural Representations
NSDF: Neural Signed Distance Functions
GIA: Gigapixel Image Approximation
NVR: Neural Volume Rendering
NGPC: Neural Graphics Processing Cluster
NFP: Neural Fields Processor
LDPU: Latent Diffusion Model Processing Unit
MLP: Multi-Layer Perceptron
SRAM: Static Random-Access Memory
GPU: Graphics Processing Unit
RTX: Nvidia RTX Series GPUs
CUDA: Compute Unified Device Architecture
DDPM: Denoising Diffusion Probabilistic Models
DDIM: Denoising Diffusion Implicit Models
DMs: Diffusion Models
HBM: High Bandwidth Memory
VR: Virtual Reality
AR: Augmented Reality
XR: Mixed Reality
TDP: Thermal Design Power
TOPS: Tera Operations Per Second
VGA: Video Graphics Array
SVGA: Super Video Graphics Array
XGA: Extended Graphics Array
WXGA: Wide Extended Graphics Array

HD: High Definition
FHD: Full High Definition
QHD: Quad High Definition
UHD: Ultra High Definition
SUHD: Super Ultra High Definition

CHAPTER 1

INTRODUCTION

The pursuit of digital realism through computer graphics has driven substantial progress in both academic and industrial research [7] [8] [9]. Classical computer graphics focuses on synthesizing photorealistic images from the physical and material properties of scenes using inherently computationally demanding algorithms [10] [11]. Concurrently, neural networks, renowned for their excellent function approximation capabilities [12] [13] [14], have spurred the development of neural graphics, a novel approach aimed at enhancing rendering processes and efficiency [7] [15].

As the field of neural graphics is in its nascent stages, the term "neural graphics" remains loosely defined and is used in various contexts [7] [4]. We define neural graphics as the technique where neural networks replace entire or parts of the rendering pipeline. Within this broad domain, neural graphics can be classified into two sub-fields: Non-generative neural graphics and generative neural graphics. In non-generative neural graphics, neural representations (NRs) are used where a conventional DNN model is trained to represent a specific scene, allowing the synthesis of novel views of the scene from previously unseen camera viewpoints and angles [16]. In this class of rendering one neural network is represents one scene or a small collection of scenes and cannot be generalized to other scenes [16] [17].

In contrast, generative neural graphics has a much wider and more generalizable scope. The field of generative artificial intelligence (GenAI) has seen significant popularity and development [18] [19] [20] [21] [22] [23], revolutionizing the use of GenAI in computer graphics [24] [25] [26] [27]. This includes advancements driven by latent diffusion models [28], which now set the standard in various generative vision-based tasks [26] [27]. Despite their ability to produce high-quality content, these models are known for their substantial computational demands and corresponding energy requirements, especially when processing at high resolutions. This dual advancement in

both generative and non-generative neural graphics and diffusion models underscores the critical need to view them from the lens of a computer architect and to understand their hardware-specific behaviors.

This dissertation investigates the detailed profiling characteristics of both generative and non-generative neural graphics pipelines and answers the question of whether specialized hardware is required to meet the computational and energy demands of these technologies. For non-generative neural graphics, we propose a novel hardware architecture, the Neural Graphics Processing Cluster (NGPC), which includes the Neural Fields Processor (NFPs). These are designed to optimize input encoding and multi-layer perceptron stages, critical for efficient non- generative neural graphics applications.

As generative models, particularly latent diffusion models (LDMs), become increasingly prevalent, their high energy consumption becomes a pressing issue. This work explores the performance and energy efficiency of LDMs, proposing the Latent Diffusion Model Processing Unit (LDPU). This scalable hardware solution is engineered from the ground up to enhance the energy efficiency of generating high-resolution images by providing hardware support for sparsity and special values processing.

Both generative and non-generative neural graphics, while capable of producing high-resolution outputs like 8K frames, face challenges in terms of rendering time and computational resource requirements. We further study super-resolution techniques as a solution to curb the computational demands and data transfer overhead of high- resolution images by rendering them at relatively lower resolutions and using super-resolution techniques to upscale them to the desired high-resolution frames. Recent advances in neural-network-based super-resolution techniques, including proprietary and open-source frameworks, offer substantial improvements in image quality and processing speed, making them ideal for integration with neural rendering processes.

This dissertation aims to bridge the gap between the fast- paced algorithmic advancements in neural graphics and the practical implementations in hardware to address the increasing demands of modern neural graphics. Neural graphics algorithms have revolutionized the field of digital content creation, making it faster, more efficient, and broadly accessible [29] [30]. Through comprehensive experimental analysis and rigorous performance evaluations, this dissertation delves into the computational characteristics and

memory access patterns of the neural graphics algorithms. The subsequent chapters will delve deeper into each of these topics, exploring both the state-of-the-art and the novel contributions of this research in the realms of non-generative neural graphics, generative neural graphics, and the potential of combining super-resolution techniques with neural graphics.

CHAPTER 2

APPLICATIONS OF NEURAL GRAPHICS

2.1 Introduction

Neural graphics, leveraging the power of neural networks, has emerged as a transformative technology in the field of computer graphics. This chapter explores the diverse range of applications where neural graphics can be effectively employed, providing a comprehensive taxonomy based on the specific needs of each application, such as energy efficiency, latency requirements, and real-time performance.

2.2 Applications of Non-Generative Neural Graphics

Non-Generative neural graphics, more formally known as Neural Representations (NRs), can conceptually be seen as a way to represent high-frequency data with high fidelity using fully connected neural networks. They have applications in various domains (Figure 2.1). This section explores the diverse applications of neural representations, focusing on their technical aspects, benefits, and challenges. We will also discuss their energy efficiency, latency/performance, and real- time needs.

2.2.1 Computer Graphics

Neural representations are extensively used in computer graphics to create hyper-realistic models and scenes. Neural Radiance Fields (NeRFs), in particular, have shown exceptional performance in rendering complex scenes with accurate lighting and reflections. These techniques have the potential to enable content-creators, filmmakers, and game developers to create immer-

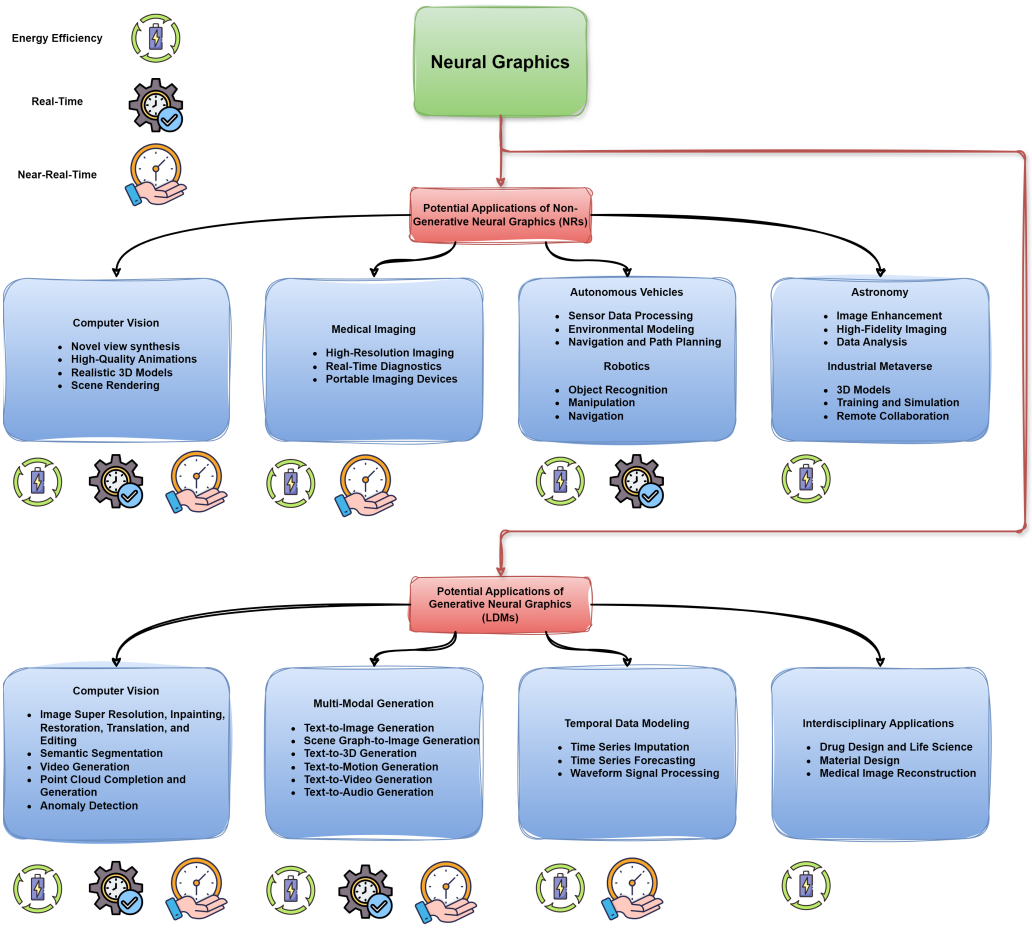


Figure 2.1: Potential Applications of Neural Graphics.

sive environments with less manual effort compared to traditional methods.

Creating high-quality visual data with neural representations require significant computational resources. Optimizing neural graphics techniques and utilizing specialized hardware can help improve energy efficiency. Latency is a crucial factor in rendering visual content, especially for interactive applications like video games. Neural representations need to be optimized to reduce inference time and provide smooth, real-time rendering. Real-time rendering of complex scenes requires highly optimized neural representations and hardware acceleration to ensure seamless performance.

2.2.2 Medical Imaging

Neural representations can potentially have transformative applications in medical imaging, where they can be used to reconstruct high-resolution images from low-quality scans [31] [32] [33] [34] [35] [36] [37] [38] [39] [40]. This technique has the potential to improve diagnostic accuracy and aid in medical research.

Energy-efficient neural representations can reduce the power consumption of medical imaging devices, making them more suitable for portable and wearable applications. Low latency is critical in medical imaging for near-real-time diagnostics and treatment planning. Optimizing neural representations for fast inference is essential. Near-real-time reconstruction of medical images can potentially enable immediate analysis and decision-making, crucial for emergency and surgical applications.

2.2.3 Autonomous Vehicles

Neural representations can potentially be employed in autonomous vehicles to process sensor data and create detailed environmental models [41] [42] [43] [44] [45] [46] [47] [48] [49]. These models can then be used for navigation, obstacle detection, and path planning.

Energy-efficient neural representations will potentially help in reducing the power consumption of autonomous vehicle systems, extending battery life, and improving overall efficiency. Latency is a critical factor in autonomous driving, where decisions need to be made in real-time (milliseconds). Opti-

mized neural representations can potentially ensure quick processing of sensor data. Real-time environmental modeling is essential for the safe operation of autonomous vehicles, requiring high- performance neural representations and hardware support.

2.2.4 Astronomy

In astronomy, neural representations can be used to enhance images of celestial objects by reconstructing high-fidelity images from noisy and low-resolution data [50] [51]. Techniques like NeRFs can improve the quality of astronomical observations and potentially enable new discoveries. Neural radiance fields are already being employed in astronomy. For example, Jarolim et al. presented SuNeRF [50], where they discuss the 3D reconstruction of the solar EUV corona using neural radiance fields. Hansen et al. [51] have used neural radiance fields for geometric modeling of lunar terrain.

Efficient neural representations are necessary to handle the large volumes of data generated by astronomical observations without excessive power consumption. While real-time processing is not required in astronomy applications, faster processing times can enable more dynamic observations and analysis.

2.2.5 Robotics

Neural representations are used in robotics for tasks such as object recognition, manipulation, and navigation [52] [53] [54] [55] [56] [57] [58]. These techniques enable robots to interact with their environment more effectively.

Energy-efficient neural representations are extremely critical for robotics applications as they can extend the operational time of battery-powered robots, making them more practical for long-term use. Low latency is also essential for real-time robotic applications, ensuring quick response times and smooth operation. Real-time object recognition and manipulation are crucial for the effective performance of robots in dynamic environments.

2.2.6 Industrial Metaverse

Neural representations has the potential to play a key role in the industrial metaverse, where they can be used to create detailed 3D models of machinery and processes [59] [60] [61] [62]. These models can facilitate training, simulation, and remote collaboration.

Efficient neural representations can reduce the power requirements for rendering and interacting with complex industrial models. High-performance neural representations are necessary to provide smooth and responsive interactions within the industrial metaverse. While real-time processing may not be as critical in initial phase of industrial applications, faster processing times can enable more dynamic observations and analysis.

2.3 Potential Applications of Generative Neural Graphics

Latent Diffusion Models (LDMs) have emerged as the state of the art technique for generative neural graphics. They have shown remarkable versatility and potential across a wide range of applications. Figure 2.1 also presents the taxonomy of potential applications of generative neural graphics using latent diffusion models. This section delves into these applications, emphasizing their technical aspects, benefits, and challenges, while also discussing their energy efficiency, latency/performance, and real-time needs.

2.3.1 Image Synthesis and Editing from Scratch

LDMs have significantly advanced the field of image synthesis, surpassing traditional methods like GANs [63]. By modeling the data distribution through progressive noise addition and removal, LDMs can generate high- fidelity images from scratch. These models have the potential to be effective in applications such as digital art creation, game design, and visual effects for movies. Image synthesis with LDMs can be computationally intensive. Techniques such as optimized network architectures and specialized hardware can improve energy efficiency. Latency in image synthesis is crucial for interactive applications like game design. LDMs can be optimized for faster inference

times, reducing latency significantly. Real-time image editing requires LDMs to perform inference in milliseconds, necessitating highly optimized models and hardware acceleration.

2.3.2 Image Super-Resolution

LDMs can be used directly to enhance the resolution of images, generating high-quality, detailed images from low-resolution inputs [64] [28] [65] [66] [67] [68]. This application is vital in fields like satellite imagery and medical imaging, where high-resolution images are crucial for analysis and decision-making. Energy-efficient models can reduce the power required for processing images, which is essential for portable devices. For real-time applications, the latency of super- resolution models must be low to process images swiftly without noticeable delays. In real-time video streaming and broadcasting, super-resolution models need to operate with minimal latency to provide enhanced visual quality without lag.

2.3.3 Inpainting and Restoration

LDMs can fill in missing parts of images (inpainting) and restore corrupted or incomplete images to their original form. This is particularly useful in restoring old photographs, enhancing security footage, and repairing damaged artworks [69] [70] [71] [72] [73] [74] [75]. Optimized neural architectures can help reduce the energy consumption of inpainting and restoration tasks. Latency is critical in applications like real-time security footage enhancement, where delays can impact the effectiveness of monitoring. Real-time inpainting requires models to quickly process and generate missing image parts, essential for applications in live video feeds and interactive content.

2.3.4 Video Generation

LDMs extend their capabilities to video generation, producing smooth and realistic video sequences from a series of input frames or from textual descriptions [76] [77] [78] [79] [80] [81] [82]. This application is essential in the film industry for creating special effects, in video game development

for generating realistic environments, and in virtual reality for immersive experiences. Efficient video generation models can significantly reduce the power consumption needed for creating high-quality video content. Performance optimization is crucial for generating video frames quickly, especially for high- resolution and high-frame-rate videos. Real-time video generation is vital for applications in virtual reality and live broadcasting, requiring models to operate with very low latency.

2.3.5 Frame Interpolation

LDMs can be used to generate intermediate frames between existing frames in a video sequence, improving the visual quality and smoothness of videos [83] [84] [85] [86] [87]. This is particularly useful in applications like video streaming and animation [88]. Energy efficient frame interpolation models can help reduce the computational cost of generating intermediate frames in videos. Low latency is essential for real- time frame interpolation to ensure smooth video playback without stuttering. North star of this application is real-time frame interpolation for applications such as live video streaming, where intermediate frames need to be generated on-the-fly.

2.3.6 Video Super-Resolution

Similar to image super-resolution, LDMs can be used to enhance the resolution of videos, converting low-resolution video frames to clearer and more detailed video content [89] [90] [91] [92] [93]. This has applications in improving the quality of video streams and broadcasts, especially in fields requiring high-definition footage. Video super-resolution models need to be optimized for energy efficiency to handle the large volume of data in videos without excessive power consumption. High performance video super-resolution models can process frames quickly to maintain smooth playback without delays. Real-time video super-resolution is necessary for live broadcasts and streaming services to enhance video quality dynamically.

2.3.7 Text-to-Image Generation

LDMs have set new standards in converting textual descriptions into corresponding images [94] [95] [96] [97] [98] [99] [100]. This capability is highly valuable in content creation, enabling artists and designers to quickly visualize ideas based on textual inputs. Tools like DALL-E [101]and MidJourney [102] are prominent examples of this application [28]. Energy-efficient text-to-image models can reduce the power consumption required for generating high-quality images from text descriptions. Latency is critical for text-to-image generation, especially in interactive applications where users expect quick feedback. Real-time text-to- image generation allows for dynamic content creation and instant visualization of ideas, essential for creative industries.

2.3.8 Scene Generation

Given a textual description, LDMs can generate entire scenes, which is beneficial for creating visual content for storytelling, advertising, and entertainment [103] [28] [104] [105] [106] [107]. This technology reduces the time and effort required to create complex scenes manually [108]. Efficient scene generation models help in managing power consumption, especially when generating large and detailed scenes. High- performance models ensure that scenes are generated quickly, providing immediate results for users. Real-time scene generation is crucial for applications in interactive storytelling and live content creation, where quick generation of scenes is needed.

2.3.9 3D Object Generation

Extending beyond 2D images, LDMs can generate 3D models from textual descriptions [109] [110] [111] [112] [113] [108], facilitating applications in virtual reality, augmented reality, and 3D printing. This technology streamlines the creation of 3D assets for various industries [88]. Optimized 3D object generation models can reduce the energy required to produce high-quality 3D assets. Latency is important in 3D object generation for applications that require quick creation and manipulation of 3D models. Real-time 3D object generation is essential for applications in virtual and augmented reality,

where interactive and dynamic 3D content is needed.

2.3.10 Temporal Data Modeling

LDMs have shown to be useful beyond graphics and vision. LDMs are adept at modeling temporal data, making them suitable for applications such as time series forecasting and signal processing [114] [115] [116] [117] [118] [108]. These models capture the underlying patterns in temporal data, enabling accurate predictions and analyses. Efficient temporal data models help in managing the energy costs associated with processing large temporal datasets. Latency is critical in temporal data modeling to provide timely and accurate predictions. Real-time needs in temporal data modeling include applications such as real-time financial forecasting and live monitoring of environmental conditions.

2.3.11 Other Applications

Researches have extended LDMs to various interdisciplinary fields, demonstrating their versatility and adaptability. In the pharmaceutical domain, LDMs can potentially simulate molecular interactions and predict the behavior of compounds [108], potentially accelerating the drug discovery process. By generating potential drug candidates and evaluating their efficacy, LDMs can potentially contribute to the development of new medications [108]. In material science, LDMs have the potential to assist in the design and discovery of new materials by modeling the properties and interactions of different compounds [108]. In medical imaging, LDMs can potentially enhance medical imaging techniques by reconstructing high-quality images from low-resolution scans [119] [120] [121] [122] [123]. This application has the potential to improve diagnostic accuracy and enable better treatment planning in healthcare [108].

2.4 Summary and Discussion

The applications of neural representations are diverse and impactful, ranging from computer graphics and medical imaging to autonomous vehicles and

industrial applications. While these technologies promise significant advancements in output quality and rendering efficiency, addressing their energy and latency challenges is crucial for their widespread adoption in real-time and near-real-time applications. Figure 2.1 presents an overview of the potential applications of neural graphics. For instance, Neural Radiance Fields (NeRF) and Neural Signed Distance Functions (NSDF) have shown great potential in rendering and 3D modeling but require substantial performance boosts to meet real-time requirements, with performance gaps ranging from 1.51× to 55.50× (Chapter 3). Future developments in hardware-software co-design will be essential to push the state of the art in neural representations.

Latent Diffusion Models (LDMs) have diverse applications, spanning from image and video generation to natural language processing and scientific research. Figure 2.1 presents the taxonomy of potential applications of LDMs. These models have shown significant promise in generating high-quality content with applications such as text-to-image generation and video super-resolution. However, the current state of the art in LDMs is too energy-intensive and latency- intensive, making them unsuitable for real-time or near-real- time applications. Addressing these challenges will require significant advancements in both hardware and software co- design.

In the rest of this thesis, we will present different solutions to improve the rendering latency and energy efficiency of neural graphics. Chapters 3 and 4 detail our proposed hardware architectures, NGPC and LDPU, and their respective performance evaluations. Chapter 5 explores the integration of neural graphics with neural super-resolution techniques to further enhance output quality and computational efficiency. These combined efforts aim to bridge the gap between the current capabilities and the demanding requirements of modern visual computing applications.

CHAPTER 3

HARDWARE ACCELERATION OF NEURAL REPRESENTATIONS (NON-GENERATIVE NEURAL GRAPHICS)

3.1 Introduction

The fundamental goal of classical computer graphics is to synthesize photo-realistic and controllable imagery. Rendering algorithms synthesize an image of a scene from the geometric and material properties of the scene, often through tracing the path of a photon from light source to the object and utilizing the information about the geometry and scattering distributions of the object to simulate the interaction of light with the object. Inverse rendering algorithms follow the reverse process of rendering. From the final image, they provide guidance about how geometry and materials of a scene need to be adjusted. Both rendering and inverse rendering algorithms are well-known to be computationally challenging tasks [124–126]. As such, search has gone on for decades to build efficient rendering and inverse rendering algorithms [125, 127–130].

Since visual data, which is the output of the classical rendering and inverse rendering algorithms, is usually resilient to approximations [131, 132], and as neural networks are considered to be particularly good function approximation algorithms, a natural question arises: *can neural networks be used to approximate the algorithms used in classical computer graphics?* Recent works [16, 133–138] have shown that such neural representations can in fact be superior in learning and representing the physical and the material properties of the scenes. Then the information learned in these networks is extremely compact and can be used to synthesize the photo-realistic imagery. This process of approximating entire or parts of computer graphics using neural networks is known as *neural graphics* [7]. Neural graphics promises a fast, deterministic time replacement for traditional rendering algorithms.

In this work we ask the question: *does neural graphics need hardware support?* We focus on four neural graphics applications recently presented as a representative set of neural graphics representations [134]: 1) Neural radiance and density fields (NeRF), 2) Neural signed distance functions (NSDF), 3) Gigapixel image approximation (GIA) and 4) Neural volume rendering (NVR). These four applications cover a wide range of graphics tasks including rendering, novel view synthesis [139], 3D shape representation [140], simulation, path planning [141], 3D modeling [142], and image approximations. We first performed an algorithmic analysis of the neural graphics applications and found that all the neural graphics applications require a multi-layer perceptron to learn the scene representations and an input encoding kernel to capture high frequency information in the visual data. We studied a wide range of input encoding algorithms and carefully picked three representative input encoding algorithms for further study, 1) *multi resolution hashgrid encoding*, 2) *multi resolution densegrid encoding* and 3) *low resolution densegrid encoding*. We analyzed the above four neural graphics applications for all three encoding types, on a modern desktop class GPU (RTX 3090). Our profiling shows that if we want to render 4k resolution frames at 60 FPS, there is a gap of $\sim 1.51\times$ to $55.50\times$ between the desired performance and the state of the art. This motivates the need to provide hardware acceleration to bridge this gap.

In order to understand the performance bottlenecks of the representative neural graphics applications, we perform kernel level performance breakdown analysis. Our results show that input encoding, and multi-layer perceptron are the two most expensive kernels in all neural graphics applications consuming 72.37%, 60.0%, and 59.96% of application time for *multi resolution hashgrid encoding*, *multi resolution densegrid encoding* and *low resolution densegrid encoding* respectively. Based on these results, we design an architecture - a *neural fields processor* (Figure 3.9) - to accelerate these stages in hardware using an input encoding engine and a hardware MLP engine. The input encoding engine directly supports operations and dataflow identified by the kernel-level analysis of the input encoding (Section 3.4). The MLP engine is similarly optimized for the small MLPs common in neural graphics multi-layer perceptron kernels. Moreover, in neural graphics, the outputs of the input encoding kernel are always consumed by the multi-layer perceptron kernel (Figure 3.4). We exploit this fact in the neural fields processor hardware

by fusing the input encoding and multi-layer perceptron engines.

We propose a scalable neural graphics processing cluster (NGPC) - consisting of several neural fields processor units - along with the existing GPC units. We evaluate the performance of the neural graphics applications on our proposed architecture for the scaling factor (number of neural fields processor units) of 8, 16, 32, and 64. For *multi resolution hashgrid encoding*, the performance benefits of our architecture are 12.94×, 20.85×, 33.73×, and 39.04× for the scaling factor of 8, 16, 32, and 64 respectively. Our results show that with multi resolution hashgrid encoding, NGPC enables the rendering of 4k Ultra HD resolution frames at 30 FPS for NeRF and 8k Ultra HD resolution frames at 120 FPS for all our other neural graphics applications.

Our work makes the following contributions:

- Neural graphics applications require dedicated hardware acceleration for real-time rendering. We quantify the performance gap between the modern hardware and the desired performance targets. $\sim$ 1.51× to 55.50× for rendering 4k resolution frames at 60 FPS.
- We studied the neural graphics applications and identified that the input encoding and the multi-layer perceptron kernels are common performance bottlenecks.
- We present an efficient hardware architecture to support neural graphics applications in real time. Our architecture accelerates the input encoding and multi-layer perceptron kernels through dedicated engines and fuses the engines for a more efficient dataflow. We show that our hardware architecture is scalable and flexible enough to support a wide range of neural graphics applications.
- We quantify the benefits of our hardware architecture and show significant performance benefits against GPU baseline for all four neural graphics applications and three input encoding types.

3.2 Neural Graphics: An Overview

Figure 3.1 depicts a high-level comparison of conventional rendering pipeline and the neural graphics pipeline. Rendering in conventional real-time graph-

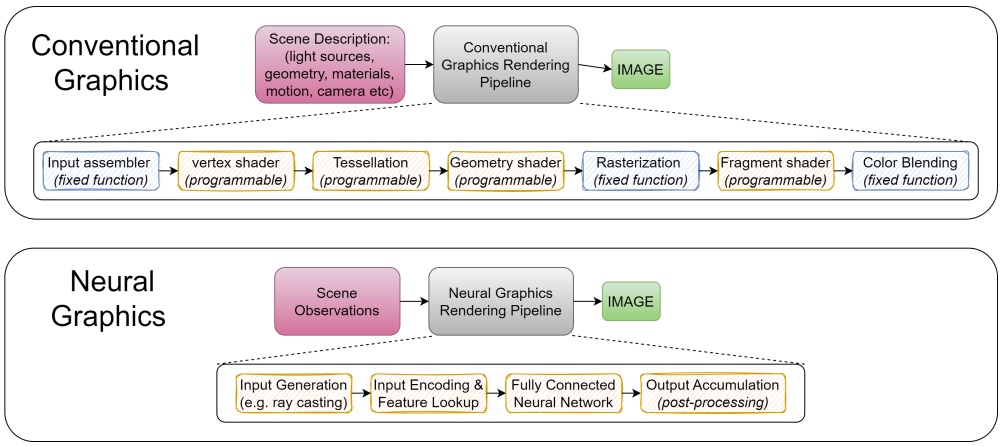


Figure 3.1: Rendering pipeline: Conventional Computer Graphics [3] VS Neural Graphics [4].

ics starts with the detailed description of the physical and material properties of the scene. This description is passed as input to the rendering pipeline which then generates a 2D image (frame) as output. In neural rendering, the description of the geometric and material properties of the scene are derived from multiple scene observations (images or video), which serves as an input to the neural rendering pipeline. Neural graphics replaces the conventional rendering pipeline with much simpler neural rendering pipeline. The neural rendering pipeline at high level consists of three stages.

1) Input stage: This stage is responsible for generating the inputs to a neural network. In classical computer vision and image processing applications, the typical goal is image classification or image transformation. In these applications, an image is typically used as input to the neural network. The input layer of the network can be vary large depending upon the resolution of the image. For example, if a network is processing a 224×224 RGB image (as in case of AlexNet [143]), the input layer has $224 \times 224 \times 3$ dimensions. Unlike classical computer vision and image processing applications, where usually an image is passed as input to the neural network, in neural graphics applications, the neural network inputs are either encoded pixel positions/coordinates (x, y, z) or encoded pixel positions/coordinates and camera viewing angle (x, y, z, θ, ϕ), depending upon the application under consideration.

2) Inference stage: This stage is responsible for learning the detailed information about the scene. The learning objective of the neural graphics applications is different from a typical computer vision and image processing application. In a typical computer vision and image processing application, the goal is usually to learn common features from a batch of training images and then, during inference phase, classify the images into a set of classes by identifying the features in them. The fully connected layers on the input side of a neural network usually become very expensive for a typical computer vision application and hence the neural networks used in classical computer vision applications are usually convolutional neural networks. In a neural graphics application, the goal is to learn the detailed representation of the scene, which usually comprises high-frequency visual data. Suitable input encodings such as frequency encoding [16] and parametric grid encodings [134], combined with strong compression capabilities of fully connected networks, have been shown to learn neural representations for such information well.

In training phase, the scene observations are learned by the neural network using classical neural network training techniques (for example gradient descent and Adam optimization), where the loss function propagates the gradients in the backward direction and adjusts the weights of the neural network. In the inference phase, the network outputs the pixel color (RGB) or pixel color and density information (RGB, σ) using the given pixel coordinates/positions (x, y, z) or pixel coordinates/positions and camera viewing angle (x, y, z, θ, ϕ) as inputs.

3) Compositing stage: The output of the fully-connected neural network is either three channel pixel color (RGB) or pixel color and density information (RGB, σ). The goal of the output stage is to accumulate the color and density information of the individual pixels and assemble the final output imagery. Classical volume rendering techniques [144–146] can be used to project the output colors and densities into an image.

Figure 3.2 shows the high level benefits of neural graphics over conventional computer graphics. 1) **More compact representations of scenes:** In classical computer graphics, "Fields" are widely used to parameterize the physical properties of an object or scene over space and time. Such space-time parameterizations, defined for all spatial and/or temporal coordinates,

are needed to synthesize 3D shapes and/or 2D images. In order to faithfully store arbitrary functions by way of classic discrete samples, high sampling rates are needed to avoid aliasing. As the complexity of the scene grows, the memory requirement to store these samples explodes. However, in neural graphics, the field is parameterized, fully or in part, by a neural network. Such parameterized fields are known as neural fields. Neural representations are known to adapt well to continuous functions with sparse discontinuities even with much lower storage capacities. Hence, neural fields enable compact and efficient representations of the scene. 2) **Simpler domain-agnostic data structures:** In conventional computer graphics, complex data structures such as 3D point clouds [147], 3D meshes [148], voxel based 3D models [149], parametric models, and depth maps [150] are used for 3D representations. These data structures are scene geometry dependent. In neural graphics, while aspects of such data structures may still be required to limit network complexity at larger scales, extended neural representations often lead to more unified, higher-level data structures and thus reduced complexity. 3) **Predictable performance:** In conventional computer graphics, the rendering time strongly depends on the complexity of the scene. In neural graphics, as large parts of the rendering are replaced by a constant-cost inference operation, the rendering time becomes less dependent on details of the scene, and instead is mostly a function of scale and higher-level scene layout. 4) **Higher-level scene definition:** As explained earlier, in conventional computer graphics, a complete description of the physical and material properties of the scene is required for rendering. In neural graphics, however, the properties of the scene can implicitly be learned from a few images or video of the scene. This leads to much simpler scene definitions.

The above benefits make neural graphics arguably the biggest advance in the field of computer graphics in decades.

