



uOttawa

AI-ASSISTED SUPER-RESOLUTION AND RESOURCE OPTIMIZATION IN CLOUD GAMING

By

Haseeb Ur Rehman

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY in ELECTRICAL AND COMPUTER ENGINEERING

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Haseeb Ur Rehman, Ottawa, Canada, 2026

Declaration

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Cloud gaming (CG) has transformed the gaming industry by enabling users to remotely access resource-intensive games on lightweight devices. However, delivering a seamless gaming experience remains challenging due to stringent bandwidth requirements, at least 25 Mbps for 1080p streaming [1], and strict latency tolerances that depend on game genre, approximately 80 ms for fast-paced first-person games and 100–150 ms for slower third-person and story-driven titles [2, 3]. This thesis addresses these challenges through the systematic development of deep learning-based super-resolution (SR) solutions integrated within the CG pipeline to reduce bandwidth consumption while preserving visual quality and interactivity.

We present four interconnected contributions. First, we propose **GameSR**, a lightweight neural super-resolution model that operates directly on encoded game frames without requiring game engine integration or source code access. By combining reparameterized convolutional blocks with a lightweight ConvLSTM for temporal learning, GameSR achieves real-time performance of up to 240 FPS on a GPU-accelerated client at comparable perceptual quality to native streaming.

Second, we develop **ARCADE**, an adaptive cloud gaming framework that jointly optimizes rendering resolution, encoding bitrate, and client-side super-resolution through offline reinforcement learning. By rendering at a lower resolution when scene complexity permits, ARCADE saves server-side resources on two fronts simultaneously: it reduces the GPU cost of *rendering* the frame and the cost of *compressing* (encoding) it, while the lower bitrate cuts transmission bandwidth. This yields reductions in server CPU and GPU utilization of up to 62% and 41% respectively, alongside up to 50% bandwidth savings compared to standard WebRTC streaming at equivalent or higher visual quality.

Third, we extend our approach to immersive virtual reality cloud gaming. We demonstrate that transmitting one color view and one monochrome view reduces VR streaming bandwidth by up to 56% (**iSR**), and address the resulting sequential pipeline bottleneck by developing **GameSRVR**, a unified model that jointly performs stereo-aware colorization and super-resolution in a single forward pass. GameSRVR reduces inference from 73.61 ms to ~ 11 ms per stereo pair at $4\times$ scaling, a $6.7\times$ speedup. Deployed on a cloud gaming testbed, GameSRVR achieves over 33% bandwidth savings and an end-to-end latency of ~ 20 ms, meeting the VR motion-to-photon budget.

Fourth, to enable reproducible AI-driven cloud gaming research, we introduce **GameLab**, an open-source, AI-enabled cloud gaming testbed built on WebRTC. GameLab provides programmable server-side and client-side hook points for integrating machine learning modules, along with a novel QR-based frame identity mechanism enabling reliable full-reference quality evaluation under real network conditions.

Extensive objective evaluations on popular games including Counter-Strike 2, Overwatch 2, Team Fortress 2, and FIFA 24, as well as VR titles Beat Saber, Pavlov VR, and VRChat, alongside subjective user studies, validate that our solutions deliver high perceptual quality while meeting real-time constraints. This thesis demonstrates that effective cloud-gaming neural upsampling can be achieved without proprietary game engine integration, offering practical solutions deployable across both legacy and modern game titles.

Acknowledgements

I am deeply thankful to the University of Ottawa for supporting my doctoral studies through the International Doctoral Scholarship and Admission Scholarship. I also acknowledge the research funding provided by Advanced Micro Devices (AMD) through the Natural Sciences and Engineering Research Council of Canada (NSERC), grant ALLRP556311-20. This support made the work in this thesis possible.

I am especially grateful to my supervisor, Prof. Shervin Shirmohammadi, for his continuous guidance, mentorship, and encouragement throughout my PhD. I would also like to sincerely thank Prof. Mohamed Hefeeda of Simon Fraser University for his support and direction during this journey. My thanks also go to Prof. Ihab Amer for his guidance during my time with AMD, and to Dr. Khaled Diab for his support in the early phase of this project.

Finally, I am profoundly thankful to my family, and especially to my wife and partner, for their unwavering support, patience, and strength throughout this journey.

And above all, all praise be to God.

Contents

List of Abbreviations	xv
1 Introduction	1
1.1 Background and Motivation	1
1.2 Research Problem	2
1.2.1 The Bandwidth-Quality Trade-off	3
1.2.2 The Latency Constraint	3
1.2.3 The Server-Side Computational Burden	3
1.2.4 The VR Extension Challenge	4
1.2.5 The Research Infrastructure Gap	4
1.3 Thesis Statement and Scope	5
1.4 Research Questions and Objectives	6
1.5 Overview of the Proposed Solutions	7
1.5.1 GameSR: Real-Time, Engine-Agnostic Super-Resolution	7
1.5.2 ARCADE: Complexity-Aware Joint Optimization of the Cloud Gaming Pipeline	7
1.5.3 GameSRVR: Joint Colorization and Super-Resolution for Cloud VR	8
1.5.4 GameLab: Reproducible Research Infrastructure for AI-Driven Cloud Gaming	8
1.6 Thesis Contributions	8
1.7 Practical Applications	9
1.8 Results Dissemination	10
1.8.1 Publications	10
1.8.2 Data and Code Sharing	10
1.9 Thesis Structure	10
2 Background and Related Work	12
2.1 Introduction	12
2.2 Deep Learning Fundamentals	12
2.2.1 Convolutional Neural Networks	13
2.2.2 Recurrent Neural Networks and Temporal Modeling	14
2.3 Super Resolution	16
2.3.1 Problem Formulation	16
2.3.2 Classical Approaches	16
2.3.3 Deep Learning-Based Super Resolution	17
2.4 Video Super Resolution	19
2.4.1 Problem Formulation	20

2.4.2	Temporal Alignment Strategies	20
2.4.3	Temporal Consistency	21
2.4.4	Relevance to Cloud Gaming	21
2.5	Super Resolution for Gaming Content	21
2.5.1	Commercial Game Upscalers	22
2.6	Content Complexity Analysis	23
2.7	Cloud Gaming	24
2.7.1	System Architecture	24
2.7.2	End-to-End Latency Pipeline	25
2.7.3	Video Encoding for Cloud Gaming	25
2.7.4	Technical Challenges	26
2.7.5	Commercial Platforms	27
2.8	Reinforcement Learning for Systems Control	28
2.8.1	Markov Decision Process Formulation	28
2.8.2	Value Functions and Q-Learning	28
2.8.3	Offline Reinforcement Learning	29
2.9	Quality Metrics for Cloud Gaming	29
2.9.1	Full-Reference Objective Metrics	29
2.9.2	No-Reference Metrics	30
2.9.3	Subjective Evaluation	31
2.10	WebRTC for Real-Time Streaming	31
2.10.1	Key Components	31
2.10.2	Application to Cloud Gaming	32
2.11	Summary and Research Gaps	33
3	GameSR: Real-Time Super-Resolution for Interactive Gaming	34
3.1	Introduction	34
3.2	Problem Definition and Design Goals	35
3.2.1	Limitations of Existing Approaches	35
3.2.2	Design Goals	36
3.2.3	Exploiting Gaming Content Characteristics	37
3.3	GameSR Architecture	37
3.3.1	Overview and Operation	37
3.3.2	Model Formulation	38
3.3.3	Feature Representation Stage	38
3.3.4	Feature Extraction Blocks	39
3.3.5	Temporal Learning with ConvLSTM	41
3.3.6	Upsampling Stage	42
3.3.7	Mobile Variant: GameSR-M	42
3.4	Training Methodology	42
3.4.1	Dataset Collection	42
3.4.2	Loss Function	43
3.4.3	Training Configuration	43
3.5	Evaluation	44
3.5.1	Performance Metrics	44
3.5.2	Comparison with State-of-the-Art SR Models	44
3.5.3	Comparison with Commercial Upscalers	45
3.5.4	Model Efficiency Analysis	46

3.5.5	Ablation Study	46
3.5.6	Cross-Game Generalization	47
3.5.7	High-Resolution Scaling	47
3.5.8	User Study	48
3.5.9	Hardware Scalability	49
3.6	End-to-End Cloud Gaming Evaluation	49
3.6.1	Bandwidth Savings	49
3.6.2	Bitrate–Quality Analysis Across Scales and Resolutions	50
3.6.3	End-to-End Latency	50
3.6.4	Processing and Power Savings	50
3.7	Summary	51
4	ARCADE: Joint Optimization of Bandwidth and Computing Resources	57
4.1	Introduction	57
4.2	Problem Analysis and Motivation	58
4.2.1	Challenges in Cloud Gaming Optimization	58
4.2.2	Limitations of Current Approaches	58
4.2.3	Underutilized Client Resources	59
4.2.4	Relationship to Other Video Streaming Systems	59
4.3	The ARCADE Approach	60
4.4	ARCADE System Architecture	60
4.4.1	System Overview	60
4.5	Joint Bandwidth-Compute Optimization	61
4.5.1	Reinforcement Learning Formulation	61
4.5.2	State Space	62
4.5.3	Action Space	62
4.5.4	Reward Function	62
4.6	Batch-Constrained Q-Learning	63
4.6.1	Training Details	64
4.7	Content Complexity Analysis	64
4.7.1	Complexity Prediction	64
4.8	System Integration	65
4.9	SR Backend Integration	66
4.10	Implementation	66
4.10.1	WebRTC Testbed	66
4.10.2	Baseline Configuration	66
4.10.3	Games and Dataset	67
4.11	Evaluation	67
4.11.1	Experimental Setup	67
4.11.2	Meeting Target Quality	67
4.11.3	Bandwidth Savings	68
4.11.4	Low-Bandwidth Scenarios	69
4.11.5	Dynamic Network Adaptation	69
4.11.6	End-to-End Latency	70
4.12	System Ablation	70
4.12.1	No-JO Variant	71
4.12.2	No-SR Variant	72
4.13	Compute and Power Savings	72

4.14	User Study	73
4.15	Generality to Different SR Backends	75
4.16	Practical Considerations and Limitations	76
4.17	Summary	77
5	GameSRVR: Joint Super-Resolution and Colorization for Cloud VR Gam-	78
	ing	
5.1	Introduction	78
5.1.1	Measurement Study	79
5.2	VR System Requirements and Limitations of Existing Approaches . . .	79
5.3	Design Goals and Key Insight	80
5.3.1	Design Goals	80
5.3.2	Key Insight: Stereo Redundancy	80
5.3.3	Potential Bandwidth Savings	81
5.4	iSR: Motivating Study	81
5.4.1	System Overview	81
5.4.2	Key Results	82
5.4.3	Limitation: Sequential Pipeline Latency	82
5.5	GameSRVR: Joint Colorization and Super-Resolution	83
5.5.1	FastSICModule: Lightweight Stereo Feature Extraction	83
5.5.2	FastColorization: Half-Resolution Attention	84
5.5.3	GameSR-Based Upsampler	85
5.5.4	Joint Training Strategy	85
5.6	Experimental Setup	86
5.6.1	VR Gaming Dataset	86
5.6.2	Experimental Configuration	86
5.7	Evaluation	86
5.7.1	Quality Comparison: GameSRVR vs. iSR	86
5.7.2	Rate-Distortion Analysis	87
5.7.3	End-to-End Latency	87
5.7.4	User Study	88
5.7.5	Comparison with State-of-the-Art	89
5.7.6	Perceptual Color Difference	90
5.7.7	Colorization and Upscaling Trade-off	90
5.8	Summary	91
6	GameLab: AI-Enabled Cloud Gaming Research Testbed	94
6.1	Introduction	94
6.2	Related Work and Positioning	95
6.3	System Architecture	96
6.3.1	Server-Side Pipeline	96
6.3.2	Client-Side Pipeline	97
6.4	Configuring and Running GameLab	98
6.5	Demonstrations	99
6.5.1	Swappable Video Upsamplers	99
6.5.2	Client-Side Analytics	100
6.5.3	Complexity-Driven Rate Control	100

6.5.4	Transport Logging	101
6.6	Reproducibility and Overheads	101
6.7	Summary	102
7	Conclusion and Future Work	103
7.1	Summary of Contributions	103
7.1.1	GameSR: Real-Time Neural Upsampling	103
7.1.2	ARCADE: Joint Optimization Framework	103
7.1.3	GameSRVR: Joint Super-Resolution and Colorization for Cloud VR	103
7.1.4	GameLab: Research Infrastructure	104
7.2	Addressing the Research Questions	104
7.3	Key Findings and Insights	104
7.4	Practical Applications	105
7.5	Limitations	105
7.6	Future Research Directions	106
7.6.1	Cross-Game and Zero-Shot Generalization	106
7.6.2	Online Reinforcement Learning	107
7.6.3	Joint Neural Codec and Super-Resolution	107
7.6.4	Extended Reality and Emerging Displays	109
7.6.5	Mobile Cloud Gaming	110
7.6.6	Perceptual Quality and User Experience	111
7.7	Concluding Remarks	111

List of Figures

2.1	ESPCN architecture: feature extraction in LR space followed by sub-pixel convolution for upsampling, enabling efficient real-time super-resolution.	18
2.2	High-level architecture of a cloud gaming system, showing the server-side rendering and encoding pipeline, network transport layer, and client-side decoding and display chain.	24
3.1	GameSR in cloud gaming: low-resolution streams are rendered at server-side and upscaled at client-side in real-time.	38
3.2	Overview of GameSR: (a) architecture where the current low-resolution input frame (the leftmost image, frame t) is downsampled with PixelUnshuffle, processed by Feature Extraction Blocks, and passed through a lightweight ConvLSTM before upsampling via PixelShuffle with a residual connection. The “Temporal Data” block $h(t-1), \dots, h(t-n)$ holds the ConvLSTM hidden states carried over from the preceding frames of the same gameplay sequence; the ConvLSTM fuses this temporal context with the features of the current (leftmost) frame to produce a motion-aware, temporally consistent reconstruction, and emits the updated state $h(t)$ for the next frame. (b) internal structure of the Feature Extraction Block (FEB), in its multi-branch training form and reparameterized single-path inference form; here Act denotes the GELU activation and Norm the LayerNorm normalization (defined in Section 3.3.4, “Activation and Normalization”).	53
3.3	Performance of GameSR on upscaling game sessions from CS2 and OW2 by a factor of $2\times$	54
3.4	Visualization of upsampling methods on Overwatch 2 frames: each row corresponds to a different map (Row 1: Junktown, Row 2: Esperança, Row 3: Nepal). Columns show DLSS, FSR, and GameSR, respectively.	54
3.5	Efficiency of GameSR compared to state-of-the-art methods for $2\times$ scaling: (a) parameter count, (b) GPU memory usage, (c) GPU utilization comparison against DLSS, FSR, and native rendering.	54
3.6	Inference latency vs. device compute capability (TFLOPS) for GameSR at $2\times$ scaling.	55
3.7	VMAF vs. bitrate for CS2 and OW2. GameSR achieves target quality levels at substantially lower bitrates.	55
3.8	End-to-end latency breakdown across bitrates (8/16/24 Mbps). GameSR adds 4–5 ms for SR but reduces overall latency by 19% through smaller frame sizes.	55

3.9	Server-side GPU and CPU utilization: baseline WebRTC streaming at 1080p versus GameSR-enabled streaming at 540p.	56
4.1	Overview of cloud gaming and its challenges.	58
4.2	Overview of ARCADE, which jointly optimizes rendering resolution, encoding bitrate, and client-side super-resolution in cloud gaming. . . .	60
4.3	Joint Optimization Module (JOM) in ARCADE.	61
4.4	Achieved VMAF for baseline WebRTC (OW2) at 30 Mbps across five client locations (Brazil, Chicago, London, Singapore, and Toronto); the shaded region is the mean $\pm$ standard deviation across 50 sessions. . . .	68
4.5	Achieved VMAF for ARCADE (OW2) at 30 Mbps across the same five client locations; ARCADE maintains the VMAF 90 target more consistently than WebRTC, especially for clients farther from the server. . . .	69
4.6	Normalized consumed bandwidth for WebRTC and ARCADE (OW2) at 30 Mbps across five locations: Brazil, Chicago, London, Singapore, and Toronto.	69
4.7	Achieved VMAF for baseline WebRTC (OW2) under the 8.5 Mbps bandwidth constraint, across five client locations. WebRTC quality is unstable, with frequent drops as low as 40.	70
4.8	Achieved VMAF for ARCADE (OW2) under the 8.5 Mbps bandwidth constraint, across five client locations. ARCADE maintains average VMAF close to 80, with some low-complexity GOPs exceeding 90. . . .	71
4.9	RTT distributions for CS2 under the 30 Mbps constraint, for WebRTC and ARCADE.	71
4.10	Dynamic bitrate adaptation under simulated congestion and delay. The red curve shows bitrate allowed by GCC; blue shows bitrate selected by ARCADE.	72
4.11	End-to-end latency breakdown for baseline WebRTC and ARCADE. . . .	72
4.12	Quality and bandwidth for baseline WebRTC and ARCADE–NoSR on OW2 with unconstrained GCC-driven bandwidth (a,b), and rate–distortion results for GameSR at different bitrates and scaling factors (c,d).	73
4.13	Comparison of normalized CPU and GPU utilization between baseline WebRTC and ARCADE.	74
4.14	Generality of ARCADE to different SR backends. We instantiate ARCADE with bicubic, RT4KSR, and GameSR at (a) $3\times$ and (b) $2\times$ upscaling. The controller and pipeline are unchanged; differences in curves arise solely from each upscaler’s latency–quality trade-off. . . .	76
5.1	Experimental setup for measuring bandwidth usage in cloud VR gaming. . . .	80
5.2	Bandwidth usage distribution for three VR games.	81
5.3	Overview of the iSR system for reducing required bandwidth in VR cloud gaming platforms. The left view is transmitted in monochrome and reconstructed to full color on the client, while the right view retains color throughout the pipeline.	82
5.4	Architecture of the lightweight colorization module used in iSR, derived from SiCNet and optimized for interactive VR cloud gaming.	82

5.5	GameSRVR pipeline. The left (monochrome) and right (color) LR views are processed jointly: the FastSICModule extracts multi-scale stereo features, FastColorization transfers color from right to left at half resolution, and the GameSR-based upsampler super-resolves both views to HR. The right view bypasses colorization and is directly super-resolved.	84
5.6	Rate–distortion comparison of iSR for $4\times$ upscaling against the baseline across different bitrates.	87
5.7	Rate–distortion comparison of iSR for $8\times$ upscaling against the baseline across different bitrates.	87
5.8	End-to-end latency breakdown across bitrates for WebRTC and iSR.	88
5.9	End-to-end latency breakdown across bitrates for WebRTC and GameSRVR (2K stereo streaming, GameLab).	88
5.10	Visual quality produced by iSR with $8\times$ bandwidth reduction on sample left-eye frames from Beat Saber, Pavlov VR, and VRChat. The upscaled frames (top) are nearly indistinguishable from the ground truth (bottom).	92
5.11	Cumulative distribution of ΔE_{2000} across all scales (from iSR evaluation).	93
6.1	Design of GameLab. The testbed provides two Python hook points: (i) server <i>pre-encode</i> for frame analysis/modification and QR frame IDs, and (ii) client <i>post-decode</i> for pluggable AI modules such as SR and object detection. Detailed logs enable offline and online analyses.	96
6.2	CS2 bandwidth sweep: SSIM/VMAF vs. achieved bitrate for three pipelines. Metrics use QR-aligned frames.	99
6.3	Client-side object detection as a post-decode plug-in. Annotated frames from CS2 and OW2 showing detector outputs computed after WebRTC decoding.	100
6.4	Time-series behavior of EVCA-driven rate control: (left) estimated spatial complexity, and (right) corresponding target bitrate from the ladder.	101
6.5	RTT and jitter traces logged from WebRTC transport statistics during CS2 bandwidth-sweep runs.	101
7.1	Proposed joint neural codec and super-resolution architecture. The server-side encoder receives low-resolution rendered frames and produces a compact latent representation $z_0, z_1, z_2, \dots$ that is transmitted to the client. The client-side decoder splits into a decompression branch (green) and a super-resolution branch (blue), whose outputs are fused to produce the reconstructed high-resolution frame. Shared early layers (yellow) enable feature reuse across both tasks.	108

List of Tables

2.1	Minimum bandwidth requirements of major cloud gaming platforms. . .	28
3.1	Feature extraction block architectural exploration on CS2. All variants use 16 base filters.	40
3.2	Quantitative comparison between state-of-the-art super-resolution models and GameSR at $2\times$ scaling on four popular games. Evaluated on NVIDIA RTX A4000 GPU.	44
3.3	Comparison of GameSR against IMDN and IMDN-G (IMDN retrained on CS2), isolating the contribution of game-specific training.	45
3.4	Quantitative comparison at $3\times$ scaling on four popular games. Evaluated on NVIDIA RTX A4000 GPU.	45
3.5	Comparing GameSR against DLSS and FSR, which require engine-level data. In contrast, GameSR upscales encoded streams. Results shown for $2\times$ scaling.	46
3.6	Ablation study of GameSR on CS2, showing the impact of ConvLSTM, PixelUnshuffle, and Reparameterization on efficiency and quality. . . .	46
3.7	Cross-game generalization of GameSR. Models trained on CS2, OW2, and a combined dataset are evaluated on both games.	47
3.8	Super-resolution performance of GameSR on high-resolution clips. All experiments conducted on NVIDIA A4000 GPU.	47
3.9	Rendered condition MOS for GameSR across participant groups. Exp.=Experienced, Occ.=Occasional, No.=Non-gamers.	48
3.10	Paired cloud gaming comparison (25 Mbps): native 1080p WebRTC vs. GameSR MOS scores with 95% CI. Δ MOS = GameSR – WebRTC. TOST equivalence margin: ± 0.5 MOS.	48
4.1	User study results: visual quality MOS (mean $\pm$ standard deviation across participants) achieved by ARCADE and WebRTC for CS2 and OW2.	74
5.1	Potential bandwidth savings from sending chrominance components of only one view for various chroma subsampling formats.	81
5.2	Perceptual quality of iSR across three VR games and all scaling factors.	83
5.3	iSR inference time across scaling factors. Only $\times 8$ and $\times 10$ meet the 20 ms VR budget, but at significant quality cost.	83
5.4	Inference time and memory usage of iSR across scaling factors.	84
5.5	Quality and inference comparison between iSR (two-stage sequential) and GameSRVR (joint model) at $4\times$ scaling. GameSRVR reduces inference from 73.61 ms to ~ 11 ms while maintaining comparable quality.	86
5.6	MOS for iSR across all VR games and scale factors.	89

5.7	Colorization results ($\times 4$ upscaling).	89
5.8	Impact of cumulative SiCNet modifications across all scales.	90
5.9	Impact of feature channel width across all scales. Bold indicates the selected configuration per scale.	91
5.10	Super-resolution comparison at $4\times$ and results across all scales.	92

List of Abbreviations

API	Application Programming Interface
BasicVSR	Basic Video Super-Resolution
BCQ	Batch-Constrained Q-Learning
BN	Batch Normalization
BPTT	Backpropagation Through Time
CAGR	Compound Annual Growth Rate
CIELAB	CIE L*a*b* color space
CDN	Content Delivery Network
CG	Cloud Gaming
CNN	Convolutional Neural Network
ConvLSTM	Convolutional Long Short-Term Memory
CPU	Central Processing Unit
CQL	Conservative Q-Learning
CUDA	Compute Unified Device Architecture
DASH	Dynamic Adaptive Streaming over HTTP
DCT	Discrete Cosine Transform
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DLSS	Deep Learning Super Sampling
DNN	Deep Neural Network
DQN	Deep Q-Network
DTLS	Datagram Transport Layer Security
EDSR	Enhanced Deep Super-Resolution
EDVR	Enhanced Deformable Video Restoration
ERP	Equirectangular Projection
ESA	Enhanced Spatial Attention
EVCA	Efficient Video Complexity Analyzer
FEB	Feature Extraction Block
FLOPs	Floating-Point Operations
FOV	Field of View
FPS	Frames Per Second
FSR	FidelityFX Super Resolution
GAN	Generative Adversarial Network
GCC	Google Congestion Control
GELU	Gaussian Error Linear Unit
GOP	Group of Pictures
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit

GT	Ground Truth
H.264	Advanced Video Coding (AVC)
H.265	High Efficiency Video Coding (HEVC)
HMD	Head-Mounted Display
HR	High Resolution
HUD	Heads-Up Display
ICE	Interactive Connectivity Establishment
IMDN	Information Multi-Distillation Network
LPIPS	Learned Perceptual Image Patch Similarity
LR	Low Resolution
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MOS	Mean Opinion Score
MSE	Mean Squared Error
NACK	Negative Acknowledgment
NPU	Neural Processing Unit
NVENC	NVIDIA Video Encoder
PLI	Picture Loss Indication
PReLU	Parametric Rectified Linear Unit
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
QP	Quantization Parameter
QR	Quick Response (Code)
ReLU	Rectified Linear Unit
ResBlock	Residual Block
RGB	Red, Green, Blue
RIR	Residual in Residual
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RRB	Reparameterizable Residual Block
RTCP	Real-Time Control Protocol
RTP	Real-Time Transport Protocol
RTT	Round-Trip Time
SBS	Side-by-Side
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SISR	Single-Image Super-Resolution
SoC	System on Chip
SOTA	State-of-the-Art
SR	Super Resolution
SRTP	Secure Real-Time Transport Protocol
SSIM	Structural Similarity Index Measure
STUN	Session Traversal Utilities for NAT
SwinIR	Swin Transformer for Image Restoration
TAA	Temporal Anti-Aliasing

TCP	Transmission Control Protocol
TD3	Twin Delayed Deep Deterministic Policy Gradient
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
VCA	Video Complexity Analyzer
VMAF	Video Multi-Method Assessment Fusion
VR	Virtual Reality
VoD	Video on Demand
VRT	Video Restoration Transformer
VSR	Video Super Resolution
WebRTC	Web Real-Time Communication
XeSS	Xe Super Sampling
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

1.1 Background and Motivation

Cloud gaming (CG) has emerged as a transformative paradigm in the gaming industry, enabling users to access resource-intensive games on lightweight devices by offloading rendering to remote servers. The global video game market generated revenues exceeding USD 184 billion in 2024 [4], and cloud gaming is among its fastest-growing segments, projected to expand at a compound annual growth rate of over 40% through 2030 [4]. This rapid growth is driven by the flexibility to play anywhere and the ability to access high-fidelity gaming experiences without expensive local hardware. Major technology companies have invested heavily in cloud gaming platforms, including Microsoft Xbox Cloud Gaming [5], NVIDIA GeForce NOW [6], Sony PlayStation Plus Premium [7], and Amazon Luna [8].

Despite its promise, delivering a seamless cloud gaming experience presents significant technical challenges. Unlike traditional video-on-demand streaming, cloud gaming content exhibits high motion intensity, complex animations, and requires low-latency compression profiles to preserve interactivity. These characteristics result in substantially higher bandwidth demands compared to conventional streaming services. For instance, streaming 1080p content on Netflix requires approximately 5 Mbps [9], whereas NVIDIA GeForce NOW demands at least 25 Mbps for the same resolution [1]. This translates to data consumption of roughly 3 GB/hour for Netflix compared to 12.6 GB/hour for cloud gaming platforms [10, 11].

The computational demands are equally challenging. High-resolution gaming at 4K and frame rates exceeding 120 fps places enormous processing loads on server-side GPUs. Modern GPUs such as the NVIDIA RTX 3080 Ti draw approximately 350 W under load [12], and complete gaming systems can consume 325–380 W at ultra settings [13], resulting in annual energy consumption of up to 1,400 kWh per system [14]. For cloud gaming providers operating large server farms, these computational and energy costs directly impact scalability and profitability.

Latency presents an additional constraint that distinguishes cloud gaming from other streaming applications. Prior studies have established that first-person shooter games tolerate a maximum end-to-end latency of approximately 80 ms before player performance degrades noticeably [2], while every additional 100 ms of delay can reduce third-person game performance by up to 25% [2]. This latency budget must accommodate client input capture, network transmission, server-side rendering and encoding, return transmission, and client-side decoding and display, leaving minimal room for additional processing

overhead.

It is important to distinguish latency from frame rate, as the two govern different aspects of the player experience and are frequently conflated. Frame rate determines the temporal smoothness of rendered motion, whereas latency (responsiveness) determines how quickly the displayed image reacts to player input. For interactive cloud gaming, latency is the more critical and distinguishing constraint: once a baseline frame rate sufficient for smooth motion is met (typically 60 fps for 2D gaming), reductions in end-to-end latency yield larger improvements in perceived playability and measured player performance than further increases in frame rate, particularly for latency-sensitive genres such as first-person shooters [2]. By contrast, conventional video-on-demand streaming is largely insensitive to latency because playback can be buffered, which is why cloud gaming is fundamentally harder to deliver than ordinary video. The two metrics are nonetheless coupled, since the target frame rate defines the per-frame processing budget: at 60 fps each frame must be produced and displayed within roughly 16.7 ms, which in turn bounds the time available for encoding, transmission, and any client-side enhancement. This thesis therefore treats a sustained interactive frame rate (60 fps for 2D and 90 fps for VR) as a hard real-time requirement, while minimizing the end-to-end latency that most directly constrains the solution space and the player experience.

The emergence of cloud Virtual Reality (VR) gaming further intensifies these challenges. VR head-mounted displays (HMDs) require high-resolution stereoscopic content at sustained frame rates exceeding 90 fps to avoid simulator sickness, placing significantly greater demands on both server-side rendering and network transmission compared to conventional flat-screen gaming [15]. Stereo video inherently doubles the data that must be transmitted, compounding the bandwidth constraints already present in standard cloud gaming. At the same time, end-to-end latency budgets for VR are considerably stricter: motion-to-photon latency must remain below 20 ms to prevent perceptual discomfort [15], leaving even less headroom for additional processing in the streaming pipeline. These amplified constraints make VR an important and largely unaddressed frontier for bandwidth-efficient cloud gaming research.

These interrelated challenges of bandwidth consumption, computational load, and latency sensitivity motivate the research presented in this thesis. We address them through two complementary strategies: (i) lightweight deep learning-based super-resolution that enables streaming at lower resolutions with client-side upscaling, reducing bandwidth requirements while preserving visual quality, and (ii) adaptive joint optimization of rendering resolution, encoding bitrate, and server-side resources using offline reinforcement learning, which reduces computational costs by dynamically matching resource allocation to scene complexity and network conditions. Both strategies operate within the strict latency constraints that interactive gaming demands.

1.2 Research Problem

Reducing bandwidth and computational requirements in cloud gaming while maintaining visual quality and interactivity presents a multifaceted research challenge. The strict latency constraints inherent to interactive gaming severely limit the solution space, as any additional processing must fit within tight timing budgets without degrading the player experience. This section identifies five interconnected sub-problems that collectively define the research scope of this thesis, progressing from the fundamental bandwidth–

quality trade-off through system-level optimization challenges to domain-specific and infrastructure requirements.

1.2.1 The Bandwidth-Quality Trade-off

A straightforward approach to reducing bandwidth is lowering the spatial resolution of the rendered stream. Transmitting 540p instead of 1080p content can reduce bitrate requirements by 40% or more [16]. However, without effective upsampling on the client side, this approach results in visibly degraded visual quality, particularly for game content featuring fine details such as text, HUD elements, and distant objects.

Content-aware and learning-based upsampling solutions have been successfully deployed in traditional (non-cloud) gaming. Major hardware manufacturers offer GPU-accelerated upscalers including NVIDIA’s Deep Learning Super Sampling (DLSS) [17], AMD’s FidelityFX Super Resolution (FSR) [18], and Intel’s Xe Super Sampling (XeSS) [19]. These solutions achieve impressive quality improvements while reducing rendering load. However, they require tight integration with the game engine and access to internal rendering data structures such as motion vectors, depth buffers, and temporal history, information that is unavailable to cloud gaming clients, which receive only compressed video frames. Moreover, incorporating these upscalers demands explicit developer effort during game development: the rendering pipeline must be modified to expose the required engine data, and the temporal upscaling modes of DLSS 2/3 and FSR 2/3 depend on temporal anti-aliasing (TAA) and sub-pixel jitter, meaning that games using forward rendering or lacking TAA support cannot benefit without substantial engine modifications. Furthermore, these upscalers are restricted to specific hardware platforms (e.g., DLSS requires NVIDIA RTX GPUs with dedicated Tensor Cores) and supported game titles. While Steam alone hosts over 86,000 games, only approximately 650 support DLSS and 350–400 support FSR, well under 1% of the catalog [20, 21]. With additional titles on Epic Games Store, PlayStation Store, and Xbox Marketplace, the relative coverage is even smaller, leaving the vast majority of games, particularly legacy titles such as Team Fortress 2 (2007) and forward-rendered titles like Counter-Strike 2, without upscaling capabilities.

1.2.2 The Latency Constraint

Any solution deployed in cloud gaming must operate within strict latency budgets. The complete round-trip latency, from player input to visual feedback, must remain below perceptible thresholds to maintain gameplay quality. Network transmission alone can consume up to 80% of the tolerable latency budget [22], leaving minimal headroom for additional client-side processing. This constraint eliminates most existing super-resolution models from consideration: state-of-the-art approaches such as SwinIR [23] achieve excellent reconstruction quality but are far too slow for real-time 1080p enhancement. Even efficiency-oriented models typically exceed the per-frame latency budget required for 60 fps interactive gaming.

1.2.3 The Server-Side Computational Burden

Cloud gaming providers face substantial costs for GPU provisioning and energy consumption. Rendering high-resolution content at high frame rates requires powerful

server-side GPUs, and the number of concurrent users a server can support is directly limited by available GPU resources. Reducing the server-side rendering resolution could alleviate this burden, but current systems lack mechanisms to adaptively balance rendering resolution, encoding bitrate, and client-side enhancement based on network conditions and content complexity.

Traditional encoders allocate bits uniformly without considering scene complexity, resulting in wasted bandwidth for visually simple scenes and insufficient quality for complex ones. Furthermore, rendering resolution is typically fixed regardless of whether a scene is visually simple or highly complex, meaning GPU cycles are wasted rendering fine details that the encoder will subsequently discard. A system that can predict scene complexity and jointly adjust rendering resolution, encoding bitrate, and client-side enhancement in response would yield efficiency gains that no single-component optimization can achieve.

1.2.4 The VR Extension Challenge

Virtual reality cloud gaming introduces additional constraints beyond traditional 2D streaming. VR applications require rendering two views (one per eye) at high resolution and frame rates exceeding 90 fps to prevent motion sickness. The latency tolerance is even stricter, with end-to-end motion-to-photon latency required to remain below 20 ms to prevent perceptual discomfort [15], a budget that must be shared across rendering, encoding, transmission, decoding, and any client-side enhancement, leaving only a few milliseconds available for super-resolution processing.

Existing VR-specific super-resolution approaches such as CVHSSR [24] achieve high quality but require nearly 3 seconds per stereo pair, making them unsuitable for interactive use. Beyond speed, these approaches treat the two stereo views as independent streams and transmit both in full color, overlooking a fundamental property of stereo content: the chrominance information across left and right views is nearly identical, since color perception does not require binocular disparity. This redundancy represents an opportunity, transmitting one full-color view and one downsampled monochrome view reduces transmission cost while the missing chroma can be reconstructed on the client. Addressing VR cloud gaming therefore requires not simply a faster SR model, but a fundamentally different transmission and reconstruction strategy that exploits stereo structure.

1.2.5 The Research Infrastructure Gap

Evaluating AI-driven solutions for cloud gaming requires infrastructure that supports controlled experimentation with neural network modules while providing reliable quality measurements. Existing cloud gaming stacks such as GamingAnywhere [25] were designed for deployment rather than research extensibility, lacking programmable insertion points for AI models and robust mechanisms for full-reference quality evaluation under realistic network conditions where packet loss, reordering, and frame duplication occur.

These interconnected challenges motivate the research questions addressed in this thesis: How can we design neural upsampling solutions that are engine-agnostic, real-time capable, and effective for both 2D and VR cloud gaming? How can we jointly optimize the rendering, encoding, and enhancement pipeline to maximize quality under bandwidth and compute constraints? How can stereo redundancy in VR be exploited

to achieve bandwidth-efficient immersive streaming within strict latency budgets? And what open, extensible infrastructure is needed to enable reproducible, AI-in-the-loop research in this domain?

1.3 Thesis Statement and Scope

This thesis argues that *client-side, engine-agnostic neural upsampling*, when combined with *complexity-aware system control* and *stereo redundancy exploitation*, can significantly reduce bandwidth and server-side rendering costs in cloud gaming while respecting stringent interactivity constraints. Underpinning these contributions is an open, AI-ready research testbed that enables reproducible evaluation of such techniques under realistic network conditions. Collectively, the solutions developed in this thesis achieve bandwidth reductions of 35–56% across 2D and VR gaming scenarios, server-side resource savings of up to 62%, and real-time inference exceeding 240 fps, all without requiring game engine integration or hardware-specific dependencies.

Scope. The focus of this thesis is on real-time cloud gaming and cloud VR streaming where the client receives a compressed video stream over WebRTC-based real-time transport, representative of modern cloud gaming delivery infrastructure, and applies post-decode enhancement. We study (i) bandwidth–quality trade-offs, (ii) end-to-end latency constraints, and (iii) server resource utilization under realistic network conditions. While the primary application context is cloud gaming, the neural super-resolution models developed in this thesis (GameSR and GameSRVR) are engine-agnostic and operate on decoded video frames alone, making them equally applicable as standalone upsamplers for local gaming, game streaming, or any real-time video enhancement scenario. The design philosophy is twofold: first, to develop high-quality, lightweight upsamplers that achieve competitive reconstruction quality regardless of deployment context, and second, to integrate these models into an end-to-end cloud gaming pipeline where their bandwidth and compute savings are most impactful.

Client compute and the “any device” premise. Shifting enhancement to the client raises a legitimate concern: a defining promise of cloud gaming is that any device, regardless of its local capability, can play, whereas requiring client-side neural upscaling appears to reintroduce the very hardware requirement that this model was meant to eliminate. Three considerations reconcile our approach with this premise. First, the heavy and genuinely prohibitive work, namely game logic, physics, AI, and high-resolution 3D rendering together with its associated discrete GPU, large memory, storage, and per-title installation, remains entirely on the server; the client adds only a single lightweight post-decode upscaling pass, not a rendering pipeline. Second, every cloud gaming client already performs hardware-accelerated video decoding, and the incremental cost of our models is small relative to that decode step: we demonstrate real-time operation across a wide hardware range, from a desktop RTX 4090 down to a mobile Adreno 750 GPU (Section 3.5.9), confirming that the requirement is within reach of commodity consumer devices rather than gaming-class hardware. Modern smartphones, tablets, laptops, smart TVs, and set-top boxes increasingly ship with capable GPUs and dedicated neural processing units (NPUs), and on-device neural upscaling is already mainstream in consumer displays. Third, the enhancement is optional and gracefully degradable: a device lacking sufficient compute can fall back to classical interpolation or simply receive the standard full-resolution stream at a higher bitrate, so no device is excluded from play. The net effect is therefore a modest and

optional client-side compute cost exchanged for substantial bandwidth savings, which for users on mobile, metered, or bandwidth-limited connections broadens rather than narrows access to high-quality cloud gaming.

Codec choice. Throughout this thesis we use H.264/AVC, via the open `libx264` encoder with low-latency settings, as the video codec for all streaming experiments, and we state this upfront because it affects how the reported bitrate results should be read. We adopt H.264 for three reasons. First, it is the most universally supported codec in practice: hardware decoders are present on virtually every client device, from smartphones and web browsers to smart TVs and set-top boxes, which is essential for the low-latency decoding that commodity cloud gaming clients require, and it is the default, most interoperable codec in the WebRTC stack used as our delivery substrate. Second, `libx264` is an open, deterministic, and widely studied reference encoder, which makes our experiments reproducible. Third, H.264 is a deliberately conservative choice with respect to the *absolute* bitrates we report: because it is less compression-efficient than newer codecs such as HEVC and AV1, the bitrates required to reach a given quality, for both the full-resolution baseline and our lower-resolution streams, would be lower under a more modern codec, so the absolute bandwidth figures in this thesis are an upper bound rather than a best case. Importantly, our savings stem from transmitting fewer pixels, a lower streaming resolution that the client reconstructs, which is a source of efficiency independent of the entropy coder; the approach therefore remains beneficial under HEVC or AV1, though the exact percentage savings would need to be re-measured, since it depends on how a given codec’s coding efficiency scales with resolution. Critically, none of the contributions of this thesis are tied to H.264. The super-resolution models (GameSR and GameSRVR) operate on decoded RGB frames and are therefore codec-agnostic, while ARCADE’s rendering-resolution and bitrate control is orthogonal to the specific codec in use. The GameLab testbed (Chapter 6) additionally supports hardware-accelerated H.264 (NVENC/CUVID) and can be extended to other codecs without any change to the methods.

1.4 Research Questions and Objectives

The challenges identified in Section 1.2 give rise to four research questions that collectively span the progression from model-level design to system-level optimization, domain extension, and research infrastructure. This thesis addresses the following research questions:

1. **RQ1:** How can we design a real-time, engine-agnostic neural upsampler for cloud gaming that achieves comparable quality to engine-coupled solutions without requiring access to game source code or rendering data structures?
2. **RQ2:** How can we jointly optimize rendering resolution, encoding bitrate, and client-side super-resolution to maximize Quality of Experience while simultaneously minimizing server-side computational resource consumption under varying bandwidth constraints?
3. **RQ3:** How can neural upsampling be extended to virtual reality cloud gaming by exploiting stereo chrominance redundancy between left and right eye views, while meeting the stricter latency requirements of immersive applications?

4. **RQ4:** What research infrastructure is needed to enable reproducible evaluation of AI-driven cloud gaming systems under realistic network conditions?

To address these questions, the thesis pursues the following objectives:

1. To develop a lightweight neural super-resolution model optimized for gaming content that achieves real-time performance on heterogeneous client devices while maintaining high perceptual quality.
2. To create an adaptive framework that intelligently coordinates server-side rendering, video encoding, and client-side enhancement based on content complexity and network conditions, with the dual objective of maximizing perceptual quality and minimizing server-side resource utilization.
3. To extend super-resolution techniques to stereoscopic VR content by exploiting redundancies between left and right eye views, achieving bandwidth reduction within VR’s strict latency budget.
4. To build an open-source cloud gaming testbed that facilitates AI model integration and provides reliable quality evaluation mechanisms under realistic network conditions.

1.5 Overview of the Proposed Solutions

This thesis follows a progression from *model-level design* to *system-level optimization*, and finally to *domain extension and reproducible experimentation*. Instead of modifying game engines or relying on privileged rendering signals (such as motion vectors, depth buffers, and temporal history accessible only within the game engine), the proposed solutions operate solely on the compressed video stream available at the cloud gaming client, enabling deployment across modern and legacy titles.

1.5.1 GameSR: Real-Time, Engine-Agnostic Super-Resolution

To address the bandwidth–quality trade-off under strict latency constraints (RQ1), we develop **GameSR**, a lightweight neural super-resolution model tailored to gaming content. GameSR upsamples received low-resolution frames to higher resolution on the client side, targeting real-time performance while preserving perceptual quality for challenging game details such as text, HUD elements, and high-frequency textures. The design emphasizes efficiency through compact feature extraction and lightweight temporal modeling, and leverages a data-centric training strategy on game-specific sequences to capture the structured, synthetic patterns common in games.

1.5.2 ARCADE: Complexity-Aware Joint Optimization of the Cloud Gaming Pipeline

Beyond a static operating point, cloud gaming requires adaptation because network bandwidth, latency, and scene complexity vary over time. To address system-level optimization (RQ2), we propose **ARCADE**, a framework that *jointly* selects rendering resolution, encoding bitrate, and client-side enhancement settings to maximize Quality

of Experience (QoE) under constraints. ARCADE uses a content complexity analyzer together with an offline reinforcement learning policy trained from logged sessions, enabling robust decision-making without risky online exploration during live gameplay.

1.5.3 GameSRVR: Joint Colorization and Super-Resolution for Cloud VR

Virtual reality introduces dual-view streaming, stricter latency budgets, and higher sensitivity to visual artifacts. To address the VR domain extension (RQ3), we introduce **GameSRVR**, a unified stereo-aware colorization and super-resolution model. GameSRVR builds on the insight from the iSR study, which showed that transmitting one color view and one monochrome view can significantly reduce VR streaming bandwidth, but replaces iSR’s slow two-stage sequential pipeline with a tightly integrated architecture. Through lightweight depthwise-separable feature extraction, half-resolution attention-based color transfer, and a GameSR-based upsampler with shared weights, GameSRVR reduces inference from 73.61 ms to ~ 11 ms per stereo pair at $4\times$ scaling (a $6.7\times$ speedup over iSR). Evaluated on the GameLab testbed, GameSRVR achieves over 33% bandwidth savings and an end-to-end latency of ~ 20 ms, meeting the VR motion-to-photon budget.

1.5.4 GameLab: Reproducible Research Infrastructure for AI-Driven Cloud Gaming

To enable controlled experimentation and fair evaluation (RQ4), we build **GameLab**, an open-source cloud gaming testbed built on WebRTC. GameLab provides programmable hook points for AI module integration and a QR-based frame identity mechanism for reliable full-reference quality evaluation under realistic network conditions. It serves as the evaluation platform for all contributions in this thesis.

1.6 Thesis Contributions

This thesis makes the following contributions to the field of cloud gaming and neural video enhancement:

1. **GameSR** (Chapter 3, addressing RQ1): A lightweight, engine-agnostic neural super-resolution model that achieves real-time performance at over 240 fps on a single GPU while reducing cloud gaming bandwidth by 35–49% at comparable perceptual quality. GameSR operates directly on encoded frames without requiring game engine integration, enabling deployment across both modern and legacy game titles.
2. **ARCADE** (Chapter 4, addressing RQ2): An adaptive cloud gaming framework employing offline reinforcement learning to jointly optimize rendering resolution, encoding bitrate, and client-side super-resolution. By rendering at a lower resolution when content permits, ARCADE saves server resources on two fronts at once, reducing both the GPU cost of rendering and the cost of compressing (encoding) each frame. ARCADE reduces server CPU and GPU utilization by up to 62%

and 41% respectively, while achieving up to 50% bandwidth savings compared to standard WebRTC streaming.

3. **GameSRVR** (Chapter 5, addressing RQ3): A joint colorization-and-super-resolution model for cloud VR gaming that exploits stereo view redundancy through asymmetric color-monochrome transmission and unified stereo-aware reconstruction. Building on the iSR study, which demonstrated up to 56% bandwidth reduction with a two-stage pipeline, GameSRVR reduces inference from 73.61 ms to ~ 11 ms at $4\times$ scaling. Deployed on the GameLab testbed, GameSRVR achieves over 33% bandwidth savings and an end-to-end latency of ~ 20 ms, meeting the VR motion-to-photon budget without requiring gaze-tracking hardware or game engine modifications.
4. **GameLab** (Chapter 6, addressing RQ4): An open-source, AI-enabled cloud gaming testbed providing programmable server and client hook points for ML model integration, along with a QR-based frame identity mechanism enabling reliable full-reference quality evaluation under realistic network conditions. GameLab underpins the evaluation of all contributions in this thesis and is publicly released to facilitate reproducible research.

1.7 Practical Applications

The methods developed in this thesis are motivated by and directly applicable to real-world cloud gaming deployment challenges.

Bandwidth-constrained delivery. Cloud gaming platforms require roughly 10–25 Mbps for high-quality 1080p play [11, 1, 26], excluding users on mobile, rural, or otherwise bandwidth-limited connections. The engine-agnostic enhancement approach developed in this thesis enables providers to stream at significantly lower resolutions and bitrates while reconstructing high-quality visuals on the client, extending platform reach without infrastructure upgrades.

Server-side scalability. GPU provisioning is among the largest operational costs for cloud gaming providers. Rendering at full resolution for every concurrent user regardless of scene complexity wastes compute on content that would be imperceptible at lower quality. The adaptive joint optimization framework reduces server GPU and CPU utilization in proportion to scene complexity, directly increasing the number of concurrent sessions a fixed hardware budget can support.

Legacy and cross-platform title support. Engine-coupled upscalers such as DLSS and FSR require per-title integration and are unavailable for the majority of the game library, particularly older titles. Because the solutions in this thesis operate on compressed video streams rather than engine internals, they apply uniformly across modern and legacy titles without developer cooperation or source code access.

Immersive VR streaming. Cloud VR gaming faces even more severe bandwidth demands than flat-screen gaming due to dual high-resolution views at high frame rates. The joint colorization-and-super-resolution model developed in this thesis achieves real-time stereo reconstruction at $4\times$ scaling within VR’s 20 ms latency budget, making it directly deployable in existing cloud VR pipelines without HMD hardware modifications and making high-quality immersive streaming viable over connections that would otherwise be insufficient.

1.8 Results Dissemination

1.8.1 Publications

The key findings and methodologies developed in this thesis have been disseminated through the following peer-reviewed publications:

1. H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “GameSR: Real-Time Super-Resolution for Interactive Cloud Gaming,” *under review at IEEE Transactions on Games*, 2026.
2. H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “ARCADE: Adaptive Resolution and Codec Allocation for Deep-learning Enhanced Cloud Gaming,” *under review at IEEE Transactions on Multimedia*, 2026.
3. G. Bakhtiariadzad, H. U. Rehman, S. Shirmohammadi, and M. Hefeeda, “iSR: Super-Resolution for Immersive Cloud VR Gaming Platforms,” *accepted at ACM Multimedia Systems Conference (MMSys '26)*, Hong Kong, 2026.
4. H. U. Rehman, G. Bakhtiariadzad, S. Shirmohammadi, and M. Hefeeda, “GameSRVR: Joint Colorization and Super-Resolution for Cloud VR Gaming,” *to be submitted to ACM Transactions on Multimedia (TOMM)*, 2026.
5. H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “GameLab: An Open-Source AI-Enabled Cloud Gaming Testbed,” *accepted at ACM Multimedia Systems Conference (MMSys '26)*, Hong Kong, 2026.

1.8.2 Data and Code Sharing

To promote research transparency and enable further work in this field, the following resources have been made publicly available:

1. GameLab Testbed: <https://github.com/haseebfazal/ai-testbed>
2. iSR: <https://github.com/haseebfazal/iSR>
3. GameSRVR: *[to be released upon publication]*

1.9 Thesis Structure

This thesis is organized as follows:

- **Chapter 2: Background and Related Work** provides technical foundations for the thesis, covering cloud gaming architectures, real-time streaming over WebRTC, quality metrics for interactive video, neural super-resolution for video, content complexity analysis, and reinforcement learning for systems control. It also reviews prior work and identifies the research gaps addressed in subsequent chapters.

- **Chapter 3: GameSR: Real-Time Super-Resolution for Interactive Gaming** presents a lightweight, engine-agnostic neural super-resolution model designed for latency-sensitive cloud gaming. The chapter details the model design principles, the game-centric training methodology, and comprehensive evaluations demonstrating real-time performance with significant bandwidth reduction.
- **Chapter 4: ARCADE: Adaptive Cloud Gaming Framework** develops an adaptive framework that jointly optimizes rendering resolution, encoding bitrate, and client-side enhancement through offline reinforcement learning. The chapter presents the complexity-aware control system, the offline training pipeline, and evaluations showing substantial reductions in server resource utilization and bandwidth consumption.
- **Chapter 5: GameSRVR: Joint Super-Resolution and Colorization for Cloud VR Gaming** extends neural upsampling to virtual reality cloud gaming by exploiting stereo-view redundancy. The chapter first summarizes the iSR study, which demonstrated the feasibility of asymmetric color-monochrome transmission with separate colorization and SR stages, then presents GameSRVR, a unified model that jointly performs stereo-aware colorization and super-resolution in a single forward pass. The chapter details the architectural design, optimization strategy, and evaluation methodology for real-time VR upscaling.
- **Chapter 6: GameLab: AI-Enabled Cloud Gaming Testbed** introduces the open-source research platform that underpins evaluation throughout this thesis, detailing its architecture, QR-based frame alignment, and integration capabilities.
- **Chapter 7: Conclusion** summarizes the key contributions of this thesis, discusses limitations and practical considerations, and outlines directions for future research in neural enhancement for cloud gaming and cloud VR.

Through this structure, the thesis presents a cohesive narrative that progresses from foundational model design (GameSR), through system-level adaptive optimization (ARCADE), to domain extension for immersive VR (GameSRVR), all underpinned by reproducible research infrastructure (GameLab).

Chapter 2

Background and Related Work

2.1 Introduction

The research contributions of this thesis draw on several intersecting technical areas: deep learning architectures for visual processing, super-resolution techniques for image and video enhancement, cloud gaming system design, and real-time streaming protocols. This chapter establishes the foundations required to understand and contextualize the proposed solutions, and reviews existing literature to identify the gaps that motivate each contribution.

Section 2.2 introduces the deep learning architectures central to the proposed solutions, convolutional neural networks for spatial feature extraction, recurrent architectures including LSTM and ConvLSTM for temporal modeling, and structural reparameterization for inference efficiency. Section 2.3 covers image super-resolution from classical interpolation methods to modern deep learning approaches. Section 2.4 extends this to video super-resolution, where temporal consistency across frames is exploited. Section 2.5 examines super-resolution specifically for gaming content, including commercial engine-coupled upscalers such as DLSS, FSR, and XeSS and their limitations in cloud gaming contexts. Section 2.6 introduces content complexity analysis, which underpins the adaptive decisions made in ARCADE. Section 2.7 describes cloud gaming system architectures, their technical challenges, and commercial platform deployments. Section 2.8 covers reinforcement learning fundamentals and offline RL methods relevant to ARCADE’s control policy. Section 2.9 introduces the quality metrics used throughout the thesis for objective and subjective evaluation. Section 2.10 describes the WebRTC protocol and its role as the transport layer in our experimental testbed. Finally, Section 2.11 synthesizes the reviewed literature to identify the research gaps addressed by this thesis.

2.2 Deep Learning Fundamentals

Deep learning has revolutionized computer vision tasks, including image and video super-resolution. This section introduces the neural network architectures that form the foundation of the solutions developed in this thesis.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed for processing data with spatial structure, such as images and video frames [27]. Unlike fully connected networks that treat each input independently, CNNs exploit the spatial correlation among neighboring data points through local receptive fields, enabling them to learn hierarchical representations of visual features [28]. This property makes CNNs particularly well-suited for image super-resolution, where the goal is to reconstruct high-frequency details from low-resolution inputs.

A CNN processes input through a sequence of layers that progressively extract and transform features. The fundamental operation is convolution, which applies learned filters across the spatial dimensions of the input. For a 2D input image $\mathbf{I}$ and a convolutional kernel $\mathbf{K}$, the convolution operation produces an output feature map:

$$(\mathbf{I} * \mathbf{K})(i, j) = \sum_m \sum_n \mathbf{I}(i + m, j + n) \cdot \mathbf{K}(m, n) \quad (2.1)$$

where (i, j) denotes the spatial coordinates in the output feature map, and the summation is performed over the kernel dimensions. By learning appropriate kernel weights during training, CNNs can detect features ranging from low-level edges and textures to high-level semantic patterns.

The following components are fundamental to the CNN architectures developed in this thesis.

Activation Functions. Activation functions introduce non-linearity into the network, enabling it to learn complex mappings between inputs and outputs. The Rectified Linear Unit (ReLU) [29] is the most widely used activation function, defined as:

$$\text{ReLU}(x) = \max(0, x) \quad (2.2)$$

ReLU is computationally efficient and mitigates the vanishing gradient problem, but it can cause “dead neurons” when inputs are consistently negative.

The Gaussian Error Linear Unit (GELU) [30] provides a smoother alternative by weighting inputs according to their probability under a standard normal distribution:

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2} \left[1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right] \quad (2.3)$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. GELU has been shown to improve training stability and is used in our feature extraction blocks.

The Parametric Rectified Linear Unit (PReLU) [31] extends ReLU with a learnable slope for negative inputs:

$$\text{PReLU}(x) = \max(0, x) + a \cdot \min(0, x) \quad (2.4)$$

where a is a learned parameter. This flexibility allows the network to adapt the activation behavior during training, potentially improving representational capacity.

Depth-to-Space (PixelShuffle). The sub-pixel convolution layer, also known as PixelShuffle [32], provides an efficient learned upsampling mechanism that has become standard in super-resolution architectures. Rather than using interpolation followed by convolution, PixelShuffle first applies a convolution that increases the channel dimension by a factor of s^2 (where s is the upscaling factor), then rearranges these channels into spatial dimensions:

$$\mathcal{P}\mathcal{S}(\mathbf{T})_{i,j,c} = \mathbf{T}_{\lfloor i/s \rfloor, \lfloor j/s \rfloor, c \cdot s^2 + s \cdot \text{mod}(j,s) + \text{mod}(i,s)} \quad (2.5)$$

This operation transforms a tensor of shape $(H \times W \times C \cdot s^2)$ to $(H \cdot s \times W \cdot s \times C)$, effectively trading channel depth for spatial resolution. PixelShuffle is computationally efficient because the convolution operates at the lower resolution, and the rearrangement is a simple memory operation.

Space-to-Depth (PixelUnshuffle). The inverse operation, PixelUnshuffle [32], rearranges spatial data into the channel dimension, transforming a tensor from $(H \cdot s \times W \cdot s \times C)$ to $(H \times W \times C \cdot s^2)$. This operation is useful for reducing spatial dimensions early in the network, allowing subsequent convolutions to operate on smaller feature maps with richer channel information. In our GameSR architecture, we apply PixelUnshuffle at the input stage to reduce computational cost: processing features at $\frac{1}{s}$ spatial resolution with s^2 times more channels requires significantly fewer operations than processing at full resolution, while preserving the same information content.