3.2.1 Input Encoding

Photo-realistic visual data usually has high frequency information. For example, the crisp RGB colors of a gigapixel image, the detailed texture, lighting, and geometry information of a 3D scene, are some examples of high frequency visual data. In neural graphics, multi-layer perceptrons are used to learn and

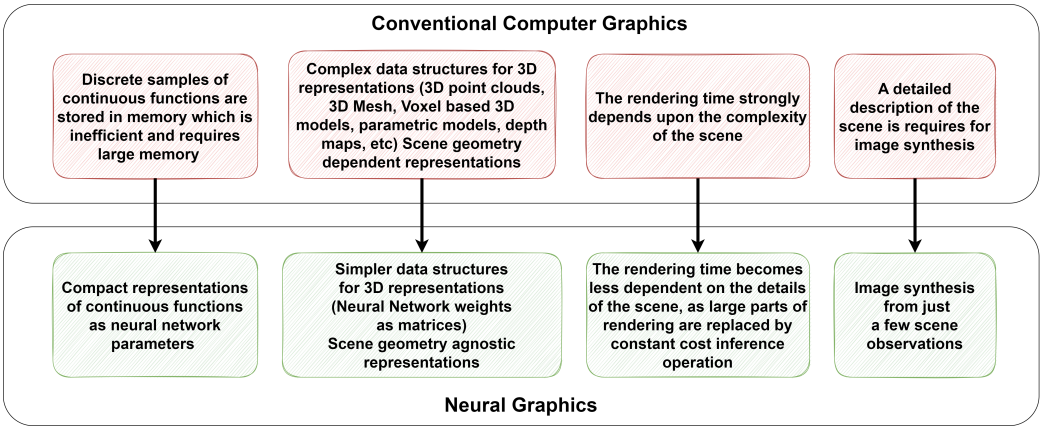


Figure 3.2: Inherent benefits of Neural Graphics over Conventional Computer Graphics.

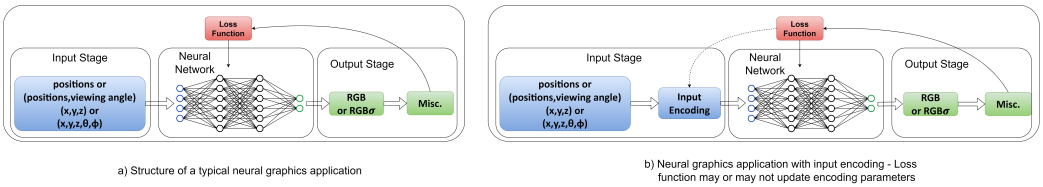


Figure 3.3: Different stages of a neural graphics pipeline.

represent this high frequency visual data. Previous works have shown that multi-layer perceptrons (Figure 3.3-a) are biased towards learning low frequency information [16, 151] and struggle with high frequency content. To address this, the original NeRF paper [16] (Vanilla-NeRF) introduced input encoding. NeRF uses high- dimensional input encodings to enable the representation of complex, high-frequency details in scenes. The core idea is to transform the 5D coordinates, comprising spatial location and viewing direction, into a higher- dimensional space, thus mitigating the inherent bias of neural networks towards low-frequency functions as shown in [151]. Directly using low- dimensional inputs limits the network's ability to capture the intricate variations in color and geometry present in real-world scenes. By mapping these inputs to a higher- dimensional space, the neural network can more effectively learn and represent the fine details and textures, resulting in more accurate and photorealistic renderings.

Moreover, the higher-dimensional encoding mitigates the risk of the network producing blurry or oversmoothed images, a common issue when dealing with high-frequency scene content. As described in [16], positional encoding techniques enable the network to maintain and reconstruct the high-

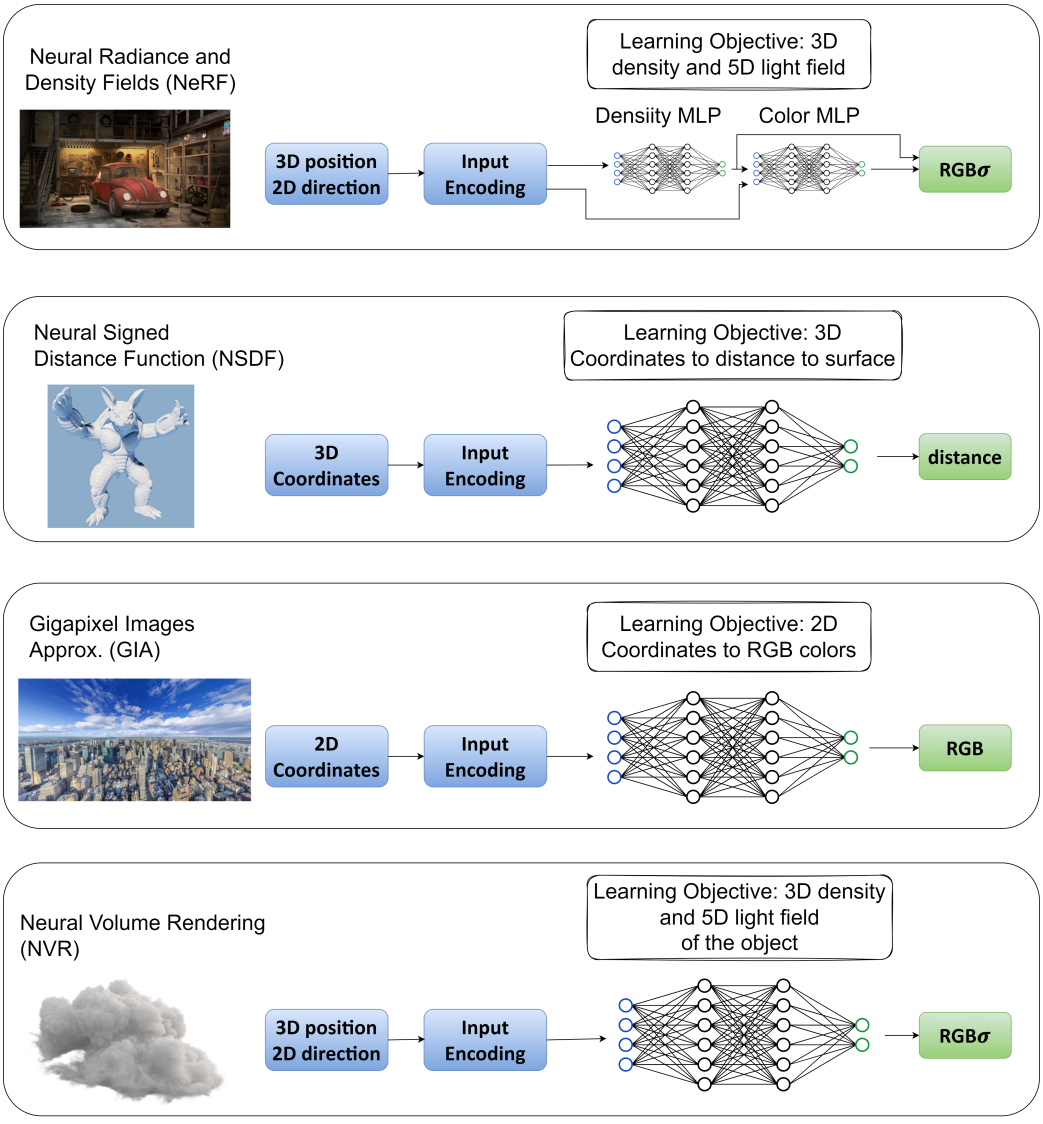


Figure 3.4: Structure of the four neural graphics applications under study.

frequency details by providing a richer, more expressive representation of the input coordinates. This encoding process facilitates the network's ability to interpolate and extrapolate the radiance field, ensuring that the generated images are sharp and visually coherent, even when synthesizing novel views from sparse input images. Thus, higher-dimensional encoding is essential for achieving high-resolution and high-fidelity results characteristic of state-of-the-art neural rendering techniques. Input encoding schemes can be divided into two high level categories based on whether they use a fixed high-frequency function ($\sin, \cos$, Fourier transform) or a learnable function (embeddings, neural networks): *1) fixed-function encodings 2) parametric encodings*.

1) Fixed-function encodings:

Vanilla-NeRF [16] was the first paper to show that, mapping the 5D vector of input positions and viewing angle to a higher dimensional space, by using high frequency sin and cos functions, and using the outputs of the high frequency functions as input features to the multi-layer perceptron, enables the multi-layer perceptron to learn the high frequency variations of the scene. Since vanilla-NeRF, there had been a lot of work on improving the input encoding. Tancik et al. [152] showed, for example, that replacing sin and cos with Fourier transform lets the network learn high frequency functions in low dimensional domains. These encoding schemes are known as *frequency encoding schemes*, as they use high frequency sinosoids for mapping the low dimensional inputs to higher dimension space. All these schemes are fixed function encoding schemes as they use a fixed compute function for mapping the low dimensional input vector to higher dimensional space.

2) Parametric encodings: Recently, state of the art results have been shown by *parametric encodings*. Parametric encoding schemes essentially use additional trainable parameters in addition to weights and biases of the neural networks to learn information about the scene. These parameters can be arranged as auxiliary data structures and used to map low dimensional positions to higher dimensional inputs, which are then used to query the neural network. In a popular class of parametric schemes [134, 135], the trainable parameters are stored in generic lookup tables where the number of parameters are picked based on the desired reconstruction quality. These

works propose to divide the scene into multiple resolution levels (grid) and use a separate lookup table for each resolution level. As the scene is divided into multiple grids, the approach is called *multi-resolution grid encoding* and the number of lookup tables (or resolution levels) can also be optimized as a hyper parameter. Different resolution levels give different output fidelity; Muller et al. [134] have shown that for most neural graphics applications, depending upon the type of grid used, 8 to 16 resolution levels produce acceptable visual fidelity.

The different steps performed in *multi-resolution grid encoding* are explained in Figure 3.6. The encoding parameters are arranged into L levels, each containing up to T feature vectors with dimensionality F. Each level is independent and stores the feature vectors at the vertices of a grid. Each corner of the grid is mapped to an entry in the level's respective feature vector array where each feature vector array has a fixed size of T. The mapping between the grid and the feature vector array is 1:1 for coarse levels because the dense grid requires fewer than T parameters for coarse levels. However, for finer levels, the feature vector can either be mapped 1:1 or treated as hash table and a hash function can be used to index into the array [134]. The input encoding, where a hash function is used to index into lookup table, is called *multi-resolution hashgrid encoding*. Whereas, the encoding where 1:1 mapping is used for all the resolution levels is known as *multi-resolution densegrid encoding*. The hash function used by the state of the art parametric encoding scheme (instant-NGP [134]) is presented in Equation 3.1.

$$h(x) = (\oplus_{i=1}^{d} x_i \pi_i) \; mod \; T \tag{3.1}$$

where $\oplus$ denotes the bit-wise XOR operation and π are unique large prime numbers. The feature vectors at each corner of the grid are linearly interpolated to ensure continuous representation. The interpolated feature vectors for all the levels are then concatenated to generate the final encoded input to the multi-layer perceptron, as shown in Figure 3.6. The number of training parameters are bounded by $T \times L \times F$. Muller et al. [134] have shown that for *multi-resolution hashgrid encoding* only using 16 levels and 2 features per array entry produces high fidelity image frames for many neural graphics applications, hence the number of trainable encoding parameters is set

$T \times 16 \times 2$ where T ranges from 2^{14} *to* 2^{24} depending upon the neural graphics application and the desired image fidelity. For *multi-resolution densegrid encoding* 2-8 levels can be used to get high visual fidelity outputs.

3.3 Does Neural Graphics Need Hardware Support?

To understand the performance characteristics of neural graphics, we focus on the following four representative neural graphics applications that cover a relatively wide range of graphics tasks, as presented by Müller et al. [135], including rendering, novel view synthesis, 3D shape representation, simulation, path planning, 3D modeling, and image approximations:

1) **Neural radiance and density fields (NeRF):** The learning objective during training phase is is to learn the 3D density and the 5D light field of a given scene from a few scene observations (images or video of the scene). The structure of the NeRF application is shown in Figure 3.4. In NeRF, two fully-connected neural networks, 1) Density MLP and 2) Color MLP, are concatenated together where density MLP learns the density information (σ) and the color MLP learns the view dependent (RGB) color information of the scene. As the density information of a scene is view direction agnostic, the encoded coordinates/positions (x, y, z) are used as input to the density MLP. As the color information depends upon position as well as view angle, the output of the density MLP along with the encoded view directions (θ, ϕ) are passed as input to the color MLP. The output of the color MLP is a three dimensional vector containing the pixel color information (RGB). The density information is then concatenated with the pixel color information to get the four dimensional vector containing the pixel color and the density information (RGB, σ).

2) **Neural signed distance functions (NSDF):** In classical computer graphics, signed distance functions (SDFs) are used to represent a 3D shape as the zero level-set of a function of position x. In graphics, SDFs are commonly used in applications such as simulation, path planning, 3D modeling, and video games [134]. In neural approximations of SDFs, the MLP learns the mapping from 3D coordinates to the distance to a surface. The structure

of the NSDF application is shown in Figure 3.4. The inputs to the MLP are encoded positions and the final output is the distance to the surface.

3) **Gigapixel image approximation (GIA):** A gigapixel image is an ultra high definition digital image bitmap, usually made by combining multiple detailed images into a single image. A Gigapixel image has billions of pixels, which is much more than the capacity of a normal professional camera. In GIA a neural network is used to approximately learn the gigapixel image in its trainable parameters. The learning objective of the MLP in GIA application is to learn the mapping from 2D coordinates to RGB colors of the gigapixel image. The structure of the GIA application is shown in Figure 3.4. As the MLP learns the 2D image, there is no density information and hence the view direction is also not required; the inputs to the MLP are the encoded pixel positions and the outputs are the corresponding pixel color (RGB). This application can be considered as an important benchmark to test neural network's ability to learn the high frequency details of visual data.

4) **Neural volume renderer (NVR):** This application is similar to NeRF. The only difference is that instead of learning the density and the emission field, the network in neural volume rendering learns the density and a reflectance field, which can be used to simulate the light transport in the volume using path tracing. The structure of the NVR application is shown in Figure 3.4. In NVR, two fully-connected neural networks, 1) Density MLP and 2) Color MLP, are concatenated together where the density MLP learns the density information (σ) and the color MLP learns the view dependent (RGB) color information of the bounded object. Similar to NeRF, the encoded positions (x, y, z) are used as input to the density MLP and the output of the density MLP along with the encoded view directions (θ, ϕ) are passed as input to the color MLP. The output is a four dimensional vector containing the pixel color and the density information (RGB, σ).

Each application can be implemented using a variety of input encodings. As parametric encodings produce strictly better output fidelity than frequency encodings [134, 135, 153, 154], we picked parametric encoding for further exploration. In order to faithfully represent the state of the art in parametric encodings, we explored three different types of *parametric en-*

codings in this work – *1) Multi resolution hashgrid encoding:* Hash function is used to generate the indices of the lookup tables while mapping the grid features to the lookup tables entries. The number of resolution levels used is 16 [134, 155]. *2) Multi resolution densegrid encoding:* 1:1 mapping is used between grid features and the lookup tables entries. The number of resolution levels used is 8 [134, 155]. *3) Low resolution densegrid encoding:* 1:1 mapping is used between grid features and the lookup tables entries. The number of resolution levels used is 2 [134, 155].

We profiled the above four neural graphics applications with the chosen three input encoding schemes using the open source code published by Müller et al [155]. The parameters for all our neural graphics applications and encoding schemes are shown in Table 3.1. Both the input encoding algorithm and the multi-layer perceptron are implemented as separate fused CUDA kernels [134, 135, 155]. Unlike standard MLPs, the fully-fused MLPs do not have any explicit biases. Due to the smaller size of the fully-fused MLPs, the intermediate activations and the partial sums can be stored on the faster on-chip memory, reducing the number of global memory accesses. A python based wrapper generates the inputs and displays the final rendered frame. We run the applications on Nvidia's RTX3090 using CUDA version 11.7 and report the total runtime of the applications.

Our results (Figure 3.5) show that, on a modern desktop class GPU (RTX3090), for multi resolution hashgrid encoding, rendering $\sim 2M$ pixels (1920×1080 frame) takes $231msec$, $27.87msec$, $2.12msec$, and $6.32msec$ for NeRF, NSDF, GIA, and NVR respectively. This is unacceptable for real-time applications. For instance, if we want to render 4k frames at 60FPS only one of our neural graphics applications (GIA) is able to meet that target, there is the performance gap of 55.50×, 6.68×, and 1.51× for NeRF, NSDF, and NVR respectively.

In order to understand the performance bottlenecks of neural graphics applications, we perform kernel level performance breakdown analysis. Figure 3.5 shows that input encoding and multi-layer perceptron are the two most expensive kernels in all neural graphics applications. For *multi resolution hashgrid encoding*, on average across all neural graphics applications, 40.24% of the total number of cycles are consumed by the input encoding kernel and 32.12% cycles are consumed by the multi-layer perceptron kernel. On average, this amounts to 72.37% of application time. For *multi resolu-*

tion densegrid encoding, on average across all neural graphics applications, 24.63% of the total number of cycles are consumed by the input encoding kernel and 35.37% cycles are consumed by the multi-layer perceptron kernel. On average, this amounts to 60.0% of application time. For *low resolution densegrid encoding*, on average across all neural graphics applications, 24.15% of the total number of cycles are consumed by the input encoding kernel, and 35.37% cycles are consumed by the multi-layer perceptron kernel. On average, this amounts to 59.96% of application time. This data motivates the need to accelerate the input encoding and multi-layer perceptron kernels. In next section, we will provide a further breakdown of the time spent in the input encoding and the multi-layer perceptron kernels to understand their performance bottlenecks.

3.4 Understanding Performance of Neural Graphics Applications

We performed further analysis (using Nvidia's nsight compute) to understand the performance bottlenecks in the input encoding and multi-layer perceptron kernels. Figure 3.8 shows the operation level breakdown of the of input encoding kernels for different input encoding types. Five most expensive operations in terms of number of cycles spent are labeled in the Figure 3.8. As explained in Section 3.2, the grid lookups are the major building block of the input encoding algorithms; our analysis also shows that the grid lookups take significant amount of cycles across all three input encoding types.

The percentage utilization of the GPU compute and memory resources for the input encoding kernel is presented in Table 3.2. On average, across all encoding kernel calls, the memory utilization of the GPU is higher than compute utilization. This is because input encoding is a memory intensive workload that requires performing lookup operations for mapping inputs to learned features even as the lookup tables for all the resolution levels do not entirely fit on the L2 cache of RTX3090. The memory wait time to resolve the cache misses also adds to the overall cycles. Our analysis also shows that the integer mapped modulo operation is one of the most expensive operations for all three input encoding types. This is because the modulo operation gets mapped to the less efficient general integer modulo operation instead of using

more efficient bitwise operations which account for the fact that hash map sizes are always powers of two.

As multi resolution densegrid and low resolution densegrid have one-on-one mapping of grid indices, the hash function is not called for these input encoding types, and hence the breakdown shows zero cycles for the hash function. However, the hash function consumes a significant number of cycles for multi resolution hashgrid. This is because the hash function cannot efficiently utilize the ALU due to the stalls caused by waiting on the long scoreboard to resolve the global memory requests. As shown in the input encoding breakdown (Figure 3.8), other relatively simple compute operations also consume significant number of cycles because they have to wait for the long scoreboard to resolve the global memory requests corresponding to the grid lookups.

Table 3.2 also presents the percentage utilization of the GPU compute and memory resources for the MLP kernel. Our results show that for MLP kernels as well, the memory utilization is higher than compute utilization. This is because, for a constant batch size, the computational cost of a neural network scales quadratically with its width due to the matrix multiplication operations required for each layer, where the number of operations is proportional to the square of the number of neurons (i.e., $O(M^2)$). In contrast, the memory traffic scales linearly with the width because each weight and activation is accessed and updated a constant number of times per layer, resulting in $O(M)$ memory accesses. This distinction arises from the fact that while computations involve all pairwise interactions between neurons in each layer, efficient memory access patterns ensure that the data movement for these operations grows only linearly with the number of neurons. This principle is discussed in detail in [135]. For relatively big MLPs with a large number of neurons, the quadratic compute cost quickly becomes the bottleneck and further optimizing memory traffic has little to no performance benefits because the cost would be overshadowed by the quadratic compute requirements. However, all our neural graphics applications have tiny MLPs, with only 2-4 hidden layers and 64 hidden neurons per layer (Table 3.1). In smaller MLPs, the compute cost and the linear memory traffic become asymptotically comparable and hence the memory traffic cost starts to matter. Moreover, the GPUs and the modern processors in general usu-

Application	Parameters
NeRF multi res. hashgrid	GridEncoding: Nmin=16 b=1.51572 F=2 T=2^19 L=16 Density model: 3–[HashGrid]–>32–[FullyFusedMLP(neurons=64;layers=3)]–>1 Color model: 3–[Composite]–>16+16–[FullyFusedMLP(neurons=64;layers=4)]–>3
NeRF multi res. densegrid	GridEncoding: Nmin=16 b=1.405 F=2 T=2^19 L=8 Density model: 3–[DenseGrid]–>16–[FullyFusedMLP(neurons=64;layers=3)]–>1 Color model: 3–[Composite]–>16+16–[FullyFusedMLP(neurons=64;layers=4)]–>3
NeRF low res. densegrid	GridEncoding: Nmin=128 b=1 F=8 T=2^19 L=2 Density model: 3–[TiledGrid]–>16–[FullyFusedMLP(neurons=64;layers=3)]–>1 Color model: 3–[Composite]–>16+16–[FullyFusedMLP(neurons=64;layers=4)]–>3
NSDF multi res. hashgrid	GridEncoding: Nmin=16 b=1.38191 F=2 T=2^19 L=16 Model: 3–[HashGrid]–>32–[FullyFusedMLP(neurons=64;layers=4)]–>1
NSDF multi res. densegrid	GridEncoding: Nmin=16 b=1.405 F=2 T=2^19 L=8 Model: 3–[DenseGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>1
NSDF low res. densegrid	GridEncoding: Nmin=128 b=1 F=8 T=2^19 L=2 Model: 3–[TiledGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>1
NVR multi res. hashgrid	GridEncoding: Nmin=16 b=1.275 F=2 T=2^19 L=16 Model: 3–[HashGrid]–>32–[FullyFusedMLP(neurons=64;layers=4)]–>4
NVR multi res. densegrid	GridEncoding: Nmin=16 b=1.405 F=2 T=2^19 L=8 Model: 3–[DenseGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>4
NVR low res. densegrid	GridEncoding: Nmin=128 b=1 F=8 T=2^19 L=2 Model: 3–[TiledGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>4
GIA multi res. hashgrid	GridEncoding: Nmin=16 b=1.25992 F=2 T=2^24 L=16 Model: 2–[HashGrid]–>32–[FullyFusedMLP(neurons=64;layers=4)]–>3
GIA multi res. densegrid	GridEncoding: Nmin=16 b=1.405 F=2 T=2^24 L=8 Model: 2–[DenseGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>3
GIA low res. densegrid	GridEncoding: Nmin=128 b=1 F=8 T=2^24 L=2 Model: 2–[TiledGrid]–>16–[FullyFusedMLP(neurons=64;layers=4)]–>3

Table 3.1: The parameters of the four neural graphics applications for different input encoding schemes.

ally have larger computational throughput than memory bandwidth, which means that, for small number of neurons, the memory cost dominates.

3.5 Neural Fields Processor: An Architecture for Neural Graphics

Since our profiling motivates the need to accelerate the input encoding and multi-layer perceptron kernels, we design an architecture - a *neural fields processor* (Figure 3.9) - to accelerate the input encoding and multi-layer perceptron kernels in hardware using separate hardware engines for the two. To accelerate input encoding kernel, our kernel level analysis (Section 3.4) suggests providing hardware support for the grid lookups and the dataflow of the input encoding. Moreover, in neural graphics applications, the outputs of the input encoding kernel are always consumed by the multi-layer perceptron kernel (Figure 3.4). In Müller et al's GPU implementation of

Table 3.2: The percentage utilization of the GPU compute and memory resources, for the input encoding and the MLP kernels for all four neural graphics applications and for different input encodings.

App.-Kernel	Grid Size/Block Size	Comp. Util. per kernel call	Mem. Util. per kernel call	Kernel Calls	Comp. Util. avg. across application	Mem. Util. avg. across application
NeRF multi res. hashgrid	(3853;16;1)/(512;1;1)	61.73	72.85	59	40.63	72.02
NeRF MLP	(3853;16;1)/(512;1;1)	34.3	65.2	118	33.36	63.07
NSDF multi res. hashgrid	(1823;16;1)/(512;1;1)	73.08	43.54	256	15.97	30.8
NSDF MLP	(1823;16;1)/(512;1;1)	38.13	71.74	256	9.76	18.28
NVR multi res. hashgrid	(403;16;1)/(512;1;1)	52.5	59.03	48	18.67	30.36
NVR MLP	(403;16;1)/(512;1;1)	36.51	67.01	48	11.51	21.05
GIA multi res. hashgrid	(4050;16;1)/(512;1;1)	82.87	62.23	1	82.87	62.23
GIA MLP	(4050;16;1)/(512;1;1)	39.1	72.22	1	39.1	72.22
NeRF multi res. densegrid	(3966;8;1)/(512;1;1)	71.39	91.81	45	57.37	72.31
NeRF MLP	(3966;8;1)/(512;1;1)	39.53	68.4	90	34.51	62.31
NSDF multi res. densegrid	(1823;8;1)/(512;1;1)	76.1	48.25	244	18.38	21.28
NSDF MLP	(1823;8;1)/(512;1;1)	41.66	73.49	244	11.06	19.41
NVR multi res. densegrid	(403;8;1)/(512;1;1)	57.38	56.8	48	17.41	22.43
NVR MLP	(403;8;1)/(512;1;1)	39.83	67.67	48	12.17	20.59
GIA multi res. densegrid	(4050;8;1)/(512;1;1)	78.53	65.83	1	78.53	65.83
GIA MLP	(4050;8;1)/(512;1;1)	42.89	73.07	1	42.89	73.07
NeRF low res. densegrid	(3980;2;1)/(512;1;1)	53.83	49.74	43	31.17	59.57
NeRF MLP	(3980;2;1)/(512;1;1)	39.41	68.17	86	35.5	64.1
NSDF low res. densegrid	(1823;2;1)/(512;1;1)	55.88	45.52	260	7.21	20.07
NSDF MLP	(1823;2;1)/(512;1;1)	41.37	72.98	260	10.34	18.14
NVR low res. densegrid	(403;2;1)/(512;1;1)	22.71	69.16	48	6.29	22.71
NVR MLP	(403;2;1)/(512;1;1)	39.2	66.58	48	12.11	20.48
GIA low res. densegrid	(4050;2;1)/(512;1;1)	66.15	59.12	1	66.15	59.12
GIA MLP	(4050;2;1)/(512;1;1)	42.87	73.02	1	42.87	73.02

these applications [134, 155], the input encoding kernel writes outputs to device memory and the multi-layer perceptron kernel fetches that data again for further processing (Figure 3.7). This leads to unnecessary memory traffic and waste of energy for DRAM accesses that can potentially be avoided. This opportunity can be exploited in the neural fields processor hardware by fusing the input encoding and multi-layer perceptron engines in such a way that the input encoding engine directly writes the outputs to the input memory of the multi-layer perceptron engine.

Figure 3.9-a presents the architecture of the input encoding hardware engine. Each input encoding hardware engine has a dedicated on-chip SRAM (*grid_sram*) to cache the lookup table for one resolution level. The size of the *grid_sram* (1MB per input encoding engine) is chosen such that the entire lookup table for one resolution level fits on the on-chip SRAM, and the off-chip memory access penalty could be avoided for grid lookups. The lookup table for one resolution level is cached once on the dedicated *grid_sram* of one input encoding engine and then lookups are performed for all the inputs for the entire frame. Our hardware architecture has 16 input encoding engines to match the maximum number of resolution levels of the neural graphics applications. As the multi resolution hashgrid input encoding has 16 resolution

levels, each of the 16 input encoding engines caches the lookup table corresponding to one resolution level and then performs the input encoding for all the resolution levels in parallel. As multi resolution densegrid input encoding has eight resolution levels, two inputs can be processed in parallel. Similarly, as low resolution densegrid input encoding has two resolution levels, the input encoding IP can process eight inputs in parallel.

The normalized input coordinates are pre-fetched into the input FIFO (Figure 3.9-a). The *grid_scale* module calculates the scale of the grid from the base-resolution of the grid and the resolution level being processed. The *pos_fract* module is responsible for converting the normalized input coordinates to the absolute coordinates. It multiplies the normalized coordinates with grid scale to compute the absolute coordinates, which are then passed to the *grid_index* module for the index calculation. The *grid_index* module is responsible for calculating the final indices for the lookup operations. The *grid_index* module can be configured to either hash in the indices for the multi resolution hashgrid encoding or compute the indices without the hashing function for the multi resolution densegrid and low resolution densegrid encoding types. The algorithm to compute the indices for the lookup operation takes the modulo of the indices with the hash-map size as an intermediate operation before the final indices could be calculated. We observe that the hash-map size is always power of two for all our neural graphics applications. We exploit this optimization opportunity in hardware and approximate the modulo operation with shift operation in input encoding engine. The final outputs of the *grid_index* modules are actual indices that are directly used to perform the feature lookups from the *grid_sram*. The *interpol_weights* module computes the interpolation weights that are then multiplied with the features to compute the final features that are fed to the multi-layer perceptron engine. As each input encoding engine calculates features for one resolution level, the outputs of the input encoding engines are concatenated together to get the final input vector for the MLP engine.

Since MLPs in neural graphics applications are small (2-4 hidden layers, 64 neurons in each hidden layer), our MLP engine has a 64×64 grid of MAC units that computes one layer of the multi-layer perceptron at a time. As the size of the hidden layer is relatively small, a dedicated small on-chip SRAM is used to store the intermediate features of the hidden layers. Keeping the intermediate features on-chip removes the off-chip memory accesses

for storing/fetching the intermediate features and improves the performance by 1OOM [135].

Figure 3.10-a shows the interaction of the neural fields processor with the GPU. A set of N NFPs are organized as a neural graphics processing cluster (NGPC) connected to the shared L2 cache. Figure 3.10-b and 3.10-c show the programming model of the NGPC. The programming model for the NGPC involves the GPU command buffer [156] configuring the NGPC and scheduling the input encoding and the multi-layer perceptron kernels on the NGPC. The rest of the kernels are scheduled just as conventional CUDA kernels on the streaming multiprocessors. The outputs of the NGPC are written back to the GPU memory and are read by the streaming multiprocessors, which then compute the rest of the kernels. The inputs are divided into batches. While the GPU is processing the rest of the neural graphics application kernels for the Nth batch of inputs, the N+1st batch is scheduled on the NGPC to compute the input encoding and MLP kernels in parallel, as shown in Figure 3.10-b.

3.6 Evaluation and Results

We provide an emulator to evaluate the performance of the neural graphics applications on our architecture. The block diagram of the emulator is in Figure 3.11. The inputs to the emulator are 1) The neural graphics application parameters such as the input encoding type, the grid resolution levels, the input encoding parameters, the multi-layer perceptron structure; 2) the architecture parameters such as the number of NFP units in the NGPC, operating frequency of the NGPC, the critical path delay of the architecture, the memory access time for different on-chip SRAM blocks of the NFP , the memory access time for device memory, the area and power estimates of the NGPC; 3) the kernel level breakdown of the performance of the neural graphics application on the GPU; and 4) the frame resolution. The outputs of the emulator are 1) the overall performance of the neural graphics application with the input encoding and the multi-layer perceptron scheduled on NGPC and the rest of the kernels scheduled on the GPU. 2) the overall area and power of the NGPC.

We evaluated the performance of the neural graphics applications on our proposed architecture for the scaling factor of 8, 16, 32, and 64, where NGPC-8 has 8 NFP unit, NGPC-16 has 16 NFP units, and so on. Figure 3.12a presents the overall performance of the neural graphics applications with multi resolution hashgrid encoding on the proposed architecture. Our results show that when we have only 8 NFP units in the NGPC, on average across all four neural graphics applications, we get overall 12.94× performance benefits compared to the GPU baseline. When we scale the number of NFP units in an NGPC to 16, 32, and 64, the performance benefits averaged across the four representative neural graphics applications increase to 20.85×, 33.73×, and 39.04× respectively.

Figure 3.12b presents the overall performance of the neural graphics applications with multi resolution densegrid encoding. Our results show that when we have only 8 NFP units in NGPC, on average across all four neural graphics applications, we get overall 9.05× performance benefits compared to the GPU baseline. When we scale the number of NFP units in an NGPC to 16, 32, and 64, the performance benefits averaged across the four representative neural graphics applications increase to 14.22×, 22.57×, and 26.22× respectively.