Structural Reparameterization. Structural reparameterization [33] is a technique that enables training complex, multi-branch architectures while deploying simplified, efficient single-path networks. During training, a reparameterizable block may contain multiple parallel branches, for example, a 3×3 convolution, a 1×1 convolution, and an identity mapping, that are summed to produce the output. This multi-branch structure provides richer gradient flow and improved optimization.

During inference, these parallel branches can be algebraically merged into a single convolution. For instance, consider a block with a 3×3 convolution $\mathbf{W}_3$, a 1×1 convolution $\mathbf{W}_1$ (zero-padded to 3×3), and an identity branch. The equivalent single kernel is:

$$\mathbf{W}_{\text{equiv}} = \mathbf{W}_3 + \text{pad}(\mathbf{W}_1) + \mathbf{I}_{3 \times 3} \quad (2.6)$$

This transformation preserves the mathematical function while eliminating the computational overhead of multiple branches. In our GameSR architecture, we employ reparameterizable Feature Extraction Blocks (FEBs) that expand features through $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$ convolution sequences during training. At inference time, these are merged into single 3×3 convolutions, substantially reducing latency without any loss in reconstruction quality.

2.2.2 Recurrent Neural Networks and Temporal Modeling

While CNNs excel at extracting spatial features from individual frames, video super-resolution requires modeling temporal dependencies across consecutive frames. Objects in motion, camera movements, and scene transitions create correlations between frames

that can be exploited to improve reconstruction quality and temporal consistency. Recurrent Neural Networks (RNNs) provide a mechanism for processing sequential data by maintaining internal state that captures information from previous time steps [34].

Unlike feedforward networks where information flows in a single direction, RNNs contain recurrent connections that form cycles, allowing information to persist across sequence elements. Given an input sequence $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$, an RNN computes hidden states $\mathbf{h}_t$ that encode information from all previous inputs:

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h) \quad (2.7)$$

where $\mathbf{W}_{hh}$ and $\mathbf{W}_{xh}$ are learnable weight matrices, $\mathbf{b}_h$ is a bias term, and ϕ is a non-linear activation function. This formulation enables the network to capture temporal patterns and dependencies in sequential data.

However, standard RNNs suffer from the vanishing gradient problem: during back-propagation through time, gradients can diminish exponentially, making it difficult to learn long-range dependencies. This limitation motivated the development of gated recurrent architectures.

Long Short-Term Memory Networks. Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber [34], address the vanishing gradient problem through a gating mechanism that regulates information flow. An LSTM cell maintains a cell state $\mathbf{C}_t$ that acts as a memory pathway, allowing gradients to flow across many time steps without degradation.

The LSTM cell contains three gates that control information flow: the **forget gate** $\mathbf{f}_t$ determines what information to discard from the previous cell state; the **input gate** $\mathbf{i}_t$ controls what new information to store; and the **output gate** $\mathbf{o}_t$ regulates what information to output. These gates are computed as:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad (2.8)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad (2.9)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad (2.10)$$

where σ denotes the sigmoid function and $[\cdot, \cdot]$ represents concatenation.

The cell state and hidden state are updated as:

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \quad (2.11)$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \quad (2.12)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t) \quad (2.13)$$

where $\odot$ denotes element-wise multiplication and $\tilde{\mathbf{C}}_t$ is the candidate cell state.

Convolutional LSTM Networks. Standard LSTMs process vectorized inputs, discarding spatial structure, a significant limitation for video processing where spatial relationships are essential. Convolutional LSTM (ConvLSTM) networks, introduced by Shi et al. [35], extend LSTMs by replacing matrix multiplications with convolution operations, preserving spatial structure throughout the recurrent computation.

In a ConvLSTM cell, the inputs $\mathbf{X}_t$, hidden states $\mathbf{H}_t$, cell states $\mathbf{C}_t$, and gates are all 3D tensors with two spatial dimensions and a channel dimension. The gate computations

become:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} * \mathbf{X}_t + \mathbf{W}_{hi} * \mathbf{H}_{t-1} + \mathbf{W}_{ci} \odot \mathbf{C}_{t-1} + \mathbf{b}_i) \quad (2.14)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} * \mathbf{X}_t + \mathbf{W}_{hf} * \mathbf{H}_{t-1} + \mathbf{W}_{cf} \odot \mathbf{C}_{t-1} + \mathbf{b}_f) \quad (2.15)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} * \mathbf{X}_t + \mathbf{W}_{ho} * \mathbf{H}_{t-1} + \mathbf{W}_{co} \odot \mathbf{C}_t + \mathbf{b}_o) \quad (2.16)$$

where $*$ denotes the convolution operation. By using convolutions, ConvLSTM captures both spatial correlations within each frame and temporal correlations across frames simultaneously, making it effective for spatiotemporal sequence modeling tasks including video super-resolution.

In our GameSR architecture, we employ a lightweight ConvLSTM module after spatial feature extraction. This design balances temporal modeling capability against computational efficiency: the ConvLSTM operates on compact feature representations rather than full-resolution frames, and its single-layer structure minimizes latency while still capturing the temporal coherence essential for flicker-free video upscaling.

2.3 Super Resolution

Super-resolution (SR) refers to the task of reconstructing high-resolution (HR) images with rich visual details from low-resolution (LR) observations. This section covers the problem formulation, classical approaches, and reviews deep learning-based methods that form the state-of-the-art.

2.3.1 Problem Formulation

The degradation process that produces LR images from HR originals can be modeled as:

$$\mathbf{y} = (\mathbf{x} \otimes \mathbf{k}) \downarrow_s + \mathbf{n} \quad (2.17)$$

where $\mathbf{y}$ is the observed LR image, $\mathbf{x}$ is the unknown HR image, $\mathbf{k}$ is a blur kernel, $\otimes$ denotes convolution, $\downarrow_s$ represents downsampling by scale factor s , and $\mathbf{n}$ is additive noise.

Super-resolution is fundamentally an ill-posed inverse problem: a single LR observation may correspond to infinitely many HR solutions that, when degraded, produce identical LR images. The goal of SR methods is to learn priors or regularization that constrain the solution space to produce visually plausible and perceptually accurate reconstructions.

2.3.2 Classical Approaches

Early SR methods employed interpolation-based techniques such as bilinear, bicubic, and Lanczos filtering. These methods compute output pixels as weighted combinations of neighboring input pixels, offering low computational complexity and real-time performance. However, interpolation cannot recover high-frequency details absent from the LR input, resulting in over-smoothed reconstructions.

Subsequent research explored reconstruction-based methods leveraging sparse coding and dictionary learning [36]. These approaches represent image patches as sparse linear combinations of learned dictionary atoms. While achieving improved quality

over interpolation, dictionary-based methods depend heavily on the similarity between training and test images, and their iterative optimization procedures are computationally expensive.

2.3.3 Deep Learning-Based Super Resolution

The introduction of deep learning to super-resolution marked a paradigm shift in the field. This section reviews the evolution of neural SR architectures, organized by their key innovations.

Pioneering Architectures. The first DNN-based SR network, Super-Resolution Convolutional Neural Network (SRCNN) [37], solved the SR problem by constructing a three-layer CNN to learn the mapping between LR patches and their corresponding HR patches. SRCNN uses bicubic interpolation to upsample the LR image early in the network, then applies convolutional layers to refine the result. The loss function employed is mean squared error (MSE):

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.18)$$

Although SRCNN is lightweight, the pre-upsampling strategy is computationally expensive since most operations occur in HR space, a significant limitation for real-time applications like cloud gaming.

Kim et al. [38] applied residual learning to SR with Very Deep Convolutional Networks (VDSR), enabling training of 20-layer networks. VDSR also uses early upsampling but learns residuals rather than direct mappings, with mean absolute error (MAE) as the loss function:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.19)$$

While achieving state-of-the-art PSNR at the time, the HR-space processing remains computationally prohibitive.

Efficient Upsampling Strategies. Shi et al. [32] introduced Efficient Sub-pixel CNN (ESPCN), which avoids pre-interpolation by placing the upsampling operation at the network’s end using the sub-pixel convolution (PixelShuffle) layer. As shown in Figure 2.1, feature extraction occurs entirely in LR space, dramatically reducing computational cost. This post-upsampling strategy has become standard in efficient SR architectures and forms the basis of our GameSR design.

Dong et al. accelerated SRCNN with Fast SRCNN (FSRCNN) [39], using an hour-glass structure that shrinks feature dimensions before mapping, then expands them before upsampling via deconvolution. Both ESPCN and FSRCNN achieve low inference times but struggle to produce adequate PSNR values for demanding applications due to their limited network capacity.

Deep Residual Architectures. Lim et al. [40] proposed Enhanced Deep Super-Resolution network (EDSR), building on SRResNet [41]. EDSR removes batch normalization, arguing that SR inputs and outputs have similar distributions where batch

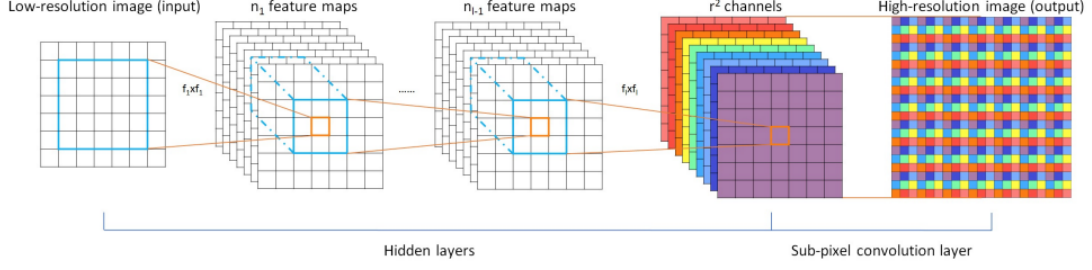


Figure 2.1: ESPCN architecture: feature extraction in LR space followed by sub-pixel convolution for upsampling, enabling efficient real-time super-resolution.

normalization reduces range flexibility, reducing GPU memory by 40% while enabling deeper networks. EDSR achieved state-of-the-art PSNR but with inference times exceeding real-time requirements.

Zhang et al. [42] introduced Residual Channel Attention Networks (RCAN), using channel attention mechanisms within a Residual-in-Residual structure. The attention mechanism allows the network to focus on informative features while bypassing abundant low-frequency information. However, RCAN’s execution time exceeds 100 ms on RTX 2080, making it unsuitable for cloud gaming.

Progressive and Pyramid Architectures. Lai et al. [43] proposed Laplacian Pyramid Super-Resolution Network (LapSRN), which progressively reconstructs HR images at multiple scales. Each pyramid level predicts residuals that are added to upsampled coarse features. While effective for large scaling factors, LapSRN’s multi-stage processing introduces latency incompatible with real-time constraints.

Perceptual Quality and GANs. Ledig et al. [41] introduced SRGAN, highlighting that MSE optimization produces high PSNR but overly smooth results lacking perceptual quality. SRGAN employs adversarial training with a perceptual loss computed on VGG features:

$$\mathcal{L}_{\text{perceptual}} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(\mathbf{I}^{HR})_{x,y} - \phi_{i,j}(G_{\theta}(\mathbf{I}^{LR}))_{x,y})^2 \quad (2.20)$$

where $\phi_{i,j}$ denotes VGG feature maps. Wang et al. [44] improved upon this with ESRGAN, using Residual-in-Residual Dense Blocks and relativistic discriminator training for enhanced perceptual quality.

Diffusion-Based Super Resolution. More recently, denoising diffusion probabilistic models [45] have emerged as a powerful generative class for super-resolution, often surpassing GANs in perceptual quality and detail realism. Methods such as SR3 [46] cast SR as iterative conditional denoising, producing highly realistic textures by progressively refining a noisy estimate toward the high-resolution image. However, this iterative formulation is fundamentally at odds with real-time constraints: generating a single image requires many sequential network evaluations (typically tens to hundreds of denoising steps), resulting in inference times on the order of seconds per image, orders of magnitude beyond the per-frame budget of interactive gaming. Efforts to accelerate diffusion-based SR, such as ResShift [47], which reduces sampling to roughly 15 steps,

and one-step distilled variants, narrow this gap but still trail purpose-built lightweight feed-forward models in latency at the resolutions required here. Moreover, their stochastic sampling can hallucinate details not present in the source and introduce temporal flicker across video frames, undermining the fidelity and frame-to-frame consistency that gaming demands. For these reasons, diffusion-based SR, despite its impressive quality, is not currently suitable for the latency-critical, temporally-consistent setting targeted in this thesis, and we instead focus on efficient feed-forward convolutional designs.

Lightweight and Efficient Models. For practical deployment, several lightweight architectures have been proposed. IMDN [48] uses information multi-distillation blocks to achieve competitive quality with reduced parameters. LatticeNet [49] employs lattice blocks combining residual and dense connections efficiently. RT4KSR [50] targets real-time 4K super-resolution through aggressive architectural optimization. Despite these advances, a fundamental tension remains: models achieving state-of-the-art quality such as SwinIR [23] require inference times on the order of seconds per frame, while models fast enough for real-time use incur noticeable quality penalties. It is worth examining why the highest-quality models are so costly. SwinIR’s quality stems from its vision-transformer backbone (it is built on the Swin Transformer), whose self-attention captures long-range dependencies that convolutions miss. This design carries two drawbacks for real-time gaming SR. First, it is computationally expensive: self-attention scales quadratically with the number of spatial tokens, and SwinIR consequently requires nearly 2 s per 1080p frame (Chapter 3), orders of magnitude beyond the per-frame budget of interactive gaming. Second, transformers are harder to train: unlike convolutions, they lack the built-in inductive biases of locality and translation equivariance, so these properties must be learned from data, and transformer models therefore require substantially larger and more diverse training datasets to generalize well. This is a poor fit for the data-centric, per-game training regime targeted in this thesis, where only a handful of representative gameplay sessions are available per title. These considerations, high inference cost and data-hungry training, motivate a convolutional rather than transformer-based design for real-time gaming super-resolution. This quality–latency trade-off is a central challenge addressed in Chapter 3, where a detailed quantitative comparison of these models is provided.

2.4 Video Super Resolution

While single-image SR operates on individual frames independently, video super-resolution (VSR) exploits temporal redundancy across consecutive frames. Adjacent frames contain complementary information, details occluded or aliased in one frame may be visible in neighbors, and leveraging this can yield substantial quality improvements over single-image approaches. The fundamental challenge in VSR is how to align and aggregate this temporal information effectively without introducing artifacts or prohibitive computational overhead.

2.4.1 Problem Formulation

Given a sequence of low-resolution frames $\{y_{t-k}^{LR}, \dots, y_t^{LR}, \dots, y_{t+k}^{LR}\}$, VSR aims to reconstruct a high-resolution output $\hat{y}_t^{HR}$ for the target frame t by exploiting information from neighboring frames within a temporal window of size $2k + 1$. Formally, a VSR model F with parameters θ optimizes:

$$\theta^* = \arg \min_{\theta} \sum_t L(F(y_{t-k:t+k}^{LR}; \theta), y_t^{HR}) \quad (2.21)$$

where L is a reconstruction loss. The key difference from SISR is that F now has access to a temporal context window, enabling it to recover details that are ambiguous or unavailable in the single target frame.

2.4.2 Temporal Alignment Strategies

VSR methods differ primarily in how they align temporal information across frames before aggregation.

Explicit Optical Flow Alignment. Early VSR methods estimate dense optical flow between frames and warp features to align them before reconstruction [51, 52]. VESPCN [51] was among the first to jointly train motion compensation and super-resolution end-to-end. While effective for smooth, predictable motion, optical flow estimation is computationally expensive and can introduce alignment artifacts in regions of occlusion, fast motion, or complex scene changes, all common in gaming content.

Deformable Convolution Alignment. Deformable convolutional networks [53] offer a learnable alternative to explicit flow estimation. Rather than computing a global motion field, deformable convolutions adapt their sampling offsets conditioned on the input, allowing the network to implicitly model local motion patterns. EDVR [54] extends this with a Pyramid, Cascading, and Deformable (PCD) alignment module operating at multiple scales, achieving state-of-the-art quality on natural video benchmarks. However, the increased flexibility comes at the cost of higher parameter counts and inference times that preclude real-time use.

Implicit Spatio-Temporal Aggregation. Rather than explicitly aligning frames before fusion, implicit methods learn to aggregate temporal context directly through 3D convolutions or spatio-temporal attention [55]. Transformer-based architectures such as VRT [56] apply multi-head attention across spatial and temporal dimensions, capturing long-range dependencies that localized convolutions miss. These approaches avoid motion estimation errors but are typically memory-intensive and poorly suited to the strict latency budgets of interactive applications.

Recurrent Propagation. Recurrent approaches process frames sequentially, propagating information through hidden states across time. BasicVSR [57] demonstrates that simple bidirectional recurrent propagation with optical flow alignment achieves competitive quality with fewer parameters than sliding-window methods. Recurrent designs naturally handle variable-length sequences with modest memory requirements, as only the current frame and hidden state need to reside in memory simultaneously,

a significant advantage for deployment on resource-constrained clients. This strategy directly informs our GameSR design, which employs a lightweight ConvLSTM module to propagate temporal context without the overhead of explicit motion estimation.

2.4.3 Temporal Consistency

A critical consideration specific to VSR is temporal consistency: processing frames independently, even with high per-frame quality, can introduce flickering artifacts as fine details shift between reconstructions. This is particularly noticeable in regions of uniform texture or gradual motion, and effective VSR methods must therefore balance reconstruction fidelity against temporal stability.

For gaming content, this challenge is amplified. Games frequently contain static HUD elements, repeating textures, and abrupt scene transitions due to player actions. Independent per-frame upsampling can cause visible inconsistencies in these elements even when per-frame PSNR is high. Temporal loss functions, which penalize differences between consecutive output frames, and recurrent architectures that carry hidden state across frames both help mitigate this issue. Our GameSR training methodology incorporates temporal sequences rather than isolated frames specifically to encourage stable reconstruction across time.

2.4.4 Relevance to Cloud Gaming

Despite the quality gains VSR offers over SISR, most state-of-the-art VSR models are designed for offline video restoration rather than real-time interactive applications. Models such as EDVR and VRT process entire temporal windows simultaneously and incur high per-frame inference cost, far beyond what cloud gaming latency budgets permit [54, 56]. Even efficiency-focused models typically assume frames can be buffered and processed in batches, an assumption that breaks down in cloud gaming where each frame must be decoded, enhanced, and displayed within a fixed deadline to preserve interactivity.

Two constraints jointly shape the VSR design space for cloud gaming. First, the per-frame latency budget: at 60 fps each frame allows approximately 16.7 ms from decode to display, leaving only a fraction of that for enhancement. Second, the causal requirement: a cloud gaming client receives frames in real time and cannot buffer a future temporal window without introducing end-to-end latency, ruling out bidirectional and sliding-window approaches that rely on future frame context.

The central design challenge is therefore not reconstruction quality in isolation, but achieving temporal consistency and quality gains within strict per-frame latency and causality constraints. This motivates the selective adoption of lightweight recurrent temporal modeling in GameSR, specifically a compact ConvLSTM that propagates hidden state causally across frames, prioritizing efficiency while retaining the key benefit of temporal context over purely single-frame approaches.

2.5 Super Resolution for Gaming Content

Gaming content presents both unique challenges and opportunities for super-resolution. Unlike natural video, game frames are synthetically rendered with precise pixels and

known degradation characteristics. Furthermore, gaming content exhibits distinctive properties:

- **Structured patterns:** Repetitive textures, geometric structures, and consistent art styles recur throughout gameplay.
- **Static elements:** HUD overlays, minimaps, and UI components remain constant or change predictably.
- **Predictable motion:** Character animations and camera movements follow smooth trajectories governed by game physics.

These properties motivate a *data-centric* approach: rather than training general-purpose models on diverse natural images, per-game models can specialize in specific visual characteristics. This strategy is employed by commercial upscalers and forms the foundation of GameSR.

2.5.1 Commercial Game Upscalers

Major hardware vendors have developed proprietary upscaling solutions that have seen widespread adoption in the gaming industry. While these solutions demonstrate the viability of neural upscaling for interactive content, they share architectural assumptions that make them unsuitable for cloud gaming deployment.

NVIDIA DLSS. Deep Learning Super Sampling [58] has evolved through several generations since its introduction in 2018. DLSS 1.0 used a convolutional autoencoder trained on game-specific 16K ground truth images to hallucinate high-frequency detail. DLSS 2.0 introduced a generalized temporal accumulation network trained across many games simultaneously, eliminating the need for per-game training while improving quality and stability. DLSS 3.0 extended the framework further with Frame Generation, synthesizing entirely new intermediate frames using optical flow to double perceived frame rates. Across all versions, DLSS relies on motion vectors exported from the game engine’s G-buffer and jittered sub-pixel rendering to accumulate temporal detail across frames. This enables impressive reconstruction quality and temporal stability but requires NVIDIA RTX hardware with dedicated Tensor Cores and explicit game engine integration to expose the necessary rendering data.

AMD FSR. FidelityFX Super Resolution [18] takes a deliberately hardware-agnostic approach. FSR 1.0 employs a purely spatial pipeline: an edge-adaptive spatial upsampling pass (EASU) reconstructs high-frequency edges, followed by a contrast-adaptive sharpening pass (RCAS) to recover perceived detail. Because it operates entirely on the final rendered frame with no engine integration required, FSR 1.0 runs on virtually any GPU. FSR 2.0 and 3.0 introduced temporal accumulation using motion vectors and depth information, substantially improving quality at the cost of engine integration similar to DLSS. The hardware-agnostic design of FSR 1.0 makes it the closest commercial analogue to the engine-agnostic setting addressed in this thesis, though its spatial-only reconstruction cannot match the quality of temporally-informed methods.

Intel XeSS. Xe Super Sampling [19] combines deep learning upscaling with optional motion vector input. In its full-quality mode, XeSS uses engine-exported motion vectors and depth data for temporal accumulation, achieving quality competitive with DLSS 2.0. On hardware without Intel Arc’s dedicated XMX matrix acceleration units, XeSS falls back to a DP4a execution path with reduced quality. Like DLSS, maximum quality requires explicit engine integration, restricting deployment to supported titles.

NVIDIA Image Scaling and Other Spatial Methods. Beyond the primary temporal upscalers, several lighter spatial solutions exist. NVIDIA Image Scaling (NIS) applies an adaptive sharpening filter similar to FSR 1.0 without requiring engine integration. AMD’s earlier Contrast Adaptive Sharpening (CAS) operates at native resolution for perceived sharpening rather than upscaling. These purely spatial methods run on any hardware and require no engine access, but produce noticeably inferior quality to temporal methods, particularly for fine text, HUD elements, and high-frequency textures, content categories critical to gaming quality.

These solutions collectively demonstrate the effectiveness of neural and adaptive upscaling for gaming content but share a critical limitation from the perspective of cloud gaming deployment. DLSS and XeSS at full quality, and FSR 2.0 and above, all require access to motion vectors, depth buffers, and sub-pixel jitter information available only inside the game engine’s rendering pipeline, data that is discarded before the video stream is encoded and transmitted to the cloud gaming client. FSR 1.0 and spatial methods avoid this dependency but sacrifice temporal quality. No existing commercial solution operates on the compressed video stream alone while achieving temporal stability, which is precisely the constraint cloud gaming clients operate under. This gap motivates the engine-agnostic design of GameSR, discussed in Chapter 3.

2.6 Content Complexity Analysis

Adaptive streaming systems benefit from understanding content complexity to optimize bitrate allocation. Prior complexity analyzers such as VCA and EVCA use spatial and temporal features to inform rate control and resource allocation decisions [59, 60].

Spatial Complexity. Spatial complexity measures the detail level within individual frames using texture energy derived from DCT coefficients. For a frame divided into C blocks of size $w \times w$, the average texture energy E is:

$$E = \frac{1}{C \cdot w^2} \sum_{k=0}^{C-1} H_{p,k} \quad (2.22)$$

where $H_{p,k}$ represents the DCT-based texture energy of block k in frame p . Frames with high spatial complexity contain fine textures, sharp edges, and detailed geometry that require more bits to encode faithfully.

Temporal Complexity. Temporal complexity evaluates changes between consecutive frames using the sum of absolute differences between texture energies:

$$h = \frac{1}{C \cdot w^2} \sum_{k=0}^{C-1} \text{SAD}(H_{p,k}, H_{p-1,k}) \quad (2.23)$$

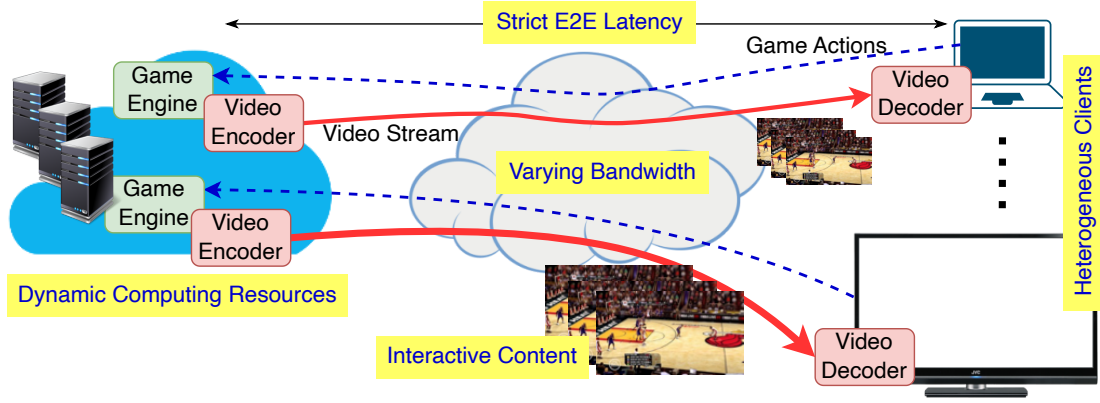


Figure 2.2: High-level architecture of a cloud gaming system, showing the server-side rendering and encoding pipeline, network transport layer, and client-side decoding and display chain.

High temporal complexity indicates rapid scene changes, fast motion, or particle effects that reduce inter-frame prediction efficiency and demand higher encoding bitrates.

These complexity metrics inform the adaptive bitrate control in ARCADE (Chapter 4), enabling content-aware optimization of rendering resolution and encoding parameters. In ARCADE, raw spatial and temporal complexity values are further quantized into discrete tiers via K-Means clustering, providing stable complexity signals suitable for reinforcement learning-based decision-making.

2.7 Cloud Gaming

Cloud gaming (CG) enables users to play computationally demanding games on lightweight devices by streaming rendered content from remote servers. Unlike video-on-demand streaming where content is pre-encoded and cached, cloud gaming renders each frame in real time in direct response to player input, making it fundamentally an interactive application with strict latency requirements. This section describes the system architecture, technical challenges, and existing optimization approaches relevant to this thesis.

2.7.1 System Architecture

A cloud gaming system comprises server-side and client-side components connected through a network transport layer. Figure 2.2 illustrates the high-level pipeline.

Server-Side Components. The server hosts the game application and performs all computationally intensive operations, isolating the client from hardware requirements.

Game Logic processes player inputs received over the network, updates the game state, and executes physics and AI simulations. The game state, including player positions, object dynamics, and world state, must be updated at the target frame rate to maintain simulation accuracy and responsiveness.

Scene Renderer generates visual frames by rasterizing 3D geometry, applying lighting models, shaders, and post-processing effects. Modern game renderers employ deferred shading pipelines that produce intermediate G-buffers containing depth,

normals, and material properties before compositing the final frame. These G-buffer contents include the motion vectors and depth information exploited by engine-coupled upscalers such as DLSS, data that is consumed internally and never transmitted to the client.

Video Encoder compresses rendered frames for network transmission. Hardware-accelerated encoders such as NVIDIA NVENC and AMD AMF encode frames using H.264 or H.265 with low-latency profiles that prioritize encoding speed over compression efficiency. This means intra-refresh intervals are kept short, B-frames are typically disabled, and rate control operates in constant-quality or variable-bitrate modes rather than the two-pass encoding used for offline video. These constraints result in lower compression efficiency compared to VoD encoding, contributing to cloud gaming’s higher bandwidth requirements.

Input Handler receives player commands, keyboard, mouse, gamepad, or touch events, from the client, applies any necessary input processing such as dead-zone filtering, and forwards them to the game logic. Minimizing the round-trip time from input arrival to frame render is critical for perceived responsiveness.

Client-Side Components. The client performs lightweight operations that can execute on low-power devices including smartphones, thin clients, and smart TVs.

Video Decoder decompresses the received bitstream using hardware-accelerated decoding, reconstructing pixel frames from the compressed stream. The decoder operates on the encoded video stream only, it has no access to the original rendered frames, the game engine, or any intermediate rendering data. This is the fundamental constraint that makes engine-agnostic enhancement necessary: any client-side processing must operate solely on the decoded pixel data.

Display Renderer presents decoded frames to the display, potentially applying post-processing operations such as color correction, scaling to the display’s native resolution, or, in the systems proposed in this thesis, neural super-resolution enhancement inserted between the decoder and the display.

Input Capture samples player input devices at high frequency and transmits events to the server with minimal buffering to reduce input latency.

2.7.2 End-to-End Latency Pipeline

The end-to-end latency in cloud gaming, from player input to visual feedback, accumulates across multiple stages: input capture and transmission, server-side game logic, rendering, video encoding, network delivery, client-side decoding, and display scan-out. The relative contribution of each stage varies with network conditions, scene complexity, frame rate targets, and hardware capabilities. Prior work has established that first-person shooter players begin to perceive degradation above 80 ms end-to-end latency [2], leaving minimal headroom for any additional client-side processing such as super-resolution enhancement. This latency budget is a central design constraint for all contributions in this thesis.

2.7.3 Video Encoding for Cloud Gaming

The encoding configuration used in cloud gaming differs substantially from offline video encoding and has direct implications for super-resolution quality. Low-latency profiles

disable or restrict several encoding features that improve compression efficiency in offline settings. B-frames, which require future frame references and introduce buffering delay, are typically disabled entirely. GOP (Group of Pictures) lengths are kept short to limit error propagation from packet loss and to enable rapid quality recovery. Rate control must respond to rapid changes in scene complexity within a single GOP, as gaming content can transition from a static menu screen to a high-motion combat scene within a single frame. These constraints mean that cloud gaming streams are encoded less efficiently than VoD content at equivalent bitrates, producing more compression artifacts at a given bandwidth, artifacts that client-side super-resolution must either avoid amplifying or actively suppress.

2.7.4 Technical Challenges

Cloud gaming faces distinct challenges compared to traditional video streaming, arising from its interactive nature and the tight coupling between player input, server computation, and visual output.

Latency Sensitivity. Interactive gaming demands end-to-end latency far below thresholds acceptable for passive video consumption. Research has established that first-person shooter games tolerate maximum latencies of approximately 80 ms before player performance degrades noticeably [2], while additional delay reduces third-person game performance by up to 25% per 100 ms [2]. Network delay alone can consume up to 80% of this budget [22], leaving minimal headroom for additional processing stages. This constraint is fundamentally different from video-on-demand streaming, where buffering several seconds of content masks network variability entirely. In cloud gaming, any increase in processing time at the client, including enhancement operations, directly adds to the input-to-display latency experienced by the player. Furthermore, latency jitter is often more disruptive than consistently high latency, as irregular delays break the player's motor adaptation to the system's response characteristics.

Bandwidth Requirements. Cloud gaming demands substantially higher bitrates than traditional streaming at equivalent resolutions. Netflix streams 1080p at approximately 5 Mbps [9], whereas NVIDIA GeForce NOW requires at least 25 Mbps for comparable resolution [61]. This disparity arises from several factors: gaming content exhibits higher temporal complexity than pre-produced video; low-latency encoding profiles disable B-frames and restrict lookahead, eliminating compression tools that VoD encoders rely on heavily; and real-time rate control forces encoders to allocate bits conservatively to avoid quality collapse during transitions. These constraints mean that even modest resolution reductions, for instance from 1080p to 720p, can yield proportionally larger bandwidth savings in cloud gaming than in VoD streaming, motivating resolution-based bandwidth management strategies.

Content Complexity Variability. Unlike pre-produced video where scene complexity can be analyzed offline, cloud gaming generates content dynamically in response to player actions. Scene complexity can vary dramatically within a single session, from a low-complexity static menu or indoor environment to a high-complexity outdoor scene with particle effects, crowds, and dynamic lighting, all within fractions of a second. Traditional fixed-bitrate or simple variable-bitrate encoders cannot respond to

this variability optimally, allocating excessive bits to simple scenes and insufficient bits to complex ones. This motivates complexity-aware adaptive approaches that anticipate and respond to scene difficulty in real time, as developed in ARCADE.

Server Computational Cost. Rendering high-resolution content at high frame rates imposes substantial GPU costs on cloud gaming providers. A single gaming session running at 4K and 60 fps requires dedicated GPU resources capable of sustaining that workload continuously for the session duration. At scale, with thousands of concurrent users, GPU provisioning represents one of the largest operational expenses for cloud gaming infrastructure. Reducing the server-side rendering resolution for scenes where full resolution is unnecessary, and compensating with client-side upsampling, directly reduces GPU utilization and increases the number of sessions a fixed hardware budget can support. However, current systems lack the mechanisms to make this trade-off adaptively based on content complexity and network conditions, a gap that the ARCADE framework addresses.

Client Heterogeneity. Cloud gaming clients span a wide range of device capabilities, from high-end gaming PCs to smartphones and smart TVs. Any client-side processing, including neural enhancement, must be feasible across this heterogeneous device landscape. This places an upper bound on the computational complexity of client-side models and motivates designs that degrade gracefully on lower-capability hardware, either by skipping enhancement entirely or switching to lighter processing modes.

2.7.5 Commercial Platforms

Major technology companies operate cloud gaming services, each with distinct infrastructure approaches. Their public technical requirements provide useful benchmarks for the bandwidth and latency constraints that motivate this thesis.

NVIDIA GeForce NOW [62] streams games from users’ existing PC libraries, rendered on NVIDIA GPU-equipped servers. It offers tiered service levels up to RTX 4080-class rendering at 4K and 120 fps, requiring up to 35 Mbps at the highest tier, explicitly acknowledging the bandwidth–quality trade-off through its pricing structure.

Microsoft Xbox Cloud Gaming [26] is integrated into Xbox Game Pass Ultimate, enabling access to a curated game library from browsers, mobile devices, and consoles. Servers are based on custom Xbox Series X hardware, targeting 1080p at 60 fps with a recommended 20 Mbps connection.

Sony PlayStation Plus Premium [63] offers cloud streaming of PlayStation titles from Sony’s own data centers, recommending 15 Mbps for 1080p streaming, lower than competing services, reflecting differences in content type and encoding strategy.

Amazon Luna [64] provides channel-based gaming subscriptions streamed from AWS infrastructure, supporting up to 4K and leveraging AWS edge servers to reduce baseline network latency for users in well-served regions.

These platforms demonstrate the commercial viability of cloud gaming while highlighting its ongoing challenges: all impose minimum bandwidth requirements substantially above VoD streaming, all face the fundamental latency–quality–cost trade-off, and none currently employ client-side neural enhancement, the opportunity addressed in this thesis. Table 2.1 summarizes these requirements, highlighting that even basic 720p streaming demands 10–15 Mbps, while 4K requires 35+ Mbps—well above the capacity

Table 2.1: Minimum bandwidth requirements of major cloud gaming platforms.

Platform	720p@60fps	1080p@60fps	4K@60fps
NVIDIA GeForce NOW [62]	15 Mbps	25 Mbps	35 Mbps
Xbox Cloud Gaming [26]	10 Mbps	20 Mbps	–
PlayStation Plus [63]	10 Mbps	15 Mbps	–
Amazon Luna [64]	10 Mbps	–	–

of many residential and mobile connections, motivating the neural bandwidth reduction approach developed in this thesis.

2.8 Reinforcement Learning for Systems Control

Reinforcement learning (RL) provides a framework for sequential decision-making under uncertainty, where an agent learns to select actions that maximize cumulative reward through interaction with an environment. This section introduces the RL concepts underlying ARCADE’s adaptive control policy.

2.8.1 Markov Decision Process Formulation

An RL problem is typically formalized as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s, a)$ is the state transition probability, $R(s, a)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor that trades off immediate and future rewards.

At each time step t , the agent observes state $s_t \in \mathcal{S}$, selects action $a_t \in \mathcal{A}$ according to a policy $\pi(a|s)$, receives reward $r_t = R(s_t, a_t)$, and transitions to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$. The objective is to find a policy π^* that maximizes the expected discounted return:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (2.24)$$

2.8.2 Value Functions and Q-Learning

The action-value function $Q^{\pi}(s, a)$ represents the expected return when taking action a in state s and following policy π thereafter:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right] \quad (2.25)$$

Deep Q-Networks (DQN) [65] approximate $Q^*(s, a)$ with a neural network $Q(s, a; \theta)$ trained by minimizing the temporal difference (TD) loss:

$$\mathcal{L}(\theta) = \mathbb{E} \left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta) \right)^2 \right] \quad (2.26)$$

where θ^- denotes the parameters of a periodically updated target network that stabilizes training. DQN demonstrated that deep RL can solve complex control tasks, but assumes

the agent can freely explore the environment, an assumption that is problematic when exploration risks degrading real users’ Quality of Experience.

2.8.3 Offline Reinforcement Learning

In many real-world applications, including cloud gaming, online exploration is undesirable because deploying untested policies may degrade user experience. Offline RL [66] addresses this by learning policies entirely from a fixed dataset of previously collected transitions $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$, without any further interaction with the environment.

The central challenge of offline RL is *distribution shift*: the learned policy may favor actions that are rare or absent in the dataset, for which Q-value estimates are unreliable. Standard Q-learning can severely overestimate values for such out-of-distribution actions, leading to poor real-world performance.

Batch-Constrained Q-Learning (BCQ). BCQ [67] directly addresses distribution shift by constraining the learned policy to remain close to the behavior policy that generated the dataset. BCQ first learns a generative model $G(a|s)$ of the behavior policy via behavior cloning, then restricts the maximization in Q-learning to actions with sufficiently high likelihood under G :

$$\mathcal{A}^{\text{filtered}}(s) = \{a \in \mathcal{A} \mid G(a|s) > \tau\} \quad (2.27)$$

where τ is a threshold controlling how closely the learned policy must adhere to the data. The policy then selects the highest-Q action within this filtered set:

$$\pi_{\text{BCQ}}(s) = \arg \max_{a \in \mathcal{A}^{\text{filtered}}(s)} Q(s, a) \quad (2.28)$$

This prevents out-of-distribution extrapolation while still improving upon the behavior policy within the data support. In ARCADE, we adopt discrete BCQ because: (i) the action space (bitrate–resolution pairs) is inherently discrete; (ii) decisions occur only at GOP boundaries where fine-grained continuous control offers little benefit; and (iii) a finite action set keeps the offline policy within the logged dataset’s support.

Alternative Offline RL Methods. Other notable offline RL approaches include Conservative Q-Learning (CQL) [68], which regularizes Q-values to be pessimistic for out-of-distribution actions, and Twin Delayed DDPG (TD3) adapted for offline settings. In preliminary experiments for ARCADE, discrete BCQ was more stable than these alternatives for our discrete action catalog, motivating its selection.

2.9 Quality Metrics for Cloud Gaming

Evaluating the effectiveness of neural enhancement in cloud gaming requires both objective computational metrics and subjective human assessment. This section introduces the quality metrics used throughout this thesis.

2.9.1 Full-Reference Objective Metrics

Full-reference metrics compare a reconstructed or enhanced frame against the original uncompressed ground truth, quantifying distortion or perceptual similarity.

Peak Signal-to-Noise Ratio (PSNR). PSNR is the most widely used pixel-level quality metric, defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \quad (2.29)$$

where MAX is the maximum possible pixel value (255 for 8-bit images) and MSE is the mean squared error between the reference and reconstructed images. Higher PSNR indicates less distortion. While simple and widely comparable, PSNR does not account for the structural or perceptual characteristics of the human visual system, and two images with identical PSNR can differ substantially in perceived quality.

Structural Similarity Index (SSIM). SSIM [69] provides a more perceptually motivated assessment by comparing luminance, contrast, and structural information between images:

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (2.30)$$

where μ_x, μ_y are the local means, σ_x^2, σ_y^2 are the local variances, σ_{xy} is the cross-covariance, and C_1, C_2 are stabilization constants. SSIM values range from -1 to 1 , with 1 indicating perfect structural similarity.

Video Multi-Method Assessment Fusion (VMAF). VMAF [70], developed by Netflix, fuses multiple elementary quality metrics using a support vector machine trained on human subjective scores. VMAF accounts for visual information fidelity, detail loss, and motion characteristics, producing scores on a 0–100 scale where higher values indicate better quality. VMAF has been shown to correlate more strongly with human perception than PSNR or SSIM for video content [70], and is used as the primary quality metric in ARCADE’s reward function.

Learned Perceptual Image Patch Similarity (LPIPS). LPIPS [71] measures perceptual distance between images using deep features extracted from a pre-trained network (typically VGG or AlexNet). Lower LPIPS scores indicate greater perceptual similarity. Unlike PSNR and SSIM, which operate on raw pixel values, LPIPS captures high-level perceptual differences that align more closely with human judgments of image quality.

2.9.2 No-Reference Metrics

In cloud gaming, reference frames are often unavailable at the client because engine non-determinism, floating-point variability, multithreaded scheduling, and event-driven randomness prevent frame-level consistency between server-rendered originals and client-received frames [72]. No-reference (blind) quality models address this limitation by predicting quality from the distorted frame alone.

NDNetGaming. NDNetGaming [73] is a deep CNN-based no-reference quality model specifically tailored to gaming content, producing MOS-like scores. It was trained on gaming video sequences rated by human participants, making it well-suited for evaluating cloud gaming visual quality.

VSFA. The Video Quality Assessment with Subjectively and Objectively Annotated data (VSFA) model [74] combines a ResNet-50 feature extractor with a GRU temporal module for blind video quality prediction. It generalizes across content types and is used as a complementary no-reference assessment in our evaluations, particularly for comparisons with commercial upscalers where reference frames are not available.

2.9.3 Subjective Evaluation

Objective metrics, while valuable for systematic comparison, do not fully capture the quality as perceived by human players. Mean Opinion Score (MOS) studies, conducted following standardized protocols, provide direct measurements of perceived quality. In such studies, participants view content under controlled conditions and rate quality on a Likert scale (typically 1–5). MOS evaluations are employed in Chapters 3, 4, and 5 to validate that the objective quality improvements achieved by our solutions translate to perceptible improvements for human observers.

2.10 WebRTC for Real-Time Streaming

WebRTC (Web Real-Time Communication) is an open-source protocol suite for real-time audio and video communication, designed to operate directly in web browsers and mobile applications without requiring third-party plugins or software [75]. Originally open-sourced by Google in 2011 and subsequently standardized as a W3C Recommendation (finalized in 2021) with ongoing IETF maintenance, WebRTC has become the de facto standard for browser-based real-time media. Importantly, it is continuously updated rather than frozen at its original 2011 form: its reference implementation tracks Chrome releases and its Google Congestion Control algorithm has evolved over time. It underpins mainstream services such as Google Meet and Discord, as well as cloud gaming platforms, most notably Google Stadia, which used WebRTC for game streaming. Its low-latency design, built-in adaptive bitrate control, and peer-to-peer architecture make it well-suited for cloud gaming applications, and it forms the transport layer of the GameLab testbed developed in this thesis.

2.10.1 Key Components

WebRTC exposes three primary APIs that together handle media capture, transmission, and data exchange.

MediaStream manages synchronized streams of audio and video tracks from capture devices or other sources such as screen capture or programmatically generated content. Each track carries timing metadata that enables synchronization between audio and video at the receiver. In cloud gaming, the MediaStream abstraction is used to carry the encoded game video from server to client.

RTCPeerConnection is the core API handling peer-to-peer media transmission. It manages the full lifecycle of a media connection: ICE (Interactive Connectivity Establishment) negotiation to discover viable network paths through NATs and firewalls, DTLS (Datagram Transport Layer Security) handshake for encrypted transport, SRTP (Secure Real-time Transport Protocol) for encrypted media delivery, and ongoing connection maintenance including STUN and TURN server coordination. ICE gathers address candidates from the local network interface, STUN servers (which reveal the

public-facing address), and TURN relay servers (used as fallback when direct peer-to-peer paths are blocked). The ICE agent then performs connectivity checks to select the optimal path, prioritizing direct connections to minimize latency.

RTCDataChannel establishes bidirectional data channels over SCTP (Stream Control Transmission Protocol) tunneled through DTLS. Data channels can be configured for reliable ordered delivery (similar to TCP) or unreliable unordered delivery (similar to UDP), making them suitable for different cloud gaming data types: game input events benefit from unreliable delivery to minimize head-of-line blocking, while configuration messages may require reliability.

Signaling. WebRTC itself does not define a signaling protocol, delegating session negotiation to the application layer. Before a peer connection can be established, the two endpoints must exchange Session Description Protocol (SDP) offers and answers, which describe the supported codecs, resolutions, and transport parameters, as well as ICE candidates. This exchange typically occurs over WebSockets, HTTP, or SIP through a signaling server. The signaling server facilitates initial connection setup but does not relay media traffic, keeping it off the latency-critical path once the connection is established.

Media Transport. WebRTC transmits media over RTP (Real-time Transport Protocol) with RTCP (RTP Control Protocol) for feedback. RTP carries timestamped media packets with sequence numbers enabling reordering and loss detection at the receiver. RTCP carries feedback messages including receiver reports (packet loss, jitter), NACK (negative acknowledgment for retransmission requests), and PLI (Picture Loss Indication) to request keyframes after loss events. For cloud gaming, PLI handling is particularly important: packet loss triggering a keyframe request causes a brief quality drop as the encoder produces a large intra-coded frame, temporarily increasing latency and bitrate.

Congestion Control. WebRTC employs Google Congestion Control (GCC) to adapt sending rates based on estimated network capacity. GCC operates in two complementary modes: a delay-based controller that estimates available bandwidth from inter-packet arrival time variations, and a loss-based controller that reduces rate in response to packet loss. REMB (Receiver Estimated Maximum Bitrate) messages carry the receiver's bandwidth estimate back to the sender, which adjusts its encoding bitrate accordingly. This built-in adaptive bitrate mechanism is essential for cloud gaming, where network conditions can fluctuate significantly, and forms the baseline adaptation strategy against which ARCADE's joint optimization is compared in this thesis.

2.10.2 Application to Cloud Gaming

WebRTC is employed as the transport layer in the GameLab testbed for several reasons. Its design for real-time communication with sub-second latency aligns with cloud gaming's strict timing requirements. Built-in congestion control provides automatic adaptation to network conditions without requiring custom rate control implementation. Native support in browsers and availability of open-source native libraries such as aiortc (Python) and libwebrtc (C++) enable diverse client deployments and research-grade instrumentation. The open-source nature of these implementations allows modification

of internal components, including congestion control parameters and media pipeline hooks, necessary for integrating AI modules at the server and client stages.

However, standard WebRTC lacks two capabilities critical for AI-driven cloud gaming research: programmable insertion points for ML model integration within the media pipeline, and reliable mechanisms for aligning transmitted and received frames under packet loss and reordering for full-reference quality evaluation. GameLab addresses both limitations, extending the standard WebRTC stack with server-side and client-side hook points and a QR-based frame identity mechanism, capabilities described in detail in Chapter 6.

2.11 Summary and Research Gaps

This chapter has established the technical foundations for neural upsampling in cloud gaming: deep learning architectures for spatial and temporal processing, super-resolution techniques from classical to state-of-the-art, content complexity analysis for adaptive streaming, reinforcement learning methods for sequential decision-making, quality metrics for objective and subjective evaluation, cloud gaming system architecture and its challenges, and real-time streaming over WebRTC.

The review reveals several research gaps that motivate this thesis:

1. **Engine-Agnostic Real-Time SR (addressed in Chapter 3):** Commercial upscalers (DLSS, FSR, XeSS) require game engine integration unavailable in cloud gaming. Existing DNN-based SR models either achieve high quality at non-real-time speeds, SwinIR requires nearly 2 seconds per 1080p frame, or run in real time with inferior quality, as with RT4KSr at 15 ms but with a PSNR penalty of over 2 dB relative to state-of-the-art. There is a need for engine-agnostic models that achieve both high quality and real-time performance for gaming content.
2. **Joint Optimization Framework (addressed in Chapter 4):** Current cloud gaming systems treat rendering, encoding, and client-side processing as independent components. No framework jointly optimizes these stages to maximize QoE under bandwidth and compute constraints, leaving potential efficiency gains unrealized.
3. **VR-Specific SR (addressed in Chapter 5):** Virtual reality cloud gaming introduces unique challenges, stereo views, stricter latency budgets, magnified artifacts, not addressed by existing SR approaches designed for 2D content. Furthermore, existing stereo SR methods such as CVHSSR achieve high quality but require nearly 3 seconds per stereo pair, making them unsuitable for interactive use.
4. **Research Infrastructure (addressed in Chapter 6):** Existing cloud gaming testbeds such as GamingAnywhere target deployment rather than research extensibility, lacking programmable hooks for AI model integration and robust mechanisms for quality evaluation under realistic network conditions.

The subsequent chapters address these gaps systematically: Chapter 3 presents GameSR, a lightweight engine-agnostic SR model; Chapter 4 develops ARCADE, a joint optimization framework using offline RL; Chapter 5 extends neural upsampling to VR with iSR; and Chapter 6 introduces GameLab, the research infrastructure underpinning all evaluations.

Chapter 3

GameSR: Real-Time Super-Resolution for Interactive Gaming

H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “GameSR: Real-Time Super-Resolution for Interactive Gaming,” under review at IEEE Transactions on Games, 2026.

3.1 Introduction

High-resolution gaming demands significant computational resources, with challenges further amplified by bandwidth and latency constraints in cloud gaming. As discussed in Chapter 2, existing upscalers such as NVIDIA DLSS and AMD FSR reduce rendering costs but require engine integration, making them unavailable for most titles, especially those released before the introduction of such upscalers. This chapter presents **GameSR**, a lightweight, engine-independent super-resolution model that operates directly on encoded game frames, addressing the first research question of this thesis: *How can we design a real-time, engine-agnostic neural upsampler for cloud gaming that achieves comparable quality to engine-coupled solutions?*

The key challenge in designing such a system is meeting strict latency constraints while achieving high visual quality. Modern games already push frame budgets to the limit, often disabling effects like motion blur or ambient occlusion at higher frame rates. Any additional processing risks introducing stutter or added input-to-display delay. Thus, an effective upscaler must be engine-agnostic, efficient, and carefully integrated to deliver perceptual gains without breaking interactivity.

GameSR addresses these challenges through a novel architecture that combines reparameterized convolutional blocks, PixelUnshuffle for efficient feature extraction, and a lightweight ConvLSTM for temporal modeling. The design exploits the unique characteristics of gaming content, its synthetic, repetitive, and structured nature, to achieve real-time upscaling with high perceptual quality.

The main contributions of this chapter are as follows:

1. A lightweight neural super-resolution model that operates directly on rendered frames **without requiring access to game source code or game engine data structures**, making it readily deployable in cloud gaming for both modern and legacy games.

2. Demonstration that GameSR achieves **near-parity with industrial upscalers** (DLSS, FSR) on no-reference perceptual metrics, despite using no motion vectors or depth buffers.
3. GameSR matches state-of-the-art quality while running **30–60× faster** than CNN baselines (EDSR, IMDN) and nearly **500× faster** than SwinIR, with up to an **order-of-magnitude smaller** model size and memory footprint.
4. Deployment in a real-time WebRTC cloud-gaming testbed demonstrating that streaming at lower resolutions and upscaling with GameSR achieves the same perceptual quality at **35–49% lower bitrate**.

3.2 Problem Definition and Design Goals

3.2.1 Limitations of Existing Approaches

As established in Chapter 2, existing game upscalers face several critical limitations that prevent their use in cloud gaming:

Engine-Integrated Upscalers. As detailed in Chapters 1 and 2, industrial upscalers (DLSS [58], FSR [18], XeSS [19]) require access to engine-internal data structures (motion vectors, depth buffers, temporal history), limiting them to fewer than 1% of Steam’s catalog [76, 77, 78] and excluding legacy and forward-rendered titles entirely.

General-Purpose SR Models. Prior work has proposed numerous SR models including EDSR [40], LapSRN [43], IMDN [48], LatticeNet [49], and SwinIR [23]. These lightweight models employ strategies like information distillation, feature distillation, Laplacian pyramids, and residual-attention mechanisms. However, they target general efficiency rather than the millisecond-level latency demands of cloud gaming. Even IMDN, the most efficient among these, takes over 120 ms to upscale a frame by $2\times$ on a high-end GPU, far exceeding real-time limits. Recurrent VSR methods such as RLSP [79], MRVSR [80], and SSL-pruned BasicVSR [81] use heavier backbones with fixed $4\times$ scaling on small inputs and report tens of milliseconds per frame. In contrast, GameSR targets $2\times$ and $3\times$ upscaling of full HD streams (540p/720p→1080p) in 4–5 ms with only 138K parameters, a more suitable operating point for real-time gaming.

Efficient Transformer and Hybrid Models. A recent line of work seeks to make transformer-based SR practical by combining convolution and self-attention. Efficient Mixed Transformer (EMT) [82], for example, interleaves convolutional blocks with lightweight mixed-attention modules to raise quality at reduced cost, and broader CNN–transformer hybrids follow the same philosophy of using convolutions for local features and attention for long-range context. While these designs narrow the efficiency gap relative to pure transformers such as SwinIR, they remain ill-suited to real-time cloud gaming for three reasons. First, they retain self-attention as a core component and therefore inherit its drawbacks discussed in Chapter 2: quadratic scaling in the number of tokens, irregular memory access that is poorly accelerated on commodity and mobile hardware, and a weaker inductive bias that demands more training data than purely

convolutional models. Second, they are designed and benchmarked for classical single-image SR on small natural-image patches (e.g., DIV2K), not for the millisecond-budget full-HD inference that gaming requires; their reported “efficiency” is relative to other transformers rather than to the real-time bar set by interactive gaming. Third, being single-image methods, they ignore the temporal continuity of gameplay that GameSR exploits through lightweight recurrence. In short, hybrid and efficient-transformer SR improve the quality–cost trade-off *within* the transformer family but still do not meet the simultaneous real-time, engine-agnostic, and temporally-consistent requirements of cloud gaming.

Cloud Gaming Constraints. In cloud gaming, clients receive only compressed video streams, so industrial upscalers and rendering-coupled research models cannot be applied, they require motion vectors, depth, and other engine-level data unavailable client-side. Even if executed server-side, such methods would not reduce streaming bitrate since frames must still be transmitted at display resolution. While VOD streaming has explored SR integration via segment-specific micro-models [83, 84, 85], this approach is infeasible for interactive cloud gaming where frames are generated in real time based on player inputs.

Client Device Feasibility. Most client devices possess underutilized compute capable of running lightweight upscalers: smartphones include powerful NPUs (iPhone 16 Pro: 35 TOPS; Dimensity 9400: 50 TOPS [86, 87]), while consoles such as the PS5 and Xbox Series X offer over 10 TFLOPS [88, 89]. However, heavy VSR models remain impractical due to power, thermal, and latency constraints. GameSR is designed to exploit this available client-side compute with a lightweight, full-frame, engine-agnostic architecture that runs within milliseconds on both mobile and desktop hardware.

3.2.2 Design Goals

Based on these limitations, we establish the following design goals for GameSR:

1. **Engine-Agnostic Operation:** The model must operate directly on rendered RGB frames without requiring access to game source code, motion vectors, depth buffers, or other engine-internal data structures.
2. **Real-Time Performance:** Inference latency must remain below 10 ms per frame to support 60+ FPS gaming with minimal impact on end-to-end latency.
3. **High Visual Quality:** Reconstruction quality should match or approach state-of-the-art methods and commercial upscalers on perceptual metrics.
4. **Temporal Consistency:** The model must maintain temporal stability to avoid flickering artifacts that degrade the gaming experience.
5. **Deployment Flexibility:** The architecture should be lightweight enough to run on diverse client hardware, from high-end GPUs to mobile devices.

3.2.3 Exploiting Gaming Content Characteristics

Unlike camera-captured video, game frames are *synthetic* outputs of a graphics engine, exhibiting visual statistics and streaming behavior that differ from natural video [90]. More importantly for super-resolution, a game is a largely *closed visual world*: it reuses a finite library of meshes, textures, shaders, and HUD elements across maps and stages, while viewpoints and motions are drawn from a small set of camera rigs (first-person, third-person) and a fixed action space (move, jump, shoot, ability casts).

This bounded, repetitive structure makes high-frequency details (edges, text, UI glyphs, weapon geometry) highly recurring and therefore learnable from representative gameplay. GameSR leverages this property by training per title: the model learns game-specific priors that generalize across unseen matches of the same game, enabling real-time reconstruction quality that is difficult to achieve with generic SR models trained for open-world natural video.

Is per-title training realistic? We argue that it is, particularly in the cloud gaming setting this thesis targets. First, the cost of producing a per-title model is low: roughly ten representative gameplay sessions suffice for training (Section 3.4.1), the model is compact (138K parameters), and a single training run yields both the desktop and mobile variants, so a new title can be onboarded with a few hours of one-time, offline effort. Second, the economics favour it: a cloud gaming provider hosts a finite, curated catalogue on infrastructure it controls, so each title need only be trained once, and the resulting model, a few hundred kilobytes, is stored in a server-side model bank and reused across millions of streamed sessions, amortizing the one-time training cost to a negligible amount per session. This mirrors established industry practice, where per-title optimization and certification are routine and early upscalers such as DLSS 1.0 were themselves trained per game. Third, one model per game is an upper bound rather than a strict requirement: our cross-game experiments (Section 3.5.6) show that a single model trained on multiple titles of the same genre performs strongly across them, so one model can serve a family of related games, while the meta-learning and genre-conditioned directions outlined in Chapter 7 aim to reduce the per-title cost further still.