Figure 3.12c presents the overall performance of the neural graphics applications with low resolution densegrid encoding. Our results show that when we have only 8 NFP units in NGPC, on average across all four neural graphics applications, we get overall 9.37× performance benefits compared to the GPU baseline. When we scale the number of NFP units in an NGPC to 16, 32, and 64, the performance benefits averaged across the four representative neural graphics applications increase to 14.66×, 22.97×, and 26.4× respectively.

Our results also show that NeRF performance plateaus for NGPC-64. I.e., increasing the number of NFP beyond 64 does not improve the overall performance of the application. This is because the time consumed by the non-input encoding and multi-layer perceptron kernels becomes the performance bottleneck. Similarly, for NSDF, NVR, and GIA, the performance plateaus for NGPC-32, NGPC-16, and NGPC-64 respectively.

In order to better understand where the application level benefits are coming from, we also compare the performance of the input encoding and the multi-layer perceptron kernels individually, with their GPU based imple-

mentations. Figure 3.13 presents the performance improvement of the input encoding kernel and the multi-layer perceptron kernels individually, on our architecture, for scaling factors of 8, 16, 32, and 64. Our results show that for multi resolution hashgrid, on average across four neural graphics applications, the neural graphics processing cluster -64 has performance improvement of 246× and 1232× for the input encoding kernel and the multi-layer perceptron kernel, respectively. For multi resolution densegrid, neural graphics processing cluster -64 has performance improvement of 379× and 1070× for the input encoding kernel and the multi-layer perceptron kernel, respectively. For low resolution densegrid, the neural graphics processing cluster -64 has performance improvement of 2353× and 1451× for the input encoding kernel and the multi-layer perceptron kernel, respectively.

In Figure 3.14, we present the number of pixels that can be rendered for a given FPS target with and without neura graphics processing cluster. Horizontal lines in the figure mark the number of pixels in HD (1280×720), FHD (1920×1080), QHD/2k (2560×1440), Ultra HD/4k (3820×2160), 5k (5120×2880), and 8k (7680×4320) frame resolutions. The vertical bars show the number of pixels rendered within the time budget of $1000/30 = 33.33$ms, $1000/60 = 16.67$ms, $1000/90 = 11.11$ms, and $1000/120 = 8.33$ms corresponding to the FPS targets of 30, 60, 90, and 120 FPS respectively. Our results show that with multi resolution hashgrid encoding, NGPC enables the rendering of 4k Ultra HD resolution frames at 30 FPS for NeRF and 8k Ultra HD resolution frames at 120 FPS for all our other neural graphics applications.

For estimating the area and power overheads, we wrote RTL for neural fields processor and synthesized it using Synopsys design compiler along with the Nangate 45nm open cell library. We used CACTI to get the area and power estimates for the SRAM blocks. Figure 3.15 shows the area and power of the different configurations of NGPC normalized with respect to Nvidia's RTX3090 area and power. In order to get iso-technode comparison, we scaled the area and power of NGPC to 7nm using often-used scaling formulas [157]. Our estimates show that NGPC-8 that has only one NFP unit increases the die area of GPU by only $\sim$ 4.52% with the power overhead of $\sim$ 2.75%. Similarly, NGPC-16, NGPC-32, and NGPC-64 increase the GPU die area by $\sim$ 9.04%, $\sim$ 18.01%, and $\sim$ 36.18% respectively and GPU power by $\sim$ 5.51%, $\sim$ 11.03%, and $\sim$ 22.06% respectively.

Table 3.3 presents the input/output bandwidth and the data access time for our NGPC architecture. Our estimates suggest that, for 60FPS, the bandwidth requirement for NGPC architecture is 231 GB/s for NeRF and 69 GB/s for all other neural graphics applications. The memory bandwidth of Nvidia RTX 3090 is 936.2 GB/s [158]. Hence, for 60FPS, the IO bandwidth of the accelerator is $\sim 24\%$ of the GPU memory bandwidth for NeRF and only $\sim 7\%$ of the GPU memory bandwidth for NSDF, NVR, and GIA. Moreover, the massively parallel nature of the workload and the high memory bandwidth of the GPU compared to the IO bandwidth of the accelerator keeps the encoding engines busy with high utilization. This translates to data access time of 4.12ms for NeRF and 1.23ms for all other neural graphics applications.

In order to get confidence in our evaluation methodology and the reported speedup numbers, we performed a sanity check against Amdahl's law and presented our analysis in figures 3.12a, 3.12b, and 3.12c. Horizontal lines in the figures 3.12a, 3.12b, and 3.12c show the peak speedup bounded by Amdahl's law and the vertical bars show the reported speedup from our emulator. Our analysis shows that the reported speedup is always under the Amdahl-driven analytical bounding of speedup. We also modeled the performance of our MLP engine using popular open-source DNN-architecture-modeling frameworks Timeloop [159] and Accelergy [160]. In figure 3.13 we also present the performance benefits of the MLP engine modeled with the timeloop and accelergy. Our analysis shows that the performance benefits reported by our emulator are within $\sim 7\%$ of the performance benefits modeled with the timeloop and accelergy.

To analyze the performance trends for the latest GPU models, although we did not have direct access to the A100 and H100 GPUs, we instead relied on performance and energy efficiency estimates sourced from existing literature. For instance, the NVIDIA RTX3090 delivers 285 TOPS for int8 calculations [161]. In comparison, the A100 and H100 GPUs have 634 TOPS [162] and 4000 TOPS [163], respectively. This equates to the A100 being approximately 2.22 times more performant than the RTX3090 for int8 operations, whereas the H100 is about 14.04 times more performant. When integrated with our proposed Neural Graphics Processing Cluster (NGPC), the performance improvement claims translate to approximately $\frac{58}{2.22} \approx 26.12$ times with the A100 and $\frac{58}{14.04} \approx 4.1$ times with the H100, relative to the GPU-only

Table 3.3: The input/output bandwidth and the data access time for our NGPC architecture.

App.	Input BW (GB/s)	Output BW (GB/s)	Totoal BW (GB/s)	Access time (ms)
NeRF	69.523	46.349	231.743	4.126
NSDF	34.761	34.761	69.523	1.238
GIA	34.761	34.761	69.523	1.238
NVR	34.761	34.761	69.523	1.238

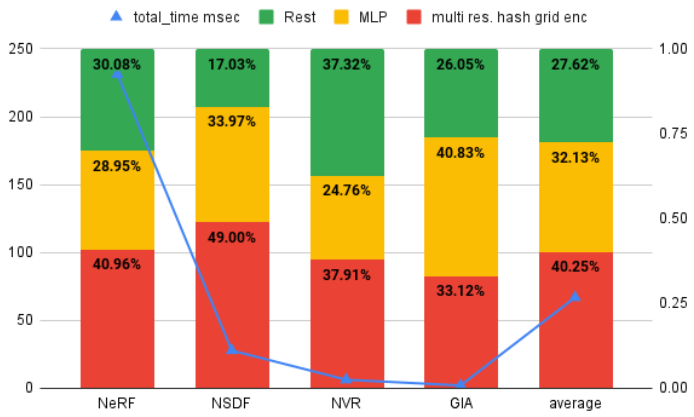
baseline performance.

3.7 Summary and Discussion

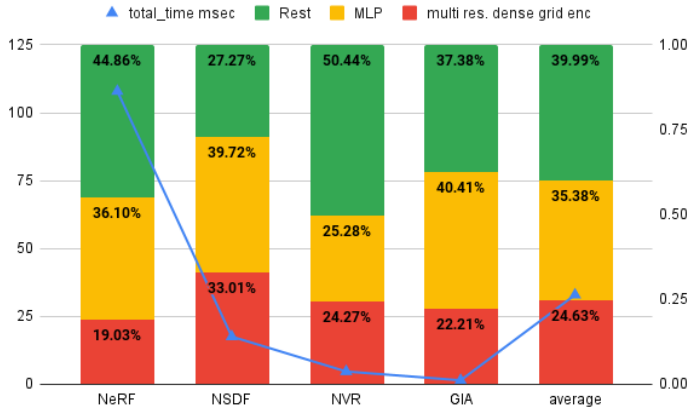
Neural graphics promises a fast, deterministic time replacement for traditional rendering algorithms. In this chapter, we address the question: *does neural graphics need hardware support?* We studied four representative neural graphics applications (NeRF, NSDF, NVR, and GIA) and showed that, if we want to render 4k resolution frames at 60FPS, there is the gap of $\sim 1.51\times$ to $55.50\times$ between the desired performance and the state of the art. For AR and VR applications, there is a larger gap of $\sim$ 2-4OOM between the desired performance and target power. Through in depth analysis, we identified that the input encoding and the multi-layer perceptron kernels are the performance bottlenecks consuming 72.37%, 60.0%, and 59.96% of application time for *multi resolution hashgrid encoding*, *multi resolution densegrid encoding*, and *low resolution densegrid encoding* respectively. Based on the compute and memory access characteristics of the input encoding and the multi-layer perceptron kernels, we proposed neural graphics processing cluster – a scalable hardware architecture that directly accelerates the input encoding and multi-layer perceptron kernels through dedicated engines and supports a wide range of neural graphics applications. To achieve good overall application level performance improvements, we also accelerate the rest of the kernels by fusion into a single kernel, leading to a $\sim 9.94\times$ speedup compared to previous optimized implementations [134] which is sufficient to remove this performance bottleneck. Our results show that, NGPC gives up to $58.36\times$ end-to-end application-level performance improvement. For *multi resolution hashgrid encoding* on average across the four neural graphics applications, the performance benefits of our architecture are $12.94\times$, $20.85\times$, $33.73\times$, and $39.04\times$ for the scaling factor of 8, 16, 32, and 64, re-

spectively. For *multi resolution densegrid encoding* on average across the four neural graphics applications, the performance benefits of our architecture are 9.05×, 14.22×, 22.57×, and 26.22× for the scaling factor of 8, 16, 32, and 64, respectively. Similarly, for *low resolution densegrid encoding* on average across the four neural graphics applications, the performance benefits of our architecture are 9.37×, 14.66×, 22.97×, and 26.4× for the scaling factor of 8, 16, 32, and 64, respectively. Our results show that with multi resolution hashgrid encoding, NGPC enables the rendering of 4k Ultra HD resolution frames at 30 FPS for NeRF and 8k Ultra HD resolution frames at 120 FPS for all our other neural graphics applications.

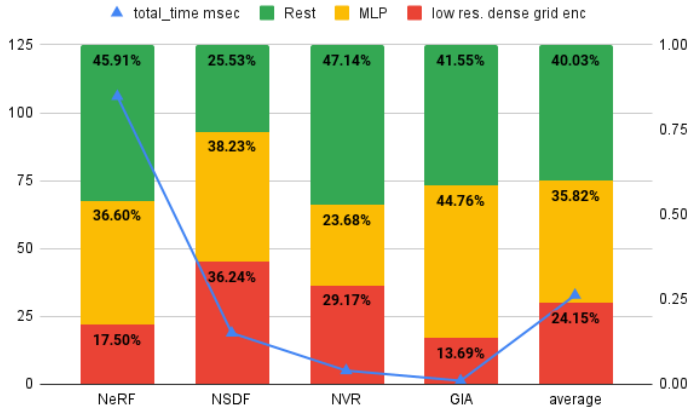
In Chapter 2, we discussed various applications of non-generative neural graphics across different domains, focusing on their real-time, near-real- time, and energy efficiency needs. Our analysis of non- generative neural graphics applications, presented in section 3.4, revealed that neural representations require significant performance boosts to meet real-time or near-real-time demands. To address these needs, we proposed the Neural Graphics Processing Cluster (NGPC) architecture, which provides up to a 58.36× performance improvement over the GPU-only baseline, enabling up to 120 FPS, which can be considered real-time. For example, rendering 4K resolution frames at 60 FPS, previously falling short by 1.51× to 55.50×, is now feasible. Additionally, near-real-time applications like medical imaging reconstruction tasks can also benefit from this architecture. However, further optimizations are needed to fully satisfy the energy efficiency demands of non- generative neural graphics applications, such as robotics, indicating a continued need for dedicated hardware support and algorithmic improvements.



(a)



(b)



(c)

Figure 3.5: Kernel level performance breakdown analysis of neural graphics applications: The number of cycles consumed by the different kernels of the neural graphics applications (as percentage of total application cycles). (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding.*

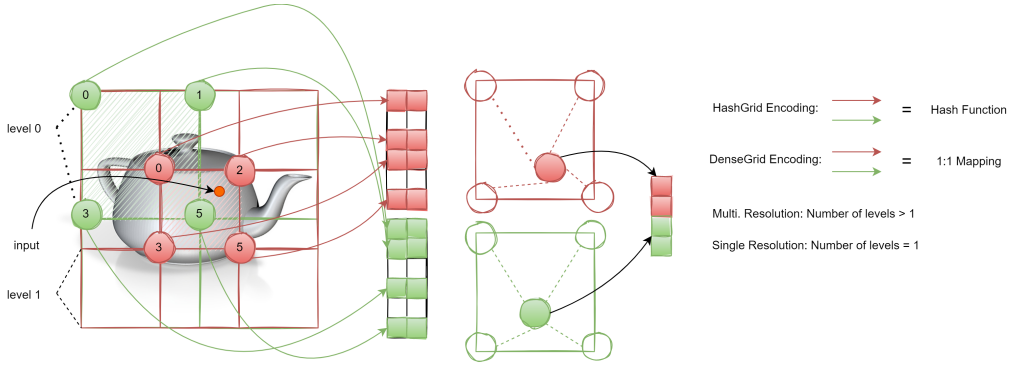


Figure 3.6: Illustration of *Multi resolution grid encoding*.

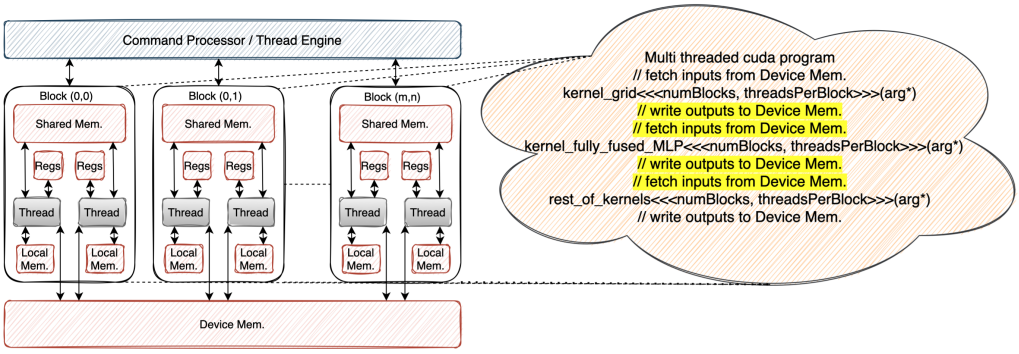


Figure 3.7: Scheduling of a typical neural graphics application on GPU.

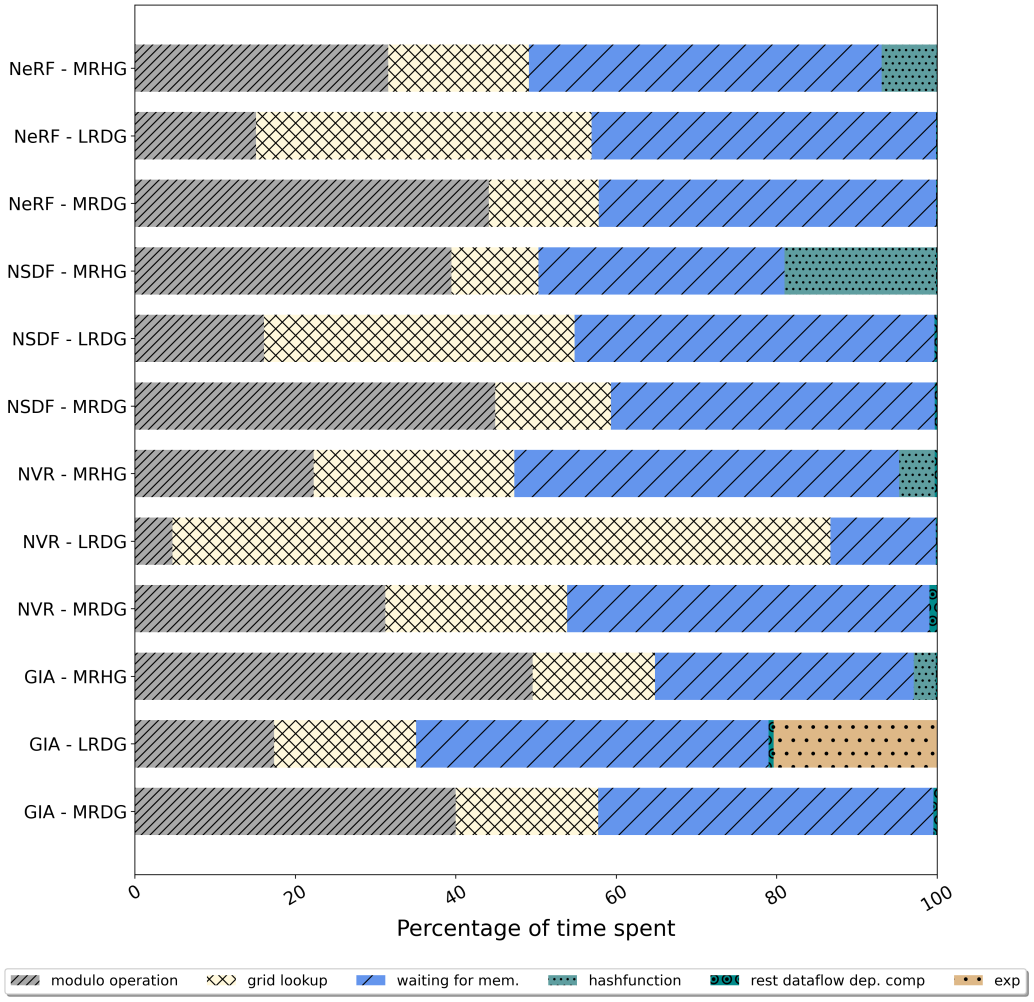


Figure 3.8: The operation level breakdown of the of input encoding kernels for different input encoding types. The five most expensive operations in terms of number of cycles spent. MRHG : *multi resolution hashgrid* , MRDG : *multi resolution densegrid* , LRDG : *low resolution densegrid* .

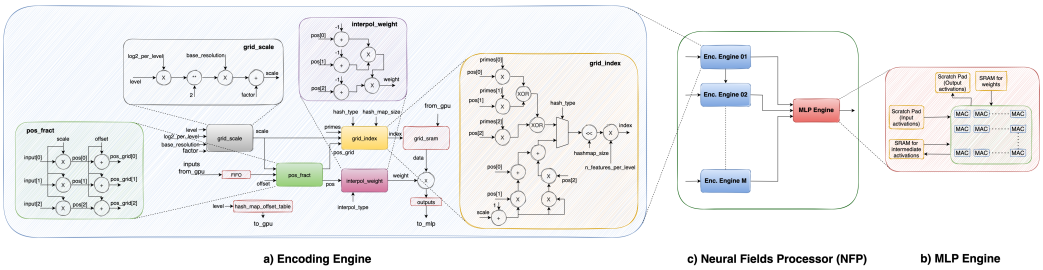


Figure 3.9: Architecture of the Neural Fields Processor (NFP)

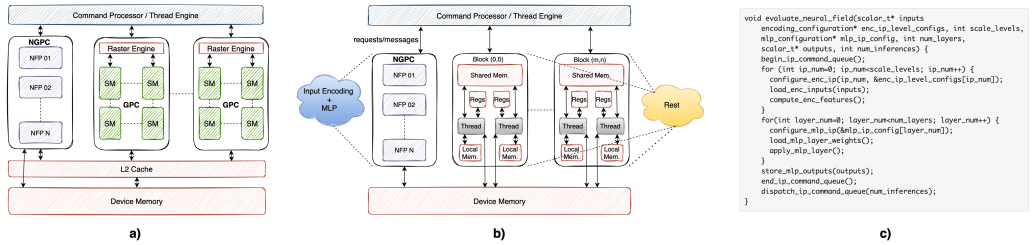


Figure 3.10: a) Interaction of the NGPC with GPU. b) Programming model for NGPC. c) Pseudocode for programming NGPC.

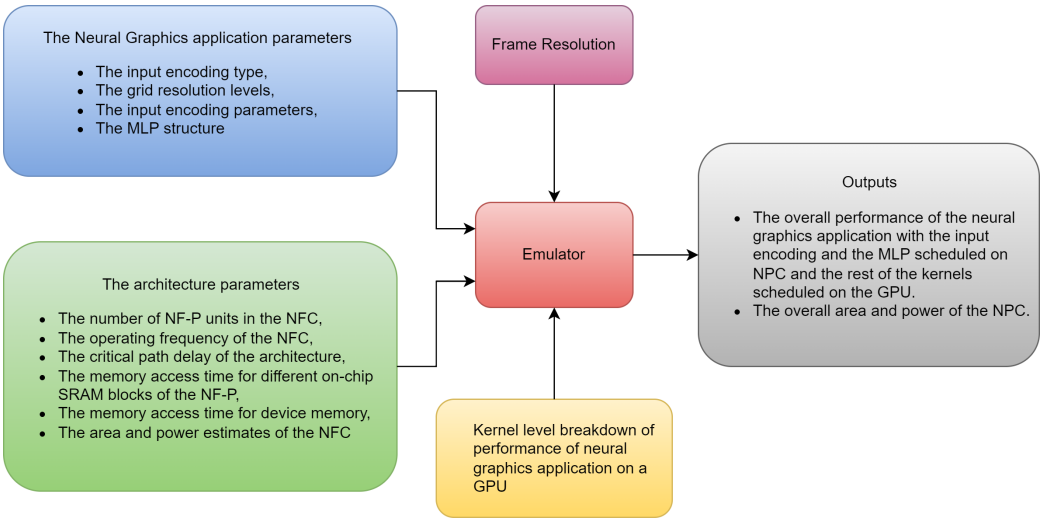
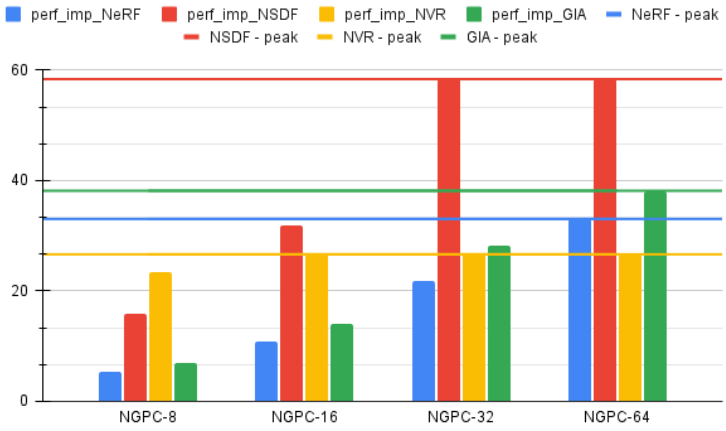
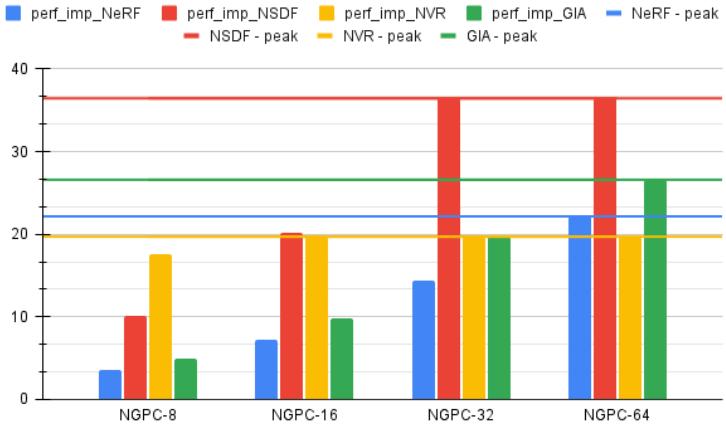


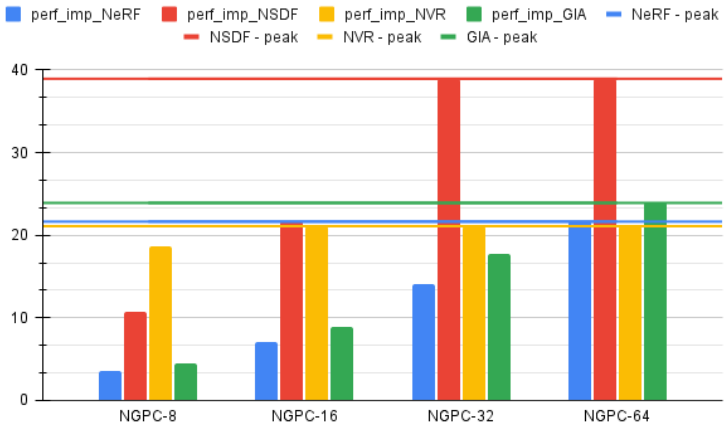
Figure 3.11: Block diagram of the emulator.



(a)

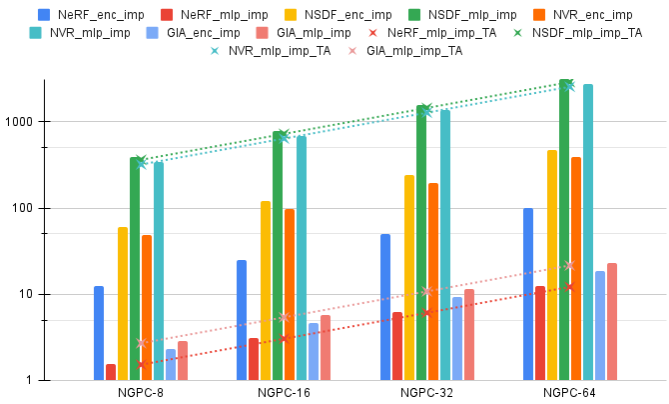


(b)

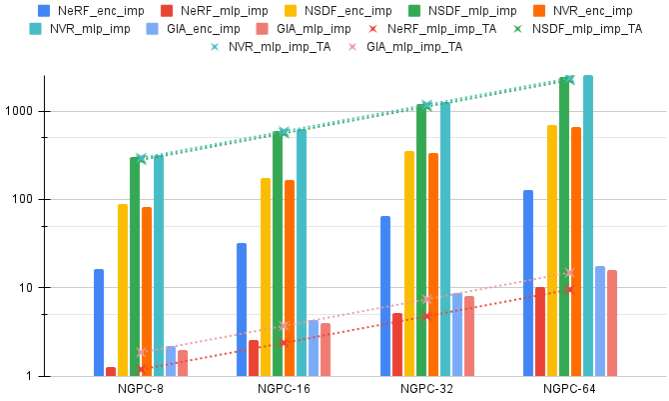


(c)

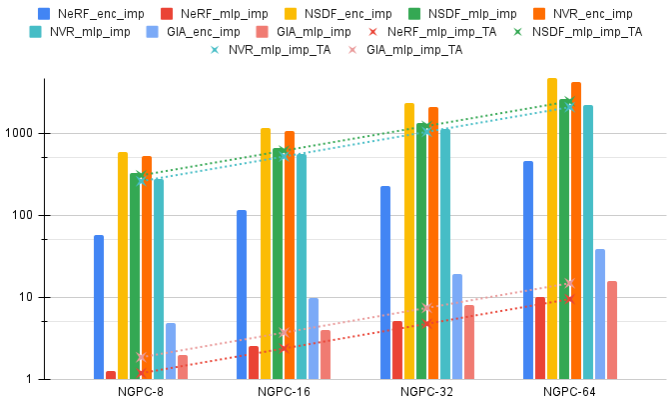
Figure 3.12: The performance of the neural graphics applications on our proposed architecture for the scaling factor of 8, 16, 32, and 64. NGPC-8 has 8 NFP units, NGPC-16 has 16 NFP units and, so on. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding.*



(a)

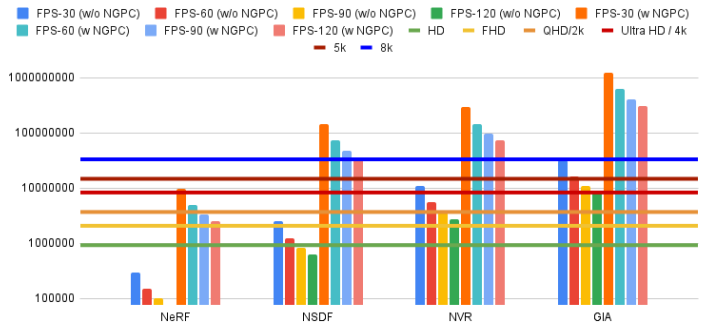


(b)

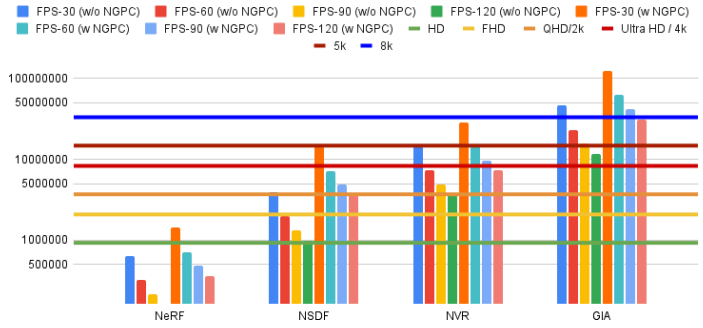


(c)

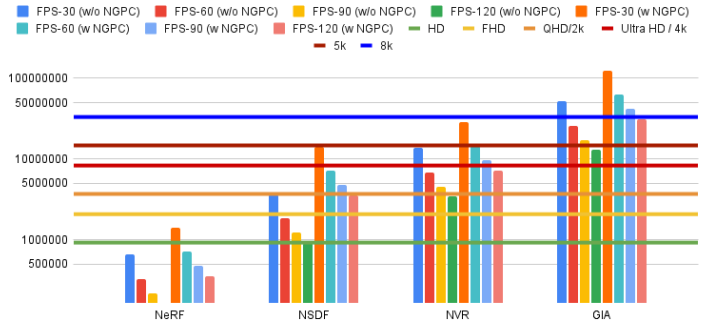
Figure 3.13: The performance improvement of the input encoding kernels and the multi-layer perceptron kernels individually, on our architecture, for the scaling factors of 8, 16, 32, and 64. For all four neural graphics applications and different input encoding types. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*, where dotted lines labeled as mlp_imp_TA are the performance improvements of MLP engine modeled with timeloop and accelergy.



(a)



(b)



(c)

Figure 3.14: The number of pixels that can be rendered for a given FPS target with and without neural graphics processing cluster. (a) *Multi resolution hashgrid encoding*, (b) *Multi resolution densegrid encoding*, (c) *Low resolution densegrid encoding*.

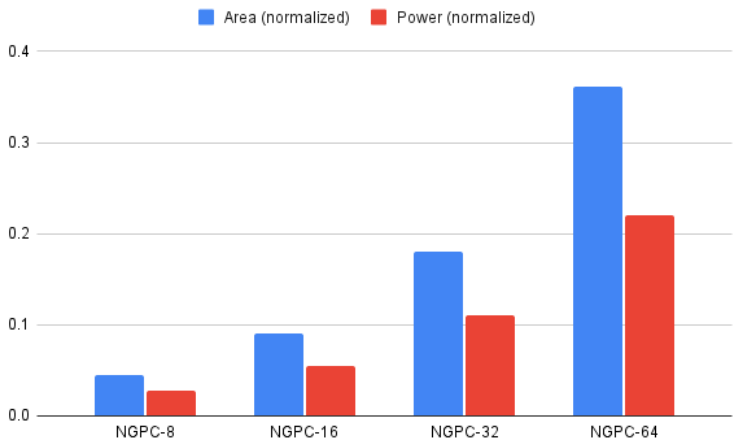


Figure 3.15: The area and Power of neural graphics processing cluster , normalized with respect to the area and power of Nvidia RTX 3090 die area and power.