3.3 GameSR Architecture

3.3.1 Overview and Operation

GameSR is designed as an engine-independent, lightweight super-resolution model for both traditional and cloud gaming. In traditional gaming, GameSR applies as a post-processing step after rendering, enhancing frames before display. In cloud gaming (Figure 3.1), frames are rendered on the server, compressed, and transmitted to clients. GameSR is inserted between the decoder and display, transparently upscaling frames in real-time as they arrive.

Beyond improving perceived quality, GameSR offers three advantages for cloud gaming:

1. **Reduced server load:** Operating at lower resolutions reduces rendering and encoding cost.
2. **Lower transmission bitrate:** Fewer pixels are streamed, reducing bandwidth requirements.



Figure 3.1: GameSR in cloud gaming: low-resolution streams are rendered at server-side and upscaled at client-side in real-time.

3. **No integration required:** Deployment requires no modification to game engines or decoders, enabling use with both modern and legacy games.

This design is also codec-agnostic: any title or streaming service outputting RGB video can benefit without depending on vendor-specific upscaling stacks.

3.3.2 Model Formulation

GameSR, and SR models in general, reconstruct high-resolution (HR) frames from low-resolution (LR) inputs by optimizing a parameterized function F :

$$\theta^* = \arg \min_{\theta} \sum L(F(y^{LR}; \theta), y^{HR}) \quad (3.1)$$

The function F performs four main stages in our design:

1. **Feature Representation:** Transform input frames into a compact feature space.
2. **Feature Learning:** Capture non-linear mappings between LR and HR features.
3. **Temporal Learning:** Leverage information from adjacent frames.
4. **Upsampling:** Map features to HR output.

Figure 3.2 illustrates the complete architecture.

3.3.3 Feature Representation Stage

The feature representation stage first applies PixelUnshuffle (space-to-depth) [32] to the input frame, reducing spatial dimensions by a factor of s and expanding channels by s^2 , followed by a single 3×3 convolution. Unlike conventional SR methods that process features at full resolution, this down-and-up scheme significantly reduces computational cost while capturing richer channel-wise feature relationships:

$$F_1(y^{LR})_{f_1 \times \frac{H}{s} \times \frac{W}{s}} = \max \left(0, W_1 * \text{PixelUnshuffle}(y^{LR}) + B_1 \right) \quad (3.2)$$

where F_1 is the output feature map of this first (feature-representation) stage, comprising f_1 channels at the reduced spatial resolution $\frac{H}{s} \times \frac{W}{s}$; W_1 and B_1 are the weights and biases of the 3×3 convolution; and the outer $\max(0, \cdot)$ applies a ReLU activation. This feature map F_1 is the input to the subsequent Feature Extraction Blocks. PixelUnshuffle with scale factor s rearranges the input from $c \times H \times W$ into $c \cdot s^2 \times \frac{H}{s} \times \frac{W}{s}$, increasing the channel dimension by s^2 while reducing spatial dimensions.

This approach offers several advantages: (i) convolutions operate on smaller spatial dimensions, reducing FLOPs; (ii) the expanded channel dimension captures richer feature relationships; and (iii) spatial information is preserved in the channel dimension rather than discarded.

3.3.4 Feature Extraction Blocks

The feature learning stage captures non-linear mappings between LR and HR features using our Feature Extraction Block (FEB). During training, each FEB processes features through a $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$ convolution sequence:

$$F_{out} = F_{1 \times 1}^{\text{compress}} \left(F_{3 \times 3} \left(F_{1 \times 1}^{\text{expand}} (F_{in}) \right) \right) \quad (3.3)$$

where an initial 1×1 convolution expands features from C to $2C$ channels, a 3×3 convolution operates in this expanded space, and a final 1×1 convolution reduces back to C channels.

Reparameterization for Inference. To optimize inference performance, we leverage structural reparameterization [33] to collapse the three convolutions into a single equivalent operation:

$$F_{out} = F_{3 \times 3}^{\text{reparam}} (F_{in}) \quad (3.4)$$

The reparameterization process combines the weights:

$$W_{3 \times 3}^{\text{reparam}} = W_{1 \times 1}^{\text{compress}} * W_{3 \times 3} * W_{1 \times 1}^{\text{expand}} \quad (3.5)$$

This transformation preserves the learned mapping while significantly reducing computational overhead during real-time processing. Our ablation study (Section 3.5.5) shows that reparameterization reduces parameters by 54% (298K $\rightarrow$ 138K) and improves inference from 6.29 ms to 4.12 ms while maintaining identical quality metrics.

Activation and Normalization. Each FEB incorporates GELU activation [30], defined as:

$$\text{GELU}(x) = x \cdot \Phi(x) \quad (3.6)$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, providing smooth gradient flow during training. For normalization, we employ LayerNorm [91]:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (3.7)$$

where γ and β are learnable parameters, ensuring stable training behavior.

Multi-Level Feature Aggregation. After extracting features through multiple FEBs, we employ additive fusion to combine representations at different abstraction levels:

$$RB_{\text{final}} = \sum_{i=0}^N RB(i) \quad (3.8)$$

where N represents the number of FEBs and $RB(i)$ denotes the output features from the i -th FEB. This additive combination offers several advantages: (i) direct additive connections create shorter paths for gradient propagation, helping mitigate the vanishing gradient problem; (ii) each subsequent layer can access and build upon features extracted at all previous levels; and (iii) unlike concatenation-based approaches that increase feature dimensionality, addition maintains constant feature dimension while preserving multi-level information.

Table 3.1: Feature extraction block architectural exploration on CS2. All variants use 16 base filters.

Variant	Mem (GB)	Params	FLOPs (B)	PSNR	SSIM	VMAF
ResBlock	0.76	31 043	0.333	32.30	0.9758	86.01
+ DepthWise	0.01	5 262	0.018	31.70	0.9755	82.15
+ SENet	0.89	31 299	0.333	32.25	0.9756	85.97
+ CBAM	0.89	31 699	0.333	32.24	0.9752	85.18
+ ESA ($r=2$)	1.41	41 795	0.354	32.34	0.9759	86.19
+ ESA ($r=4$)	1.27	34 083	0.341	32.33	0.9759	86.13
+ ESA ($r=8$)	1.21	32 003	0.336	32.29	0.9758	85.78
1-Conv	0.65	21 763	0.237	32.16	0.9756	85.26
No outer Conv	0.74	28 723	0.309	32.25	0.9757	85.35
1-Conv + No outer	0.61	19 443	0.214	32.17	0.9756	84.69
ResNeXt-16	0.77	22 403	0.333	32.16	0.9757	85.24
ResNeXt-18	0.86	28 119	0.418	32.18	0.9757	85.56
ResNeXt-20	0.95	34 483	0.513	32.24	0.9758	85.37
ResNeXt-24	1.12	49 155	0.732	32.25	0.9759	86.27
ConvNeXt-16	3.03	24 195	0.662	32.09	0.9756	85.36

Architectural Exploration. Before settling on the final FEB design, we evaluated 15 feature extraction block variants on CS2 gameplay sequences, including attention-augmented blocks (SENet, CBAM, ESA), group convolutions (ResNeXt), depthwise convolutions (ConvNeXt), and several ResBlock simplifications. Table 3.1 summarizes the results.

Several findings guided our design. SENet and CBAM attention blocks increase parameters without improving quality at our small filter size, likely because the channel attention lacks sufficient capacity to be effective. ESA with reduction factor 4 yields marginal PSNR gains (+0.03 dB) but at 67% higher memory, making it unsuitable for latency-sensitive deployment. ResNeXt-24 matches ResBlock quality but requires $3.5\times$ more memory and higher FLOPs, and notably, ResNeXt with fewer parameters can still incur *higher* inference time than ResBlock due to the computational overhead of group convolutions. ConvNeXt consumes $4\times$ the memory of ResBlock for lower quality. These results justified the standard ResBlock topology as the basis for our reparameterizable $1\times 1-3\times 3-1\times 1$ design.

GAN-Based Fine-Tuning. We also explored adversarial training to improve perceptual texture quality. Using GameSR as the generator, we designed a game-object-driven content loss where the VGG19 feature extractor was trained on a GameNet dataset containing categorically annotated gaming assets (weapons, props, environmental elements, effects). Compared against SRGAN [41] and ESRGAN [44] using the Perceptual Index (PI, lower is better), our game-specific content loss achieved the best score (PI: Ours 2.94, ESRGAN 3.31, SRGAN 4.79), confirming that domain-specific feature representations improve perceptual quality on gaming content. However, we ultimately did not adopt the GAN-based approach due to several limitations: (i) training instability—when the

discriminator reaches high confidence, the generator’s gradient vanishes, causing training collapse; (ii) artificial traces and unnatural texture deformations in generated frames; and (iii) while GAN-upscaled frames appear visually appealing in isolation, side-by-side comparison with ground truth reveals texture hallucination, which is unacceptable for competitive gaming where visual fidelity directly affects gameplay. We therefore employ pixel-based and Charbonnier losses, which provide consistent, artifact-free reconstruction better suited to the fidelity-critical cloud gaming use case.

3.3.5 Temporal Learning with ConvLSTM

Video super-resolution leverages temporal information across frames to enhance quality, making it especially relevant for gaming sequences. Unlike single-image SR, video SR exploits motion continuity through either explicit (optical flow) or implicit (3D/deformable convolutions) alignment. However, both alignment strategies are too computationally heavy for real-time deployment, and quantifying their cost makes the gap concrete. Explicit alignment first estimates a dense optical-flow field and then warps neighbouring frames; in our measurements the flow-estimation step *alone* consumes roughly 14–16 ms per frame, already exhausting the entire 16.7 ms budget available at 60 fps before any super-resolution is performed. Implicit alignment via deformable convolutions avoids explicit flow but is also expensive: each deformable layer adds an offset-prediction convolution (producing $2k^2$ offset values per $k \times k$ kernel, plus modulation masks in the v2 formulation) followed by irregular, non-contiguous memory sampling, making it roughly $2\text{--}3\times$ slower than a standard convolution of the same kernel size and channel count and poorly supported on commodity and mobile accelerators. Compounding this, the backbones that host these alignment modules are large: EDVR [54] uses about 20.6 M parameters and BasicVSR++ [81] about 7.3 M, one to two orders of magnitude beyond GameSR’s 138 K, and they report per-frame latencies in the tens of milliseconds. Because sustained real-time operation is the central design goal of this thesis, GameSR deliberately avoids both optical-flow and deformable-convolution alignment, instead capturing temporal continuity with a lightweight ConvLSTM built entirely from standard convolutions, as described next.

To balance temporal modeling and efficiency, we adopt a lightweight variant of ConvLSTM [35] after feature extraction. ConvLSTM replaces matrix multiplications in standard LSTMs with convolutions, preserving spatial resolution while capturing long-range dependencies. The ConvLSTM operates over spatial features with dimensions $(C, H/s, W/s)$ and uses a single-layer structure with decoupled gates (input, forget, output, and cell), each implemented with independent 2D convolutions:

$$i_t = \sigma(W_i * [x_t, h_{t-1}] + b_i) \quad (3.9)$$

$$f_t = \sigma(W_f * [x_t, h_{t-1}] + b_f) \quad (3.10)$$

$$o_t = \sigma(W_o * [x_t, h_{t-1}] + b_o) \quad (3.11)$$

$$\tilde{c}_t = \tanh(W_g * [x_t, h_{t-1}] + b_g) \quad (3.12)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (3.13)$$

$$h_t = o_t \odot \tanh(c_t) \quad (3.14)$$

where $*$ denotes 2D convolution and $[\cdot, \cdot]$ is channel-wise concatenation.

This modular design enables better parallelization on modern GPUs while minimizing sequential overhead. During inference, frames are processed sequentially using

hidden states from prior frames, enabling effective motion-aware upsampling. Our ablation study shows that removing ConvLSTM causes PSNR to drop approximately 5 dB (from 37.99 dB to 32.99 dB on CS2), demonstrating the critical importance of temporal modeling for gaming content.

3.3.6 Upsampling Stage

The upsampling stage utilizes PixelShuffle for spatial resolution enhancement, avoiding checkerboard artifacts common in deconvolution methods [92]. This approach reshapes feature channels into spatial dimensions efficiently. We incorporate a residual connection by combining upsampled ConvLSTM output with the original input, preserving fine details and textures:

$$\hat{y}_{c \times H \times W}^{SR} = \text{Conv}\left(\text{PixelShuffle}\left(\max\left(0, W_{\text{up}} * (RB)_{f_1 \times H \times W} + B_{\text{up}}\right)\right)\right) \quad (3.15)$$

The global residual connection adds the bicubic-upsampled input to the network output, allowing the model to focus on learning high-frequency residuals rather than reconstructing the entire image from scratch.

3.3.7 Mobile Variant: GameSR-M

To support deployment on mobile devices and SoCs with limited compute and memory, we design **GameSR-M**, a mobile-optimized variant that restricts operations to those natively accelerated on mobile NPUs and GPUs. Specifically, GameSR-M avoids operations with poor mobile support such as PixelUnshuffle and replaces them with depthwise separable convolutions. The ConvLSTM gates are implemented with pointwise convolutions to minimize memory bandwidth. These changes reduce per-operation latency on mobile hardware at the cost of a modest quality reduction relative to the full GameSR model, as quantified in Section 3.5.9. The standard GameSR model and GameSR-M share the same training procedure and game-specific data pipeline; only the inference graph differs, allowing a single training run to produce both variants.

3.4 Training Methodology

3.4.1 Dataset Collection

We evaluate GameSR on four distinct games spanning different genres and visual styles:

- **Counter-Strike 2 (CS2):** A realistic tactical FPS on Valve’s Source 2 engine with fine geometric detail and high-contrast lighting.
- **Overwatch 2 (OW2):** A fast-paced hero shooter on Blizzard’s engine with stylized characters and dense particle effects.
- **FIFA 24 (FC24):** A sports simulation with fast camera movements and detailed player models.
- **Team Fortress 2 (TF2):** A legacy game (2007) serving as a test case for older titles.

Using VirtualDub [93], we captured uncompressed 1080p gameplay at 30/60 FPS across diverse maps, motions, camera dynamics, and lighting conditions. Five players of varying skill levels recorded five sessions each per game (25 sessions per title), yielding 40k frames for CS2, 54k for OW2, and 30k for TF2, plus an additional FC24 corpus of comparable scale covering representative match scenarios (e.g., build-up play, counterattacks, set pieces, and goal events). We used 10 sessions per game for training and 15 unseen sessions for testing.

3.4.2 Loss Function

Beyond architectural choices, the training objective affects both convergence and reconstruction quality. While prior SR work commonly uses MSE (L2) or MAE (L1), we optimize GameSR with the Charbonnier loss [94]:

$$\mathcal{L}_{\text{char}} = \mathbb{E}_{(z,y) \sim P_{\text{data}}} [\rho(y - G(z))], \quad \rho(x) = \sqrt{x^2 + \varepsilon^2} \quad (3.16)$$

Here z denotes the low-resolution input, y the high-resolution ground-truth frame, $G(z)$ the super-resolved output of GameSR, and $y - G(z)$ the per-pixel reconstruction residual; the penalty $\rho(\cdot)$ is applied element-wise and averaged over the data distribution P_{data} . The constant ε is a small positive value that smooths the penalty at the origin: whereas the L1 (MAE) penalty $|x|$ has a non-differentiable kink at $x = 0$, the Charbonnier function $\sqrt{x^2 + \varepsilon^2}$ is differentiable everywhere, and as $\varepsilon \rightarrow 0$ it recovers L1 exactly. Its gradient, $\rho'(x) = x / \sqrt{x^2 + \varepsilon^2}$, is bounded in $[-1, 1]$: it grows approximately linearly for small residuals (behaving like L2 and giving smooth, stable updates that refine sub-pixel detail) and saturates toward ± 1 for large residuals (behaving like L1 and preventing a few large errors from dominating the gradient). This combination is particularly well suited to gaming content. The L2 (MSE) loss penalizes large errors quadratically and therefore over-weights outliers, driving the network toward the blurry conditional mean and washing out exactly the high-frequency structures, edges, HUD text, and fine textures, that matter most in games; L1 preserves such edges better but its constant-magnitude gradient and non-smoothness near zero slow fine convergence. The Charbonnier loss retains L1’s robustness to outliers, and hence its edge- and detail-preserving behaviour, while restoring a smooth gradient near zero for stable optimization. In our experiments, this consistently yielded better training dynamics and sharper reconstructions than either MSE or MAE alone across all four games.

3.4.3 Training Configuration

We trained GameSR in PyTorch 2.0.1 on an NVIDIA RTX A4000 with an Intel Xeon Gold 5220 CPU and 32 GB RAM. Training used:

- Optimizer: AdamW [95] ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
- Learning rate: 10^{-3} , halved every 2×10^5 iterations
- Batch size: 16
- Data split: 80/20 for training/validation

Table 3.2: Quantitative comparison between state-of-the-art super-resolution models and GameSR at $2\times$ scaling on four popular games. Evaluated on NVIDIA RTX A4000 GPU.

Model	Inference (ms)	Counter-Strike 2			Overwatch 2			Team Fortress 2			FIFA24		
		PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
Bicubic	–	32.50	0.998	0.189	35.22	0.999	0.128	34.07	0.998	0.128	31.82	0.997	0.190
SwinIR	1971.7	35.92	0.998	0.084	40.74	0.999	0.016	40.10	0.999	0.046	34.91	0.998	0.155
LapSRN	239.7	33.63	0.998	0.107	38.09	0.999	0.030	36.77	0.998	0.144	33.48	0.998	0.172
EDSR	160.0	35.30	0.998	0.091	40.16	0.999	0.018	39.41	0.999	0.050	34.56	0.998	0.157
LatticeNet	154.4	35.46	0.998	0.088	40.29	0.999	0.017	39.36	0.999	0.050	34.71	0.998	0.159
IMDN	121.2	35.38	0.998	0.089	40.33	0.999	0.018	39.36	0.999	0.050	34.71	0.998	0.154
RT4KSR	15.0	33.74	0.998	0.154	37.11	0.999	0.101	36.04	0.999	0.093	33.35	0.998	0.163
GameSR	4.12	37.99	0.999	0.095	40.36	0.999	0.021	40.88	0.999	0.051	34.84	0.998	0.143

The ConvLSTM hidden state is reset between training clips to prevent temporal state leakage across unrelated sequences, ensuring the model learns to initialize effectively from a cold start as occurs at the beginning of each streaming session. For deployment, we compiled the model with Torch-TensorRT using kernel fusion and mixed precision (FP32 inputs, FP16 kernels) to improve throughput and memory efficiency.

3.5 Evaluation

3.5.1 Performance Metrics

We evaluate quality using commonly used metrics:

- **PSNR and SSIM:** Pixel-based metrics measuring reconstruction fidelity.
- **VMAF:** Netflix’s perceptual quality metric correlating with human judgment [96].
- **LPIPS:** Learned perceptual metric based on deep features [97].

In gaming, reference frames are often unavailable due to engine non-determinism, floating-point variability, and event-driven randomness, which prevent frame-level consistency [98]. Thus, we also employ two no-reference models for comparisons with commercial upscalers:

- **NDNetGaming:** Tailored to gaming content with MOS-like scores [99].
- **VSFA:** A ResNet-50+GRU model for blind quality assessment [74].

3.5.2 Comparison with State-of-the-Art SR Models

Before comparing with other models, we first demonstrate GameSR’s consistent performance across diverse gameplay sessions. Figure 3.3 shows the distribution of quality metrics when upscaling sessions from CS2 and OW2 by a factor of $2\times$. GameSR consistently achieves high-quality results: PSNR ranges from 36–40 dB, SSIM exceeds 0.998, and VMAF scores fall within the 90–95 range, indicating excellent quality [96].

Table 3.2 presents a quantitative comparison between GameSR and state-of-the-art super-resolution models at $2\times$ scaling on four popular games. All experiments were conducted on an NVIDIA RTX A4000 GPU.

Table 3.3: Comparison of GameSR against IMDN and IMDN-G (IMDN retrained on CS2), isolating the contribution of game-specific training.

Model	PSNR	SSIM	LPIPS
IMDN	35.37	0.9988	0.0898
IMDN-G	38.17	0.9990	0.0835
GameSR	37.99	0.9990	0.0954

Table 3.4: Quantitative comparison at $3\times$ scaling on four popular games. Evaluated on NVIDIA RTX A4000 GPU.

Model	Inference (ms)	Counter-Strike 2			Overwatch 2			Team Fortress 2			FIFA24		
		PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
SwinIR	881.2	32.53	0.996	0.178	36.79	0.998	0.052	36.38	0.998	0.106	31.41	0.992	0.264
LatticeNet	70.5	32.25	0.996	0.189	36.42	0.999	0.056	35.81	0.998	0.113	31.20	0.992	0.268
EDSR	76.6	32.26	0.996	0.188	36.32	0.999	0.059	35.82	0.998	0.113	31.13	0.992	0.278
IMDN	55.2	32.29	0.996	0.186	36.40	0.999	0.055	35.85	0.998	0.110	31.20	0.992	0.272
RT4KSR	9.0	30.99	0.995	0.253	33.75	0.997	0.194	33.17	0.997	0.174	29.77	0.991	0.279
GameSR	4.09	35.46	0.996	0.180	36.99	0.998	0.059	38.10	0.998	0.111	31.23	0.992	0.253

GameSR achieves near state-of-the-art quality (PSNR/SSIM/LPIPS) while running orders of magnitude faster. GameSR reaches $\sim 240+$ fps (~ 4.1 ms/frame), compared to <10 fps for EDSR/LatticeNet and <1 fps for SwinIR. While SwinIR attains the best perceptual scores due to its Transformer-based design, its computational cost (1971.7 ms/frame) prevents deployment in latency-sensitive settings. RT4KSR [100], the only other real-time candidate, yields noticeably lower quality on gaming content.

To further validate the benefit of game-specific training independently of architectural choices, we retrained IMDN on our CS2 dataset (IMDN-G). As shown in Table 3.3, game-specific training substantially closes the quality gap relative to the generic IMDN model. GameSR achieves comparable quality to IMDN-G with only a 0.18 dB PSNR difference, while using $5\times$ fewer parameters and $4.5\times$ less GPU memory, demonstrating that both the data-centric training strategy and the architectural efficiency of GameSR are independently valuable.

Table 3.4 presents results for $3\times$ scaling (360p $\rightarrow$ 1080p). The trends mirror those observed at $2\times$: GameSR delivers quality on par with state-of-the-art SR models while being orders of magnitude faster.

3.5.3 Comparison with Commercial Upscalers

We compare GameSR against industry-standard upscalers DLSS and FSR using no-reference quality metrics, as shown in Table 3.5. FSR and DLSS were applied using in-game settings, whereas we rendered frames natively at 540p and upscaled them using GameSR. This setup is conservative for GameSR: while DLSS and FSR have access to additional renderer data (motion vectors, depth buffers), GameSR relies solely on input frames. Specifically, we compared GameSR to FSR 1.0 in CS2 and FSR 2.2/DLSS 3.5 in OW2, all using “Performance” mode ($2\times$ upsampling) to match GameSR’s scaling factor.

GameSR achieves near-parity with FSR and DLSS in perceptual quality, with differences of only 0.1 (NDNetGaming) and 0.06 (VSFA) in CS2, and within 0.05 (FSR) and 0.17 (DLSS) for NDNetGaming in OW2. Crucially, GameSR’s engine-independent

Table 3.5: Comparing GameSR against DLSS and FSR, which require engine-level data. In contrast, GameSR upscales encoded streams. Results shown for $2\times$ scaling.

Model	Counter-Strike 2		Overwatch 2		Team Fortress 2	
	NDNet $\uparrow$	VSFA $\uparrow$	NDNet $\uparrow$	VSFA $\uparrow$	NDNet $\uparrow$	VSFA $\uparrow$
DLSS	–	–	4.93	0.88	–	–
FSR	5.00	0.89	4.81	0.81	–	–
GameSR	4.90	0.83	4.76	0.81	4.79	0.78

Table 3.6: Ablation study of GameSR on CS2, showing the impact of ConvLSTM, PixelUnshuffle, and Reparameterization on efficiency and quality.

Model	#Params (K)	Memory (MiB)	Inference (ms)	PSNR	SSIM	LPIPS
No ConvLSTM	65	436	3.05	32.99	0.998	0.116
No PixelUnshuffle	125	1174	13.13	38.65	0.999	0.087
No Reparameterization	298	608	6.29	37.99	0.998	0.095
GameSR (Final)	138	604	4.12	37.99	0.998	0.095

nature makes it deployable on titles like CS2 and TF2 where modern upscalers (DLSS 2+, FSR 2+) cannot be adopted without changes to the rendering pipeline.

Figure 3.4 shows side-by-side comparisons of upscaled frames from DLSS, FSR, and GameSR on Overwatch 2 sequences. The first, second, and third rows correspond to the Junktown, Esperança, and Nepal maps, respectively. We include OW2 here since it natively supports both DLSS and FSR, allowing direct visual comparison against GameSR.

3.5.4 Model Efficiency Analysis

GameSR’s efficiency comes from three design choices: ConvLSTM captures temporal dependencies, reparameterization enables wide training but lightweight inference, and PixelUnshuffle reduces spatial processing cost. Figure 3.5 compares parameter counts and GPU memory usage.

GameSR uses only 138K parameters and 604 MiB memory, compared to 1.37M/11.9 GiB for EDSR and 910K/3.9 GiB for SwinIR. IMDN and LatticeNet also require 5–6 $\times$ more memory. At $\times 2$ scale (and similarly at $\times 3$), GameSR achieves order-of-magnitude savings in size and memory over state-of-the-art upscalers.

3.5.5 Ablation Study

To validate the contributions of each major component, we performed an ablation study removing ConvLSTM, PixelUnshuffle, and Reparameterization one at a time. Table 3.6 compares inference time, memory, parameters, and quality metrics.

ConvLSTM. Temporal processing proves critical: removing ConvLSTM causes the largest quality degradation of any component (~ 5 dB PSNR), despite reducing parameters to 65K and inference to 3.05 ms. This confirms the importance of temporal modeling noted in Section 3.3.5.

Table 3.7: Cross-game generalization of GameSR. Models trained on CS2, OW2, and a combined dataset are evaluated on both games.

Test Sequence	Trained on CS2		Trained on OW2		Trained on CS2+OW2	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
CS2	37.80	0.999	34.63	0.999	37.53	0.999
OW2	37.17	0.999	38.81	0.999	38.74	0.999

Table 3.8: Super-resolution performance of GameSR on high-resolution clips. All experiments conducted on NVIDIA A4000 GPU.

Source $\rightarrow$ Target	Time (ms)	PSNR (dB)	SSIM	VMAF
720p $\rightarrow$ 2K	7.0	36.24	0.998	91
1080p $\rightarrow$ 4K	14.3	39.25	0.999	93
720p $\rightarrow$ 4K	15.3	34.83	0.999	88.13

PixelUnshuffle. This component balances quality and efficiency. Without it, the model achieves the highest quality (PSNR: 38.65 dB), but inference time triples to 13.13 ms and memory nearly doubles to 1174 MiB, impractical for real-time applications.

Reparameterization. As described in Section 3.3.4, collapsing the three-convolution FEB into a single equivalent operation reduces parameters by 54% and improves inference by 34% without any quality degradation.

3.5.6 Cross-Game Generalization

To assess generalization, we trained models on CS2, OW2, and their combination. Table 3.7 shows that models perform best in-domain but cross-game evaluations still yield competitive results. The combined CS2+OW2 model performs strongly on both titles, suggesting that shared motion and visual structures within the shooter genre improve robustness and enable effective multi-game training.

3.5.7 High-Resolution Scaling

We evaluate GameSR on 2K (2560 $\times$ 1440) and 4K (3840 $\times$ 2160) gameplay sequences. Table 3.8 reports reconstruction quality and latency for three high-resolution mappings.

For 1080p $\rightarrow$ 4K, GameSR achieves up to 39.25 dB PSNR, SSIM above 0.999, and VMAF above 93 while keeping inference below 16 ms per frame (real-time 60 FPS). For 720p $\rightarrow$ 2K, latency is only 7 ms per frame (approximately 143 FPS), sufficient even for 120 Hz gaming. All 2K/4K clips were played by different users, collected after training, and remain strictly unseen during training and validation, mirroring the protocol used for the 1080p test set. Despite being trained solely on 1080p content, GameSR can be directly applied to substantially higher-resolution inputs without fine-tuning while preserving temporal stability.

Table 3.9: Rendered condition MOS for GameSR across participant groups. Exp.=Experienced, Occ.=Occasional, No.=Non-gamers.