CHAPTER 4

HARDWARE ACCELERATION OF LATENT DIFFUSION MODELS (GENERATIVE NEURAL GRAPHICS)

4.1 Introduction

Diffusion models [26–28,63,67,101,108,121,164–169] have recently taken over the world of generative AI, challenging the dominance of more mature generative techniques such as Generative Adversarial Networks (GANs) [170], Variational Auto Encoders (VAEs) [171], and flow based models [172]. They are now considered the state-of–art in generative tasks [28] and have demonstrated capabilities across a vast number of domains, including computer vision [26], computer graphics [27], natural language processing (NLP) [108] [173], scientific computing [174] [175] [176].

Unfortunately, diffusion models also have high energy footprint, especially due to their scale of impact. Since 2023, over 15 billion images have been generated using text-to-image algorithms [177]. To put this in perspective, it took photographers 150 years, from the first photograph taken in 1826 until 1975, to reach the same number of images. This means that by 2023, AI has created as many images in a single year as photographers have in 150 years. Since the launch of DALL-E 2 [178], people are creating an average of 34 million images per day [177]. At the recommended FaceBook image resolution of $\approx 756 Kpixels/image$ [179], we estimate that the total energy requirement for generating these many images using most optimized software implementation of Stable Diffusion [180], by far the most popular diffusion model, on RTX3090, a state-of-art GPU, would be 13.2 terajoules or 3.67 GWh. For context, the average U.S. residential electric-utility customer used approximately 10,791 kilowatt-hours (kWh) of electricity annually [181]. The energy requirements may only increase as generative becomes even more widely used [177] and as high resolutions cameras and displays become more mainstream [182].

Such extraordinary energy requirements for diffusion models, even for optimized model software, motivate the need for optimizing diffusion model hardware for energy efficiency.

In this chapter, we study diffusion model performance and energy efficiency, focusing on latent diffusion models for graphics applications. We observe that the cost of generating images scales superlinearly with the desired image resolution. Our analysis shows that, when the desired image resolution is increased from VGA to 8K resolution, the effective number of pixels increases by 108× whereas the compute time increases by 1710× on RTX3090. We identify that the superlinear increase in the generation cost of latent diffusion models is due to the underlying attention mechanism. We also observe that natively the latent diffusion models have very low percentage of sparsity and special values (values that are '0', '1' or powers' of '2'). But with methods such as quantization and model fine-tuning, the percentage of sparsity and special values can be significantly increased.

Based on these key insights, we propose Latent Diffusion Model Processing Unit (LDPU) a scalable architecture targeting energy efficient execution of latent diffusion models. LDPU has hardware support for sparsity and special values. The architecture of LDPU is described in depth in section 4.5.

We also present a comparative analysis of the LDPU architecture against conventional GPU implementations when running a latent diffusion model. The study reveals that the LDPU architecture, with four systolic arrays and 64×64 Diffusion Processing Element (DPE) grid dimensions, achieves a performance improvement of approximately 1.32× and an energy efficiency of 108× in Int8 configuration of LDPU. For 256×256 DPE grid dimensions, the performance scales to 16×. For the Int4 variant, the improvements are even more pronounced, with performance improvements of up to 25.43× and energy efficiency reaching 572× the GPU baseline. In essence, the energy impact of operating at 8K resolution with the LDPU is equivalent to that of operating at XGA resolution without the LDPU.

Our work makes the following contributions:

- We show that the generation cost of latent diffusion models escalates superlinearly with the desired resolution. We quantify the cost scaling on modern hardware and pinpoint that the superlinear escalation of generation cost is due to the underlying attention mechanism.

- We show that latent diffusion models inherently exhibit a low percentage of sparsity and special values, which include zeros, ones, or powers of two. However, through techniques such as quantization and model fine-tuning, it is possible to significantly increase the incidence of these values.
- We propose Latent Diffusion Model Processing Unit (LDPU), a scalable architecture designed from the ground up to enhance energy efficiency for latent diffusion models, featuring hardware support for sparsity and special values.
- We quantify the benefits of our hardware architecture and show significant performance and energy efficiency benefits against GPU baseline.

To this best of our knowledge, this is the first work on hardware acceleration of latent diffusion models.

4.2 Diffusion Models: An Overview

Diffusion models are probabilistic generative models that progressively decompose the data into a noisy state and then reconstruct the data by reversing the noising process. Formally known as Denoising Diffusion Probabilistic Models (DDPMs), these models operate using two Markov chains: a forward chain that progressively adds noise to data until a simple prior distribution (typically Gaussian) is achieved, and a reverse chain that aims to reconstruct the original data from this noisy state. Figure 4.2 shows the forward and reverse diffusion chains. In the forward diffusion process, Gaussian noise is incrementally added to the input data over several time steps, transforming it from a clean state to pure noise. This process is modeled as a Markov chain where each step involves adding Gaussian noise based on a predefined schedule, ensuring the data becomes increasingly noisy, eventually resembling a simple distribution like a standard normal distribution. The forward diffusion process establishes a well-defined pathway from structured data to unstructured noise, enabling the model to learn the characteristics of the noise distribution at each step. This step is crucial as it sets up the reverse process, where the model will reconstruct the original data by learning the noise patterns [183] [184].

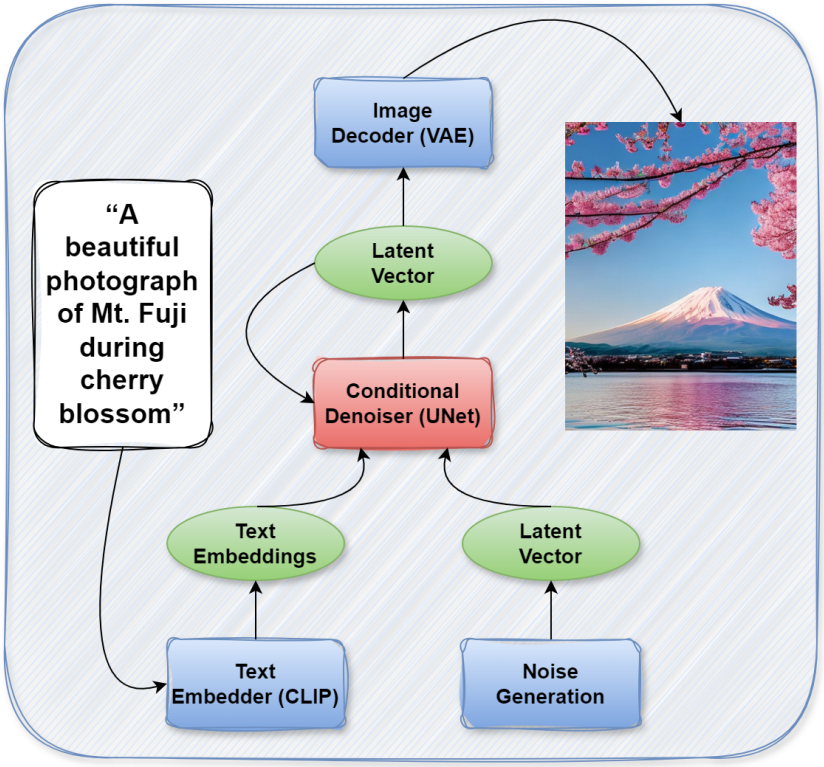


Figure 4.1: A schematic of diffusion model based rendering pipeline. [5]

The reverse diffusion process is the generative phase of the model, where it aims to reverse the noise addition steps performed during the forward diffusion. Starting from pure noise, the model gradually denoises the data, step-by-step, using learned parameters that estimate the noise added at each stage of the forward process. This reverse process is modeled as a Markov chain in reverse order, where each step predicts and removes the noise incrementally. Typically, the reverse diffusion is parameterized by a neural network, often a U-Net, which takes as input the noisy data and the current step in the diffusion process to output the denoised data for the previous step. By iteratively applying this process, the model generates data that closely approximates the original, clean data distribution. The success of the reverse process relies heavily on the accuracy of the noise predictions and the predefined variance schedule used during the forward process [185] [186].

This work focuses on DDIMs (Denoising Diffusion Implicit Models) [187]—a variant of DDPMs [88] which offer faster image generation by allowing some steps in the denoising process to be skipped while maintaining high image quality, making DDIMs particularly suitable for applications requiring

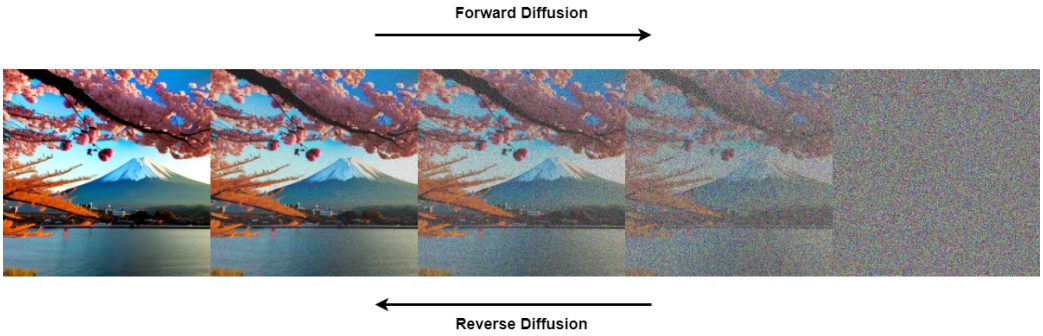


Figure 4.2: Forward and reverse diffusion chains in diffusion models, aka Denoising Diffusion Probabilistic Models (DDPMs).

quicker output without a full sequence generation. Latent Diffusion Models (LDMs) take these principles into lower-dimensional and more computationally efficient realms. LDMs operate in a compressed latent space rather than directly in pixel space, drastically reducing computational demands while maintaining high fidelity in image synthesis [28]. This approach uses an autoencoder to compress images into a latent representation, which is then processed by the diffusion model. This method has shown impressive results in tasks such as text-to-image synthesis, achieving new state-of-the-art performance metrics [28] [180] [188]. Stable Diffusion, a specific LDM, leverages latent space operations optimized for faster performance and can generate detailed images conditioned on textual descriptions rapidly.

Figure 4.1 shows a schematic of the different components of a stable diffusion-based rendering pipeline. The rendering pipeline consists of three major components: 1) a text embedding module, 2) a denoising module utilizing a diffusion model (UNet architecture), and 3) a final transformation from latent space to image space via a variational autoencoder (VAE).

The initial stage in the Stable Diffusion based rendering pipeline (Figure 4.3) involves the text embedding module, which uses a pre-trained CLIP (Contrastive Language–Image Pre-training) [189] module. This module interprets the input text and converts it into a high-dimensional vector that captures the semantic essence of the text. This vector, or embedding, acts as a guiding signal for the image synthesis process, ensuring that the generated images align with the textual input. The effectiveness of CLIP lies in its training on a vast array of images and text, allowing it to understand a wide range of descriptive text and translate these descriptions into visual elements

accurately.

Following text embedding, the process enters the denoising phase which uses a UNet architecture-based diffusion model to iteratively remove noise from an initially random pattern of pixels (latent space representation). The detailed description of the UNet-based Diffusion Model is shown in Figure 4.3. It consists of various down- sampling and up-sampling blocks connected with each other using skip connections. The down-sampling and up-sampling blocks of UNet contain a cascade of Transformer [190] and ResNet [191] modules (Figure 4.4) which are responsible for text and time embeddings respectively. The Transformer modules use an attention mechanism to keep track of contextual text embeddings whereas the ResNet modules use convolutional layers (Conv2D) and are responsible for tracking the timesteps in the iterative denoising process. Table 4.1 summarizes the specific model architecture configurations used for our evaluations. These configurations are recommended by StabilityAI [188]. The model operates with input and output channels each set at 4, built with 320 model channels, integrates attention mechanisms at scales of 4, 2, and 1, comprises 2 residual blocks, utilizes a channel multiplier sequence of [1, 2, 4, 4], equipped with 8 attention heads, and incorporates a spatial transformer with a depth of 1. The context dimension for the transformer is 768. The iterative nature of this denoising process allows for the gradual emergence of coherent and crisp images from a noisy starting point.

The final stage involves the transformation of the denoised latent representation into a tangible image. This is achieved using a Variational Autoencoder (VAE) (Figure 4.3). The VAE reinterprets the processed latent vectors back into pixel data, producing the final image. This component is crucial as it bridges the gap between abstract feature representations in the latent space and the detailed, high-resolution images that are the end product of the Stable Diffusion model. The VAE ensures that the generated images are not only visually appealing but also maintain fidelity to the original textual description.

Training Phase: During the training phase of Latent Diffusion Models, both the forward and reverse diffusion processes are actively engaged. The forward phase involves systematically adding Gaussian noise to the clean data at each step, following a predetermined noise schedule, until the data is entirely obscured, resembling random noise. This gradual degradation helps

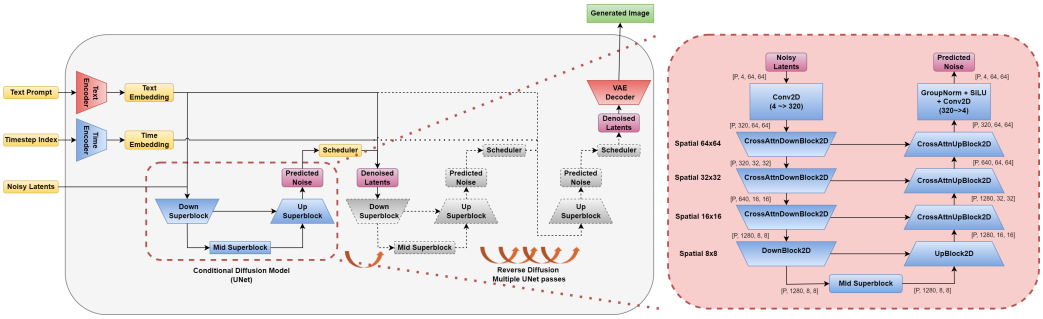


Figure 4.3: Different stages of latent diffusion model based rendering pipeline.

the model learn the precise characteristics of the noise at various stages, crucial for the reverse phase. During the reverse phase, the model reconstructs the original data from its noisy state by iteratively adjusting the model's parameters to minimize the difference between the original data and its reconstructed version from the noise. The optimization of these parameters is achieved through backpropagation, allowing the model to refine its ability to reverse the noise addition effectively and enhance its denoising capabilities.

Inference Phase: In the inference phase, the model utilizes only the reverse diffusion process, employing the parameters fine-tuned during the training stage. Starting with a distribution resembling pure noise, the model sequentially denoises this input, aiming to synthesize data that mirrors the characteristics of the original dataset or generate novel data that aligns with the learned distributions. This phase does not involve any modification to the model's parameters; instead, it leverages the trained network to transform unstructured noise into structured outputs. The effectiveness of this phase heavily depends on the accuracy of the noise modeling and the precision of the learned denoising steps during training. By systematically reducing the noise, the model can produce outputs that are visually compelling and retain the complexity and subtlety of the original data features.

4.3 Do Latent Diffusion Models Need Hardware Support?

To understand the performance characteristics of stable diffusion, we profiled several stable diffusion applications – Text-to-Image Generation [28] [164],

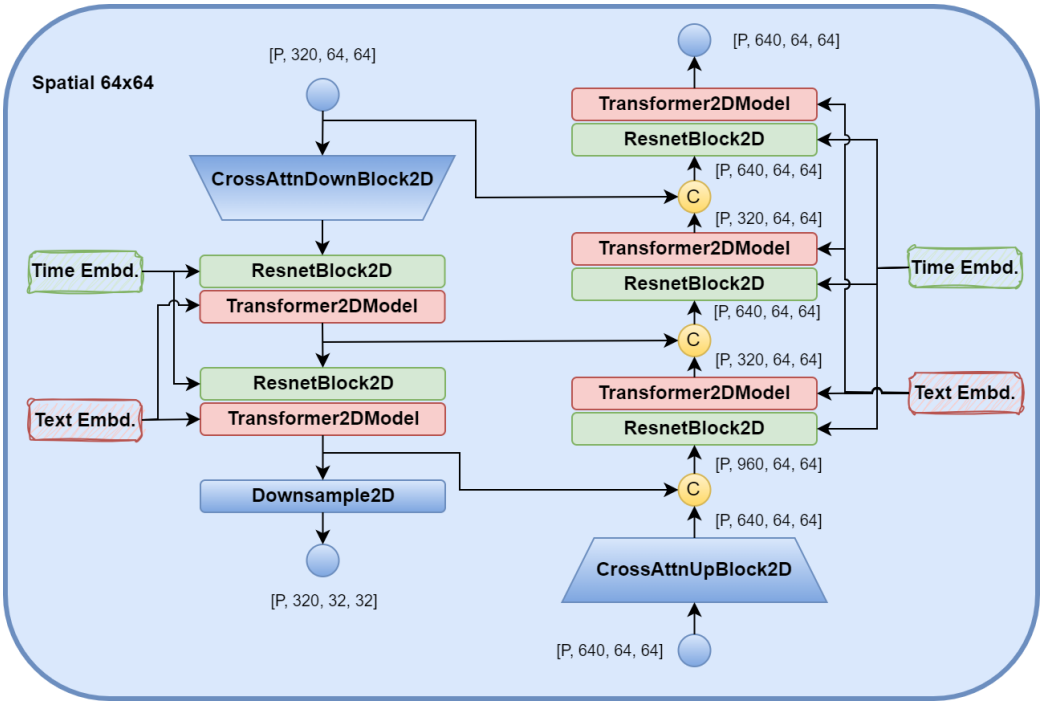


Figure 4.4: Architecture of the primitive block of latent diffusion model (UNet backbone).

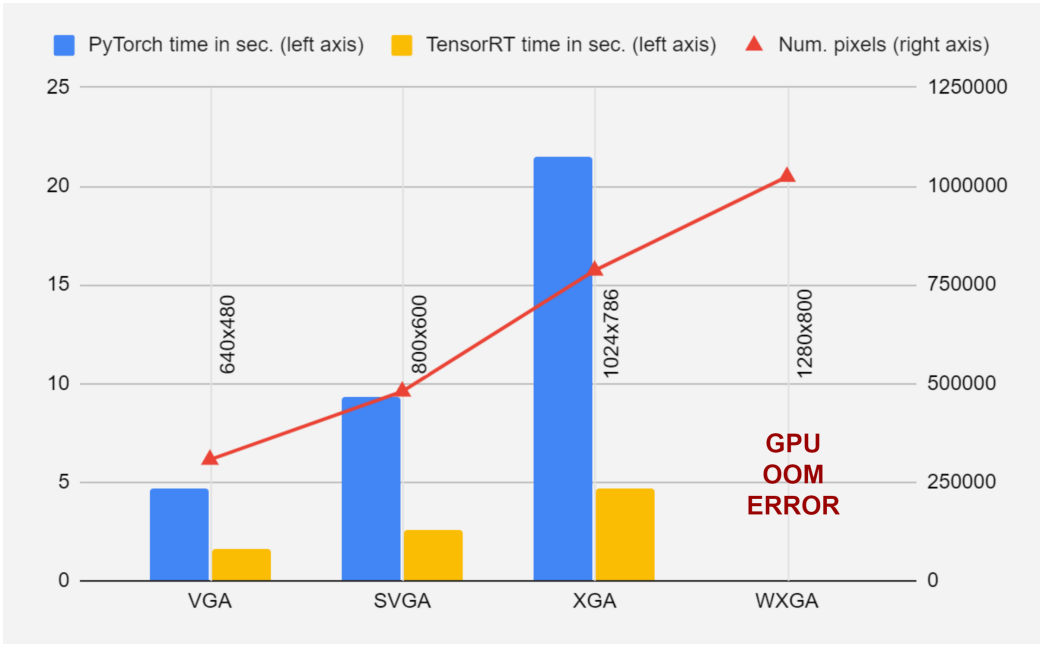


Figure 4.5: The performance characteristics of stable diffusion running on a desktop-class GPU (RTX 3090, 24 GB). For the out-of-the-box PyTorch based stable diffusion implementation with vanilla attention mechanism.

Table 4.1: UNet configuration for latent diffusion model [1].

Parameter	**Value**
Model Target	UNetModel
Image Size	Dynamic
Input Channels	4
Output Channels	4
Model Channels	320
Attention Resolutions	[4, 2, 1]
Number of Residual Blocks	2
Channel Multiplier	[1, 2, 4, 4]
Number of Attention Heads	8
Use Spatial Transformer	True
Transformer Depth	1
Context Dimension	768
Use Checkpointing	True
Legacy Mode	False

Image-to-Image Generation [28], Image In-painting [28] [165], Super Resolution [26] [28] [67], Depth-to-Image Generation [28] [121], and Image Variations [28] [166]. Quickly we observed that all of these applications have similar computational and memory footprints with diffusion models dominating the hardware resource consumption.

Figure 4.5 illustrates the performance characteristics of stable diffusion running on a desktop-class GPU (RTX 3090, 24 GB). Initially, we utilized the out-of-the-box PyTorch-based stable diffusion implementation from Huggingface [192] and profiled it on an RTX 3090. Our findings, depicted in Figure 4.5, indicate that rendering even a moderately high-resolution image, such as an XGA resolution image containing only $1024 \times 786 = 786,432$ pixels, takes approximately 21 seconds on this high-end GPU. Furthermore, we observed that, without additional optimizations, this GPU encounters out-of-memory (OOM) errors when attempting to render images beyond XGA resolution.

To address the performance issues, we used Nvidia's TensorRT [193] library to optimize the stable diffusion process. TensorRT enhances performance by constructing an optimized engine that first profiles the stable diffusion application with a sequence of mock inputs to analyze its runtime behavior. This analysis aids in identifying opportunities for operator fusion, which involves merging multiple layers and operations of the diffusion model into

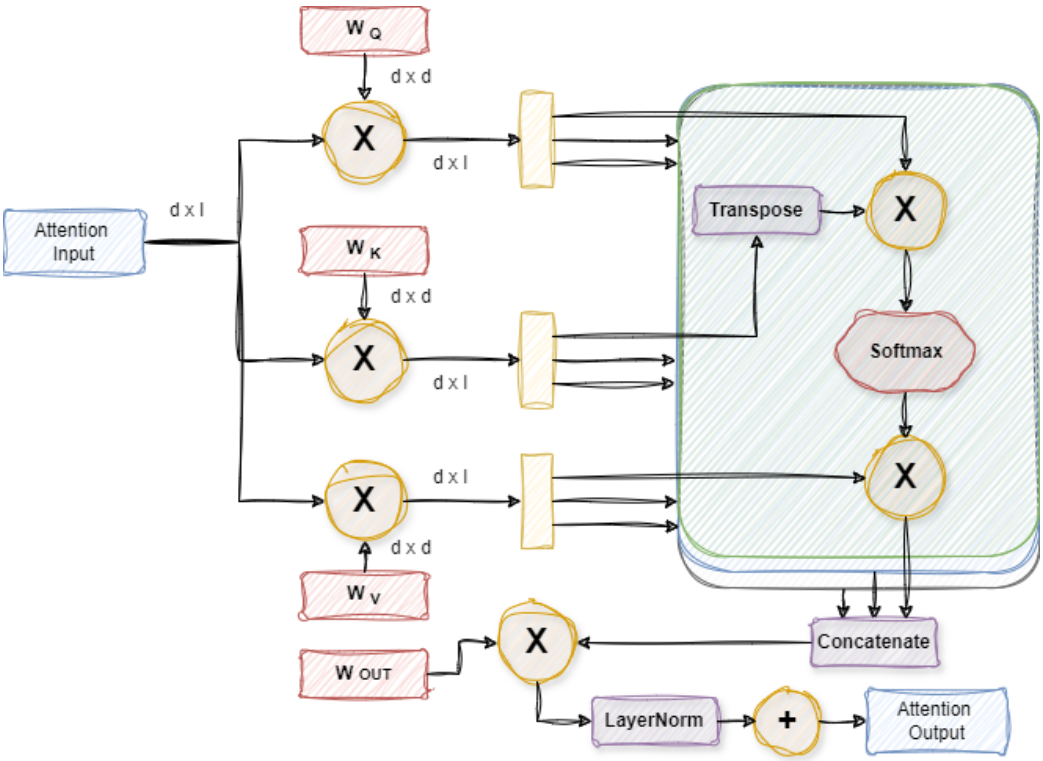


Figure 4.6: The dataflow of the attention mechanism.

new dataflow graphs with fused kernels. Using the optimized engine created by TensorRT, we observed that stable diffusion runs $\approx 3.6\times$ faster than the PyTorch based implementation. Nevertheless, it still requires multiple seconds to render a moderately high-resolution image; specifically, it takes $\approx 4.65 seconds$ to render approximately $786 Kpixels$, which corresponds to XGA resolution.

Our profiling further revealed that even with the optimized TensorRT implementation, the GPU still encounters out-of-memory (OOM) issues when processing images beyond XGA resolution using the out-of-the-box stable diffusion model without further algorithmic enhancements. This limitation underscores the necessity for additional optimizations to handle higher resolutions effectively.

4.3.1 Attention Mechanism

We were able to quickly isolate the out of memory error to the attention mechanism, which is the backbone of the Transformer block of the latent

diffusion model. The dataflow of the attention mechanism is shown in Figure 4.6. The attention mechanism computes the matrix product of queries and keys, applies a softmax function to obtain the weights, and then multiplies the resulting matrix by the values. This process results in an output matrix that combines the input features in a data-dependent way. However, a major challenge with vanilla attention arises when scaling to larger input sizes or matrix dimensions. The memory and computational requirements grow quadratically with the size of the input, primarily due to the necessity to store and manipulate large intermediate matrices QK^T and softmax(QK^T). These quadratic complexities impose significant limitations, especially in applications requiring high-resolution feature maps or large input sequences. This is the reason that the RTX3090 runs into OOM error when the desired image resolution is increased beyond XGA resolution.

FlashAttention [194] addresses these scalability issues through tiling [195], which splits the input matrices into smaller blocks that fit within the faster on-chip SRAM, avoiding frequent access to the slower off- chip HBM (High Bandwidth Memory). Additionally, FlashAttention employs an online softmax optimization where the softmax computation is decomposed and processed incrementally across these blocks. This approach ensures that the softmax calculations are performed without the need to instantiate the entire attention matrix in memory at once, thereby preserving computational resources while maintaining the exactness of the attention outputs.

In order to understand the benefits of FlashAttention in the context of diffusion models, we profiled the stable diffusion algorithm where the conventional vanilla attention mechanism was substituted with FlashAttention. Our profiling spanned various hardware platforms, including a desktop-class CPU (Intel i9-10920X), a server-class CPU (Intel Xeon Gold 6430L), and a high-end GPU (RTX3090). The results, as illustrated in Figures 4.7 and 4.8, quantify the performance of stable diffusion on these different platforms across a spectrum of rendering resolutions from VGA to 8K.

On the desktop-class CPU, the i9-10920X took approximately 58 *seconds* to render an image at VGA resolution and significantly longer, about 90610 seconds (which is roughly 1510 minutes or 25 hours), to render an $8K$ resolution image. Shifting focus to a more powerful server-class CPU, the Intel Xeon Gold 6430L demonstrated an average performance improvement of 6.27× over the i9-10920X. It rendered VGA resolution in ≈11 seconds

and $8K$ resolution in approximately 12783 seconds, translating to about 213 minutes or 3.55 hours, as presented in Figure 4.7.

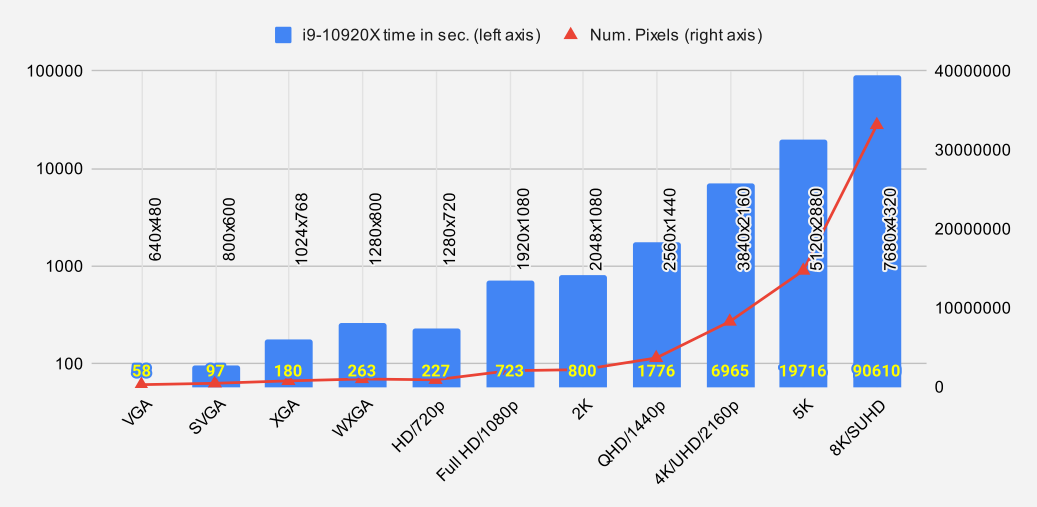
Profiling on the RTX3090 revealed an even more pronounced performance advantage, averaging around $\approx 9\times$ higher than that of the Intel Xeon Gold 6430L. Rendering times on this GPU were approximately one second for VGA resolution and 950 seconds (or about 16 minutes) for 8K resolution images, detailed in Figure 4.8.

Overall, we see that in spite of our best efforts at optimizing the performance of a latent diffusion model through software optimizations (through tensorRT and FlashAttention), rendering performance continues to be low, especially at high resolutions. This motivates the need for hardware support.

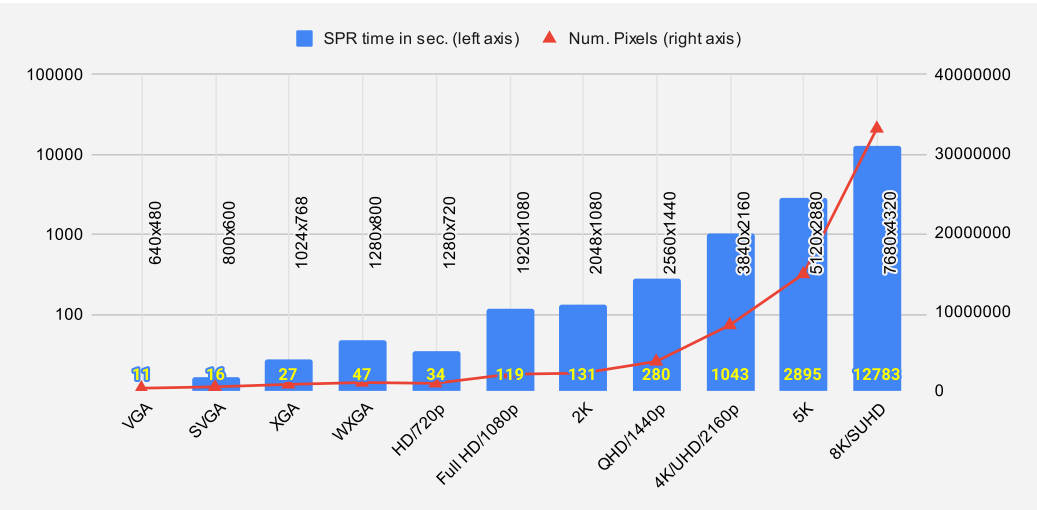
4.4 Performance Diagnosis of Latent Diffusion Models

To better understand where the compute resources are actually spent (and, therefore, what leads to such high rendering times), we performed a model level breakdown of the stable diffusion pipeline. Figure 4.9 presents the proportion of compute cycles allocated to each phase within the stable diffusion rendering pipeline, as described in Section 4.2. Our findings indicate that a substantial portion of the compute cycles, approximately 89.3%, are dedicated to the denoising process via the UNet-based latent diffusion model for VGA resolution. Conversely, only 7.9% and 2.8% of compute cycles are attributed to the CLIP and VAE stages, respectively. As the rendering resolution increases, the computational load shifts more toward the latent diffusion model. At $8K$ resolution, a staggering 99.7% of compute cycles are consumed by the UNet- based latent diffusion model, with a mere 0.3% utilized by CLIP and VAE combined. This analysis confirms that the UNet-based latent diffusion model is the primary performance bottleneck in the stable diffusion rendering pipeline.