Game	MOS	95% CI	Exp.	Occ.	No.
CS2	4.73	[4.63, 4.80]	4.62	4.75	4.84
OW2	4.70	[4.59, 4.82]	4.68	4.75	4.68

Table 3.10: Paired cloud gaming comparison (25 Mbps): native 1080p WebRTC vs. GameSR MOS scores with 95% CI. Δ MOS = GameSR – WebRTC. TOST equivalence margin: ± 0.5 MOS.

Game	WebRTC MOS [CI]	GameSR MOS [CI]	Δ MOS	Wilcoxon p	TOST p
CS2	4.37 [4.20, 4.53]	4.48 [4.28, 4.69]	+0.12	0.106	<0.001
OW2	4.43 [4.27, 4.59]	4.48 [4.28, 4.69]	+0.05	0.582	<0.001

3.5.8 User Study

To further assess perceptual quality, we conducted a subjective evaluation following ITU-T Recommendation P.910 and ITU-R BT.500 [101, 102]. A total of **15 participants** took part, comprising experienced gamers (40%), occasional gamers (40%), and non-gamers (20%). The study included two conditions: (i) a *rendered* condition, where participants rated GameSR-upscaled frames directly, and (ii) a *paired cloud gaming* condition, where participants viewed both native 1080p WebRTC streams and 540p streams upscaled by GameSR at 25 Mbps, rating each on a 5-point MOS scale across **26 stimuli** per participant. Including the native 1080p WebRTC stream as an explicit baseline allows GameSR to be judged *against* the full-resolution reference under identical streaming conditions, rather than in isolation.

Rendered condition. In the rendered condition, participants rated the visual quality of GameSR-upscaled output without a baseline comparison. As shown in Table 3.9, GameSR achieved an overall MOS of **4.73/5** for CS2 and **4.70/5** for OW2, with narrow 95% confidence intervals (± 0.08 – 0.11) indicating low inter-participant variability. Scores were stable across all experience groups (Kruskal–Wallis: $p = 0.09$ for CS2, $p = 0.50$ for OW2), confirming that gamers and non-gamers alike perceive the upscaled output as high quality.

Paired cloud gaming comparison. The paired condition provides the key statistical evidence, directly comparing GameSR against the native 1080p baseline: participants rated both native 1080p WebRTC streams and GameSR-upscaled streams under identical cloud gaming conditions at 25 Mbps. Table 3.10 summarizes the results. A TOST equivalence test with a margin of ± 0.5 MOS (half a quality category) confirms perceptual equivalence for both CS2 and OW2 ($p < 0.001$), positively asserting that the quality difference falls within an imperceptible range. Wilcoxon signed-rank tests show no significant difference for either game (CS2: $p = 0.106$; OW2: $p = 0.582$), with mean differences of only +0.12 and +0.05 MOS—far smaller than what users can perceive. A Kruskal–Wallis test confirmed no significant differences across experience groups (all $p > 0.41$), meaning experienced gamers, occasional gamers, and non-gamers all rated GameSR quality similarly.

Paired comparisons against DLSS and FSR were not included because these up-

scalers execute server-side during rendering, whereas GameSR operates client-side after decoding; native 1080p WebRTC streaming serves as the appropriate upper-bound baseline. The subjective evaluation focused on CS2 and OW2 as the most visually complex titles; objective metrics (Tables 3.2 and 3.4) and the end-to-end evaluation (Section 3.6) confirm consistent trends across all four titles. At $N = 15$, the study provides 80% power to detect large effects (Cohen’s $d \geq 0.78$), and TOST equivalence confirmation validates that GameSR delivers quality statistically equivalent to native 1080p streaming while enabling 35–49% bandwidth reduction.

3.5.9 Hardware Scalability

We evaluate the scalability of GameSR across a diverse set of client devices ranging from modern mobile SoCs to high-end desktop GPUs. Figure 3.6 reports inference latency for $2\times$ upscaling as a function of device TFLOPS.

GameSR-M (designed with mobile-friendly operations only) achieves real-time performance on an Adreno 750 mobile GPU (4.7 TFLOPS), with 26.56 ms per 1080p frame for $2\times$ and 14.55 ms for $3\times$, the $3\times$ setting meets the 60 FPS (16.7 ms) budget. On discrete GPUs, latency decreases roughly with available compute: mid-range GPUs such as the RTX 4060 Ti, 4070 Ti, and A4000 sustain about 1.7–4.0 ms per frame for $2\times$, while high-end cards like the RTX 3090 and 4090 reduce latency to about 2.3 ms and 1.3 ms, respectively.

Client-Side Resource Selection. To automatically adapt to heterogeneous client hardware, we implement a lightweight resource allocator that runs once at session start. The allocator conducts proxy inferences using dummy inputs across candidate network filter sizes (24, 32, 64), since sizes below 24 yield unacceptable quality. It then selects the largest configuration satisfying two criteria: inference time under 10 ms and model memory fitting available hardware. This probe takes 30–100 ms depending on the device, a negligible one-time cost that enables automatic deployment across the full range of client capabilities without manual configuration.

3.6 End-to-End Cloud Gaming Evaluation

We evaluate client-side SR in a real WebRTC cloud-gaming pipeline using GameLab [103], an open-source aiortc-based testbed (described in Chapter 6). The server and client are connected via a dedicated 1 Gbps Ethernet switch. We compare three 1080p-display configurations:

1. **WebRTC-1080p:** Baseline streaming at 1080p.
2. **LR+Bicubic:** Stream 540p (for $\times 2$) or 360p (for $\times 3$) and upscale with bicubic interpolation.
3. **LR+GameSR:** Same LR streams upscaled with GameSR.

3.6.1 Bandwidth Savings

Figure 3.7 shows VMAF as a function of bitrate for CS2 and OW2. Using standard VMAF targets [96], achieving *good* quality ($\text{VMAF} \geq 80$) in CS2 requires:

- 8.2 Mbps with GameSR ($\times 2$)
- 12.5 Mbps with bicubic
- 16.0 Mbps with WebRTC-1080p

This represents 34.6% and 48.9% bandwidth savings, respectively. For OW2, GameSR requires 8.1 Mbps versus 14.4 Mbps (bicubic) and 15.8 Mbps (WebRTC-1080p), yielding 43.7% and 48.4% savings.

For the *excellent* target ($\text{VMAF} \geq 90$), GameSR ($\times 2$) reaches 90 at 19.4 Mbps (CS2) and 18.4 Mbps (OW2), whereas neither bicubic nor WebRTC-1080p reaches $\text{VMAF} \geq 90$ within the tested range up to 25 Mbps (maximum VMAF: 84.2/87.4 in CS2 and 83.5/88.4 in OW2 for bicubic/WebRTC-baseline, respectively).

3.6.2 Bitrate–Quality Analysis Across Scales and Resolutions

To further characterize GameSR’s bandwidth savings, we evaluate PSNR and SSIM as a function of bitrate across multiple scale factors ($\times 2$, $\times 3$, $\times 4$) and target resolutions (1080p, 1440p, 2160p). Streams are encoded using H.264 with zero-latency settings to mirror realistic cloud gaming conditions, then decoded and upscaled before metric computation.

Figure 3.7 presents VMAF vs. bitrate for CS2 and OW2 at 1080p across scale factors. GameSR consistently outperforms the no-SR baseline at low-to-mid bitrates across all games and scales, confirming that transmitting at lower resolution and upscaling is more bandwidth-efficient than transmitting at target resolution when bandwidth is constrained. At higher bitrates, quality saturates for both approaches: the encoder reaches maximum efficiency for the given resolution, and the SR model reaches its reconstruction limit. This saturation point defines the practical operating range where GameSR delivers meaningful savings. For $\times 4$ scaling at 1080p, performance gains are smaller than $\times 2$ and $\times 3$, as the aggressive downsampling discards information the model cannot fully recover. For 2K and 4K targets, GameSR demonstrates effective generalization to resolutions beyond its 1080p training distribution, maintaining quality advantages at lower bitrates without fine-tuning.

3.6.3 End-to-End Latency

Figure 3.8 shows the end-to-end latency breakdown across bitrates (8/16/24 Mbps). Although GameSR adds $\sim 4\text{--}5$ ms of client-side SR compute, it **reduces end-to-end latency by approximately 19% on average**, driven by lower encoder, transport, and decoding times when streaming fewer pixels at lower resolutions.

3.6.4 Processing and Power Savings

Streaming and rendering frames at lower resolutions yield gains in processing efficiency, as shown in Figures 3.9a and 3.9b. When rendering at 1080p, the system exhibits an average CPU utilization of 62.74%, GPU load averaging 38.77%, and power consumption of 68.67 W. In contrast, rendering at 540p significantly reduces system load: average CPU and GPU utilizations drop to 24.13% and 24.46%, respectively, while power usage decreases to 55.37 W. These reductions, a 13.30 W ($\sim 19\%$) drop in power alongside

roughly $2.6\times$ lower CPU utilization, highlight the computational saving of rendering at a lower resolution, which GameSR enables by reconstructing the full-resolution image on the client.

However, owing to limitations in Steam’s rendering API, which does not expose control over the internal rendering resolution, and the absence of corresponding high-resolution reference frames, objective evaluation of low-resolution rendering quality is infeasible: since the game engine directly outputs frames at 540p, there is no ground-truth 1080p version of the same frame to compare against, precluding full-reference metrics such as PSNR or VMAF. For our objective evaluations we therefore stream by downsampling frames before sending them to the encoder. Even in this constrained scenario, simply downsampling the game frames before encoding yielded a 13–15% reduction in CPU load.

Overheads. The client-side SR process requires 496 MiB of GPU memory, representing different fractions of total memory across GPU models: about 2% on an RTX 4090 (high-end), 4% on an RTX 3060 (mid-range), and 12% on an RTX 2050 (low-end). While this fixed memory requirement has a varying relative impact depending on the GPU’s total memory, it remains manageable across all hardware tiers, ensuring sufficient resources remain available for other operations even on entry-level devices. When considering the total system impact, the trade-off is favorable: in exchange for this small, fixed client-side overhead, rendering at 540p instead of 1080p reduces server-side GPU usage from 38.77% to 24.46% and power consumption from 68.67 W to 55.37 W. Because server resources are typically more constrained and shared among many concurrent users, shifting a modest reconstruction cost to the client in return for substantial server-side savings is an advantageous trade in cloud gaming.

3.7 Summary

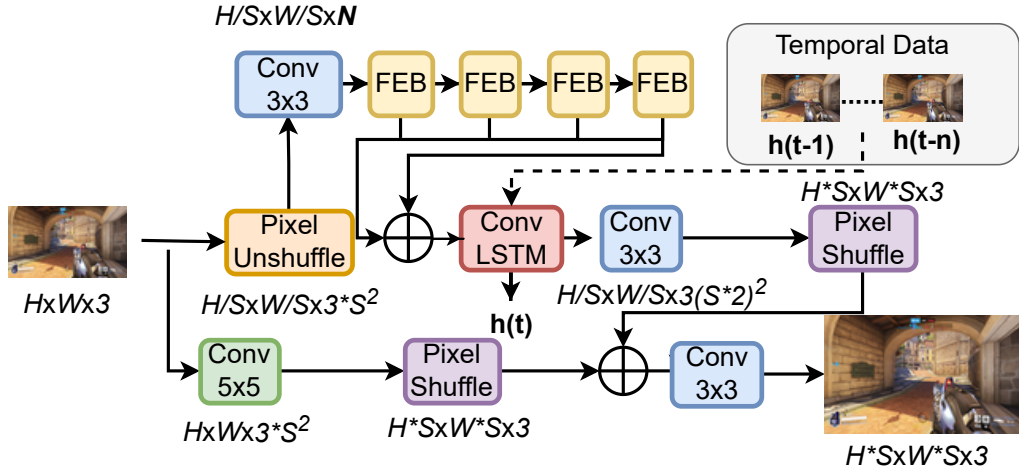
This chapter presented GameSR, a fast and engine-agnostic super-resolution model for gaming. Unlike current upscalers such as DLSS and FSR, GameSR requires no renderer data, enabling deployment in both modern and legacy games. Through efficient feature extraction with PixelUnshuffle, reparameterization for inference optimization, and lightweight temporal modeling with ConvLSTM, GameSR achieves ~ 4 ms inference time while preserving high quality.

The key findings of this chapter are:

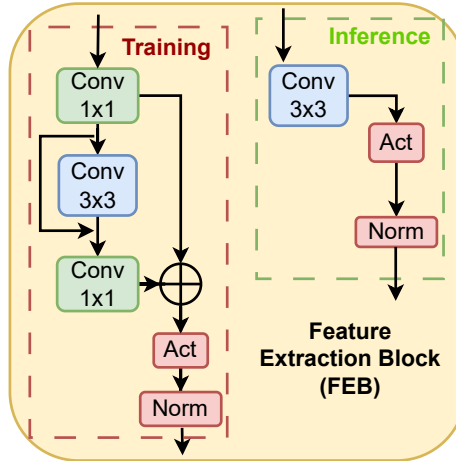
1. GameSR achieves near-parity with commercial upscalers (DLSS, FSR) on perceptual metrics without requiring engine integration.
2. GameSR runs $30\text{--}60\times$ faster than CNN baselines (EDSR, IMDN) and nearly $500\times$ faster than SwinIR, while matching or exceeding their quality on gaming content.
3. In end-to-end cloud gaming evaluation, GameSR reduces bandwidth by 35–49% to achieve target quality levels while actually reducing overall latency by 19%.
4. User studies confirm high perceptual quality (MOS 4.70–4.73/5) consistent across gamer and non-gamer populations.

5. GameSR scales effectively across hardware from mobile GPUs to high-end desktop cards, meeting real-time requirements across the spectrum.

GameSR addresses the first research question of this thesis by demonstrating that engine-agnostic, real-time neural upsampling is achievable through careful architectural design and exploitation of gaming content characteristics. The next chapter extends these ideas by integrating GameSR within ARCADE (Chapter 4), a joint optimization framework that coordinates rendering resolution, encoding bitrate, and super-resolution to maximize quality of experience under bandwidth and compute constraints.



(a) GameSR architecture



(b) Feature Extraction Block (FEB)

Figure 3.2: Overview of GameSR: (a) architecture where the current low-resolution input frame (the leftmost image, frame t) is downsampled with PixelUnshuffle, processed by Feature Extraction Blocks, and passed through a lightweight ConvLSTM before upsampling via PixelShuffle with a residual connection. The “Temporal Data” block $h(t-1), \dots, h(t-n)$ holds the ConvLSTM hidden states carried over from the preceding frames of the same gameplay sequence; the ConvLSTM fuses this temporal context with the features of the current (leftmost) frame to produce a motion-aware, temporally consistent reconstruction, and emits the updated state $h(t)$ for the next frame. (b) internal structure of the Feature Extraction Block (FEB), in its multi-branch training form and reparameterized single-path inference form; here **Act** denotes the GELU activation and **Norm** the LayerNorm normalization (defined in Section 3.3.4, “Activation and Normalization”).

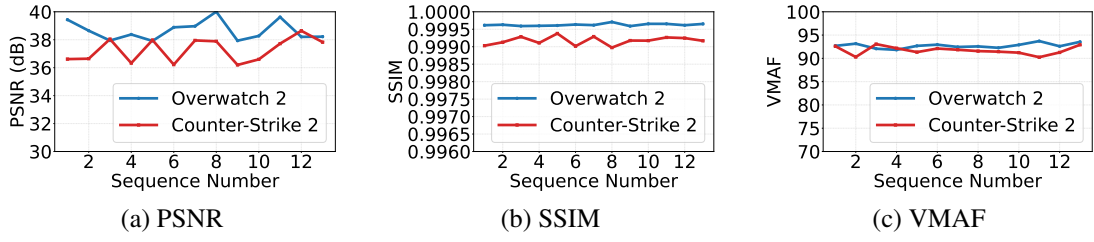


Figure 3.3: Performance of GameSR on upscaling game sessions from CS2 and OW2 by a factor of $2\times$.



Figure 3.4: Visualization of upsampling methods on Overwatch 2 frames: each row corresponds to a different map (Row 1: Junktown, Row 2: Esperança, Row 3: Nepal). Columns show DLSS, FSR, and GameSR, respectively.

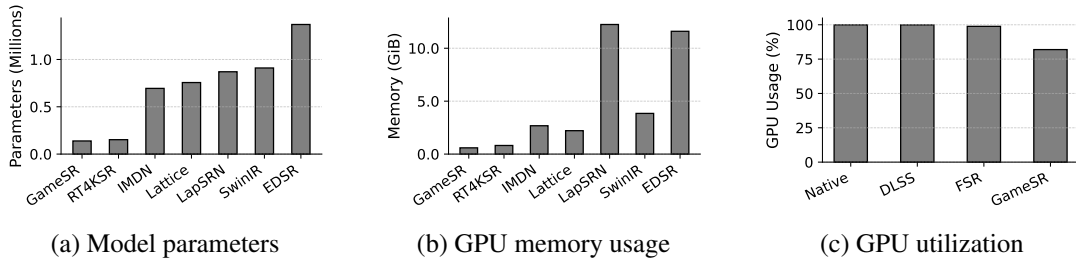


Figure 3.5: Efficiency of GameSR compared to state-of-the-art methods for $2\times$ scaling: (a) parameter count, (b) GPU memory usage, (c) GPU utilization comparison against DLSS, FSR, and native rendering.

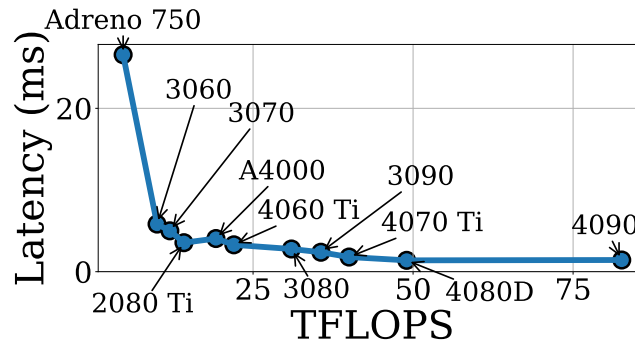
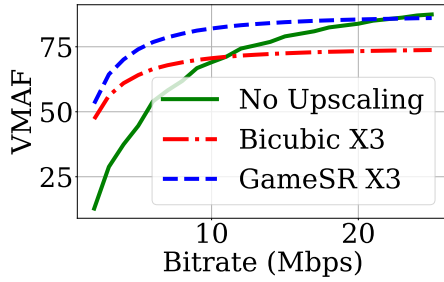
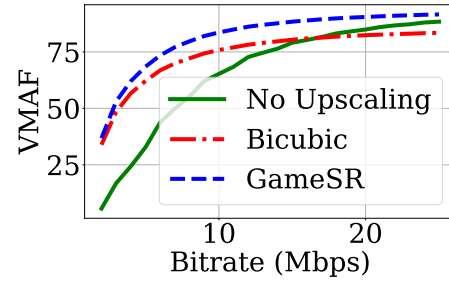


Figure 3.6: Inference latency vs. device compute capability (TFLOPS) for GameSR at $2\times$ scaling.



(a) CS2



(b) OW2

Figure 3.7: VMAF vs. bitrate for CS2 and OW2. GameSR achieves target quality levels at substantially lower bitrates.

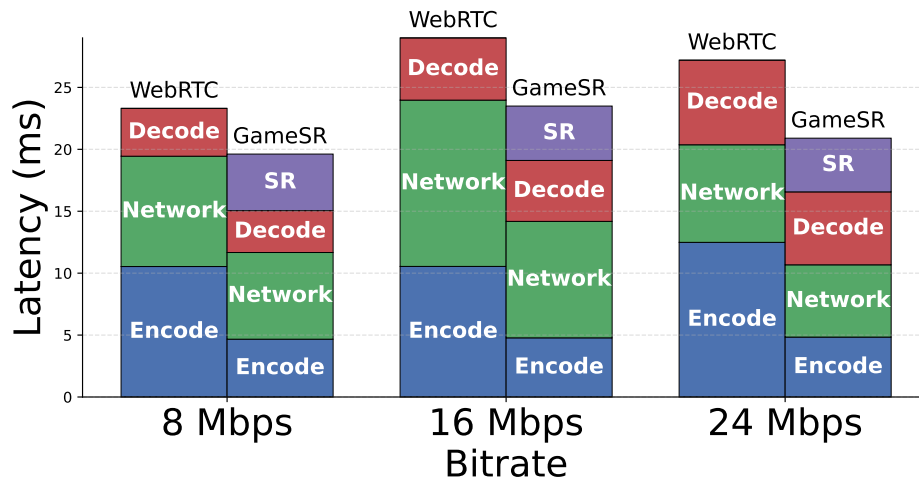
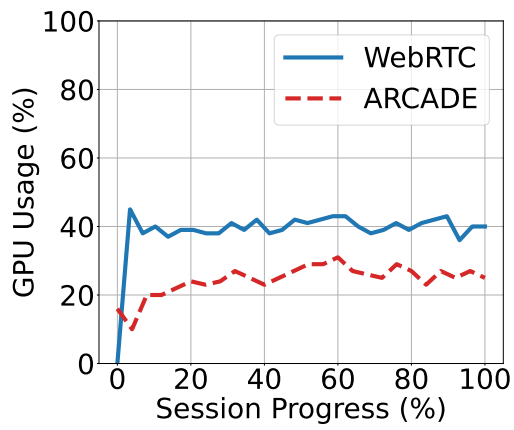
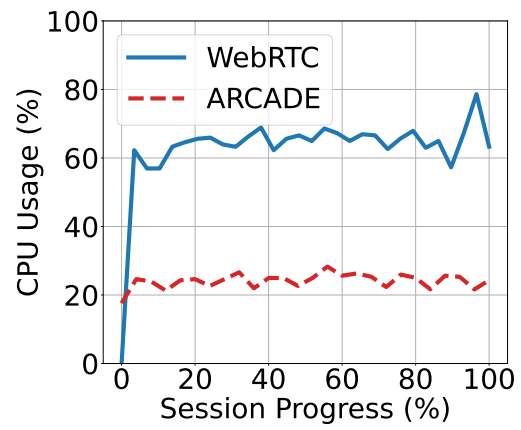


Figure 3.8: End-to-end latency breakdown across bitrates (8/16/24 Mbps). GameSR adds 4–5 ms for SR but reduces overall latency by 19% through smaller frame sizes.



(a) GPU utilization



(b) CPU utilization (normalized)

Figure 3.9: Server-side GPU and CPU utilization: baseline WebRTC streaming at 1080p versus GameSR-enabled streaming at 540p.

Chapter 4

ARCADE: Joint Optimization of Bandwidth and Computing Resources

H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “ARCADE: Joint Optimization of Bandwidth and Processing Resources for Cloud Gaming,” under review at IEEE Transactions on Multimedia, 2026.

4.1 Introduction

Cloud gaming streams interactive gameplay from remote servers, enabling users to play on commodity devices without owning high-end hardware. However, as established in Chapters 1 and 2, cloud gaming demands $\sim 5\times$ more bandwidth than VoD streaming and imposes strict latency constraints. On the server side, each session runs a full game instance with real-time GPU rendering and low-latency encoding, and studies show that excessive rendering and encoding waste compute and energy in data centers [104, 105, 106].

Chapter 3 presented GameSR, a lightweight super-resolution model that can upscale low-resolution game streams to high resolution in real-time. However, simply deploying SR is not sufficient; there is no mechanism to *jointly* reason over rendering resolution, bitrate, and upscaling strategy. Current systems render at fixed resolution regardless of scene difficulty, and while clients may apply bilinear/bicubic scaling, they do not coordinate these decisions with network conditions and server load.

This chapter presents **ARCADE** (Adaptive Resource-aware Cloud gAming DElivery), an end-to-end framework that jointly optimizes bandwidth and server-side computational costs while meeting quality and latency constraints. ARCADE addresses the second research question of this thesis: *How can we jointly optimize rendering resolution, encoding bitrate, and client-side upscaling to minimize bandwidth and compute costs while maintaining perceptual quality?*

The main contributions of this chapter are:

1. To the best of our knowledge, ARCADE is the **first framework that jointly selects rendering resolution, encoding bitrate, and upscaling ratio** in real time based on bandwidth, scene complexity, and computing resources.
2. An **offline RL-based algorithm** for joint optimization of bandwidth and processing resources in cloud gaming, using Batch-Constrained Q-learning (BCQ) to learn from logged gameplay sessions.

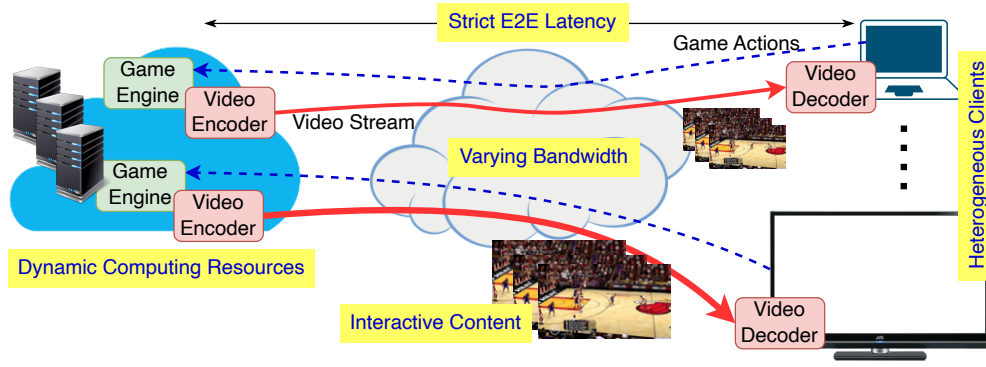


Figure 4.1: Overview of cloud gaming and its challenges.

3. Implementation on an end-to-end WebRTC testbed with extensive objective and subjective evaluation demonstrating **savings of up to 50% bandwidth, 62% CPU, and 41% GPU** compared to baseline WebRTC while maintaining or improving perceptual quality.

4.2 Problem Analysis and Motivation

4.2.1 Challenges in Cloud Gaming Optimization

As illustrated in Figure 4.1, cloud gaming optimization faces four key challenges:

Strict End-to-End Latency. Latency tolerance is genre-dependent: fast-paced first-person games tolerate only about 80 ms end-to-end latency [2], while slower third-person titles tolerate roughly 100–150 ms [3], leaving little headroom across the pipeline. Any optimization must operate within this budget without adding significant processing delay.

Highly Variable Bandwidth. Available network rate fluctuates on sub-second timescales, requiring fine-grained bitrate adaptation [107, 108]. Optimization decisions must respond quickly to changing conditions.

Content Complexity. Scene difficulty changes abruptly based on player actions, a static menu differs vastly from a chaotic firefight. Frames are generated on the fly, making complexity unpredictable.

Client Heterogeneity. Devices differ in display resolution and compute capability. A solution must adapt to diverse hardware from mobile phones to gaming PCs.

4.2.2 Limitations of Current Approaches

Existing systems suffer from several limitations that ARCADE addresses:

Post-Render Adaptation Only. Current cloud gaming systems adapt the encoder’s bitrate to available network bandwidth but treat the renderer as fixed. This adaptation occurs *after* the engine renders high-resolution frames, optimizing only the encoded bitrate. When the congestion controller reports low rates, the encoder increases quantization parameter (QP) aggressively, discarding details that were expensive to render and wasting server GPU cycles.

Ignoring Content Complexity. Current systems render at fixed resolution regardless of whether the scene is a static HUD or a high-motion firefight, potentially over-allocating bits in simple segments or under-allocating when fidelity matters. They also ignore perceptual thresholds like Just Noticeable Difference (JND) [109]: beyond JND, increasing bitrate yields no visible improvement, and under tight budgets, rendering simpler content that compresses well can outperform complex frames that must be heavily quantized.

SR Models Unsuitable for Cloud Gaming. As discussed in Chapter 2, engine-integrated upscalers (DLSS, FSR, XeSS) require internal buffers unavailable to cloud-gaming clients. Generic SR models either violate real-time latency budgets or, when lightweight, incur noticeable quality loss.

4.2.3 Underutilized Client Resources

As discussed in Chapter 3, modern client devices, smartphones with 10–50 TOPS NPUs and consoles with ~ 10 –12 TFLOPS GPUs [86, 87, 88, 89], possess substantial compute that is largely idle during remote rendering. Exploiting this latent capacity for client-side SR enables low-latency upscaling while offloading rendering and encoding work from the server.

4.2.4 Relationship to Other Video Streaming Systems

Cloud gaming streams video, but its constraints differ fundamentally from Video on Demand (VoD), live streaming, and conferencing. VoD adaptive bitrate systems (e.g., DASH [110]) rely on pre-encoded multi-bitrate segments and client-side playback buffers [111, 112, 113]; in cloud gaming, frames are rendered on the fly and must adapt at frame/GOP timescales with minimal buffering. Live streaming and conferencing tolerate higher latency and face less extreme motion and compute demands [114, 115]. Cloud gaming instead combines fast motion and GPU-intensive content with strict interactivity targets [3].