We then perform breakdown analysis of the latent diffusion model (Table 4.1) at the granularity of PyTorch blocks and find that the UNet backbone has a total of 22 ResNet and 16 transformer blocks. While there are more Resnet blocks than Transformer blocks, our analysis shows that the Transformer blocks consume $\approx$ 57.8% of compute cycles and the ResNet blocks consume $\approx$ 35.6% of compute cycles for VGA resolution. All other



(a) Intel i9-10920X



(b) Intel Gold 6430L

Figure 4.7: Profiling of FlashAttention based Stable diffusion running on a desktop-class CPU (Intel i9-10920X) and a server-class CPU (Intel Xeon Gold 6430L). The x-axis has different resolution levels. The bars show time spent in seconds and the line shows the number of pixels rendered.

blocks collectively consume only $\approx 6.6\%$ of compute cycles.

To further identify the primitive kernels of the latent diffusion model that may be responsible for its significant time overhead and its poor scaling with image resolution, we conducted a CUDA kernel-level breakdown analysis using Nvidia's Nsight Compute [196]. We used CUDA compiler version 11.7 for compiling the PyTorch based latent diffusion model implementation to CUDA kernels which then run on the GPU. The operational breakdown of

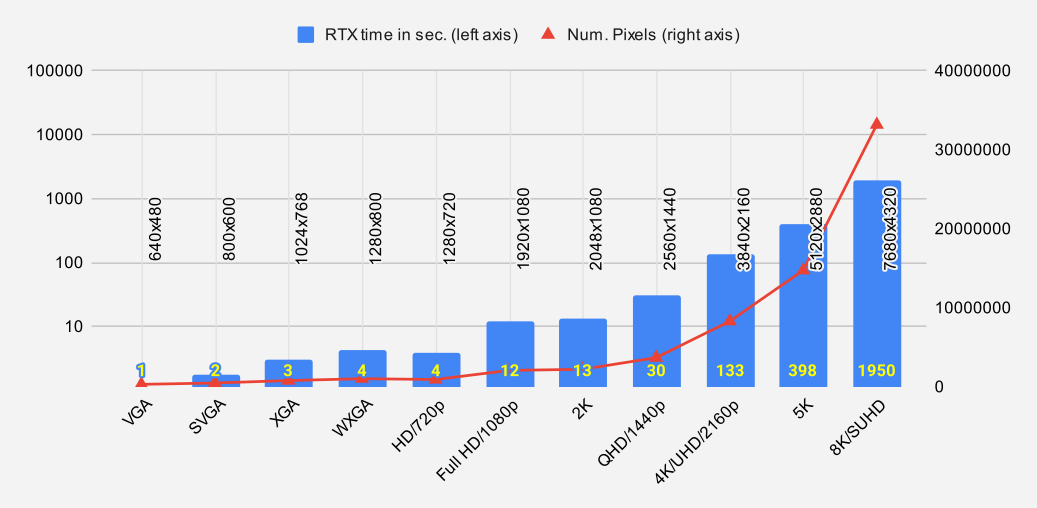


Figure 4.8: Profiling of FlashAttention based Stable diffusion running on a desktop-class GPU (RTX3090, 24GB). The x-axis has different resolution levels. The bars show time spent in seconds and the line shows the number of pixels rendered.

Table 4.2: The number of ResNet and Transformer blocks in latent diffusion model and the percentage time spent on those blocks for VGA resolution.

Blocks	Time spent (%)	Num blocks
ResNet	35.6	22
Transformer	57.8	16
Rest	6.6	

diffusion models for various rendering resolutions is shown in Figure 4.10. Our breakdown analysis reveals that the five most computationally intensive kernels are: 1) the Attention mechanism, 2) Element-wise operations like normalization and exponentiation, 3) Explicit convolution kernels, 4) Fused GEMM operations which integrate various layers of the diffusion models (including implicit convolution and matrix multiplication for dense layers combined with activation functions), and 5) Explicit GEMM kernels associated with fully connected layers. The breakdown can be understood by noting that the compiler optimizes the PyTorch layers by mapping the some of the convolution and feed- forwad layers to explicit or fused GEMM kernels. The attention kernels in the CUDA kernel level breakdown analysis belong to the Transformer blocks and the explicit convolution kernels belong to the ResNet blocks, whereas the GEMM and the fused GEMM kernels can belong either to the ResNet or the feed forward layers of the Transformer block.

We also see that the attention mechanism's share of compute cycles grows super-linearly with increasing resolution. At VGA resolution, the attention mechanism accounts for approximately 15.2% of the compute cycles. In contrast, for higher resolutions, such as 8K, the attention mechanism alone consumes 96.7% of the compute cycles, leaving a mere 3.3% for all other kernels combined. This identifies the attention mechanism to be the principal contributor to the super-linear scaling of rendering times observed in latent diffusion model.

4.4.1 Quantization and Sparsity in Latent Diffusion Models

Sparsity and quantization are extensively explored within the realm of neural networks, including in transformers, to enhance computational efficiency. Notably, transformers like BERT and GPT-3, while typically operating on dense matrix calculations, do not inherently exploit sparsity. Sparse Transformers [197] introduce sparse attention mechanisms, significantly reducing computational complexity by limiting attended positions, thereby allowing transformers to manage longer sequences more efficiently [197]. Beyond designing inherently sparse architectures, techniques like quantization also induce sparsity by reducing the precision of weights and activations. This not only shrinks model size and accelerates inference but also enhances energy

efficiency by pruning near-zero values, thus reducing the memory footprint.

In this context, our research quantifies the application of sparsity within latent diffusion models, where sparsity provides a substantial opportunity to reduce computational demands. We analyzed both static and dynamic sparsity in these models. Static sparsity was assessed by examining the trainable parameters in various layers, while dynamic sparsity was evaluated through PyTorch hooks [198] that analyze activations during the model's forward pass.

Our findings reveal that latent diffusion models, specifically in FP32 and FP16 variants from [192], exhibit very low natural static and dynamic sparsity levels ($\leq$ 1%). To introduce meaningful sparsity, we employed Q-Diffusion [199], a state-of-the-art approach for quantizing these models down to 4-bit precision without significant quality degradation. Starting with the FP32 checkpoint from [192], we obtained Int8 and Int4 quantization parameters for each tensor and applied these to analyze induced sparsity and special values in weights and activations.

Results from our quantization experiments show minimal natural sparsity in these models, necessitating induced sparsity through transformations. For instance, the 8-bit quantized model exhibited a static sparsity of 0.14% and special weight values of 1.60%, with dynamic sparsity and special activations at slightly higher levels. Fine-tuning on specific scenes like bedrooms and churches further influenced these metrics, highlighting the variability in sparsity based on training data and quantization depth.

Figure 4.11 presents the distribution of these sparsity metrics, and Table 4.3 summarizes our comprehensive sparsity analysis. The next section (Section 4.5) will detail the proposed hardware architecture designed to exploit these findings for enhanced energy efficiency in implementing latent diffusion models.

4.5 LDPU: A Scalable Architecture for Latent Diffusion Models

To accelerate latent diffusion model-based applications, we introduce a systolic array like architecture, termed the Latent Diffusion Processing Unit (LDPU). Based on the analysis discussed in Section 4.4, this architecture

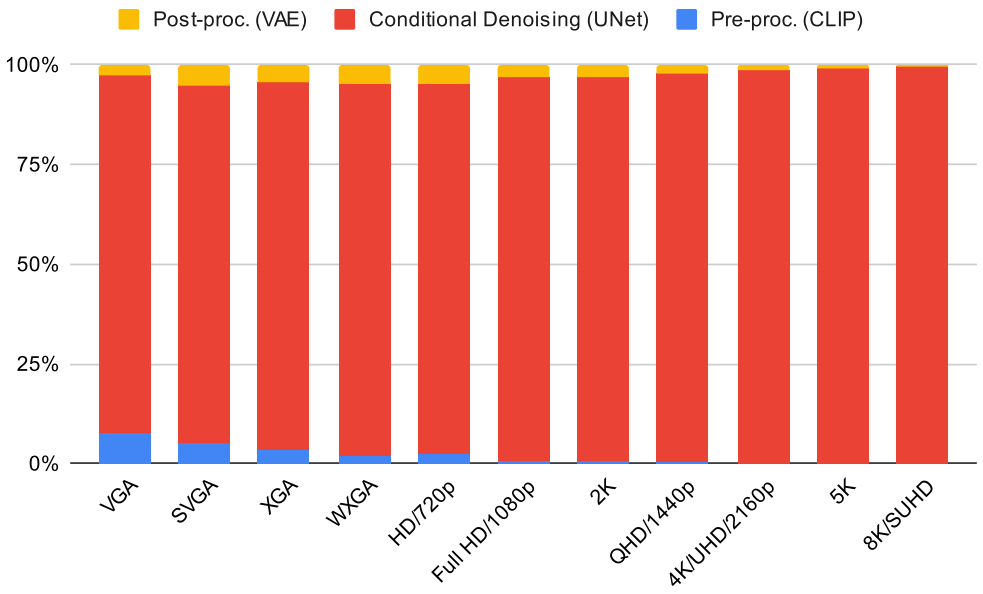


Figure 4.9: Breakdown of compute cycles spent on different stages of stable diffusion based rendering pipeline. On average 95% of the GPU compute cycles are spent on latent diffusion model (UNet backbone).

Table 4.3: The percentage of static sparsity, dynamic sparsity, special weight values and special activation values in quantized and fine-tuned latent diffusion models.

Model		w8a8	w4a8
Stable diffusion UNet	static sparsity	0.14	0.24
	special weights	1.60	24.46
	dynamic sparsity	1.22	1.12
	special activations	5.17	4.93
Fine-tuned bedroom scenes UNet	static sparsity	0.13	0.20
	special weights	1.71	25.50
	dynamic sparsity	23.01	22.73
	special activations	36.45	36.12
Fine-tuned church scenes UNet	static sparsity	0.15	0.30
	special weights	1.62	24.38
	dynamic sparsity	6.65	6.63
	special activations	16.23	16.14

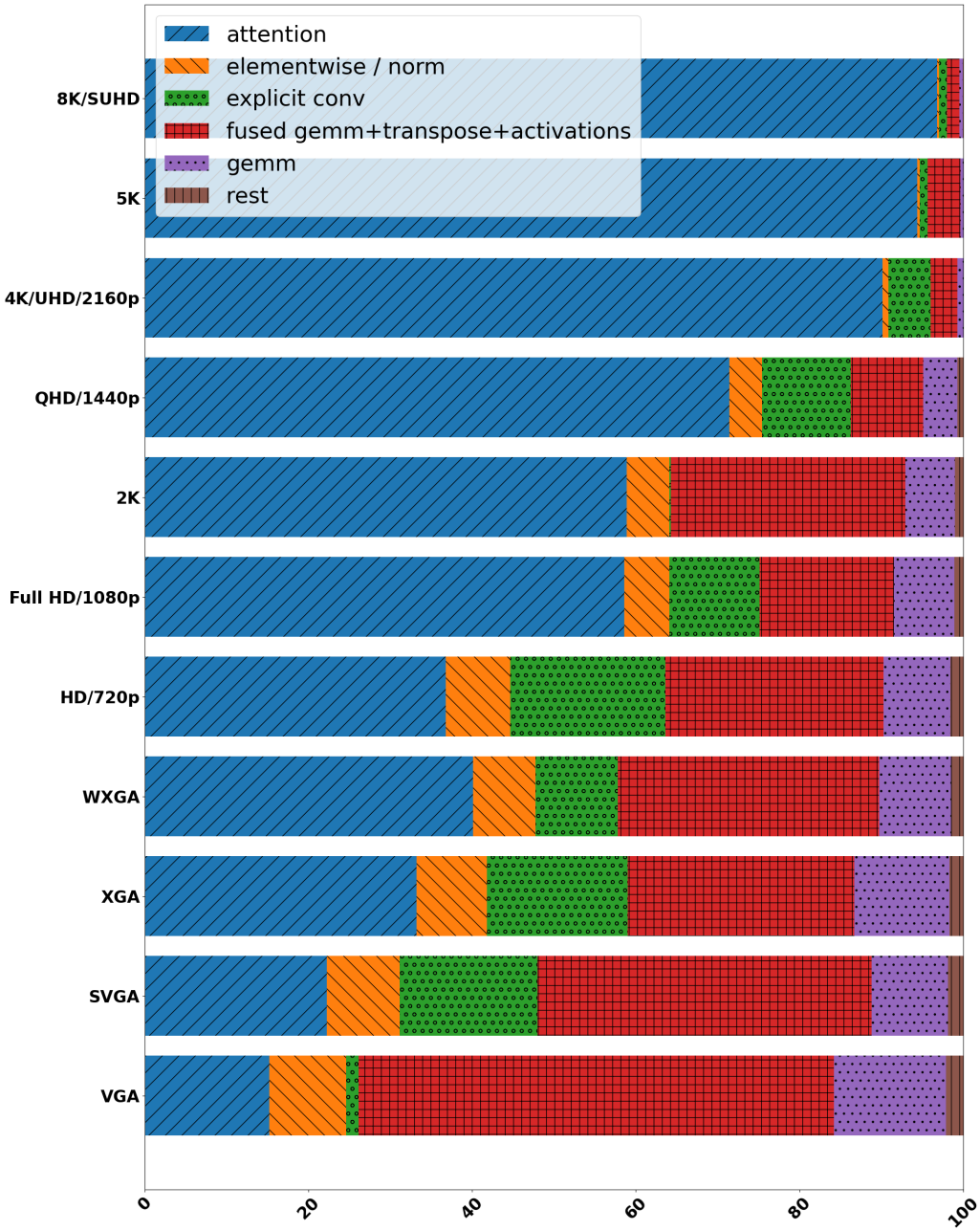
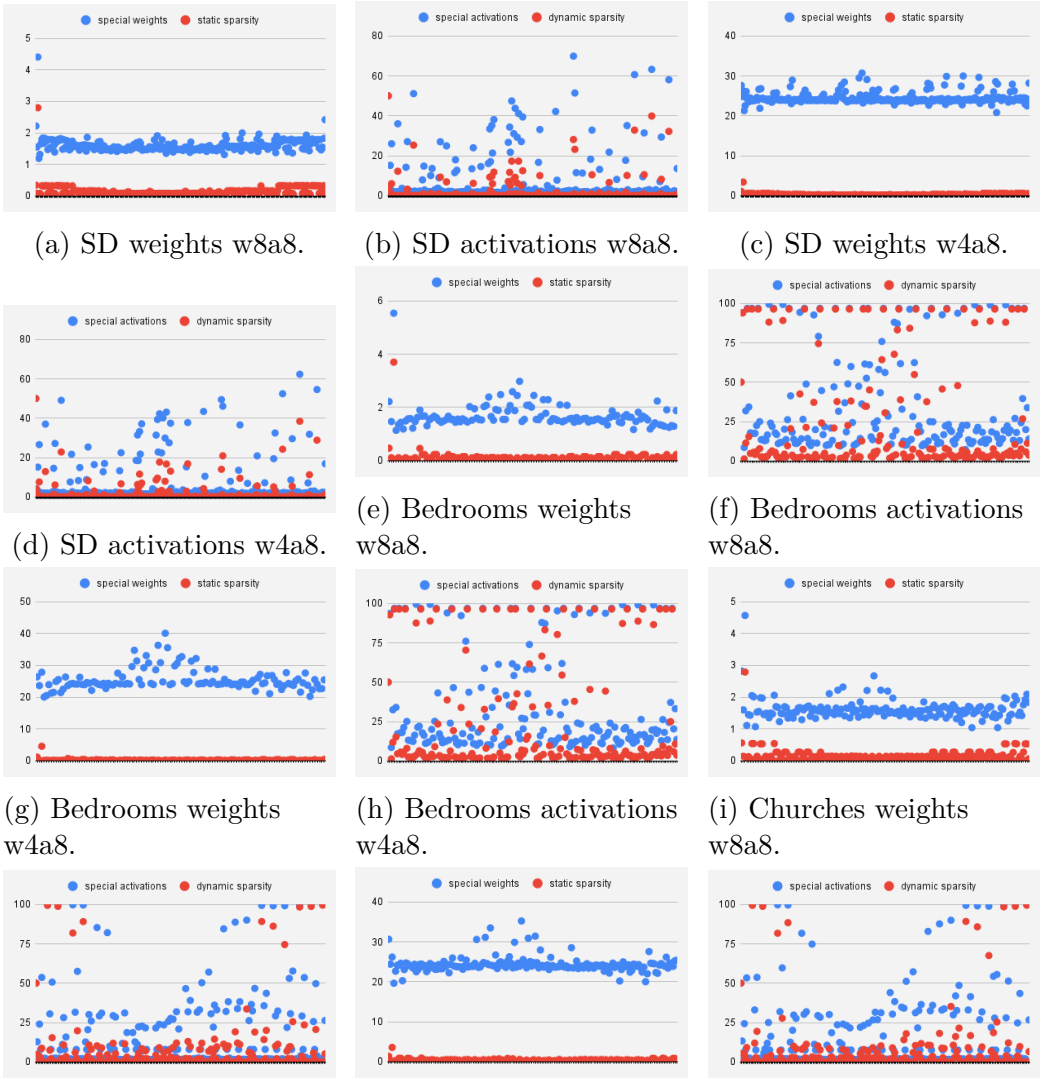


Figure 4.10: The operation level breakdown of the of UNet for different rendering resolutions. The five most expensive kernels in terms of number of cycles spent. Attention mechanism, element-wise operations such as normalization, exponent etc, explicit convolution kernels, fused gemm operations for NN layers (implicit convolution and MatMul for dense layers with activations), explicit gemm kernels.



(a) SD weights w8a8.

(b) SD activations w8a8.

(c) SD weights w4a8.

(d) SD activations w4a8.

(e) Bedrooms weights w8a8.

(f) Bedrooms activations w8a8.

(g) Bedrooms weights w4a8.

(h) Bedrooms activations w4a8.

(i) Churches weights w8a8.

(j) Churches activations w8a8.

(k) Churches weights w4a8.

(l) Churches activations w4a8.

Figure 4.11: The distribution of static sparsity, dynamic sparsity, special weight values, and special activation values as percentages across various layers of quantized and fine-tuned latent diffusion models at Int8 and Int4 quantization levels.

targets latent diffusion models by supporting efficient execution of Transformer and ResNet blocks, along with accommodating lower precisions (int8 and int4), sparsity, and special values.

Our analysis in Section 4.4 further suggests that the hardware should support attention mechanisms and elementwise operations such as exponent, sum, min, and max (for online softmax calculations). Additionally, it should

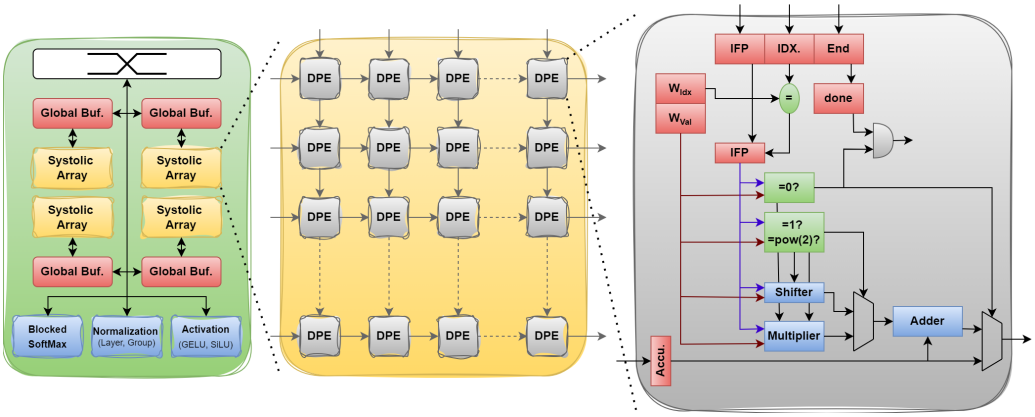


Figure 4.12: Architecture of the Latent Diffusion Processing Unit (LDPU)

handle neural network activations like GeLU and SiLU, normalization operations, and GEMM operations.

Figure 4.12 shows the block diagram of the LDPU. The architecture consists of a grid of systolic arrays and includes on-chip memory to feed data to these arrays. It also features a normalization block to support layer normalization and group normalization layers specific to latent diffusion models, an activations block with hardware support for activations such as GELU and SiLU, and a Softmax block designed to efficiently handle online-blocked-softmax operations to support the FlashAttention mechanism.

The number of systolic arrays in the grid can be adjusted based on power, performance, and area targets. Each systolic array consists of a grid of processing elements, which are designed from the ground up to support both static and dynamic sparsity, static special values (such as weights that are 0, 1, or powers of 2), and dynamic special values (such as special activation values).

We have developed both Int8 and Int4 versions of the LDPU hardware to perform design space exploration and to assess their area, power, and performance overheads. This flexible design allows scalability in the grid of systolic arrays, where the number of arrays can be varied to meet different performance, power, and area requirements.

4.5.1 Microarchitecture of Diffusion Processing Element (DPE)

Figure 4.12 illustrates the architecture of the Diffusion Processing Element (DPE) within the Latent Diffusion Processing Unit (LDPU). The DPE is engineered with an integer Multiply-Accumulate (MAC) unit at its core. In this work, we consider two data formats for the input data values — 8-bit integer (Int8) and 4-bit integer (Int4).

The Processing Element (PE) within the DPE includes registers for storing sparse matrix indices and values. A hardware comparator in the DPE checks the index of incoming vector elements against the stored matrix index. Another comparator checks if the matrix value in the register is zero. During computation initiation, each DPE retrieves a value, index entry from the incoming left stream and holds it in its registers.

Input vector elements are streamed from the top edge of the array to the DPEs, which also receive accumulated values from neighboring DPEs on the left. To minimize the critical path delay in matrix computations—particularly between the first and last vector element groups—a delayed accumulation approach is employed. This method allows overlapping streams of input element groups into the systolic array, as depicted in Figure 4.12.

In this delayed accumulation process, the DPE latches the vector element upon an index match but postpones the MAC computation until all elements in the vector group have been received. This ensures the functional correctness of the computation. To support this optimization, two additional registers are required: one for storing the vector element and another for a 1-bit signal that indicates the end of a vector group, ensuring precise timing for operations.

While this approach delays computations in some DPEs, it accelerates the overall streaming and processing of vector groups, thereby reducing total computational latency. Additionally, the DPE incorporates hardware logic to recognize special values such as 1 or powers of 2 (e.g., 2, 4, 8). When these values are detected, the DPE uses a shifter instead of the general-purpose multiplier, thus significantly reducing power consumption by gating the multiplier and adder units.

4.5.2 Modes of Operation of Diffusion Processing Element (DPE)

The operational modes of a DPE are determined by the incoming element's index, the matrix values, and the end signal of an element group. These modes — Hold, Latch, Special, Sparse, and Accumulate— are illustrated in Figure 4.14. Regardless of the DPE's mode, input vector elements are always passed downward to the next DPE.

1) Hold Mode: The PE selects matching vector elements from a streaming group. If there is no match and the index is non-zero, the PE retains the accumulated value (Accu) from the previous DPE. 2) Latch Mode: Upon encountering a matching element index that holds a non-zero and non-special value, the PE copies and stores the input value until an end signal is received. 3) Accumulate Mode: The PE begins MAC operations only after all elements have been streamed through it. Upon receiving a delayed end signal, it multiplies the stored value with the matrix value (W_val), adds it to the accumulated sum (Accu), and passes the result to the next DPE. 4) Sparse Mode: In scenarios where the matrix is sparse and contains zero values, the DPE conserves energy by bypassing the accumulator from one DPE to the next without engaging the compute elements. 5) Special Mode: If the DPE encounters an element with a special non-zero value that matches the index, it stores the input value until the end signal arrives. In this mode, the general-purpose multiplier is deactivated, and a shifter mechanism is engaged for multiplication, utilizing an appropriate right shift operation.

Figure 4.13-a shows the interaction of the LDPU with the GPU. The LDPU consisting of a cluster of N systolic arrays, on-chip SRAM and the non-linear blocks is connected to the shared L2 cache of the GPU. Figures 4.13-b and 4.13-c show the programming model of the LDPU. The programming model of the LDPU involves the GPU command buffer configuring the LDPU and scheduling the different layers of the latent diffusion model on the LDPU. The rest of the kernels, belonging to the CLIP and the VAE, are scheduled just as conventional CUDA kernels on the streaming multiprocessors of the GPU. The outputs of the LDPU are written back to the GPU memory and are read by the streaming multiprocessors, which then compute the rest of the kernels.

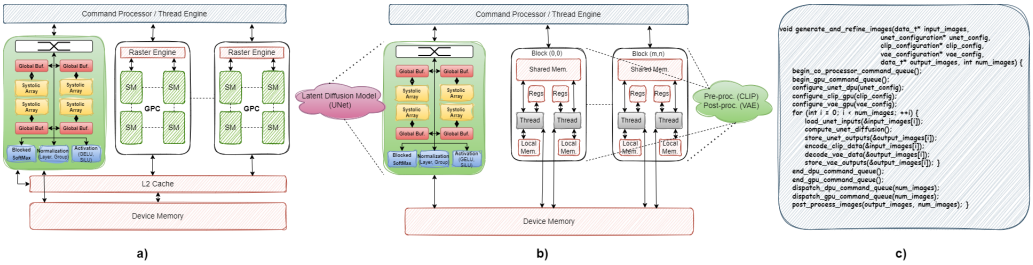


Figure 4.13: Interaction of the LDPU architecture with GPU. b) Programming model for the LDPU. c) An example pseudocode for programming LDPU.

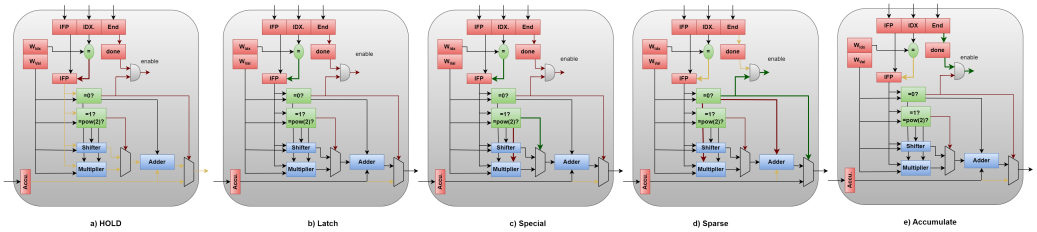


Figure 4.14: Different modes of operation supported by diffusion processing element (DPE).

4.6 Evaluation and Results

We developed an analytical model to estimate the performance of the latent diffusion model on our architecture. The block diagram of the analytical model is shown in the Figure 4.15. The inputs to the model are 1) A dictionary containing the detailed parameters of the latent diffusion model, 2) the dataflow graph of the latent diffusion model, 3) the number of systolic arrays in the LDPU grid, 4) the number of DPEs in the systolic arrays 5) the on-chip memory parameters, 6) the memory access time for DRAM, and 7) the operating frequency of the LDPU. 8) the memory access time and access energy for different on-chip SRAM blocks and 9) the synthesized area and power of different components of LDPU. 10) The critical path delay of different components of LDPU. The outputs of the analytical model are 1) The overall performance of the architecture with the latent diffusion model mapped to the LDPU architecture. 2) The overall area and power overhead of the LDPU for different architectural configurations. 3) The overall energy efficiency of the architecture.

To validate our custom performance model, we modeled a TPU [200] [201] [202] like systolic array, containing 128×128 PEs, using popular DNN modeling

frameworks Timeloop and Accelergy [6] [160]. We also model the same systolic array with our custom analytical model and compared the performance in terms of number of cycles consumed for different matrix sizes. Our results, presented in Figure 4.16 show that the analytical model is slightly on the conservative side, with an average percentage error of $\approx 3.9\%$.

For estimating the area and power overheads, we wrote RTL for different components of the LDPU architecture and synthesized it using Synopsys design compiler [203] along with the Nangate 45nm open cell library. We used CACTI to get the area and power estimates for the on-chip SRAM blocks. In order to obtain power estimates for various operating modes of the DPE, we initially generated timing information (VCD files) through simulations. These simulations of the different operating modes of the DPE were conducted using ModelSim [204]. Each mode was initialized using a SystemVerilog- based test bench. Table 4.4 presents the synthesized area of the DPE and the power estimates for various operating modes the DPE. We scaled the area and power numbers to 7nm technology node using often used scaling formulas [2].

We evaluated the performance and energy efficiency of the latent diffusion model on the LDPU architecture across various scaling factors. The architecture provides three scaling parameters: the number of systolic arrays, the dimensions of the DPE grid in each systolic array, and the precision of the computational units.

1) We explored different configurations by varying the number of systolic arrays within the LDPU architecture, specifically analyzing designs with 1, 2, and 4 arrays. 2) We also varied the DPE grid dimensions of each systolic array, conducting design space explorations for 32×32, 64×64, 128×128, and 256×256 grid dimensions. Our findings indicate that scaling the DPE grid dimension beyond 256×256 does not yield additional performance or energy efficiency gains due to decreased systolic array utilization from memory bandwidth constraints.

Using the state-of-the-art FlashAttention-based GPU implementation as a baseline, we compared the performance and energy efficiency of the LDPU architecture. Figures 4.17 and 4.18 present the performance improvements and energy efficiencies for the Int8 and Int4 versions of the LDPU architecture across different hardware scaling factors.

For the Int8 version, the architecture with four systolic arrays and 64×64

DPEs per systolic array achieved a performance improvement of approximately 1.32× and an energy efficiency of approximately 108×. The largest architecture configuration, with four systolic arrays and 256×256 DPEs per systolic array, showed a performance improvement of approximately 16×, although the energy efficiency benefits dropped to approximately 86× due to lower hardware utilization as bandwidth requirements increased.

In contrast, the Int4 version demonstrated that even with the largest architecture configuration, the performance improvement scaled to approximately 25.43×, without a drop in energy efficiency, which remained significantly higher due to the lower bandwidth requirements of the Int4 architecture.

Our sensitivity studies further indicated that the energy efficiency benefits increased by approximately 28% for models fine-tuned on bedroom scenes and by approximately 9% for those fine-tuned on church building scenes. These improvements are attributed to increased sparsity and specialized values in the fine-tuned models compared to the general-purpose model.

Figure 4.19 illustrates the area and power overheads for different scaling factors of the Int8 and Int4 variants. The Int8 variant with four systolic arrays and 256×256 DPEs had a maximum area and power overhead of approximately 21.71% and 19.68%, respectively, compared to GPUs. Meanwhile, the Int4 variant with the same configuration had significantly lower overheads of approximately 4.19% for area and 4.44% for power.

To provide context to these results, Figure 4.20 shows the energy and environmental impact of using LDPU (the Int4 variant,) vs a GPU, comparing the energy consumption and CO2 emissions to U.S. households and cars. Beyond reinforcing the high energy footprint of diffusion models, our analysis estimates that the impact of operating at 8K resolution with the LDPU is equivalent to that of operating at XGA resolution without the LDPU.

In order to estimate the energy efficiency trends for the latest GPU models, although we did not have access to the A100 and H100 GPUs, we referred to published literature to estimate the energy efficiency of our proposed architecture in comparison with these advanced GPUs. For our empirical measurements, we utilized the RTX3090. The RTX3090, with a TDP of 350 watts, achieves 0.81 operations per watt [161]. In contrast, the A100 and H100 GPUs, with TDPs of 400 watts and 700 watts respectively, achieve 2.5 [162] and 5.71 [163] TOPS per watt. This makes the A100 approximately 3.09 times more energy efficient and the H100 about 7.05 times more effi-

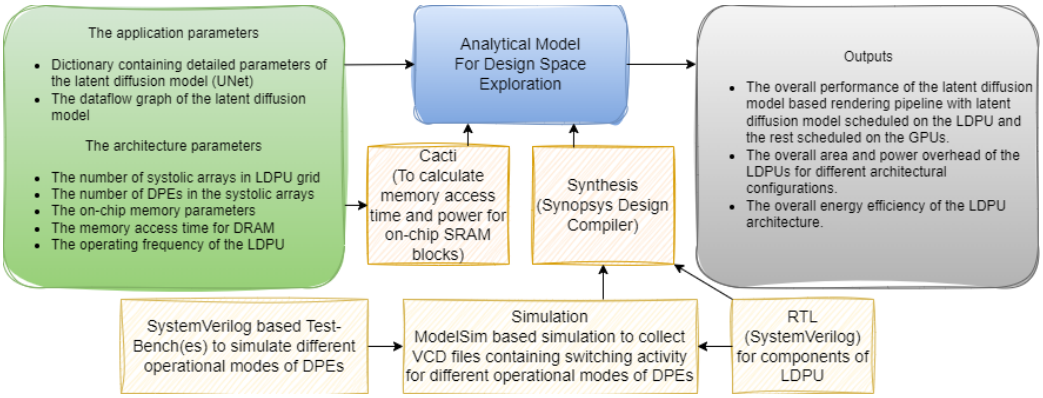


Figure 4.15: The block diagram of the evaluation methodology.