Neural SR systems like LiveNAS [116] target passive one-to-many streaming (server-side SR at ingest) and do not capture the per-session coupling between rendering, encoding, and latency in interactive cloud gaming. ARCADE addresses this gap by operating without client-side buffering, using a pluggable real-time SR backend on the client, and jointly adapting rendering resolution, encoder bitrate, and SR factor at fine timescales under explicit bandwidth and compute constraints.

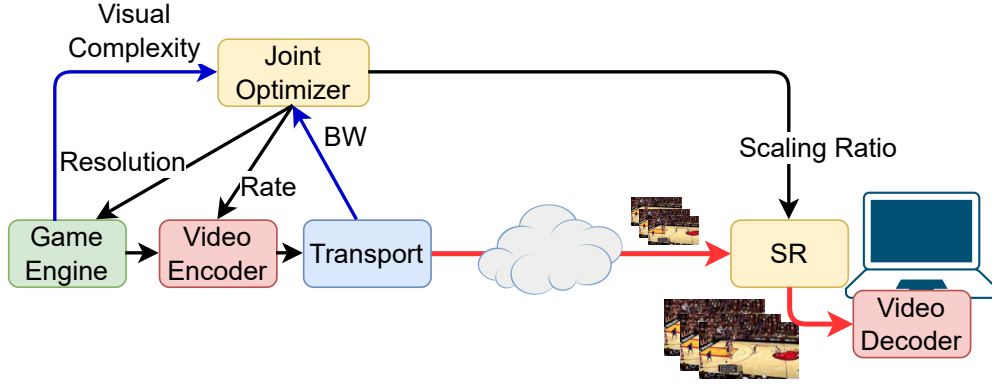


Figure 4.2: Overview of ARCADE, which jointly optimizes rendering resolution, encoding bitrate, and client-side super-resolution in cloud gaming.

4.3 The ARCADE Approach

ARCADE overcomes these limitations by providing an end-to-end framework that jointly controls rendering resolution, encoder bitrate, and client-side super-resolution, as shown in Figure 4.2. It consists of two components:

1. An **RL-based controller** that jointly optimizes bandwidth and computation given network capacity, scene complexity, and server-side resources.
2. An **optional pluggable SR backend** that upscales decoded game video on the client (instantiated with GameSR from Chapter 3).

Crucially, ARCADE’s core contribution, the joint optimizer, operates independently of any specific SR backend. The system delivers bandwidth and compute savings even when SR is unavailable, falling back to RL-driven adaptation only while still yielding complexity-aware savings over static-resolution WebRTC.

4.4 ARCADE System Architecture

4.4.1 System Overview

Rather than treating the renderer, encoder, and client-side SR as separate components, ARCADE coordinates them through a single control loop at GOP (Group of Pictures) granularity. The server hosts three modules:

- **Complexity Analyzer:** Predicts the complexity tier of the next GOP.
- **Joint Bandwidth–Compute Optimizer:** Selects bitrate–resolution pairs based on complexity, bandwidth, and server load.
- **Model Bank:** Stores SR backends with different accuracy–latency trade-offs.

The client hosts a lightweight **Model Selector** and SR backend. At session start, the client benchmarks its hardware and selects a feasible SR configuration (or disables SR if insufficient); this choice persists unless conditions change.

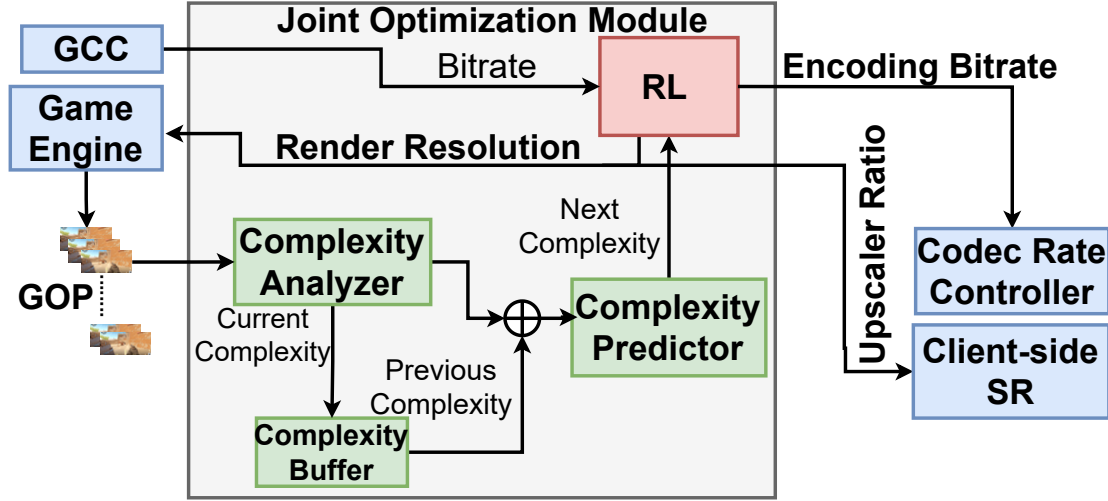


Figure 4.3: Joint Optimization Module (JOM) in ARCADE.

During runtime, the system collects frame complexity, bandwidth, and compute at each GOP. Mid-GOP changes are infeasible due to codec constraints, so ARCADE updates bitrate and rendering resolution only at GOP boundaries. The optimizer selects configurations trading off server-side cost, network bitrate, and expected client-side upscaling gain, propagating decisions to the renderer, encoder, and SR backend.

4.5 Joint Bandwidth-Compute Optimization

The Joint Optimizer balances server-side rendering/encoding cost, transmission bandwidth, and client-side upsampling to optimize QoE under varying conditions. Its modular design enables integration with any congestion control algorithm. Figure 4.3 illustrates the Joint Optimization Module.

4.5.1 Reinforcement Learning Formulation

At each GOP, the system selects a bitrate–resolution pair balancing server-side cost, available bandwidth, and expected perceptual benefit from upscaling. These decisions interact over time: aggressive actions during low-complexity segments can deplete encoder buffers or trigger QP shifts degrading later segments, while conservative actions waste bandwidth. This sequential, multi-objective problem is poorly suited to hand-crafted heuristics (myopic under non-stationary dynamics) or supervised predictors (cannot capture long-term consequences).

RL formalizes this as sequential decision-making, maximizing long-horizon QoE through value functions with function approximation. The state space $(\phi_t, b_t^{\text{rec}}, u_t^{\text{gpu}}, u_t^{\text{cpu}})$ and action space (b_t, r_t) are high-dimensional, making tabular methods intractable. Since VMAF is unavailable during live gameplay (clients cannot compute full-reference metrics without original frames) and online exploration risks degrading QoE, we adopt *offline* RL on logged transitions containing ground-truth quality and resource measurements.

4.5.2 State Space

At each GOP t , the system observes:

$$s_t = [\phi_t, b_t^{\text{rec}}, u_t^{\text{gpu}}, u_t^{\text{cpu}}] \in \mathbb{R}^4 \quad (4.1)$$

where:

- ϕ_t is the predicted complexity tier
- b_t^{rec} is the GCC-recommended bitrate
- $u_t^{\text{gpu}}, u_t^{\text{cpu}}$ are server-side GPU and CPU utilization

4.5.3 Action Space

We define a fixed catalogue of bitrate–resolution options:

$$\mathcal{A}^\circ = \{(b_i, r_j) \mid b_i \in \mathcal{B}, r_j \in \mathcal{R}\} \quad (4.2)$$

where $\mathcal{B}$ is a discrete bitrate range and $\mathcal{R}$ is the resolution set (e.g., 360p, 540p, 1080p). At GOP t , the congestion controller provides an upper bound b_t^{rec} , yielding the feasible action set:

$$\mathcal{A}_t = \{(b_i, r_j) \in \mathcal{A}^\circ \mid b_i \leq b_t^{\text{rec}}\} \quad (4.3)$$

We use discrete actions because: (i) decisions apply only at GOP boundaries where fine granularity offers little benefit, (ii) resolution is inherently discrete, and (iii) a finite action set keeps the offline RL policy within the logged dataset’s support, reducing value overestimation. Each action specifies the server-side rendering/encoding level and implied client-side upscaling ratio.

4.5.4 Reward Function

The reward is defined over outcomes induced by action $a_t = (b_t, r_t)$:

$$R_t = \lambda_q \text{VMAF}_{t+1} - \lambda_d \text{RTT}_{t+1} - \lambda_c (u_{t+1}^{\text{gpu}} + u_{t+1}^{\text{cpu}}) - \lambda_b \frac{b_t}{B_{\max}} \quad (4.4)$$

where $\lambda_q, \lambda_d, \lambda_c, \lambda_b \geq 0$ trade off quality, latency, compute, and bandwidth, respectively. Actions already satisfy $b_t \leq b_t^{\text{rec}}$ via $\mathcal{A}_t$; the term $-\lambda_b b_t / B_{\max}$ ($B_{\max} = \max \mathcal{B}$) regularizes bandwidth within the GCC-constrained range, preferring lower bitrates when quality and compute costs are comparable.

Choice of reward weights. Rather than weighting the four objectives equally, we set them to reflect their relative priority for interactive cloud gaming, taking quality as the anchor: $\lambda_q = 1.0$, $\lambda_c = 0.75$, $\lambda_d = 0.5$, and $\lambda_b = 0.25$. The rationale for this ordering is as follows:

- **Quality (VMAF), $\lambda_q = 1.0$:** perceptual quality is the primary objective the entire pipeline is designed to preserve, so it receives the highest weight and anchors the reward scale. A policy that saves resources or bandwidth at the cost of visible quality degradation defeats the purpose of the system.

- **Compute**, $\lambda_c = 0.75$: reducing server-side rendering and encoding load is a core goal of ARCADE and translates directly into GPU provisioning cost and the number of concurrent sessions a server can host. It is therefore weighted second, below quality but above the remaining terms.
- **Latency (RTT)**, $\lambda_d = 0.5$: latency is critical for interactivity, but the low-latency encoder configuration and the GCC-bounded action space already curb the dominant sources of queuing delay. A moderate weight is thus sufficient to discourage actions that inflate RTT without over-penalizing otherwise good configurations.
- **Bandwidth**, $\lambda_b = 0.25$: this term receives the smallest weight because bandwidth is already protected by a hard guardrail, every action is constrained to a bitrate within GCC’s recommended rate ($b_t \leq b_t^{\text{rec}}$ via $\mathcal{A}_t$), so the agent can never overshoot the safe network envelope. The penalty therefore acts only as a soft regularizer that breaks ties toward lower bitrates when quality and compute are comparable, and does not need a large weight to do so.

Because rewards are Z-score normalized before training (Section 4.6), these weights set the *relative* priority among objectives rather than absolute scales. We found the resulting ordering $\lambda_q > \lambda_c > \lambda_d > \lambda_b$ to be robust: moderate perturbations of the individual weights did not change the qualitative behavior of the learned policy, whereas the ordering itself encodes the design intent of prioritizing quality first, then server-side savings, then latency, with bandwidth handled primarily by the GCC guardrail.

4.6 Batch-Constrained Q-Learning

Standard off-policy Q-learning can severely overestimate values for actions rare or absent in the dataset. Recent offline RL methods extend standard algorithms (DQN [65], DDPG [117], TD3 [118]) to avoid over-optimistic Q-values on out-of-distribution actions. Batch-Constrained Q-learning (BCQ) [119] addresses this by modeling the empirical behavior policy $G(a | s)$ and restricting maximization to actions with sufficiently high likelihood under G , preventing out-of-distribution extrapolation while still improving upon the behavior policy.

We train $Q(s, a)$ using Discrete BCQ, which maintains a Q-network and behavior model $G(a | s)$ trained via behavior cloning. Given threshold τ , the filtered action set is:

$$\mathcal{A}^{\text{filtered}}(s) = \{a \in \mathcal{A}^\circ \mid G(a | s) > \tau\} \quad (4.5)$$

The Bellman target for transition (s_t, a_t, R_t, s_{t+1}) is:

$$y_t = R_t + \gamma \max_{a' \in \mathcal{A}^{\text{filtered}}(s_{t+1})} Q_{\text{target}}(s_{t+1}, a') \quad (4.6)$$

minimizing TD loss $(Q(s_t, a_t) - y_t)^2$. At inference:

$$\pi_{\text{BCQ}}(s_t) = \arg \max_{a \in \mathcal{A}^{\text{filtered}}(s_t)} Q(s_t, a) \quad (4.7)$$

We selected BCQ over alternative offline RL algorithms through preliminary experimentation. Conservative Q-Learning (CQL) [68], which regularizes Q-values for rare actions globally, was also evaluated but proved less stable for our discrete resolution-bitrate catalogue. Discrete BCQ’s explicit behavior cloning with filtered action sets was better suited to the safety-critical requirements of bandwidth- and compute-constrained control.

4.6.1 Training Details

We train the discrete BCQ agent offline on 25 sessions per title using DiscreteBCQ from d3rlpy. Observations and rewards are Z-score normalized. We train for up to 50,000 gradient steps (batch size 32, $\gamma = 0.99$), monitoring TD loss on a held-out validation set. The `action_flexibility` hyperparameter controls filtering aggressiveness around the behavior policy.

To promote generalization, the dataset spans multiple maps and players with five geographically diverse latency traces, covering varied bandwidth regimes and content complexities. We split by *player* and *trace* so evaluation uses held-out gameplay/trace combinations. We use early stopping to reduce overfitting. Our subjective study (Section 4.14) provides additional validation: participants play unseen sessions under untrained network conditions and consistently report high perceived quality.

4.7 Content Complexity Analysis

We adopt EVCA [60], computing *spatial complexity* (SC) and *temporal complexity* (TC) from DCT-domain energy. Let $C_{t,b}(u, v)$ denote the DCT coefficient at frequency (u, v) for block b in GOP t , and $\mathcal{H}$ the high-frequency indices:

$$SC_t = \frac{1}{|\mathcal{B}_t|} \sum_{b \in \mathcal{B}_t} \sum_{(u,v) \in \mathcal{H}} |C_{t,b}(u, v)| \quad (4.8)$$

$$TC_t = \frac{1}{|\mathcal{B}_t|} \sum_{b \in \mathcal{B}_t} \sum_{(u,v) \in \mathcal{H}} |C_{t,b}(u, v) - C_{t-1,b}(u, v)| \quad (4.9)$$

High SC/TC indicates complex or fast-motion scenes. Raw SC/TC creates unstable ranges for RL, and small variations often yield negligible perceptual changes. We therefore quantize complexity into tiers using K-Means clustering on feature vectors:

$$\mathbf{x}_t = [SC_t, TC_t, TC_t^{(2)}, VMAF_t]^\top \quad (4.10)$$

Each GOP t is thus described by four features: the spatial complexity SC_t and temporal complexity TC_t of Eqs. (4.8) and (4.9); the second-order temporal complexity $TC_t^{(2)}$, i.e., the change in temporal complexity between consecutive GOPs, which captures how abruptly motion intensity itself is changing (such as scene cuts and sudden action onsets); and $VMAF_t$, the full-reference quality measured for GOP t . These same four per-GOP features are used both for the tier clustering described here and as the building block of the predictor input described next.

Using the elbow method, we identified three tiers (High, Mid, Low) for both games from 25 sessions across five players, with K-Means achieving Silhouette Scores above 0.5, indicating well-separated clusters.

4.7.1 Complexity Prediction

To predict upcoming complexity, we train a multilayer perceptron (MLP) on features from the current and previous two GOPs:

$$\mathbf{z}_t = [\hat{\mathbf{x}}_{t-2}^\top, \hat{\mathbf{x}}_{t-1}^\top, \hat{\mathbf{x}}_t^\top]^\top \in \mathbb{R}^{12} \quad (4.11)$$

$$\mathbf{h}_t = \sigma(W_1 \mathbf{z}_t + \mathbf{b}_1), \quad \mathbf{p}_t = \text{softmax}(W_2 \mathbf{h}_t + \mathbf{b}_2) \quad (4.12)$$

where $W_1 \in \mathbb{R}^{64 \times 12}$ and $W_2 \in \mathbb{R}^{3 \times 64}$. The 12-dimensional input $\mathbf{z}_t$ stacks the four-feature vectors of three consecutive GOPs (the current GOP t and the previous two, $t-1$ and $t-2$) to predict the tier of the next GOP $t+1$; it does not use only one or two GOPs of history. The MLP (one hidden layer, 64 neurons, batch normalization, 0.4 dropout) was trained with Adam (lr=0.005, weight decay=1e-4), achieving 70% accuracy via 5-fold cross-validation. The predictor is trained and evaluated, via 5-fold cross-validation, on the same logged gameplay dataset used to train the offline RL agent (Section 4.6): GOP-level feature vectors extracted from the per-title corpus of 25 sessions across five players (Section 3.4.1).

Although we also tested LSTMs and Transformers for complexity prediction, their performance was comparable to the MLP with substantially higher computational cost, making the MLP the preferred choice for a module that must run at every GOP boundary with negligible overhead.

Effect of predictor accuracy in practice. The 70% figure is a three-class accuracy (Low/Mid/High), well above the 33% chance baseline, and although modest its real-world impact on the system is limited by how the prediction is used. First, complexity is re-estimated at every GOP boundary, so a misprediction is corrected after a single GOP (≈ 1.5 s) and errors do not accumulate over a session. Second, errors occur overwhelmingly between *adjacent* tiers (e.g., Low $\leftrightarrow$ Mid rather than Low $\leftrightarrow$ High), so the action selected for a misclassified GOP differs only marginally from the optimal one, yielding a small quality or bitrate deviation rather than a large one. Third, the Joint Optimizer operates within a bounded, GCC-constrained action space, bitrate capped by GCC’s recommended rate and resolution restricted to 360p/540p/1080p, so a wrong tier can never produce an unsafe action such as network overload; it can only shift the chosen operating point by one step. The two error directions are both benign: over-estimating complexity wastes a little bandwidth for one GOP with no quality loss, while under-estimating it causes a brief, single-GOP quality dip that the next prediction corrects. Fourth, because the offline RL policy is trained on logged sessions that already contain these same imperfect complexity labels, it learns a control strategy robust to noisy complexity input rather than one that assumes a perfect oracle. Finally, and most directly, all of the end-to-end results reported in this chapter, the bandwidth, compute, and quality gains together with the user study, are obtained with this 70%-accurate predictor in the loop; the headline numbers therefore already reflect its imperfection, and a more accurate predictor could only improve them. In the worst case, a single misclassified GOP costs at most ≈ 1.5 s of a slightly suboptimal bitrate–resolution choice, which is imperceptible in practice and is corrected at the next boundary.

4.8 System Integration

We integrate the RL agent into the sender-side rate-control loop of our WebRTC stack. Google Congestion Control (GCC) observes RTT, delay gradients, and loss, outputting a recommended rate $\tilde{b}_t$. Default WebRTC clamps this as:

$$\text{bitrate} = \max(\text{MIN_BITRATE}, \min(\tilde{b}_t, \text{MAX_BITRATE})) \quad (4.13)$$

In ARCADE, the RL agent receives $b_t^{\text{rec}} \triangleq \tilde{b}_t$ and selects $(b_t, r_t) \in \mathcal{A}_t$ with $b_t \leq b_t^{\text{rec}}$, jointly considering complexity ϕ_t and server utilization.

Client-side SR runs on the receiver without feeding metrics back to the agent. SR configurations are pre-profiled: at session start, the device selects one meeting per-frame latency and memory budgets, switching to lighter configurations if needed during playback. Thus, server-side RL optimizes under network and server-compute constraints, while client-side SR selection is handled independently for heterogeneous devices.

4.9 SR Backend Integration

In ARCADE, the client-side SR stage is *modular*: any real-time upscaler meeting a device’s latency and memory budget can be plugged into the post-decode hook. We instantiate this backend with GameSR [120] (Chapter 3), a lightweight, engine-independent gaming SR model. Importantly, GameSR is *not* a contribution of ARCADE; it serves as a default upsampler demonstrating end-to-end benefits and can be replaced with alternative SR models or classical scalers.

GameSR targets ARCADE’s operating points (540p→1080p for 2×, 360p→1080p for 3×) using PixelUnshuffle for efficient feature extraction, reparameterized convolutions for runtime efficiency, and lightweight ConvLSTM for temporal consistency without engine signals. The model is compact ($\sim 10^5$ parameters) and runs within a few milliseconds per frame on commodity GPUs, meeting ARCADE’s latency budget.

4.10 Implementation

4.10.1 WebRTC Testbed

We implement ARCADE in GameLab (Chapter 6), an open-source Python WebRTC stack. The system has three learned components: a pluggable SR backend (instantiated with GameSR), a Complexity Analyzer, and the Joint Optimizer. Multiple SR variants are stored in a server-side Model Bank; at session start, the client selects the fastest model meeting latency budgets, or disables SR if none is suitable.

The Complexity Analyzer (MLP predicting next GOP’s tier) runs once per GOP on the server. The Joint Optimizer selects bitrate–resolution pairs subject to GCC’s recommended rate, configuring renderer and encoder accordingly. The SR backend runs on the client (GPU/NPU when available).

For the reasons discussed in Chapter 1 (universal client support, reproducibility, and a conservative bandwidth baseline), we use H.264 as our codec; concretely, we encode with `libx264` (H.264) using low-latency settings (`tune=zerolatency`, `bframes=0`, `preset=veryfast`, High profile Level 4.2) and a fixed GOP of ≈ 1.5 s. ARCADE’s rendering-resolution and bitrate control is codec-agnostic and applies unchanged to more efficient codecs such as HEVC or AV1. Because most games do not expose render buffers, we use screen capture: `mss` [121] on Linux and `DXcam` [122] on Windows. Rate control follows GCC’s send-bitrate estimate without application-level QP overrides.

4.10.2 Baseline Configuration

Our baseline is unmodified WebRTC (`libwebrtc`) with Google Congestion Control (GCC), as used in Chrome’s WebRTC stack, RTP/RTCP over SRTP/UDP (ICE/STUN/TURN), NACK/RTX enabled, FEC disabled, and rendering at 1080p (30/60 fps). We use vanilla

WebRTC/GCC as our baseline because it is the de facto, widely deployed congestion-control stack for interactive real-time media and is the most transparent and reproducible open implementation available to the community [123]. Although WebRTC was first open-sourced by Google in 2011, it is neither a static nor a dated technology: it was formally standardized as a W3C Recommendation in 2021, its reference implementation (libwebrtc) is continuously maintained and released alongside Chrome, and the GCC algorithm itself has evolved substantially since its introduction rather than remaining fixed at its original form. It also remains the dominant real-time media stack in production today, underpinning mainstream services such as Google Meet and Discord as well as the browser-based clients of major cloud gaming platforms; notably, Google Stadia relied on WebRTC for its game streaming before its shutdown. This combination of continued real-world dominance and an open, reproducible implementation is precisely why it is the appropriate baseline: ARCADE’s gains are measured against the same actively maintained stack that real cloud gaming and real-time media services actually deploy, rather than against a research prototype.

4.10.3 Games and Dataset

We evaluate ARCADE on Counter-Strike 2 (CS2) and Overwatch 2 (OW2), the same titles used in Chapter 3. These represent distinct visual styles and engine architectures common across the FPS genre, which dominates cloud gaming traffic [124]. We use the same dataset and capture methodology described in Section 3.4.1: uncompressed 1080p gameplay at 30/60 fps from five players with varying skill levels, using 10 sessions per game for training and 15 unseen sessions for evaluation.

4.11 Evaluation

4.11.1 Experimental Setup

Our testbed uses a gaming server (Intel i7-11800H, 16 GB RAM, NVIDIA RTX 3060) and client (Intel Xeon Gold 5220, 32 GB RAM, NVIDIA RTX A4000), both running Ubuntu 22.04, connected via 1 Gbps Ethernet. We evaluate quality using PSNR, SSIM, and VMAF [125], and report bandwidth, CPU, and GPU usage. GPU utilization and power were measured using `nvidia-smi`, while CPU utilization and power were measured using `turbostat`, both averaged over 60-second intervals during active gameplay.

We emulate real-world conditions using latency traces from a global cloud gaming tournament [126], replayed with Ubuntu NetEm. The server runs in Ottawa, Canada, and we emulate clients in Toronto, Chicago, London, Rio de Janeiro, and Singapore. NetEm enforces an average throughput cap of 30 Mbps, with instantaneous bandwidth fluctuating based on traces.

We stream 50 sessions (25 per game) spanning different complexities and latency profiles, targeting VMAF 90 as the high-quality operating point [127, 128].

4.11.2 Meeting Target Quality

We compare ARCADE against baseline WebRTC by streaming 50 gaming sessions twice: once using WebRTC and once using ARCADE. Figures 4.4 and 4.5 show the

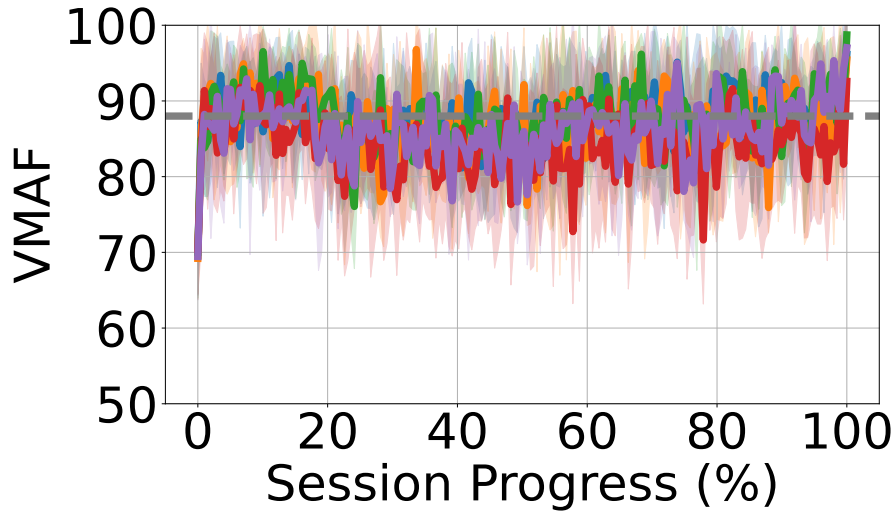


Figure 4.4: Achieved VMAF for baseline WebRTC (OW2) at 30 Mbps across five client locations (Brazil, Chicago, London, Singapore, and Toronto); the shaded region is the mean $\pm$ standard deviation across 50 sessions.

achieved VMAF for WebRTC and ARCADE respectively, averaged across all gaming sessions and color-coded by client geographical location; shaded regions indicate the confidence interval (mean $\pm$ standard deviation).

Each curve aggregates 50 sessions across different maps, players, and network traces, spanning a wide range of spatial/temporal complexities. The results show that while both WebRTC and ARCADE maintain the VMAF target score at about 90, ARCADE has clearly more consistent performance, especially supporting much better clients located farther from the server.

4.11.3 Bandwidth Savings

Examining the corresponding bandwidth usage in Figure 4.6, ARCADE utilizes significantly less bandwidth. WebRTC uses the entire 30 Mbps most of the time, whereas ARCADE uses approximately 15 Mbps on average. The **proposed system achieves up to 50% bandwidth savings** while achieving the same quality.

ARCADE simultaneously accounts for frame complexity and network conditions, using the Joint Optimizer to adjust rendering resolution, encoding rate, and upscaling factor at each GOP. In contrast, baseline WebRTC relies solely on GCC, which infers congestion from delay and packet loss: it periodically probes upward toward the estimated path capacity and then reduces the sending rate when increased delay or loss is observed. In dynamic traces, this standard probing/backoff behavior can manifest as short-lived overshoots followed by sharp rate reductions, producing visible swings between high and low quality segments.

We also observe that ARCADE exhibits a brief bitrate spike above 20 Mbps during the first GOP of each session. This is a startup transient: the Joint Optimizer begins making constrained decisions only after collecting complexity and utilization statistics over the first two GOPs. Once initialized, it quickly stabilizes the bitrate given frame complexity and GCC’s recommended rate, and the startup spike has negligible impact on the session-average bandwidth or quality stability.

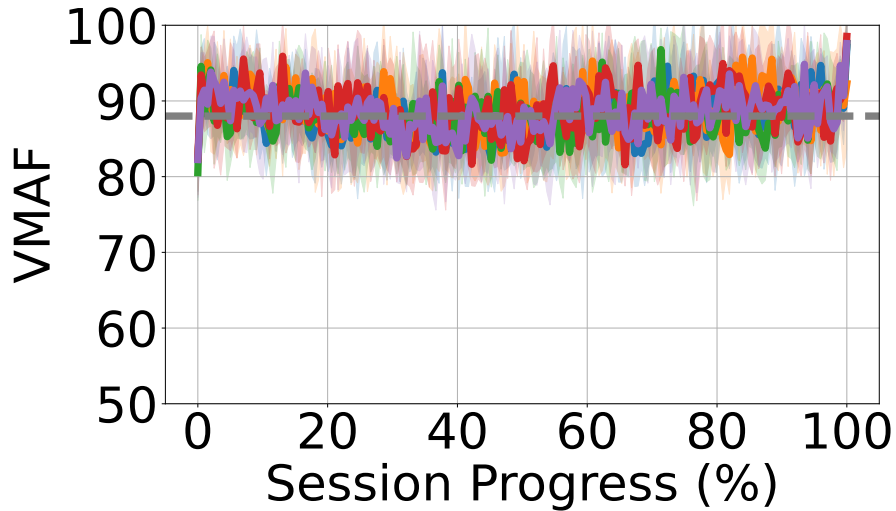


Figure 4.5: Achieved VMAF for ARCADE (OW2) at 30 Mbps across the same five client locations; ARCADE maintains the VMAF 90 target more consistently than WebRTC, especially for clients farther from the server.