Table 4.4: The synthesized area of the DPE and the power estimates for various operating modes the DPE. We scaled the area and power numbers to 7nm technology node using often used scaling formulas [2].

		Power (mW) Int8	Area (um2) Int8	Power (mW) Int4	Area (um2) Int4
LDPU DPE	Idle	0.0053	130.04	0.001361	25.15280464
	Hold	0.2560		0.065473	
	Latch	0.2560		0.065140	
	Special	0.1829		0.046529	
	Sparse	0.2560		0.065810	
	Accu.	0.2720		0.069565	

cient compared to the RTX3090. Further applying these efficiency metrics to the Latent Diffusion Model Processing Unit (LDPU) architecture, which is approximately 108 times more energy efficient than the RTX3090, translates to the LDPU being about $\frac{108}{3.09} \approx 34.95$ times more energy efficient than the A100 and $\frac{108}{7.05} \approx 15.31$ times more than the H100. Despite the enhanced capabilities of the newer GPU models (A100, H100, etc.), the general trends observed with the RTX3090 provide valuable insights into the performance and efficiency of Latent Diffusion Models (LDMs) across various hardware platforms.

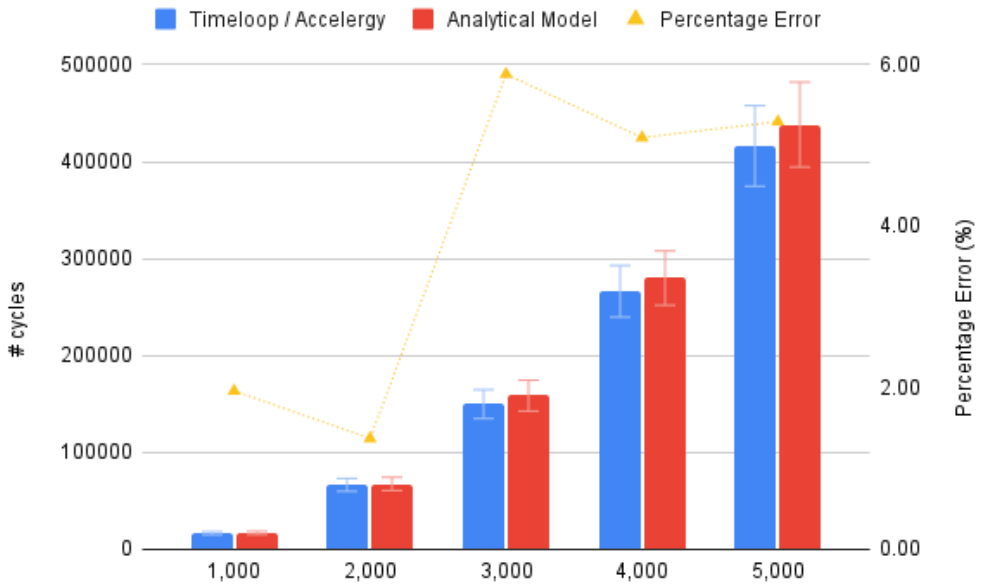
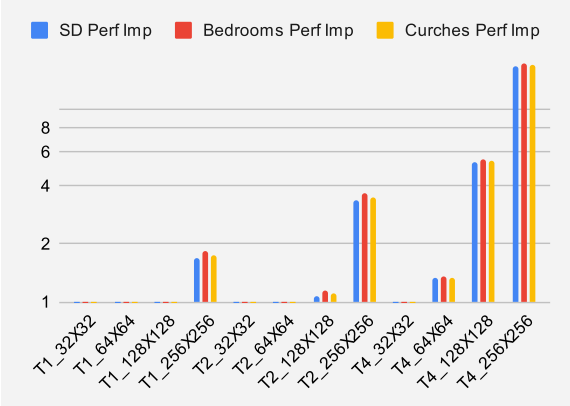


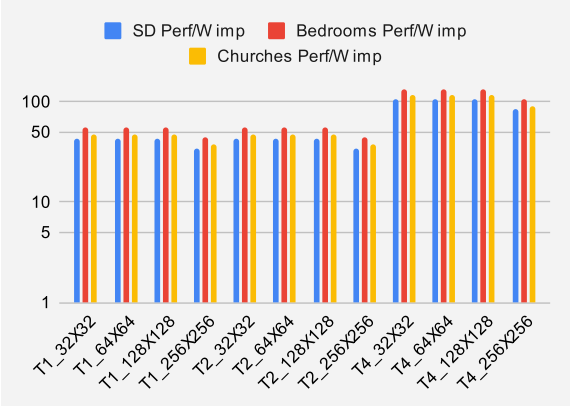
Figure 4.16: Comparison of number of cycles estimated for systolic array modeled using Timeloop/Accelergy [6] and our custom analytical model.

4.7 Summary and Discussion

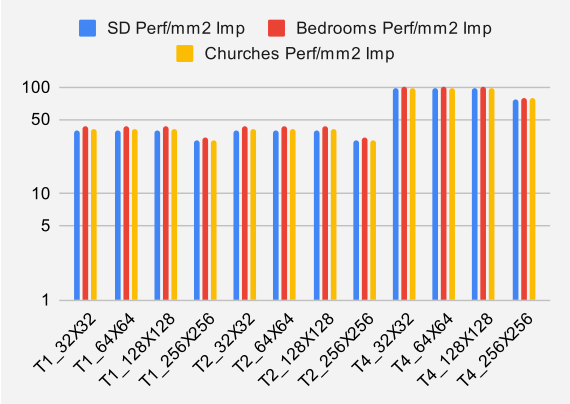
In this chapter, we argued that since the energy requirements are extraordinary for diffusion models even for optimized software implementations, hardware accelerators are needed for these models to optimize energy efficiency. To design a hardware accelerator for diffusion models, we studied diffusion model performance and scalability in depth, focusing on latent diffusion models for graphics applications, and showed that the generation cost of latent diffusion models escalates superlinearly with the desired resolution. We quantified the cost scaling on modern hardware and pinpointed that the superlinear escalation of generation cost is due to the underlying attention mechanism. We showed that latent diffusion models inherently exhibit a low percentage of sparsity and special values, which include zeros, ones, or powers of two. However, the incidence of these values can be increased significantly through techniques such as quantization and model fine-tuning. We proposed Latent Diffusion Model Processing Unit (LDPU), a scalable architecture designed from the ground up to enhance energy efficiency for laent diffusion models, featuring hardware support for sparsity and special values. We quantified the benefits of our hardware architecture and showed



(a) Performance improvement (Int8 configuration)

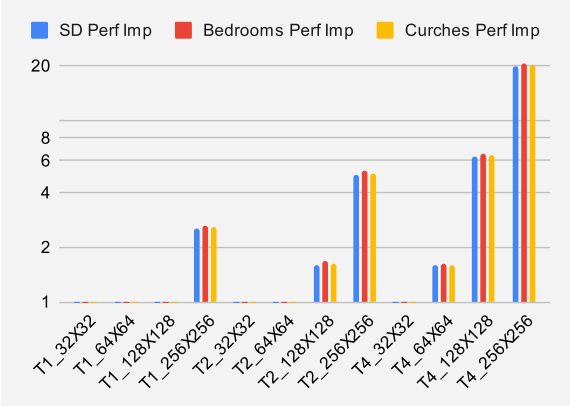


(b) Energy efficiency (Int8 configuration)

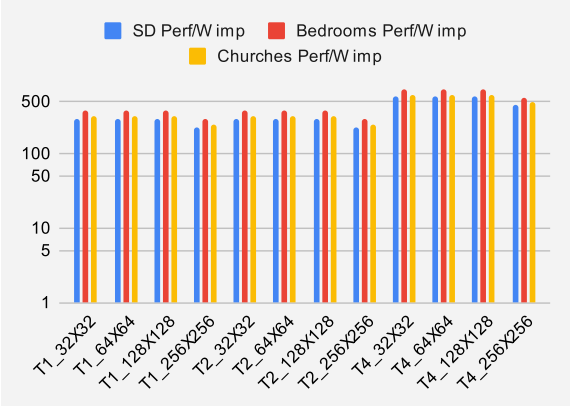


(c) Performance improvement / mm2 (Int8 configuration)

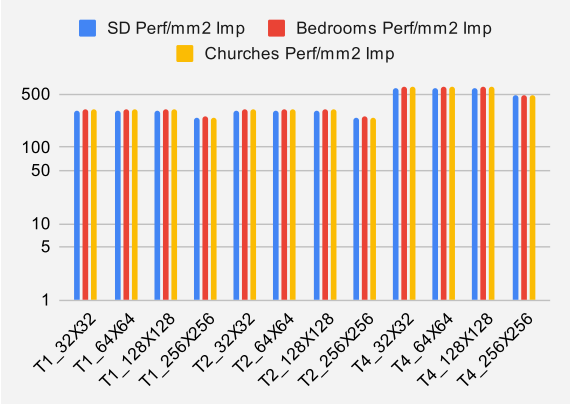
Figure 4.17: The estimated performance improvement, energy efficiency and performance improvement / mm2 for Int8 variants of LDPU architecture, for different scaling factors.



(a) Performance improvement (Int4 configuration)

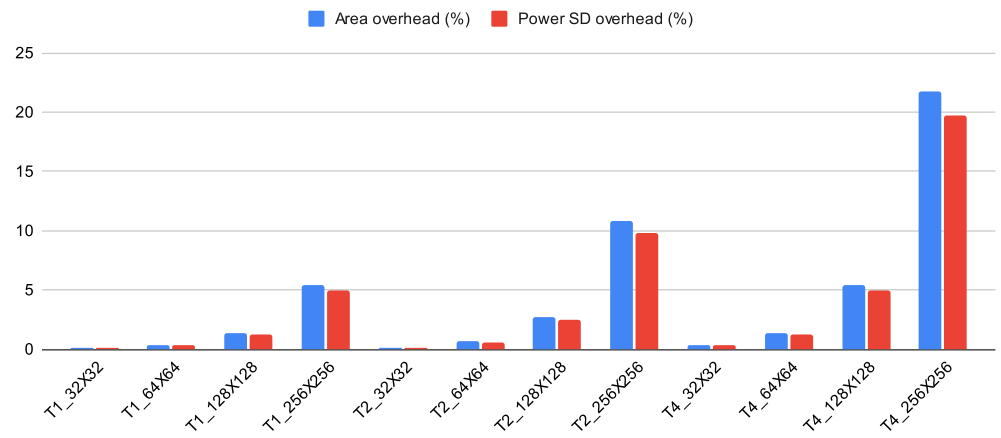


(b) Energy efficiency (Int8 configuration)

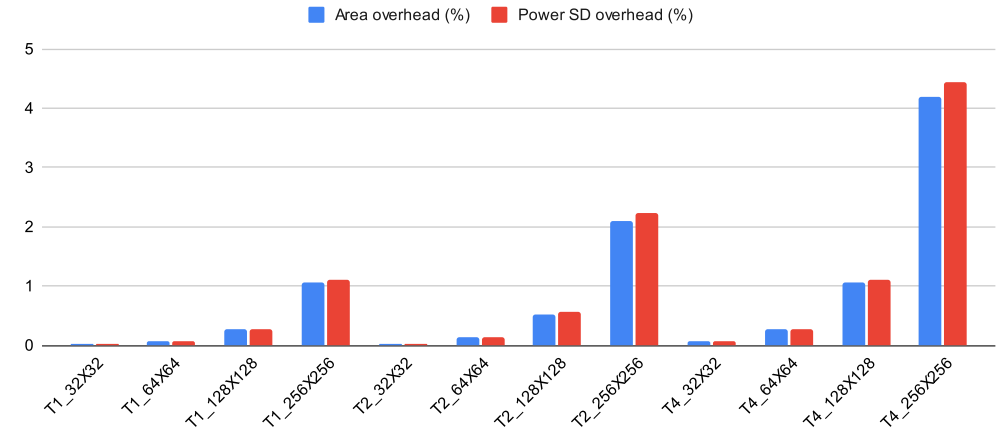


(c) Performance improvement / mm2 (Int4 configuration)

Figure 4.18: The estimated performance improvement, energy efficiency and performance improvement / mm2 for Int4 variants of LDPU architecture, for different scaling factors.



(a) LDPU Int8 variant



(b) LDPU Int4 variant

Figure 4.19: Area and Power overhead of different variants of the LDPU architecture for different scaling factors.

significant performance and energy efficiency benefits against GPU baseline. The Int8 version of the LDPU architecture achieves an energy efficiency of 108× over a GPU baseline. For the Int4 variant, the energy efficiency improvements increase to 572×. To this best of our knowledge, this is the first work on hardware acceleration of latent diffusion models.

In Chapter 2, we discussed the potential of generative neural graphics and its applications across various computing domains. In Section 4.4, we analyzed the latency and energy demands of these applications on modern hardware (GPUs). These applications demand high computational power and energy efficiency, particularly at high resolutions. To address these challenges, we proposed the Latent Diffusion Model Processing Unit (LDPU),

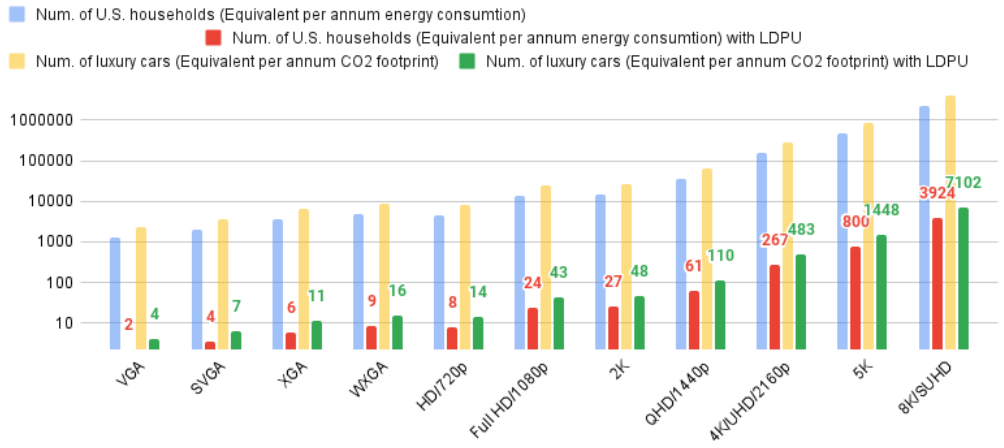


Figure 4.20: The estimated annual energy consumption and corresponding CO2 emissions from generating images using diffusion models at various resolutions, both with and without LDPU, expressed in terms of the equivalent number of U.S. households' energy consumption per year and the equivalent annual CO2 footprint of a certain number of advanced cars.

designed to minimize energy consumption. Our evaluation demonstrated a remarkable increase in energy efficiency by up to 572× compared to the traditional GPU-only baseline, significantly reducing the energy burden for high-resolution image and video generation. For instance, the energy demands of text-to- image generation and video super-resolution, which depending upon the use-case may not require real-time or near-real-time processing, were substantially reduced, making these applications approximately three orders of magnitude more energy-efficient. Despite these advancements, the latency requirements for applications like interactive gaming and real-time video generation still need further optimization, underscoring the necessity for ongoing research in hardware acceleration and model optimization.

CHAPTER 5

IMPROVING NEURAL RENDERING LATENCY WITH SUPER-RESOLUTION

5.1 Introduction

Neural rendering is a computationally intensive process. For instance, rendering a single high-resolution 8K frame on a desktop class GPU (RTX3090 with 24GB of memory) takes 34 seconds for Neural Representations (NRs) and 1950 seconds for Latent Diffusion Models (LDMs). Given these high rendering times on high-end client devices, one might consider cloud-based solutions, where computationally complex scenes are rendered on the cloud and streamed to heterogeneous client devices. However, this approach incurs significant data movement overhead between the cloud and the end-user [205].

In conventional computer graphics, techniques such as foveated rendering—rendering only the foveal regions of vision at high resolution while peripheral regions are rendered at lower resolution—have been employed to reduce both rendering time and data movement overhead [206–208]. These techniques, however, are generally only applicable to head-mounted devices, where the focus of eye is on a specific region only.

This chapter focuses on super-resolution, a technique that recovers high-resolution (a.k.a., target-resolution) frames from corresponding low-resolution frames. Since rendering low-resolution frames is relatively less computationally demanding than their high-resolution counterparts, the overall rendering overhead (including upscaling) is typically much lower [209]. In the context of cloud-based rendering, this approach not only reduces rendering times but also significantly cuts down the data movement overhead as frames can be rendered and transmitted at low resolution and then upscaled at the target device. Alternatively, super-resolution can be used at the server itself to improve server density. Figure 5.1 illustrates the application of super-resolution in various system settings.

For example, in client-based rendering scenarios (Figure 5.1), a super-resolution module could be placed between the neural renderer and the post-processing module. The neural renderer produces low-resolution frames, and the super-resolution module upscales them to high resolution. In cloud-based scenarios (Figure 5.1), super-resolution reduces network bandwidth requirements since frames are rendered and transmitted at low resolution and upscaled at the client side. This combined approach of low-resolution rendering and super-resolution not only enhances server density but also reduces overall computational costs.

Recent advances in super-resolution have largely leveraged neural-network-based methods, yielding state-of-the-art image quality [210–213]. Technologies such as Nvidia's DLSS [214], Facebook's NSRR [215], and AMD's FSR [216] focus on improving the super-resolution latency and quality. While state-of-the-art, such as DLSS, rely on proprietary, closed-source technologies, AMD's open-source framework utilizes compute shaders of GPUs, without employing neural-network techniques, which results in relatively lower visual quality / performance compared to its counterparts [214, 215].

In this chapter, we explore the performance implications of open-source, state-of-the-art super-resolution frameworks in the context of neural rendering. We observed that many high-performance super-resolution algorithms operate predominantly in low-resolution space, utilizing all network layers except the output for compute efficiency. We also observed that a typical super-resolution network also incorporates a convolutional long short-term memory (ConvLSTM) module for temporal stability [217].

Combining super-resolution with neural representations and latent diffusion models has shown promising results. Our analysis suggests that integrating super-resolution improves the rendering latency of neural representations by approximately 1.91× across various resolution levels. The benefits are even more pronounced for latent diffusion models, with a latency improvement of approximately 3.55×.

Later sections will delve deeper into the state-of-the-art in super-resolution and discuss insights from combining it with neural representations and latent diffusion models.

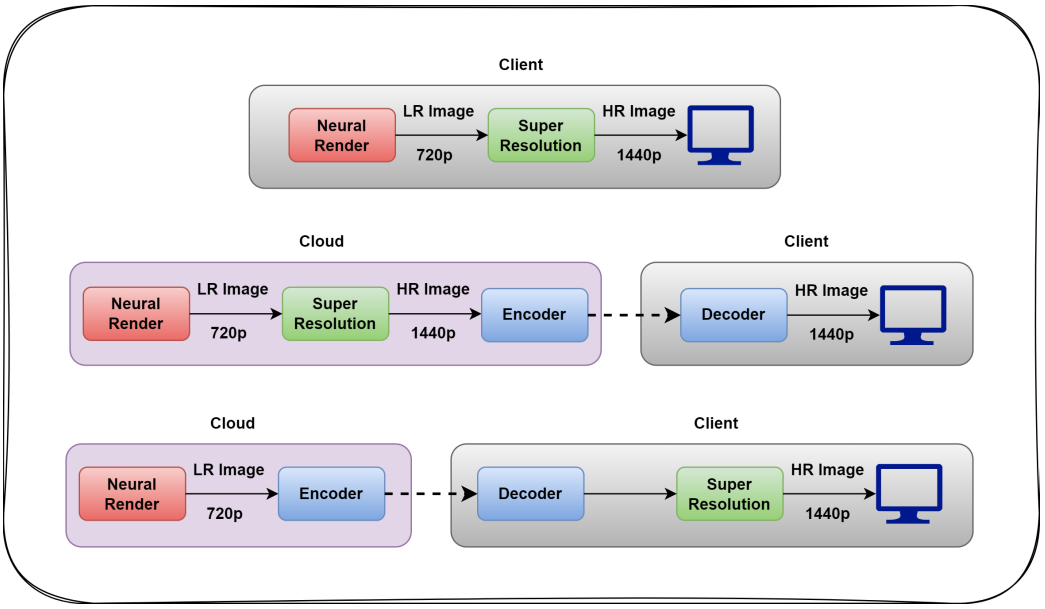


Figure 5.1: In client-only scenario super resolution improves quality, frame-rate, and reduces frame latency. In cloud-SR-to-client scenario SR improves server density as more clients can be addressed. In cloud-to-client-SR super-resolution reduces strain on bandwidth requirements and also improves the server density.

5.2 Motivation

Rendering using neural representations (NRs) and latent diffusion models (LDMs) is computationally intensive. Our analysis shows that rendering a single high-resolution frame, such as an 8K resolution, requires significant time even on advanced hardware like the RTX3090 with 24GB of memory (34 seconds for NeRF and 1950 seconds for LDM). We also performed sensitivity studies to understand the performance scaling trends of the neural representations and the latent diffusion models with the desired image resolution. The number of pixels in the horizontal and the vertical dimensions corresponding to the selected resolution levels are presented in table 5.1. The computational demands increase dramatically with image resolution, as we scale the resolution from VGA to 8K.

Figure 5.2 illustrates the linear relationship between the number of pixels and the rendering time for NeRF, attributed to the linear increase in NeRF queries per pixel. Each pixel in NeRF requires independent input encoding and a separate MLP query, resulting in a direct correlation between the number of pixels and rendering time.

Conversely, rendering times for LDM exhibit a super-linear increase with resolution. Our sensitivity analysis, illustrated in Figures 5.3, 5.4, and 5.5, presents the normalized rendering times across different hardware platforms: Intel i9-10920X, Intel Xeon Gold 6430L, and RTX 3090 (24GB). The rendering time and the number of pixels are normalized with respect to VGA resolution. Our breakdown analysis suggests that the super-linear scaling is due to the attention mechanism of the transformer blocks within the UNet architecture of LDMs 5.2, refer to section 4.4 of chapter 4 for further details. For instance, for LDMs, the rendering time approximately multiplies by 10× when the resolution is escalated from 720p to 1440p, doubling the number of pixels.

In traditional computer graphics, super-resolution algorithms are extensively used to manage the computational overhead of high-resolution rendering. Rendering at a lower resolution and then upscaling can significantly reduce the computational load compared to native high-resolution rendering [218]. This approach is especially beneficial in cloud-based rendering systems, where round-trip latency is also a critical factor for achieving satisfactory Quality of Service (QoS) and Quality of Experience (QoE) targets. Round trip latency can be expressed as:

$$L_RT = T_c + T_n + T_s \tag{5.1}$$

where L_RT is round trip latency, T_c is the total time for the client device to decode, display image, and gather feedback, T_n is the transport delay through the network, and T_s is the time taken for the cloud system to render the new frame. Studies suggest a target round-trip latency of 100ms for cloud-based systems to maintain user experience [219, 220], a target difficult to achieve with increasing resolutions and frame rates.

Google Stadia's tests highlight the latency challenge, showing an increase from 125-150ms for 1080p to as high as 279ms for 4K rendering [221]. Given the escalating demand for higher resolutions and the direct impact on latency and bandwidth requirements, super-resolution emerges as a compelling alternative. Stadia offers three resolution modes: $1280x720(720p)$, $1920x1080(1080p)$, and $3840x2160(4K)$. The recommended minimum connection bandwidth is $10Mbps$, $28Mbps$, and $35Mbps$ for $720p$, $1080p$, and $4K$ resolutions, respectively [221]. This underscores the need for alternate approaches to make

Table 5.1: The number of pixels in the x and y dimensions for the selected image resolutions.

Resolution	Num. Pixels in x dimension	Num. Pixels in y dimension	Total pixels
VGA	640	480	307200
SVGA	800	600	480000
XGA	1024	768	786432
WXGA	1280	800	1024000
HD/720p	1280	720	921600
Full HD/1080p	1920	1080	2073600
2K	2048	1080	2211840
QHD/1440p	2560	1440	3686400
4K/UHD/2160p	3840	2160	8294400
5K	5120	2880	14745600
8K/SUHD	7680	4320	33177600

high resolution cloud based rendering a reality and motivates the use of super-resolution-based techniques. E.g. rendering 720p input frames and super resolving them to 1440p frames, can potentially reduce the round trip latency, limit the minimum network bandwidth requirement to 10 Mbps, and make high resolution cloud-based-rendering practical.

In later sections, we explore various open-source, state-of- the-art super-resolution algorithms and assess their potential to enhance the efficiency and feasibility of high-resolution rendering in both NeRF and LDMs, aiming to address the escalating computational costs and network demands in cloud-based rendering environments.

5.3 Open-source, state-of-the-art super-resolution algorithms

Figure 5.6 presents the block diagram of the model architecture of a typical convolution-based super-resolution network. A typical CNN-based super-resolution network consists of three main blocks: a feature extractor, a ConvLSTM and a reconstruction module. Super-resolution networks are typically fully convolutional and are optimized to operate in low-resolution space, to optimize the computationally cost. The upsampling to desired resolution

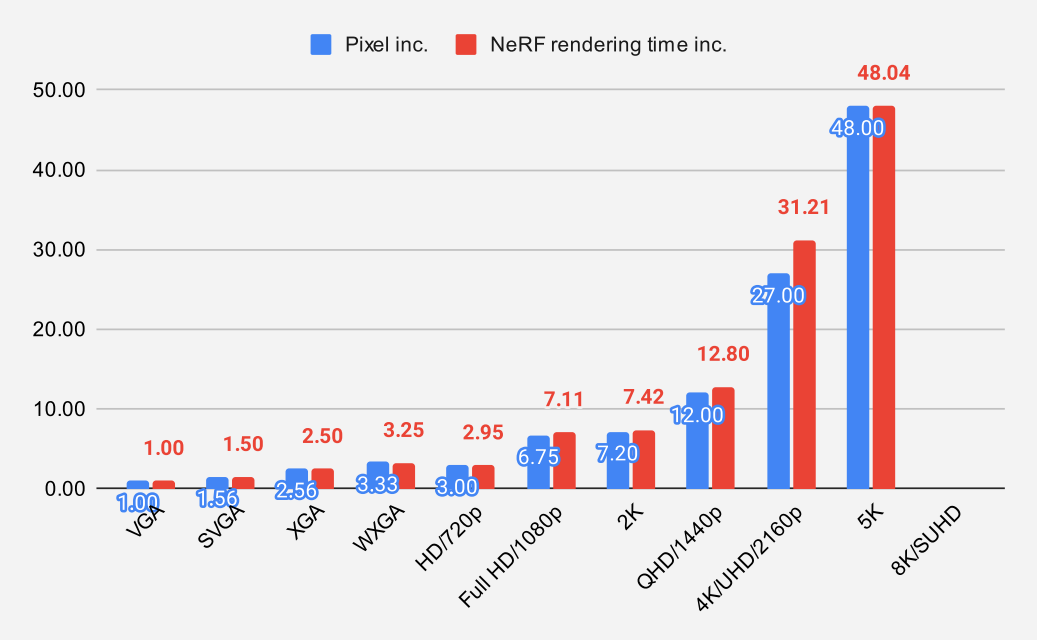


Figure 5.2: Normalized pixel count and rendering time for Neural Representations (NRs) on an RTX 3090 (24GB memory), relative to VGA resolution.

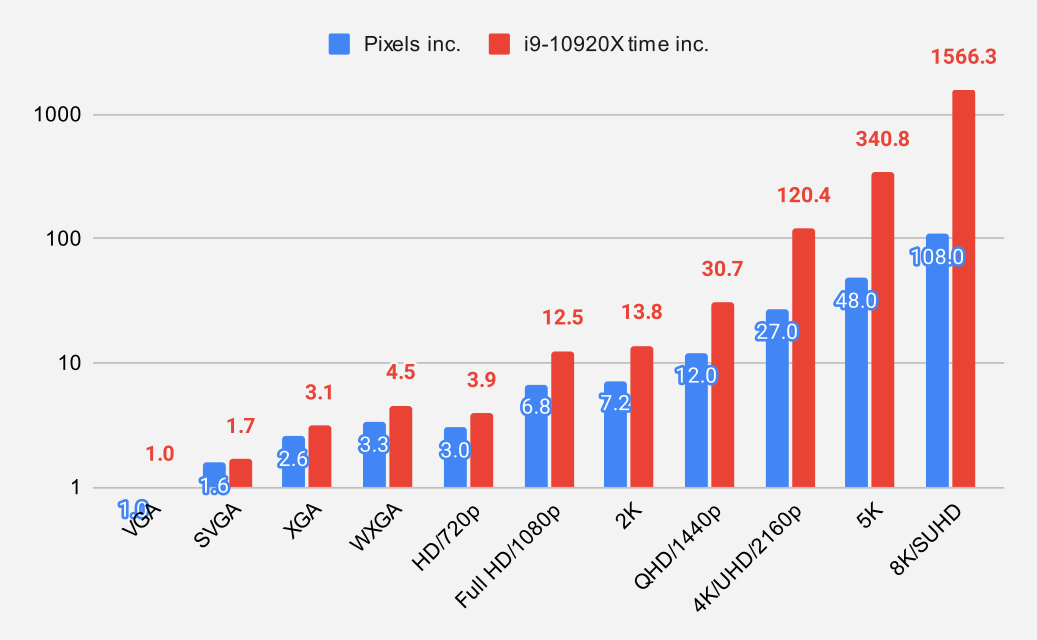


Figure 5.3: Normalized pixel count and rendering time for Latent Diffusion Model on an intel i9-10920X, relative to VGA resolution.

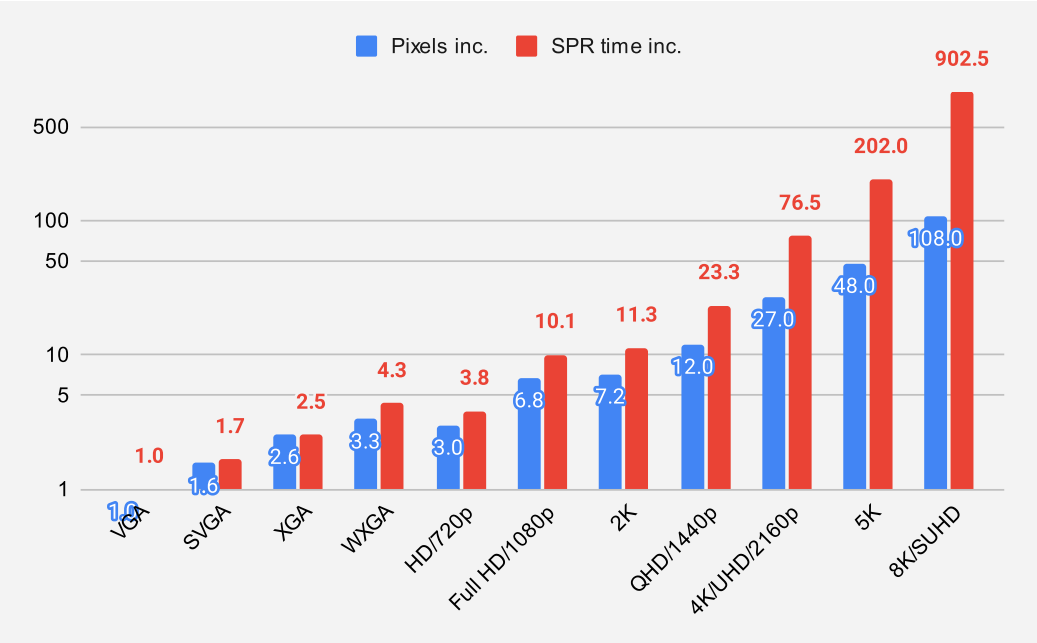


Figure 5.4: Normalized pixel count and rendering time for Latent Diffusion Model on an intel Xeon Gold 6430L, relative to VGA resolution.

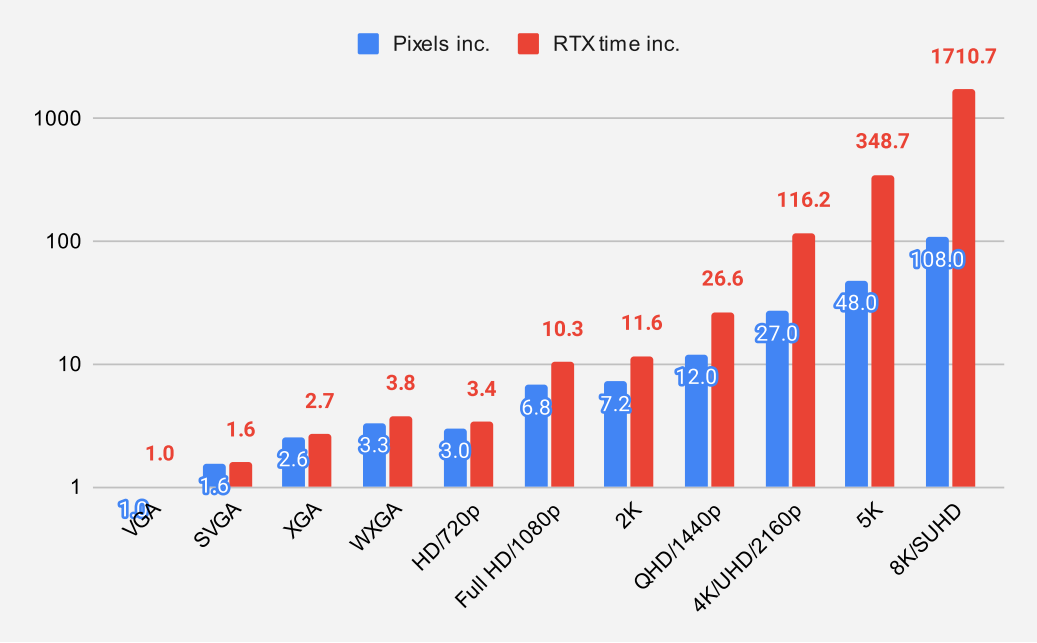


Figure 5.5: Normalized pixel count and rendering time for Latent Diffusion Model on an RTX3090 (24GB memory), relative to VGA resolution.

Table 5.2: The percentage of time spent on the Transformer and the ResNet blocks for different image resolutions.

Resolution	ResNet2D (%)	Transformer2D (%)	Rest (%)
VGA	45.34	24.33	30.33
SVGA	55.07	29.15	15.78
XGA	59.24	26.77	13.99
WXGA	45.20	24.42	30.38
HD/720p	54.81	29.07	16.12
Full HD/1080p	63.67	24.42	11.91
2K	31.77	58.50	9.73
QHD/1440p	9.83	86.25	3.92
4K/UHD/2160p	3.05	95.72	1.23
5K	1.65	97.69	0.66
8K/SUHD	0.25	99.65	0.10

is delayed towards the output layer.

- Feature Extractor: The feature extractor takes the low-resolution frame along with relevant meta-data information (typically anti-aliasing mask buffer, object overlap buffer etc) as input and passes them through a multi-layer (3-4 layers) convolutional neural network. The extracted features are then typically concatenated with the input channels before being fed into the ConvLSTM block.

- ConvLSTM: Extensive research on super-resolution has established that the use of hardware motion vectors causes visual artifacts and incurs significant computation overhead. To improve temporal stability and motion handling, a recurrent neural network module is usually incorporated in super-resolution networks, since it preserves temporal coherence in adjacent frames naturally. The ConvLSTM [217] block is responsible to handle the motion between successive frames. Abstractly, the ConvLSTM module merges historical features with the features of the current frame. Note that the feature extractor's output skips over the ConvLSTM module and gets concatenated with ConvLSTM's output before reconstruction.

- Reconstruction block: In super-resolution network, a CNN-based UNet [222] with skip connections is used as the reconstruction module to recover the high-resolution image. UNet includes a series of encoder blocks that downsample the input followed by decoder blocks that reverse this process. A typical UNet usually has identical input and output dimensions, but in super-resoluton UNet takes inputs from the low-resolution space and outputs a high-resolution frame. This results in an unbalanced UNet in which the length of the encoding path is reduced based on the network upsampling scale. For instance, one level of downsampling path is employed if the super-resolution upsampling scale is two. Usually, bi-linear interpolation is applied to the UNet's decoding phase instead of deconvolution because the computational cost is lower.

We profiled several open-source, state-of-the-art super-resolution networks including SISR networks (PAN [213], RCAN [212]), VSR networks (ESPCN [223], EDVR [211]) and rendering based super-resolution networks (NSRR [215] and CISR [224]).

Performance is measured as the inference time for one full frame within the PyTorch framework. All experiments are performed on an Intel Xeon platform with an Nvidia RTX3090 GPU. Table 5.3 reports the model size and performance comparison of the state-of-the-art open-source super-resolution frameworks for 2× up-sampling.

Our profiling suggests that the CISR outperforms all frameworks other than ESPCN by a substantial margin (ESPCN has a very small model size which results in poor quality images). EDVR and RCAN have a long inference time because of their large model size, hindering their usage in practice. PAN has 50% fewer parameters than CISR but its inference time is still triple that of CISR. The reason is that PAN has a deeper but narrower architecture, which reduces parallelism. NSRR, which also targets super-resolution for rendering, is $\approx 2.6\times$ slower than CISR. The main reason for this is that although NSRR has similar model size and depth to CISR, a large portion of NSRR's computation is spent in the target-resolution space instead of in the low-resolution space. We further optimized the CISR model with Nvidia TensorRT [225] using FP16 and the path optimized run-time is $\approx 27.7\ ms$. In-depth inference optimization is not covered in this thesis and we leave it

to future work.

We conduct image quality experiments on three popular scenes which are typically used for image quality comparison experiments: *zeroday*[1], *sun temple*[2], and *courtyard*[3].

We evaluate image quality with four metrics:

- PSNR: peak signal-to-nose ratio
- SSIM: structural similarity index
- TEMP: temporal stability (temporal L1 loss)
- PERC: perceptual loss (pre-trained VGG19 network)

The image quality comparison between each framework is shown in Table 5.4. ESPCN has the fastest inference time but fails to yield competitive image quality because of its small model capacity. The quality of EDVR and RCAN is good, but these are overly complex architectures designed for photographic images and do not work well with rendered content due to higher aliasing. NSRR adopts a feature re-weighting module with goal of easing the effects caused by hardware motion vectors. However, this module cannot deal with the amount of artifacts introduced by using four previous motion- compensated frames, leading to image quality degradation. In addition, NSRR uses a fixed learning rate and this causes instability and convergence problems towards the end of the training. The quality of PAN images is closest to CISR, but CISR is $\approx 2.8\times$ faster. To sum up, CISR offers superior performance for quality metrics, especially PSNR, temporal stability, and perceptual loss.

We further optimized the CISR model by quantizing its weights and activations to 8-bits using pytorch dynamic quantization API [226]. We used the 8-bit quantized model to perform super resolution and studied the quality of super-resolved image. Our results showed an insignificant drop in the output (super-resolved) image quality. For Dota2, the PSNR of the super-resolved image dropped from 34.93*db* to 34.88 and the structural similarity index (SSIM) dropped from 0.9738 to 0.9736. Without any further compiler optimizations, the inference latency of the quantized model is reduced by $1.6\times$

[1]https://developer.nvidia.com/orca/beeple-zero-day

[2]https://developer.nvidia.com/ue4-sun-temple

[3]https://assetstore.unity.com/packages/essentials/tutorial-projects/the-courtyard-49377

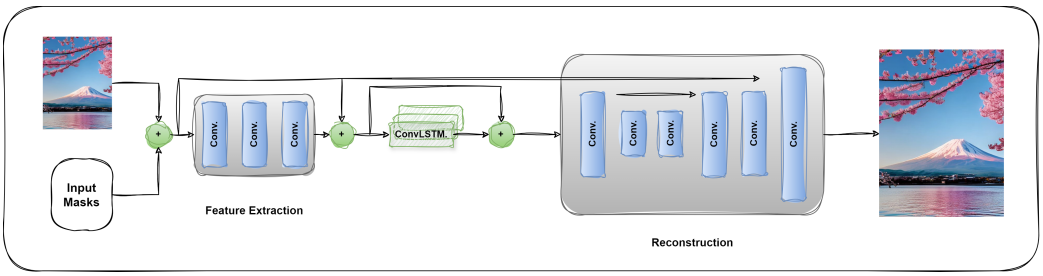


Figure 5.6: Model architecture of a typical super-resolution network, including a feature extractor, a convolutional long short-term memory (ConvLSTM) network and an unbalanced UNet for reconstruction.

Table 5.3: Model size and performance comparison of the state-of-the-art open-source super-resolution frameworks for 2× up-sampling.

	Num Parameters	Time (msec)	TensorRT (msec)
ESPCN	21284	11	-
EDVR	20633827	4952	-
RCAN	15444667	3264	-
PAN	261403	178	-
NSRR	546999	165	-
CISR	543523	64	27.7

on a GPU, compared to the original un-quantized model. Table 5.4 presents the quality comparison of the state-of-the-art open-source super-resolution frameworks for 2× up-sampling.

5.4 Combining Super Resolution with Neural Graphics

As neural rendering techniques advance, the challenge of achieving high resolutions while maintaining reasonable rendering times continues to be an active area of research. Neural network based super-resolution techniques offer a promising solution to this challenge by potentially reducing rendering times without significantly compromising on image quality. This section explores the effectiveness of 2× super-resolution in improving rendering latency for both neural representations and latent diffusion models. By applying super-resolution, we aim to achieve substantial reductions in latency, facilitating smoother and more efficient rendering processes.

Figure 5.7 illustrates the rendering time of neural representations at vari-

Table 5.4: Quality comparison of the state-of-the-art open-source super-resolution frameworks for 2×2 up-sampling.

	ZeroDay			
	PSNR	SSIM	TEMP	PERC
ESPCN	38.92	0.973	6.7	0.87
EDVR	38.33	0.958	6.92	0.82
RCAN	39.83	0.983	5.41	0.62
PAN	40.5	0.984	4.72	0.61
NSRR	39.93	0.98	5.58	0.58
CISR	41.29	0.984	4.27	0.54

	Courtyard			
	PSNR	SSIM	TEMP	PERC
ESPCN	37.02	0.956	8.08	1.07
EDVR	37.78	0.957	6.04	1.12
RCAN	39.6	0.965	5.28	0.9
PAN	39.94	0.965	5.11	0.91
NSRR	38.25	0.949	7.23	0.89
CISR	40.13	0.964	4.95	0.79

	Sun Temple			
	PSNR	SSIM	TEMP	PERC
ESPCN	31.97	0.898	15.98	1.63
EDVR	34.25	0.935	10.04	1.31
RCAN	34.06	0.924	11.03	1.38
PAN	34.15	0.932	10.4	1.4
NSRR	33.23	0.925	12.35	1.35
CISR	34.28	0.933	10.18	1.23

ous image resolution levels, both with and without 2× super-resolution. Our findings indicate that super-resolution enhances the rendering latency of neural representations by factors ranging from 1.89× to 1.95×. On average, across all tested resolution levels, super-resolution improves the rendering latency by approximately 1.91×.

For latent diffusion models, the latency improvement is even more significant due to their super-linear scaling of rendering time with resolution. Figure 5.8 reports the rendering times for latent diffusion models at different resolution levels, comparing performance with and without 2× super-resolution. Our results show that super-resolution can enhance the rendering latency of latent diffusion models by factors ranging from 2.57× to 4.99×, with an average improvement across all resolutions of approximately 3.55×.

The implementation of super-resolution has demonstrated significant benefits in managing rendering times across different neural rendering techniques (NeRF, LDMs) and resolutions. These improvements are crucial not only for enhancing user experiences but also in cloud-based rendering scenarios where they help reduce bandwidth requirements and link latency by allowing lower resolution images to be transmitted over the network and then upscaled on the client side. This reduction in data transfer also increases the server density as now more clients can be served for a given link bandwidth. Future work will focus on further optimizing these super-resolution techniques, exploring their direct integration into neural rendering frameworks, and extending their use to even higher upscaling factors (higher than 2× up-sampling). This continued advancement holds the potential to redefine the boundaries of feasible neural rendering speeds and resolutions in modern visual computing, making high-quality neural rendering more accessible and efficient.

5.5 Related Work

Prior work on high quality super-resolution has largely focused on single image super-resolution (SISR) or video super-resolution (VSR). SISR takes just one low-resolution image as input and outputs its corresponding high-resolution image [212, 213, 223, 227–232]. In contrast to SISR, VSR takes multiple low-resolution frames as input and reconstructs one high-resolution

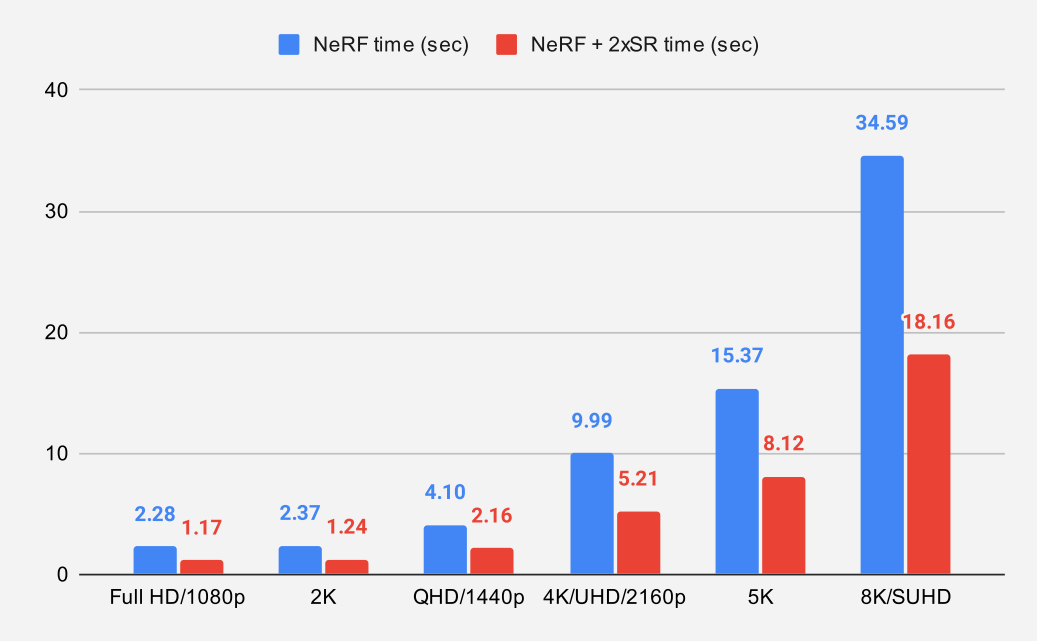


Figure 5.7: Rendering time of neural representations for different image resolution levels with and without 2× super-resolution.

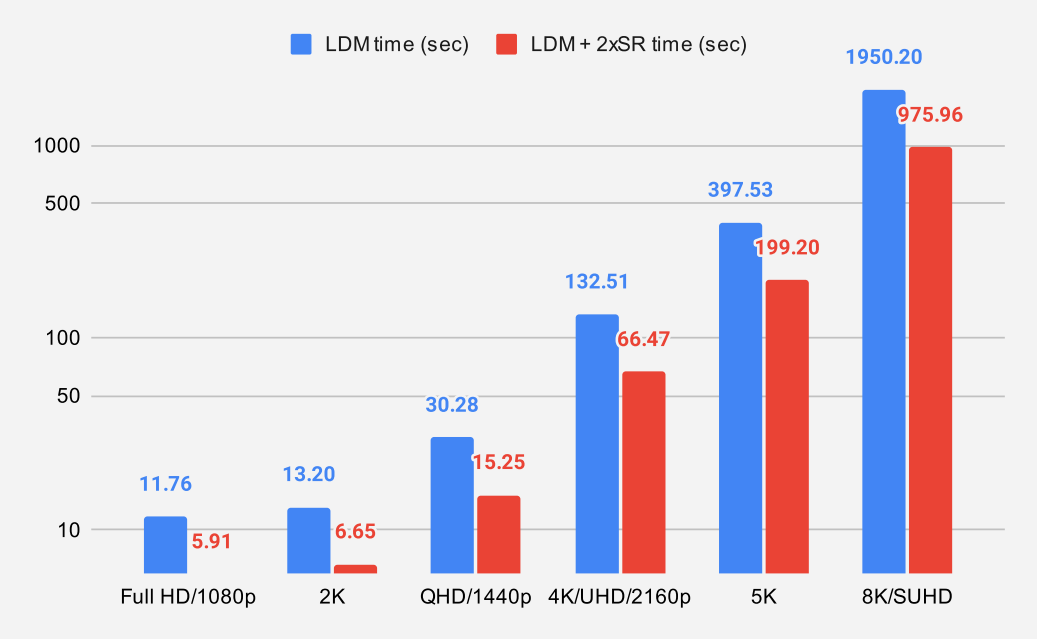


Figure 5.8: Rendering time of latent diffusion models for different image resolution levels with and without 2× super-resolution.

frame [210, 211, 233–240]. However, the SISR and VSR methods typically do not apply to rendered content. The fundamental difference in image formation between rendered and photographic images is that each sample in the rendering is temporally placed, i.e. represented in both space and time, in contrast to a pixel area integral in photographic images. Therefore, the rendered content is typically highly aliased, especially at a low resolution. SISR and VSR methods show severe issues in terms of quality and/or latency when applied to rendering.

There has been relatively limited work on super-resolution for rendering. Nvidia's DLSS [214] represents state-of-the-art but is proprietary. Besides, DLSS may be significantly limited in context of split rendering due to high network bandwidth requirement [241] [242] [243]. Since DLSS is a closed-source framework, quantitative profiling of its different pipeline stages is not possible.

Another closely related work is *NSRR* [215]. NSRR relies heavily on hardware motion vectors, which are considered unreliable. In addition, NSRR operates in the target-resolution space instead of the low-resolution space, leading to unnecessary computational cost. Unlike DLSS which is closed-source, NSRR is open-source. We, therefore, implement NSRR and provide GPU based runtime analysis.

There have been several FPGA-based attempts at accelerating SISR [244] [245] and VSR [246] algorithms. These accelerators are often limited by high on-chip memory requirements for these super-resolution algorithms (compared to the limited amount of memory available on FPGAs). ASIC-based solutions [247] [248] [249] have also been proposed to achieve energy savings and performance. However, they have been focused on classical super-resolution, SISR, and VSR. For example, [247] presents a dedicated super-resolution processor that delivers a throughput of 25-to-60fps at 211mW. This throughput is achieved for 480×270 input images when the upscaling factor of four is applied. Lee and Park [248] propose a Lagrange interpolation method with a sharpening algorithm which was implemented in hardware, achieving 4K UHD video streams from HD and FHD video streams at 30 fps speed. Another work [249] proposes edge orientation and learned linear mappings to perform super-resolution and achieves 60 fps as well.

5.6 Summary and Discussion

This chapter explores the computational challenges, high latency, and energy consumption associated with high- resolution neural rendering, emphasizing the role of super- resolution techniques in enhancing performance and energy efficiency. By employing 2× super-resolution, the research demonstrates a significant reduction in rendering latency for both neural representations and latent diffusion models. Specifically, 2× super-resolution has improved the rendering latency of neural representations by an average factor of approximately 1.91×, with improvements ranging from 1.89× to 1.95× across different resolution levels.

For latent diffusion models, which exhibit super-linear scaling of rendering time with resolution, the benefits of super-resolution are even more pronounced. The rendering latency improvements range from 2.57× to 4.99×, with an average enhancement of approximately 3.55×. These substantial reductions in rendering times facilitate smoother and more efficient rendering processes, which are crucial not only for improving user experiences but also for enhancing the efficiency of cloud-based rendering scenarios.

Integrating super-resolution with neural rendering not only achieves lower rendering latencies but also reduces data transfer demands, thereby enhancing server density and enabling more clients to be served within the same network bandwidth. Future research will focus on further enhancing these super-resolution techniques, integrating them directly into the neural rendering pipeline, and extending their capabilities to upscaling factors beyond 2×. Such progress has the potential to transform the limits of neural rendering speeds and resolutions, making high-quality neural rendering more accessible and efficient across modern visual computing platforms.

Chapter 2 discusses the real-time, near- real-time, and energy efficiency demands of neural graphics. In this chapter, we discussed the potential of combining neural graphics with neural super-resolution techniques to improve rendering latency and reduce computational costs of high-resolution images. By employing super-resolution methods, we demonstrated the capability to render at lower resolutions and upscale to high resolutions, achieving the desired visual fidelity without the associated computational expense of natively rendering at higher resolutions. Our results showed improvements in both performance and energy efficiency. By employing 2× super-resolution, the

rendering latency of neural representations improves by an average factor of approximately 1.91×, with improvements ranging from 1.89× to 1.95× across different resolution levels. For latent diffusion models, the rendering latency improvements range from 2.57× to 4.99×, with an average enhancement of approximately 3.55×. These advancements make the merger of neural graphics and super-resolution particularly beneficial for applications in virtual reality and augmented reality, where real-time or near-real-time performance is critical. However, further research is needed to optimize the balance between rendering speed and quality for ultra-high-definition content, indicating that while significant progress has been made, the quest for optimal real-time performance continues.

CHAPTER 6

DISCUSSION AND FUTURE WORK

This dissertation has thoroughly explored the computational and memory access patterns of neural graphics applications and the defining characteristics of specialized hardware accelerators for addressing the intensive computational demands of neural graphics. Our in-depth analysis spans across three distinct yet interconnected areas of this field: non-generative neural graphics, generative neural graphics via LDMs, and neural super-resolution techniques.

In non-generative neural graphics, the Neural Graphics Processing Cluster (NGPC) is introduced to specifically tackle the bottlenecks in input encoding and MLP kernels, prevalent across neural graphic applications such as NeRF, NSDF, NVR, and GIA. Our evaluations indicate that the NGPC enhances the rendering performance significantly, enabling 4K resolutions at 30 FPS for NeRF and 8K resolutions at 120 FPS for other applications, marking performance gains from 12.94× to 39.04× depending on the hardware scaling factors.

For generative neural graphics, the Latent Diffusion Processing Unit (LDPU) addresses the high energy and computational inefficiencies of LDMs. The LDPU architecture, with its specialized support for sparsity and special values, delivers substantial improvements in energy efficiency, achieving up to a 572× increase over traditional GPUs.

Additionally, the integration of super-resolution techniques mitigates the high computational demands of neural rendering. By rendering at lower resolutions and employing upscaling, these techniques efficiently enhance image quality and reduce the overall computational load, enabling more practical applications in real-time environments.

This dissertation has explored the computational and memory characteristics of neural graphics, alongside their integration with relatively mature technologies such as super-resolution. Building on the findings from this research, there are several promising directions for further exploration. This

section outlines potential areas for future research that aim to advance the integration of neural graphics within traditional rendering pipelines, enhancing both their efficiency and application scope.

6.1 Integration of Neural Graphics with Conventional Rendering Pipelines

The current research primarily focuses on standalone neural graphics pipelines. However, there is significant potential in more tightly integrating neural graphics within conventional rendering pipelines. Future studies could explore methodologies for offloading specific rendering tasks, such as dynamic lighting adjustments, to neural networks directly within the rendering pipeline. For example, conventional graphics could render static scene elements while neural graphics handle dynamic components like lighting, which could then be seamlessly integrated back into the main pipeline for further processing. This hybrid approach could reduce computational overhead and improve real-time rendering capabilities, especially in dynamic environments.

6.2 Bringing Systolic Arrays Closer to Graphics Pipelines

While GPU architectures have evolved to include ML accelerators like Tensor cores and systolic arrays, these are often underutilized in conventional rendering tasks. Future research should investigate methods to physically and logically integrate systolic arrays closer to graphic pipelines. This involves developing algorithms that can schedule ML workloads directly on these accelerators during specific stages of graphics processing, allowing for direct interaction and data flow between systolic arrays and the graphics pipeline. Such integration could dramatically increase the efficiency of rendering processes that benefit from neural network computations.

6.3 Unified Hardware Architecture for Generative and Non-Generative Neural Graphics

Integrating neural radiance fields with latent diffusion models presents unique challenges due to their differing computational and memory demands. Future work could focus on developing a unified hardware architecture that efficiently supports both generative and non-generative neural graphics applications. This integration would likely offer novel microarchitectural and system-level design opportunities, enhancing the capability of hardware platforms to handle complex neural graphics tasks seamlessly.

6.4 Tiny Neural Network Architectures for Resource-Constrained Environments

There is a growing need for tiny neural network architectures, particularly for generative models, to operate effectively in resource-constrained environments. While initiatives like instant-NGP have demonstrated the feasibility of using tiny MLPs for real-time rendering, there is potential to extend these concepts to generative models such as tiny transformers. Research in this area could explore trade-offs between model generality and efficiency, potentially leading to significant speed-ups through optimized network structures.

6.5 Simulation and Emulation Tools for Neural Graphics

There is a notable absence of frameworks capable of evaluating neural graphics hardware platforms. Given the distinct characteristics of neural graphics hardware, as demonstrated in this research, there is a clear need to develop simulation and emulation tools for this new class of architectures. This would not only facilitate better design and optimization of neural graphics systems but also open up new research avenues in hardware simulation and architecture evaluation.

REFERENCES

[1] CompVis, "Github file: Stable diffusion v1 inference configuration," https://github.com/CompVis/stable-diffusion/blob/main/configs/stable-diffusion/v1-inference.yaml, 2024, accessed: 18 Apr. 2024.

[2] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[3] "Vulkan tutorial - graphics pipeline basics," 2022. [Online]. Available: https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Introduction

[4] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi et al., "Advances in neural rendering," in *Computer Graphics Forum*, vol. 41, no. 2. Wiley Online Library, 2022, pp. 703–735.

[5] Y.-H. Chen, R. Sarokin, J. Lee, J. Tang, C.-L. Chang, A. Kulik, and M. Grundmann, "Speed is all you need: On-device acceleration of large diffusion models via gpu-aware optimizations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4650–4654.

[6] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[7] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner et al., "State of the art on neural rendering," in *Computer Graphics Forum*, vol. 39, no. 2. Wiley Online Library, 2020, pp. 701–727.

[8] K. Agusanto, L. Li, Z. Chuangui, and N. W. Sing, "Photorealistic rendering for augmented reality using environment illumination," in *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* IEEE, 2003, pp. 208–216.

[9] J. Kronander, F. Banterle, A. Gardner, E. Miandji, and J. Unger, "Photorealistic rendering of mixed reality scenes," in *Computer Graphics Forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 643–665.

[10] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 143–150.

[11] H. Shum and S. B. Kang, "Review of image-based rendering techniques," in *Visual Communications and Image Processing 2000*, vol. 4067. SPIE, 2000, pp. 2–13.

[12] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

[13] Z. Zainuddin and O. Pauline, "Function approximation using artificial neural networks," *WSEAS Transactions on Mathematics*, vol. 7, no. 6, pp. 333–338, 2008.

[14] S. Yang, T. Ting, K. L. Man, and S.-U. Guan, "Investigation of neural networks for function approximation," *Procedia Computer Science*, vol. 17, pp. 586–594, 2013.

[15] S. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor et al., "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[16] 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," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[17] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-nerf: Scalable large scene neural view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8248–8258.

[18] A. Jo, "The promise and peril of generative ai," *Nature*, vol. 614, no. 1, pp. 214–216, 2023.

[19] F. Fui-Hoon Nah, R. Zheng, J. Cai, K. Siau, and L. Chen, "Generative ai and chatgpt: Applications, challenges, and ai-human collaboration," pp. 277–304, 2023.

[20] K. I. Roumeliotis and N. D. Tselikas, "Chatgpt and open-ai models: A preliminary review," *Future Internet*, vol. 15, no. 6, p. 192, 2023.

[21] V. Alto, *Modern Generative AI with ChatGPT and OpenAI Models: Leverage the capabilities of OpenAI's LLM for productivity and innovation with GPT3 and GPT4*. Packt Publishing Ltd, 2023.

[22] K. S. Kalyan, "A survey of gpt-3 family large language models including chatgpt and gpt-4," *Natural Language Processing Journal*, p. 100048, 2023.

[23] S. Tingiris and B. Kinsella, *Exploring GPT-3: An unofficial first look at the general-purpose language processing API from OpenAI*. Packt Publishing Ltd, 2021.

[24] R. C. Basole and T. Major, "Generative ai for visualization: Opportunities and challenges," *IEEE Computer Graphics and Applications*, vol. 44, no. 2, pp. 55–64, 2024.

[25] Y. Ye, J. Hao, Y. Hou, Z. Wang, S. Xiao, Y. Luo, and W. Zeng, "Generative ai for visualization: State of the art and future directions," *arXiv preprint arXiv:2404.18144*, 2024.

[26] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[27] H. Cao, C. Tan, Z. Gao, Y. Xu, G. Chen, P.-A. Heng, and S. Z. Li, "A survey on generative diffusion models," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[28] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[29] P. Esser, J. Chiu, P. Atighehchian, J. Granskog, and A. Germanidis, "Structure and content-guided video synthesis with diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 7346–7356.

[30] D. Z. Chen, Y. Siddiqui, H.-Y. Lee, S. Tulyakov, and M. Nießner, "Text2tex: Text-driven texture synthesis via diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 18 558–18 568.

[31] A. Molaei, A. Aminimehr, A. Tavakoli, A. Kazerouni, B. Azad, R. Azad, and D. Merhof, "Implicit neural representation in medical imaging: A comparative survey," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2381–2391.

[32] M. Wysocki, M. F. Azampour, C. Eilers, B. Busam, M. Salehi, and N. Navab, "Ultra-nerf: neural radiance fields for ultrasound imaging," in *Medical Imaging with Deep Learning*. PMLR, 2024, pp. 382–401.

[33] A. Corona-Figueroa, J. Frawley, S. Bond-Taylor, S. Bethapudi, H. P. Shum, and C. G. Willcocks, "Mednerf: Medical neural radiance fields for reconstructing 3d-aware ct-projections from a single x-ray," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2022, pp. 3843–3848.

[34] Z. Chen, L. Yang, J.-H. Lai, and X. Xie, "Cunerf: Cube-based neural radiance field for zero-shot medical image arbitrary-scale super resolution," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 21 185–21 195.

[35] X. Wang, S. Hu, H. Fan, H. Zhu, and X. Li, "Neural radiance fields in medical imaging: Challenges and next steps," *arXiv preprint arXiv:2402.17797*, 2024.

[36] T. J. Jang and C. M. Hyun, "Nerf solves undersampled mri reconstruction," *arXiv preprint arXiv:2402.13226*, 2024.

[37] H. Li, H. Chen, W. Jing, Y. Li, and R. Zheng, "3d ultrasound spine imaging with application of neural radiance field method," in *2021 IEEE International Ultrasonics Symposium (IUS)*. IEEE, 2021, pp. 1–4.

[38] K. Iddrisu, S. Malec, and A. Crimi, "3d reconstructions of brain from mri scans using neural radiance fields," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2023, pp. 207–218.

[39] Q. Y. Chen, "3d reconstruction of endoscopy images with nerf," Ph.D. dissertation, New York University Tandon School of Engineering, 2023.

[40] B. Huang, Y. Wang, A. Nguyen, D. Elson, F. Vasconcelos, and D. Stoyanov, "High-fidelity endoscopic image synthesis by utilizing depth-guided neural surfaces," *arXiv preprint arXiv:2404.13437*, 2024.

[41] T. Yuan, Y. Mao, J. Yang, Y. Liu, Y. Wang, and H. Zhao, "Presight: Enhancing autonomous vehicle perception with city-scale nerf priors," *arXiv preprint arXiv:2403.09079*, 2024.

[42] O. A. Støle, "From roadside to render: An end-to-end nerf pipeline for view synthesis and 3d reconstruction in the context of autonomous driving," M.S. thesis, NTNU, 2023.

[43] M.-Y. Shen, C.-C. Hsu, H.-Y. Hou, Y.-C. Huang, W.-F. Sun, C.-C. Chang, Y.-L. Liu, and C.-Y. Lee, "Driveenv-nerf: Exploration of a nerf-based autonomous driving environment for real-world performance validation," *arXiv preprint arXiv:2403.15791*, 2024.

[44] C. Lindström, G. Hess, A. Lilja, M. Fatemi, L. Hammarstrand, C. Petersson, and L. Svensson, "Are nerfs ready for autonomous driving? towards closing the real-to-simulation gap," *arXiv preprint arXiv:2403.16092*, 2024.

[45] L. He, L. Li, W. Sun, Z. Han, Y. Liu, S. Zheng, J. Wang, and K. Li, "Neural radiance field in autonomous driving: A survey," *arXiv preprint arXiv:2404.13816*, 2024.

[46] J. Cao, Z. Li, N. Wang, and C. Ma, "Lightning nerf: Efficient hybrid scene representation for autonomous driving," *arXiv preprint arXiv:2403.05907*, 2024.

[47] Z. Wu, T. Liu, L. Luo, Z. Zhong, J. Chen, H. Xiao, C. Hou, H. Lou, Y. Chen, R. Yang et al., "Mars: An instance-aware, modular and realistic simulator for autonomous driving," in *CAAI International Conference on Artificial Intelligence*. Springer, 2023, pp. 3–15.

[48] J. Zhang, F. Zhang, S. Kuang, and L. Zhang, "Nerf-lidar: Generating realistic lidar point clouds with neural radiance fields," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 7, 2024, pp. 7178–7186.

[49] C. Jiang, R. Niu, Z. Chen, C. Hua, X. Kuang, X. Fu, B. Yu, and H. Zhu, "Large-scale neural scene disentanglement approach for self-driving simulation," *IEEE Transactions on Intelligent Vehicles*, 2024.

[50] R. Jarolim, B. Tremblay, A. Muñoz-Jaramillo, K.-M. Bintsi, A. Jungbluth, M. Santos, A. Vourlidas, J. P. Mason, S. Sundaresan, C. Downs et al., "Sunerf: 3d reconstruction of the solar euv corona using neural radiance fields," *The Astrophysical Journal Letters*, vol. 961, no. 2, p. L31, 2024.

[51] M. Hansen, C. Adams, T. Fong, and D. Wettergreen, "Analyzing the effectiveness of neural radiance fields for geometric modeling of lunar terrain," in *2024 IEEE Aerospace Conference*. IEEE, 2024, pp. 1–12.

[52] E. Šlapak, E. Pardo, M. Dopiriak, T. Maksymyuk, and J. Gazda, "Neural radiance fields in the industrial and robotics domain: applications, research opportunities and use cases," *arXiv preprint arXiv:2308.07118*, 2023.

[53] S. Lee, L. Chen, J. Wang, A. Liniger, S. Kumar, and F. Yu, "Uncertainty guided policy for active robotic 3d reconstruction using neural radiance fields," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 070–12 077, 2022.

[54] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager, "Vision-only robot navigation in a neural radiance world," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.

[55] L. Yen-Chen, P. Florence, J. T. Barron, T.-Y. Lin, A. Rodriguez, and P. Isola, "Nerf-supervision: Learning dense object descriptors from neural radiance fields," in *2022 international conference on robotics and automation (ICRA)*. IEEE, 2022, pp. 6496–6503.

[56] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint, "Learning multi-object dynamics with compositional neural radiance fields," in *Conference on robot learning*. PMLR, 2023, pp. 1755–1768.

[57] J. Ichnowski, Y. Avigal, J. Kerr, and K. Goldberg, "Dex-nerf: Using a neural radiance field to grasp transparent objects," *arXiv preprint arXiv:2110.14217*, 2021.

[58] C. Liu, K. Shi, K. Zhou, H. Wang, J. Zhang, and H. Dong, "Rgbgrasp: Image-based object grasping by capturing multiple views during robot arm movement with neural radiance fields," *IEEE Robotics and Automation Letters*, 2024.

[59] L. Fabra, J. E. Solanes, A. Muñoz, A. Martí-Testón, A. Alabau, and L. Gracia, "Application of neural radiance fields (nerfs) for 3d model representation in the industrial metaverse," *Applied Sciences*, vol. 14, no. 5, p. 1825, 2024.

[60] B. Salome, "Nerf-based scene perception for an industrial delta parallel robot," in *The 19th International scientific and practical conference "Creative business management and implementation of new ideas"(May 14–17, 2024) Tallinn, Estonia. International Science Group. 2024. 281 p.*, 2024, p. 214.

[61] M. Hillemann, R. Langendörfer, M. Heiken, M. Mehltretter, A. Schenk, M. Weinmann, S. Hinz, C. Heipke, and M. Ulrich, "Novel view synthesis with neural radiance fields for industrial robot applications," *arXiv preprint arXiv:2405.04345*, 2024.

[62] H. Turki, D. Ramanan, and M. Satyanarayanan, "Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 922–12 931.

[63] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[64] H. Li, Y. Yang, M. Chang, S. Chen, H. Feng, Z. Xu, Q. Li, and Y. Chen, "Srdiff: Single image super-resolution with diffusion probabilistic models," *Neurocomputing*, vol. 479, pp. 47–59, 2022.

[65] Z. Yue, J. Wang, and C. C. Loy, "Resshift: Efficient diffusion model for image super-resolution by residual shifting," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[66] S. Gao, X. Liu, B. Zeng, S. Xu, Y. Li, X. Luo, J. Liu, X. Zhen, and B. Zhang, "Implicit diffusion models for continuous super-resolution," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 10 021–10 030.

[67] B. B. Moser, A. S. Shanbhag, F. Raue, S. Frolov, S. Palacio, and A. Dengel, "Diffusion models, image super-resolution and everything: A survey," *arXiv preprint arXiv:2401.00736*, 2024.

[68] A. Niu, T. X. Pham, K. Zhang, J. Sun, Y. Zhu, Q. Yan, I. S. Kweon, and Y. Zhang, "Acdmsr: Accelerated conditional diffusion models for single image super-resolution," *IEEE Transactions on Broadcasting*, 2024.

[69] Z. Luo, F. K. Gustafsson, Z. Zhao, J. Sjölund, and T. B. Schön, "Refusion: Enabling large-size realistic image restoration with latent-space diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[70] Y. Zhao, T. Hou, Y.-C. Su, X. Jia, Y. Li, and M. Grundmann, "Towards authentic face restoration with iterative diffusion models and beyond," *arXiv preprint arXiv:2304.12345*, 2023.

[71] C. Ma, W. Tan, R. He, and B. Yan, "Pretraining a foundation model for generalizable fluorescence microscopy-based image restoration," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[72] S. Xie, Y. Zhao, Z. Xiao, K. Chan, Y. Li, Y. Xu, K. Zhang, and T. Hou, "Dreaminpainter: Text-guided subject-driven image inpainting with diffusion models," *arXiv preprint arXiv:2312.03771*, 2023.

[73] M. Gong, S. Xie, W. Wei, M. Grundmann, K. Batmanghelich, and T. Hou, "Semi-implicit denoising diffusion models (siddms)," *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[74] P. Li, Z. Zhang, and Z. He, "I3fdm: Iris inpainting via inverse fusion of diffusion models," *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2024.

[75] G. Couairon, A. Grechka, and M. Cord, "Gradpaint: Gradient-guided inpainting with diffusion models," *Computer Vision and Image Understanding*, vol. 240, p. 103928, 2024.

[76] D. Zhang, J. Wu, J. Liu, R. Zhao, L. Ran, Y. Gu, D. Gao, and M. Shou, "Marrying pixel and latent diffusion models for text-to-video generation," *arXiv preprint arXiv:2309.15818*, 2023.

[77] H. Chen, Y. Zhang, X. Cun, M. Xia, X. Wang, C. Weng, and Y. Shan, "Videocrafter2: Overcoming data limitations for high-quality video diffusion models," *arXiv preprint arXiv:2401.09047*, 2024.

[78] Y. Xing, Y. He, Z. Tian, X. Wang, and Q. Chen, "Seeing and hearing: Open-domain visual-audio generation with diffusion latent aligners," *arXiv preprint arXiv:2402.17723*, 2024.

[79] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," *Advances in Neural Information Processing Systems*, pp. 1087–1098, 2017.

[80] A. Gupta, L. Yu, K. Sohn, X. Gu, M. Hahn, L. Fei-Fei, I. Essa, and L. Jiang, "Photorealistic video generation with diffusion models," *arXiv preprint arXiv:2312.06662*, 2023.

[81] Y. Wang, X. Chen, X. Ma, S. Zhou, Z. Huang, Y. Wang, C. Yang, Y. He, and J. Yu, "Lavie: High-quality video generation with cascaded latent diffusion models," *arXiv preprint arXiv:2309.15103*, 2023.

[82] X. Ma, Y. Wang, G. Jia, X. Chen, Z. Liu, Y. Li, C. Chen, and Y. Qiao, "Latte: Latent diffusion transformer for video generation," *arXiv preprint arXiv:2401.03048*, 2024.

[83] Z. Huang, Y. Yu, L. Yang, C. Qin, B. Zheng, Z. Zhou, Y. Wang, and X. Zhang, "Motion-aware latent diffusion models for video frame interpolation," in *arXiv preprint arXiv:2404.13534*, 2024.

[84] W. Weng, Y. Zhang, and Z. Xiong, "Event-based blurry frame interpolation under blind exposure," *arXiv preprint arXiv:2311.18834*, 2023.

[85] Y. Wu, Q. Wen, and Q. Chen, "Optimizing video prediction via video frame interpolation," in *arXiv preprint arXiv:2403.13807*, 2024.

[86] Z. Yuan, M. Cao, X. Wang, Z. Qi, C. Yuan, and Y. Shan, "Customnet: Zero-shot object customization with variable-viewpoints in text-to-image diffusion models," *arXiv preprint arXiv:2310.19784*, 2023.

[87] G. Lin, J. Han, M. Cao, Z. Zhong, and Y. Zheng, "Event-guided frame interpolation and dynamic range expansion of single rolling shutter image," *arXiv preprint arXiv:2312.01027*, 2023.

[88] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[89] Z. Chen, F. Long, Z. Qiu, T. Yao, W. Zhou, J. Luo, and T. Mei, "Learning spatial adaptation and temporal coherence in diffusion models for video super-resolution," *arXiv preprint arXiv:2403.17000*, 2024.

[90] S. Zhou, P. Yang, J. Wang, Y. Luo, and C. C. Loy, "Upscale-a-video: Temporal-consistent diffusion model for real-world video super-resolution," *arXiv preprint arXiv:2312.06640*, 2023.

[91] Z. Qiu, H. Yang, J. Fu, D. Liu, C. Xu, and D. Fu, "Learning spatiotemporal frequency-transformer for low-quality video super-resolution," in *arXiv preprint arXiv:2212.14046*, 2022.

[92] H. Wang, X. Xiang, Y. Tian, W. Yang, and Q. Liao, "Stdan: deformable attention network for space-time video super-resolution," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[93] T. Wu, C. Si, Y. Jiang, Z. Huang, and Z. Liu, "Freeinit: Bridging initialization gap in video diffusion models," *arXiv preprint arXiv:2312.07537*, 2023.

[94] Y. Wei, Y. Zhang, Z. Ji, J. Bai, L. Zhang, and W. Zuo, "Elite: Encoding visual concepts into textual embeddings for customized text-to-image generation," in *ICCV 2023 (Oral)*, 2023.

[95] J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, and Y. Yang, "Scaling autoregressive models for content-rich text-to-image generation," *arXiv preprint arXiv:2206.10789*, 2022.

[96] H. Chen, Y. Zhang, S. Wu, X. Wang, X. Duan, Y. Zhou, and W. Zhu, "Disenbooth: Identity-preserving disentangled tuning for subject-driven text-to-image generation," in *The Twelfth International Conference on Learning Representations*, 2023.

[97] W. Hong, M. Ding, W. Zheng, X. Liu, and J. Tang, "Cogview: Mastering text-to-image generation via transformers," *arXiv preprint arXiv:2205.15868*, 2022.

[98] J. Chen, J. Yu, C. Ge, L. Yao, E. Xie, Y. Wu, Z. Wang, J. Kwok, P. Luo, H. Lu, and Z. Li, "Pixart-: Fast training of diffusion transformer for photorealistic text-to-image synthesis," *arXiv preprint arXiv:2310.00426*, 2023.

[99] M. Ding, Z. Yang, W. Hong, W. Zheng, C. Zhou, D. Yin, J. Lin, X. Zou, Z. Shao et al., "Cogview: Mastering text-to-image generation via transformers," in *Advances in Neural Information Processing Systems*, 2021, pp. 19 822–19 835.

[100] Y. Tewel, R. Gal, G. Chechik, and Y. Atzmon, "Key-locked rank one editing for text-to-image personalization," *ACM SIGGRAPH 2023 Conference Proceedings*, pp. 1–11, 2023.

[101] OpenAI, "Dall-e 2," https://openai.com/dall-e-2, 2024, accessed: 18 Apr. 2024.

[102] "Midjourney," https://www.midjourney.com/, accessed: 2024-05-28.

[103] J. Shue, E. Chan, R. Po, Z. Ankner, J. Wu, and G. Wetzstein, "3d neural field generation using triplane diffusion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[104] S. Cai, E. Chan, S. Peng, M. Shahbazi, A. Obukhov, and L. Van Gool, "Diffdreamer: Consistent single-view perpetual view generation with conditional diffusion models," *arXiv preprint arXiv:2211.12131*, 2022.

[105] L. Yang, Z. Huang, Y. Song, S. Hong, G. Li, W. Zhang, B. Cui, and B. Ghanem, "Diffusion-based scene graph to image generation with masked contrastive pre-training," *arXiv preprint arXiv:2211.11138*, 2022.

[106] Z. Guo, X. Gao, J. Zhou, X. Cai, and B. Shi, "Scenedm: Scene-level multi-agent trajectory generation with consistent diffusion models," *arXiv preprint arXiv:2311.15736*, 2023.

[107] G. Kim, H. Kim, H. Seo, D. Kang, and S. Chun, "Beyondscene: Higher-resolution human-centric scene generation with pretrained diffusion," *arXiv preprint arXiv:2404.04544*, 2024.

[108] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[109] B. Poole, A. Jain, J. Ho, J. T. Barron, H.-X. Liu, B. Q. Ayan, D. Sala, T. Nguyen, and M. Norouzi, "Dreamfusion: Text-to-3d using 2d diffusion," in *arXiv preprint arXiv:2209.14988*, 2022.

[110] Z. Lin, A. Gritsenko, J. Huo, S. Ermon, E. R. Chan, R. Anil, P. Prashanth, S. Xu, E. Franklin, E. Jang, and et al., "Magic3d: High-resolution text-to-3d content creation," *arXiv preprint arXiv:2211.10440*, 2022.

[111] R. Wang, D. Sala, H. Liu, M. Norouzi, and J. T. Barron, "Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation," *arXiv preprint arXiv:2211.13745*, 2022.

[112] Y. Chen, A. Gritsenko, J. Wu, T. Zhao, C. Ma, Z. Wang, L. Xie, Y. Bai, X. Tian, and et al., "Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation," *arXiv preprint arXiv:2310.15647*, 2023.

[113] Z. Chen, J. Yu, T. Luong, T. Kurth, Y. Yao, and M. Yang, "Diffdreamer: Consistent single-view perpetual view generation with conditional diffusion models," *arXiv preprint arXiv:2305.18823*, 2023.

[114] H. Wen, Y. Lin, Y. Xia, H. Wan, Q. Wen, R. Zimmermann, and Y. Liang, "DiffSTG: Probabilistic spatio-temporal graph forecasting with denoising diffusion models," in *Proceedings of the ACM International Conference on Advances in Geographic Information Systems*, 2023.

[115] S. Kim, S. Park, and Y. Kim, "Temporal fusion diffusion models for time series forecasting," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[116] Y. Li, H. Wong, Y. Yuan, and B. Ren, "Learning temporal dynamics with denoising diffusion probabilistic models," *arXiv preprint arXiv:2209.12478*, 2022.

[117] F. Wang, H. Zhang, and P. Zhang, "Denoising diffusion models for spatio-temporal data," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[118] H. Yang, Y. Jin, F. Liu, and P. Zheng, "Stdiff: Spatio-temporal diffusion models for time series forecasting," *Advances in Neural Information Processing Systems*, 2022.

[119] F. Bieder, J. Wolleb, A. Durrer, R. Sandkuehler, and P. C. Cattin, "Memory-efficient 3d denoising diffusion models for medical image processing," in *Medical Imaging with Deep Learning*, 2023.

[120] Y. Chen, N. Konz, H. Dong, and M. A. Mazurowski, "Anatomically-controllable medical image generation with segmentation-guided diffusion models," *arXiv preprint arXiv:2402.05210*, 2024.

[121] A. Kazerouni, E. K. Aghdam, M. Heidari, R. Azad, M. Fayyaz, I. Hacihaliloglu, and D. Merhof, "Diffusion models in medical imaging: A comprehensive survey," *Medical Image Analysis*, p. 102846, 2023.

[122] C. Cao, Z. Cui, Y. Wang, S. Liu, T. Chen, H. Zheng, D. Liang, and Y. Zhu, "High-frequency space diffusion model for accelerated mri," *IEEE Transactions on Medical Imaging*, 2024.

[123] O. Dalmaz, M. Ozbey, S. U. Dar, H. A. Bedel, S. Ozturk, A. Gungor, and T. Cukur, "Unsupervised medical image translation with adversarial diffusion models," *IEEE Transactions on Medical Imaging*, 2023.

[124] K. Ward, F. Bertails, T.-Y. Kim, S. R. Marschner, M.-P. Cani, and M. C. Lin, "A survey on hair modeling: Styling, simulation, and rendering," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 2, pp. 213–234, 2007.

[125] B. Nicolet, A. Jacobson, and W. Jakob, "Large steps in inverse rendering of geometry," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 6, pp. 1–13, 2021.

[126] H. F. Shi and S. Payandeh, "Gpu in haptic rendering of deformable objects," in *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications*. Springer, 2008, pp. 163–168.

[127] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, "A practical evaluation of popular volume rendering algorithms," in *Proceedings of the 2000 IEEE symposium on Volume visualization*, 2000, pp. 81–90.

[128] C. M. Wittenbrink, *Survey of parallel volume rendering algorithms*. Hewlett Packard Laboratories, 1998.

[129] G. Patow and X. Pueyo, "A survey of inverse rendering problems," in *Computer graphics forum*, vol. 22, no. 4. Wiley Online Library, 2003, pp. 663–687.

[130] S. R. Marschner, *Inverse rendering for computer graphics*. Cornell University, 1998.

[131] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[132] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–8.

[133] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "Nerf in the wild: Neural radiance fields for unconstrained photo collections," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7210–7219.

[134] 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.

[135] T. Müller, F. Rousselle, J. Novák, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pp. 1–16, 2021.

[136] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5855–5864.

[137] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 318–10 327.

[138] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. Kaplanyan, and M. Steinberger, "Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks," in *Computer Graphics Forum*, vol. 40, no. 4. Wiley Online Library, 2021, pp. 45–59.

[139] S. Avidan and A. Shashua, "Novel view synthesis in tensor space," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 1997, pp. 1034–1040.

[140] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 165–174.

[141] P. Raja and S. Pugazhenthi, "Optimal path planning of mobile robots: A review," *International journal of physical sciences*, vol. 7, no. 9, pp. 1314–1320, 2012.

[142] Y. M. Tang and H. L. Ho, "3d modeling and computer graphics in virtual reality," in *Mixed Reality and Three-Dimensional Computer Graphics*. IntechOpen, 2020.

[143] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[144] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *ACM Siggraph Computer Graphics*, vol. 22, no. 4, pp. 65–74, 1988.

[145] B. Lichtenbelt, R. Crane, and S. Naqvi, *Introduction to volume rendering*. Prentice-Hall, Inc., 1998.

[146] L. Westover, "Interactive volume rendering," in *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, 1989, pp. 9–16.

[147] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 12, pp. 4338–4364, 2020.

[148] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 52–67.

[149] R. Tahir, A. B. Sargano, and Z. Habib, "Voxel-based 3d object reconstruction from single 2d image using variational autoencoders," *Mathematics*, vol. 9, no. 18, p. 2288, 2021.

[150] W. T. Reeves, D. H. Salesin, and R. L. Cook, "Rendering antialiased shadows with depth maps," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 283–291.

[151] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5301–5310.

[152] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7537–7547, 2020.

[153] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," in *European Conference on Computer Vision*. Springer, 2020, pp. 608–625.

[154] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser et al., "Local implicit grid representations for 3d scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6001–6010.

[155] T. Müller, A. Evans, C. Schied, and A. Keller, "Nvlabs - instant-ngp," https://github.com/NVlabs/instant-ngp, 2022.

[156] F. Giesen, "A trip through the graphics pipeline 2011," 2011. [Online]. Available: https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/

[157] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[158] "Nvidia geforce rtx 3090," 2022. [Online]. Available: https://www.techpowerup.com/gpu-specs/geforce-rtx-3090.c3622

[159] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[160] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[161] "Nvidia geforce rtx 3090 specs," 2024, accessed: 2024-06-03. [Online]. Available: https://www.techpowerup.com/gpu-specs/geforce-rtx-3090.c3622

[162] "Nvidia a100 pcie 80 gb specs," 2024, accessed: 2024-06-03. [Online]. Available: https://www.techpowerup.com/gpu-specs/a100-pcie-80-gb.c3821

[163] "Nvidia h100 pcie 80 gb specs," 2024, accessed: 2024-06-03. [Online]. Available: https://www.techpowerup.com/gpu-specs/h100-pcie-80-gb.c3899

[164] C. Zhang, C. Zhang, M. Zhang, and I. S. Kweon, "Text-to-image diffusion model in generative ai: A survey," *arXiv preprint arXiv:2303.07909*, 2023.

[165] S. Parida, V. Srinivas, B. Jain, R. Naik, and N. Rao, "Survey on diverse image inpainting using diffusion models," in *2023 2nd International Conference on Paradigm Shifts in Communications Embedded Systems, Machine Learning and Signal Processing (PCEMS)*. IEEE, 2023, pp. 1–5.

[166] M. N. Everaert, M. Bocchio, S. Arpa, S. Süsstrunk, and R. Achanta, "Diffusion in style," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2251–2261.

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

[168] Google Research, "Imagen," https://imagen.research.google/, 2024, accessed: 18 Apr. 2024.

[169] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8633–8646, 2022.

[170] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[171] L. Pinheiro Cinelli, M. Araújo Marins, E. A. Barros da Silva, and S. Lima Netto, "Variational autoencoder," in *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer, 2021, pp. 111–149.

[172] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "Videoflow: A flow-based generative model for video," *arXiv preprint arXiv:1903.01434*, vol. 2, no. 5, p. 3, 2019.

[173] H. Zou, Z. M. Kim, and D. Kang, "Diffusion models in nlp: A survey," *arXiv preprint arXiv:2305.14671*, 2023.

[174] A. Shmakov, K. Greif, M. Fenton, A. Ghosh, P. Baldi, and D. Whiteson, "End-to-end latent variational diffusion models for inverse problems in high energy physics," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[175] Y. Yuan, J. Song, U. Iqbal, A. Vahdat, and J. Kautz, "Physdiff: Physics-guided human motion diffusion model," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 16 010–16 021.

[176] M. Letafati, S. Ali, and M. Latva-aho, "Diffusion models for wireless communications," *arXiv e-prints*, pp. arXiv–2310, 2023.

[177] Everypixel, "Ai image statistics," https://journal.everypixel.com/ai-image-statistics, 2024, accessed: 18 Apr. 2024.

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

[179] Facebook, "Help center: Business help," https://www.facebook.com/business/help/469767027114079?id=271710926837064, 2024, accessed: 18 Apr. 2024.

[180] CompVis, "Github repository: Latent diffusion," https://github.com/CompVis/latent-diffusion, 2024, accessed: 18 Apr. 2024.

[181] U.S. Energy Information Administration, "Faq: How much carbon dioxide is produced per kilowatthour of u.s. electricity generation?" https://www.eia.gov/tools/faqs/faq.php?id=97&t=3, 2024, accessed: 18 Apr. 2024.

[182] Blueprint Digital, "The future of 8k resolution," https://blueprintdigital.com/8k-future-resolution/, 2024, accessed: 18 Apr. 2024.

[183] H. Face, "Introduction to diffusion models," https://huggingface.co/course/chapter7/6?fw=pt, accessed: 2024-05-27.

[184] SuperAnnotate, "Introduction to diffusion models for machine learning," https://www.superannotate.com/, accessed: 2024-05-27.

[185] A. AI, "Introduction to diffusion models for machine learning," https://www.assemblyai.com/, accessed: 2024-05-27.

[186] S. Elflein, "A practical guide to diffusion models," https://selflein.github.io/, accessed: 2024-05-27.

[187] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv:2010.02502*, October 2020. [Online]. Available: https://arxiv.org/abs/2010.02502

[188] Stability AI, "Home page," https://stability.ai/, 2024, accessed: 18 Apr. 2024.

[189] OpenAI, "Clip: Contrastive language-image pre-training," https://openai.com/research/clip, 2024, accessed: 18 Apr. 2024.

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

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

[192] Hugging Face, "Stable diffusion," https://huggingface.co/CompVis/stable-diffusion, 2024, accessed: 18 Apr. 2024.

[193] NVIDIA, "Tensorrt," https://developer.nvidia.com/tensorrt, 2024, accessed: 18 Apr. 2024.

[194] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[195] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of gpu algorithms for matrix-matrix multiplication," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 133–137.

[196] NVIDIA, "Nsight compute," https://developer.nvidia.com/nsight-compute, 2024, accessed: 18 Apr. 2024.

[197] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019, pp. 11 967–11 976.

[198] PyTorch, "Documentation for register_module_forward_hook," https://pytorch.org/docs/stable/generated/torch.nn.modules.module.register_module_forward_hook.html, 2024, accessed: 18 Apr. 2024.

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

[200] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[201] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, "Sparse-tpu: Adapting systolic arrays for sparse matrices," in *Proceedings of the 34th ACM international conference on supercomputing*, 2020, pp. 1–12.

[202] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[203] Synopsys, "Dc ultra: Rtl synthesis and test," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, 2024, accessed: 18 Apr. 2024.

[204] Siemens EDA, "Modelsim," https://eda.sw.siemens.com/en-US/ic/modelsim/, 2024, accessed: 18 Apr. 2024.

[205] L. Valente, A. Conci, and B. Feijó, "Real time game loop models for single-player computer games," in *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment*, vol. 89. Citeseer, 2005, p. 99.

[206] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo, "Deepfovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–13, 2019.

[207] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3d graphics," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, pp. 1–10, 2012.

[208] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, "Towards foveated rendering for gaze-tracked virtual reality," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016.

[209] A. Watson, "Deep learning techniques for super-resolution in video games," *arXiv preprint arXiv:2012.09810*, 2020.

[210] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé, and N. Thuerey, "Learning temporal coherence via self-supervision for gan-based video generation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 75–1, 2020.

[211] X. Wang, K. C. Chan, K. Yu, C. Dong, and C. Change Loy, "Edvr: Video restoration with enhanced deformable convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[212] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 286–301.

[213] H. Zhao, X. Kong, J. He, Y. Qiao, and C. Dong, "Efficient image super-resolution using pixel attention," 2020.

[214] "NVIDIA DLSS," https://www.nvidia.com/en-us/geforce/technologies/dlss/.

[215] L. Xiao, S. Nouri, M. Chapman, A. Fix, D. Lanman, and A. Kaplanyan, "Neural supersampling for real-time rendering," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 142–1, 2020.

[216] "AMD FSR," https://www.amd.com/en/technologies/radeon-software-fidelityfx-super-resolution.

[217] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[218] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang, and L. Zhang, "Image super-resolution: The techniques, applications, and future," *Signal processing*, vol. 128, pp. 389–408, 2016.

[219] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: Qoe and the users' perspective," *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2883–2894, 2013, information System Security and Performance Modeling and Simulation for Future Mobile Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895717711007771

[220] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of qoe in cloud gaming based on subjective tests," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2011, pp. 330–335.

[221] M. Carrascosa and B. Bellalta, "Cloud-gaming: Analysis of google stadia traffic," *arXiv preprint arXiv:2009.09786*, 2020.

[222] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[223] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[224] A. Shenoy, "Image super-resolution," https://github.com/amanshenoy/image-super-resolution, 2024.

[225] "TensorRT," https://developer.nvidia.com/tensorrt.

[226] PyTorch, "Introduction to quantization on pytorch," 2021. [Online]. Available: https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

[227] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *European conference on computer vision*. Springer, 2016, pp. 391–407.

[228] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[229] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang et al., "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[230] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 136–144.

[231] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[232] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2472–2481.

[233] J. Caballero, C. Ledig, A. Aitken, A. Acosta, J. Totz, Z. Wang, and W. Shi, "Real-time video super-resolution with spatio-temporal networks and motion compensation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4778–4787.

[234] M. Haris, G. Shakhnarovich, and N. Ukita, "Recurrent back-projection network for video super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3897–3906.

[235] Y. Jo, S. Wug Oh, J. Kang, and S. Joo Kim, "Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3224–3232.

[236] A. Kappeler, S. Yoo, Q. Dai, and A. K. Katsaggelos, "Video super-resolution with convolutional neural networks," *IEEE Transactions on Computational Imaging*, vol. 2, no. 2, pp. 109–122, 2016.

[237] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, and T. Huang, "Robust video super-resolution with learned temporal dynamics," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2507–2515.

[238] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, X. Wang, and T. S. Huang, "Learning temporal dynamics for video super-resolution: A deep learning approach," *IEEE Transactions on Image Processing*, vol. 27, no. 7, pp. 3432–3445, 2018.

[239] M. S. Sajjadi, R. Vemulapalli, and M. Brown, "Frame-recurrent video super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6626–6634.

[240] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia, "Detail-revealing deep video super-resolution," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4472–4480.

[241] E. Carles Ramon, "Application of deep learning upscaling technologies in cloud gaming solutions," B.S. thesis, Universitat Politècnica de Catalunya, 2022.

[242] H. Long, Y. Yang, C. Liu, and J. Jia, "Srssis: Super-resolution screen space irradiance sampling for lightweight collaborative web3d rendering architecture," in *International Conference on Computer-Aided Design and Computer Graphics*. Springer, 2023, pp. 295–313.

[243] S. Kushchi, Y. Kfir, and S. Weiss, "Cloud computing through limited bandwidth inflight airplane wifi communication," in *2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*. IEEE, 2014, pp. 1–4.

[244] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient fpga-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, pp. 281–295, 2020.

[245] M.-C. Yang, K.-L. Liu, and S.-Y. Chien, "A real-time fhd learning-based super-resolution system without a frame buffer," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 12, pp. 1407–1411, 2017.

[246] K. Sun, M. Koch, Z. Wang, S. Jovanovic, H. Rabah, and S. Simon, "An fpga-based residual recurrent neural network for real-time video super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2021.

[247] J. Lee, D. Shin, J. Lee, J. Lee, S. Kang, and H.-J. Yoo, "A full hd 60 fps cnn super resolution processor with selective caching based layer fusion for mobile devices," in *2019 Symposium on VLSI Circuits*, 2019, pp. C302–C303.

[248] J. Lee and I.-C. Park, "High-performance low-area video up-scaling architecture for 4-k uhd video," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 4, pp. 437–441, 2017.

[249] Y. Kim, J.-S. Choi, and M. Kim, "2x super-resolution hardware using edge-orientation-based linear mapping for real-time 4k uhd 60 fps video applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 9, pp. 1274–1278, 2018.