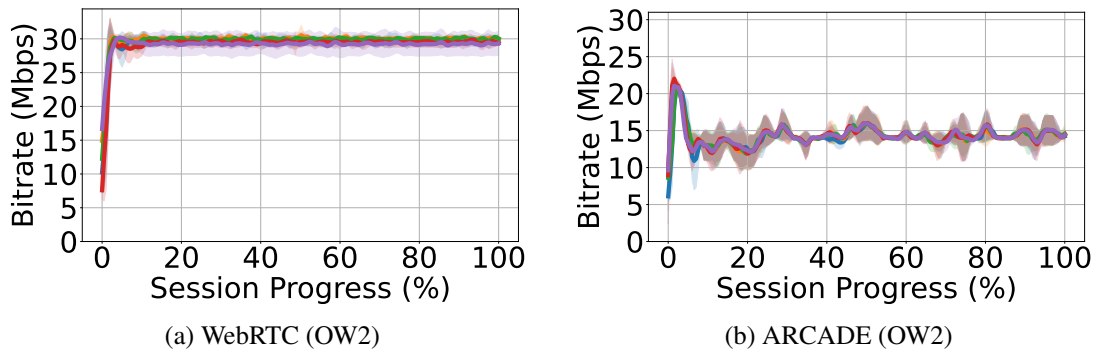


Figure 4.6: Normalized consumed bandwidth for WebRTC and ARCADE (OW2) at 30 Mbps across five locations: Brazil, Chicago, London, Singapore, and Toronto.

4.11.4 Low-Bandwidth Scenarios

To evaluate performance when bandwidth is insufficient for full target quality, we repeated experiments with available bandwidth set to only 8.5 Mbps. The results in Figures 4.7 and 4.8 show that ARCADE achieves much better quality than WebRTC, improving average VMAF by up to 33%. ARCADE maintains average VMAF scores close to 80, with some low-complexity GOPs even exceeding the target threshold of 90. In contrast, baseline WebRTC exhibits unstable quality, with frequent VMAF drops as low as 40 and overall averages slightly above 60.

4.11.5 Dynamic Network Adaptation

To stress-test ARCADE’s stability under rapidly varying network capacity, we emulate a highly dynamic link using `tc netem`, forcing congestion and network delay. Figure 4.10 plots the bitrate recommended by GCC based on network conditions and the actual encoding bitrate used by ARCADE under two congestion scenarios.

ARCADE consistently tracks the GCC envelope without overshooting it, even

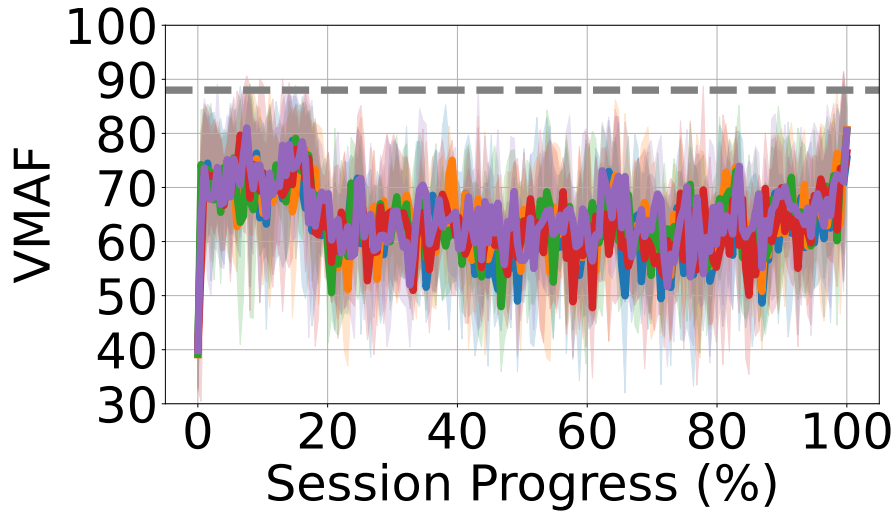


Figure 4.7: Achieved VMAF for baseline WebRTC (OW2) under the 8.5 Mbps bandwidth constraint, across five client locations. WebRTC quality is unstable, with frequent drops as low as 40.

during abrupt capacity drops, while also avoiding unnecessary bitrate usage during low-complexity segments in high-capacity periods. WebRTC simply tracks the GCC recommendation and largely saturates the available rate.

4.11.6 End-to-End Latency

Figure 4.11 breaks down average per-frame end-to-end latency into encoding, network, decoding, and client-side processing for baseline WebRTC and ARCADE at 8, 16, and 24 Mbps.

Although GameSR adds about 4–5 ms of SR processing on the client, ARCADE encodes at lower resolution and typically operates at a lower effective bitrate, reducing encoder, network, and decoding delay so that its total network latency remains consistently below that of baseline WebRTC. Consistent with this breakdown, RTT distributions collected via the `aiortc` API (Figure 4.9) also shift left under ARCADE, reflecting reduced queuing from sending lower-resolution, lower-bitrate streams.

4.12 System Ablation

We ablate ARCADE to quantify the contributions of the Joint Optimizer and the SR backend by comparing two restricted variants against the full system:

- **No-JO (fixed policy):** Disables the Joint Optimizer and uses fixed bitrate–resolution settings.
- **No-SR:** Keeps encoder and renderer at 1080p; Joint Optimizer reasons only about bitrate under bandwidth/compute constraints.

We deliberately do not ablate the Complexity Analyzer as a third, separate variant, because it is not an independent add-on but the content signal that makes the Joint Optimizer content-aware in the first place. The complexity predictor exists solely to

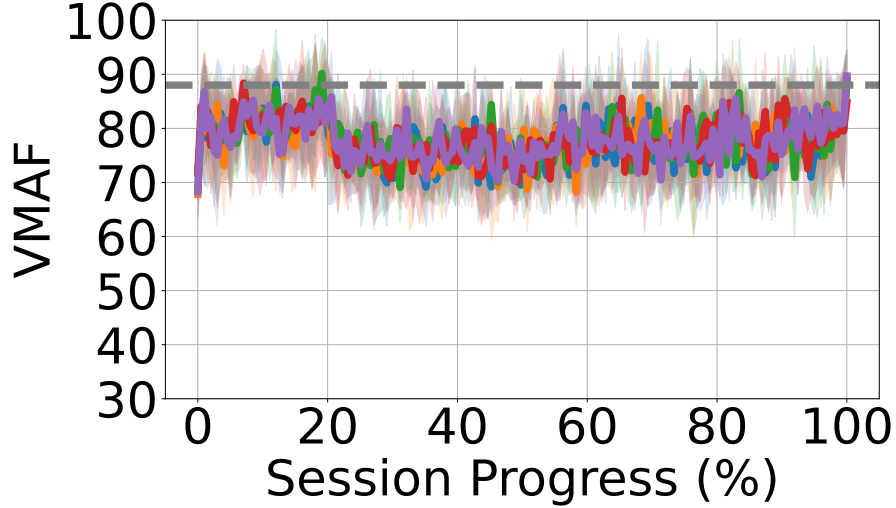


Figure 4.8: Achieved VMAF for ARCADE (OW2) under the 8.5 Mbps bandwidth constraint, across five client locations. ARCADE maintains average VMAF close to 80, with some low-complexity GOPs exceeding 90.

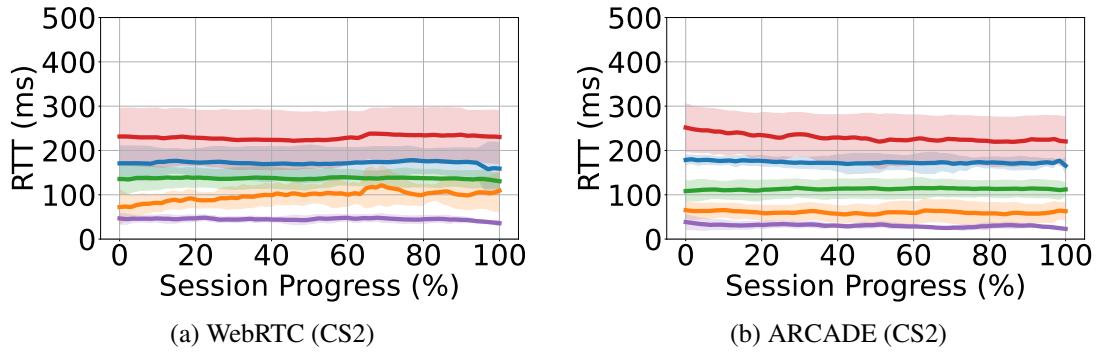


Figure 4.9: RTT distributions for CS2 under the 30 Mbps constraint, for WebRTC and ARCADE.

feed the Joint Optimizer; removing it while retaining the optimizer would leave the optimizer with no content input, collapsing it into a purely network-driven (GCC-only) rate controller. That degenerate configuration is precisely what the **No-JO** variant already evaluates: with the Joint Optimizer disabled, ARCADE adapts to network conditions alone, which is equivalent to standard adaptive WebRTC streaming with a client-side super-resolution backend, i.e., the GameSR setting of Chapter 3. The contribution of the Complexity Analyzer is therefore captured implicitly by the gap between the full ARCADE system and the No-JO variant, and a standalone “No-Complexity-Predictor” ablation would be redundant with it. The two variants above thus span the design space: No-JO isolates the value of content-complexity-driven joint optimization, while No-SR isolates the value of the client-side upscaling backend.

4.12.1 No-JO Variant

We evaluate the end-to-end pipeline over multiple sessions under varying network conditions (2–30 Mbps) and fixed resolutions (360p, 540p, 1080p). Rate–distortion

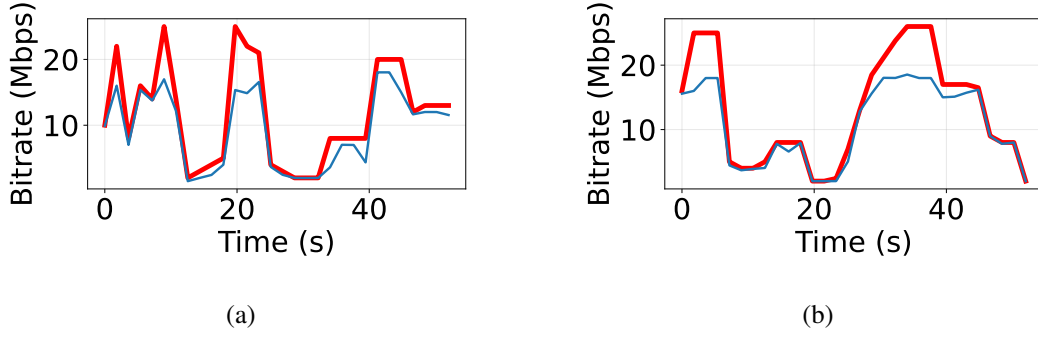


Figure 4.10: Dynamic bitrate adaptation under simulated congestion and delay. The red curve shows bitrate allowed by GCC; blue shows bitrate selected by ARCADE.

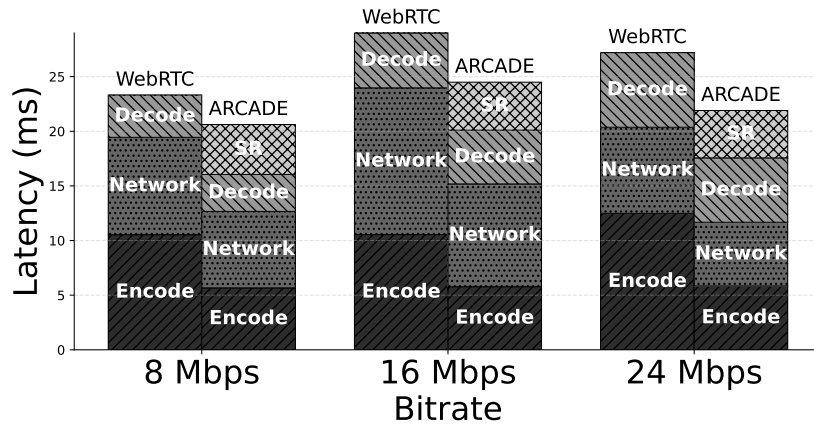


Figure 4.11: End-to-end latency breakdown for baseline WebRTC and ARCADE.

curves (Figure 4.12) show that using client-side SR at a fixed resolution consistently improves perceptual quality at low bitrates, saturating around 15 Mbps for 360p. For a target VMAF of 90, different complexity scenes require different bitrates, underscoring the need for content-adaptive encoding.

4.12.2 No-SR Variant

We disabled the SR backend and ran ARCADE as a pure bitrate–resolution controller (ARCADE–NoSR) streaming at 1080p with no explicit bandwidth cap. As shown in Figure 4.12(a,b), ARCADE–NoSR maintains VMAF comparable to baseline WebRTC while achieving an average of **24% lower sending rate**. This shows that even without client-side upscaling, the RL controller can exploit content complexity and server load to avoid over-encoding easy segments.

4.13 Compute and Power Savings

We profile CPU/GPU utilization and power at both server and client. On the server side, ARCADE yields substantial savings when operating in the renderer-controlled regime, where the game is rendered at 540p/360p and upscaled on the client. As shown in Figure 4.13, this reduces server load by up to **62% CPU** and **41% GPU** compared to

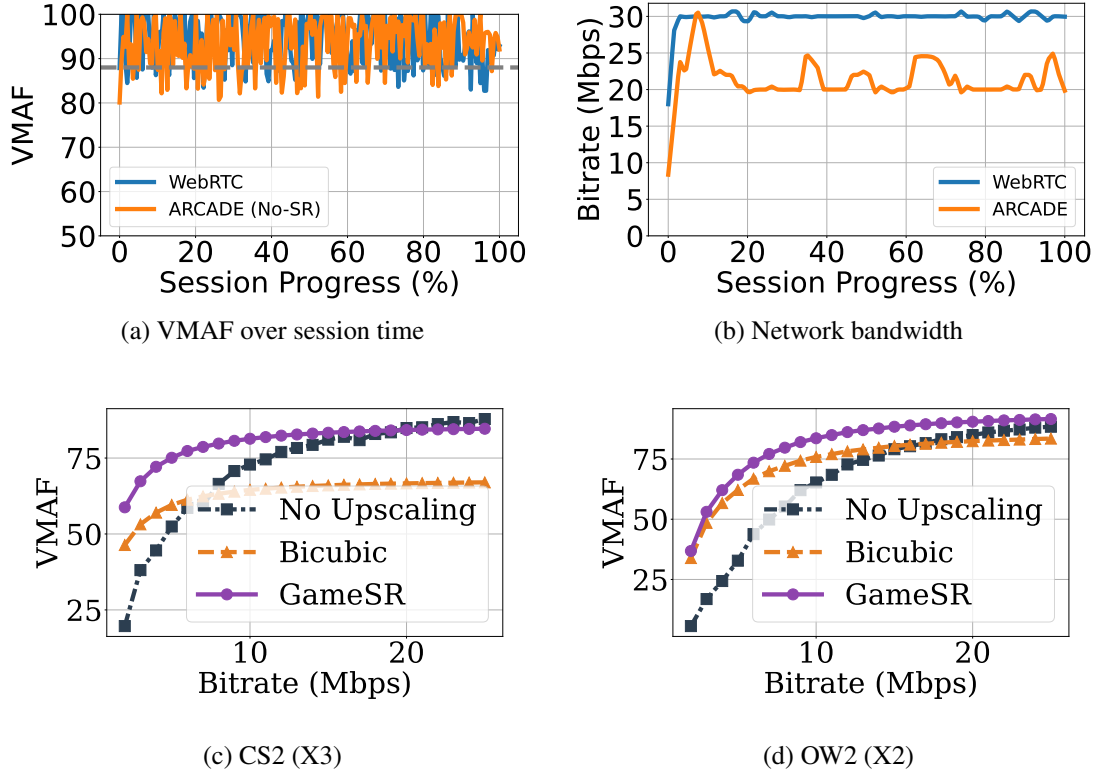


Figure 4.12: Quality and bandwidth for baseline WebRTC and ARCADE-NoSR on OW2 with unconstrained GCC-driven bandwidth (a,b), and rate-distortion results for GameSR at different bitrates and scaling factors (c,d).

1080p WebRTC.

The additional compute for the GOP-level complexity predictor and Joint Optimizer is modest, contributing only about $\sim 5\%$ server GPU overhead together with an additional 226 MiB of server GPU memory to host the MLP-based next-GOP complexity predictor and the offline RL policy, and is only incurred at GOP boundaries. This modest overhead enables intelligent bitrate and resolution decisions that yield system-wide efficiency gains far exceeding it. Even when native low-resolution rendering is unavailable and we fall back to the downsampled path, lowering the encoded resolution still yields **22–25% server CPU** savings.

On the client side, the SR backend introduces a median $\sim 18\%$ GPU overhead at $2\times$ upscaling, but this is offset by lower decoding and post-processing cost per frame. On our RTX A4000 client, this configuration results in a $\sim 20\%$ reduction in average system power compared to 1080p WebRTC. These gains arise because baseline WebRTC always renders and encodes at 1080p, whereas ARCADE adaptively lowers render resolution based on bandwidth, scene complexity, and current utilization while offloading reconstruction to the client-side SR module.

4.14 User Study

To assess real-world perceptual quality, we conducted a subjective user study, which was submitted to the University of Ottawa’s Research Ethics Board and received a letter of exemption. We compared ARCADE against baseline WebRTC using CS2 and OW2

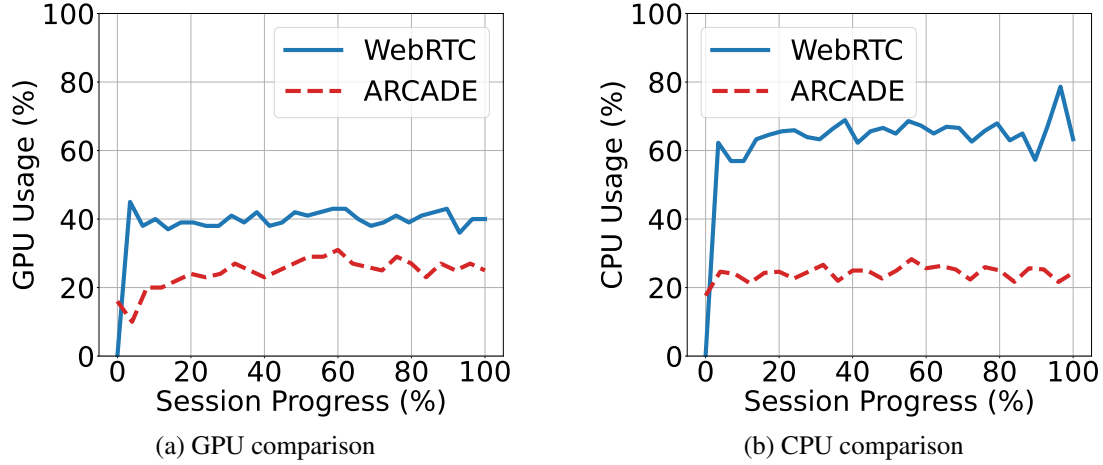


Figure 4.13: Comparison of normalized CPU and GPU utilization between baseline WebRTC and ARCADE.

Table 4.1: User study results: visual quality MOS (mean $\pm$ standard deviation across participants) achieved by ARCADE and WebRTC for CS2 and OW2.

System	30 Mbps (OW2)	30 Mbps (CS2)	8.5 Mbps (OW2)	8.5 Mbps (CS2)
WebRTC	4.40 $\pm$ 0.26	4.34 $\pm$ 0.17	2.84 $\pm$ 0.49	3.06 $\pm$ 0.43
ARCADE (Downsampled)	4.55 $\pm$ 0.38	4.54 $\pm$ 0.37	3.90 $\pm$ 0.32	4.09 $\pm$ 0.34
ARCADE (Rendered)	–	4.31 $\pm$ 0.28	–	–

under the same testbed setup, but with real users playing and rating the sessions. Each scenario was tested under two bandwidth scenarios (30 Mbps and 8.5 Mbps), using latency traces from [126].

We evaluated two system variants: (1) **Downsampler-controlled**, where the game renders at 1080p server-side and frames are downsampled before encoding, simulating low-resolution streaming; and (2) **Renderer-controlled**, where the game engine natively renders at lower resolution (e.g., 540p) to directly reduce processing load. Native 540p rendering was tested only in CS2 at 30 Mbps, where the ARCADE RL policy consistently selected it; at 8.5 Mbps, 360p was selected in high-complexity scenes, though API constraints prevented mid-session resolution switching, requiring adaptive downsampling. In OW2, rendering constraints blocked direct capture of low-resolution renders, so live sessions used downsampling.

In total, **15 participants** each completed **18 sessions** across both games (~ 270 sessions total). Among participants, 33% were experienced gamers, 40% were occasional gamers, and 27% had little or no prior gaming experience. Half of the sessions were served by baseline WebRTC, the other half by ARCADE. The order was randomized, and participants were unaware of which system was being used. After each session, participants rated overall quality on a 1–5 MOS scale [129]. Session length depended on each game’s structure: every OW2 session lasted a fixed 5 minutes, matching its fixed match objective, whereas CS2 sessions ended when the player was eliminated and therefore varied between roughly 2 and 8 minutes.

The results in Table 4.1 show that for the constrained bandwidth condition (8.5 Mbps),

ARCADE improves average perceived quality by **38%** and **34%** for OW2 and CS2, respectively. For example, the average MOS was 3.06 (poor to average quality) for CS2 using WebRTC, which improved to 4.09 (good quality) using ARCADE. Across all conditions the standard deviations remain modest (0.17–0.49 MOS; Table 4.1), indicating consistent agreement among participants.

For the high-bandwidth case (30 Mbps), ARCADE’s principal benefit is a substantial reduction in consumed bandwidth (by up to 50%) and CPU/GPU resources (by up to 62%/41%), while perceived visual quality remains essentially on par with WebRTC: the MOS gains of **3.4%** (+0.15) and **4.6%** (+0.20) for OW2 and CS2 are small and, as the analysis below shows, not statistically significant after multiple-comparison correction.

Statistical analysis. To determine whether the differences in perceived quality between ARCADE and the WebRTC baseline were statistically meaningful, we performed a per-condition statistical analysis on the Mean Opinion Scores collected from our 15 participants. Because the MOS ratings are ordinal and each participant rated both systems under identical conditions, we used the two-sided Wilcoxon signed-rank test, a non-parametric test for paired ordinal data that does not assume a normal distribution. As the comparison was repeated across four conditions, two games (CS2 and OW2) at two bandwidth levels (30 Mbps and 8.5 Mbps), we applied the Holm–Bonferroni correction to control the family-wise error rate, and all reported p -values are post-correction. To characterise the magnitude of each difference, we additionally report the matched-pairs rank-biserial correlation r as an effect size, computed from the same signed ranks used by the Wilcoxon test, together with 95% bootstrap confidence intervals on the median difference.

The analysis reveals a clear bandwidth-dependent pattern. Under the constrained 8.5 Mbps condition, ARCADE yielded a large and statistically significant improvement in perceived visual quality for both games: the mean MOS rose from 2.84 to 3.90 for OW2 (a gain of +1.06) and from 3.06 to 4.09 for CS2 (a gain of +1.03), both with $p = 0.004$ and a maximal effect size of $r = 1.0$, indicating that every participant rated the ARCADE session at least as high as the paired WebRTC session. Under the high-bandwidth 30 Mbps condition, the differences were small and did not reach significance after correction: the mean MOS gain was +0.15 for OW2 ($p = 0.19$) and +0.20 for CS2, where the raw p -value of 0.03 rose to 0.06 once the Holm–Bonferroni correction was applied, falling just short of the 0.05 threshold. We report these corrected values transparently, as the result reflects an expected ceiling effect: at 30 Mbps both systems already score near the top of the scale (approximately 4.5), leaving little headroom for a visible quality improvement. Notably, when the same analysis was applied to the overall-experience rating rather than the visual sub-score alone, ARCADE’s improvement was statistically significant across all four conditions (Holm-corrected $p \leq 0.025$), indicating that even at high bandwidth, where ARCADE’s principal benefit is a substantial reduction in bandwidth and server compute rather than a visible quality gain, participants still rated the overall experience higher.

4.15 Generality to Different SR Backends

While ARCADE is SR-agnostic, we validate its plug-in design by instantiating the client-side backend with three interchangeable upscalers under the *same* pipeline and

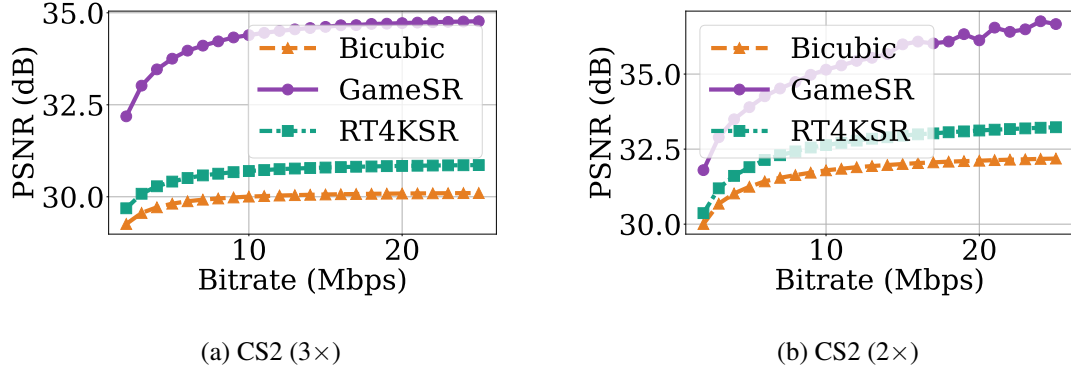


Figure 4.14: Generality of ARCADE to different SR backends. We instantiate ARCADE with bicubic, RT4KSR, and GameSR at (a) $3\times$ and (b) $2\times$ upscaling. The controller and pipeline are unchanged; differences in curves arise solely from each upscaler’s latency–quality trade-off.

RL controller: (i) bicubic, (ii) RT4KSR [50], and (iii) GameSR. All expose the same LR→HR interface at a fixed scale, and ARCADE itself is unchanged when swapping them. Heavier SR models such as EDSR [40], LatticeNet [49], or SwinIR [23] offer strong quality but typically exceed cloud-gaming latency and memory budgets.

Figure 4.14 plots PSNR vs. bitrate for CS2 at $3\times$ and $2\times$ upscaling. With identical RL and rate control, differences reflect only each backend’s latency–quality trade-off: RT4KSR outperforms bicubic, while GameSR achieves the highest PSNR. This is expected because GameSR is designed specifically for gaming content and exploits the structured, repetitive visual patterns common in games. Thus, ARCADE supports diverse SR modules and can adopt future upscalers that meet per-device latency budgets.

As shown in the quantitative SR comparison in Table 3.2 (Chapter 3), GameSR achieves the best PSNR on CS2 (37.99 dB) at the lowest inference time (4.12 ms), making it the most suitable backend for latency-constrained cloud gaming. Higher-quality models such as SwinIR require ~ 2 seconds per frame, ruling them out for interactive use.

4.16 Practical Considerations and Limitations

Overhead Analysis. At the server side, the Complexity Analyzer and Joint Optimizer run once per GOP with negligible overhead. At the client side, the SR backend reuses existing GPU/NPU and the Model Selector runs only at session start. As quantified in Section 4.13, this translates to overall lower server load, reduced bitrate, and power reduction. As demonstrated in the hardware scalability analysis of Chapter 3, the SR backend meets real-time latency across heterogeneous hardware, from mobile SoCs to high-end GPUs.

Generalization to Different Games. ARCADE is game-agnostic, evaluated on two distinct titles. Adapting to a new game requires offline training of three components: the SR backend, Complexity Analyzer, and Joint Optimizer. Like commercial upscalers (e.g., DLSS), GameSR is trained per game on captured gameplay, and about ten diverse

sessions suffice. The Joint Optimizer is trained offline on traces collected under emulated network conditions.

Robustness. ARCADE is robust to occasional mispredictions. The Complexity Analyzer predicts the next GOP’s complexity from a sliding window of the current and previous two GOPs (three GOPs in total), so errors do not accumulate, each affects only a single GOP before correction. The Joint Optimizer enforces a bounded action space: bitrate is capped by GCC and resolution restricted to 360p/540p/1080p, preventing network overload or invalid actions. With these safeguards, errors remain locally contained. If SR is disabled or too slow, ARCADE falls back to complexity-aware bitrate control at 1080p, performing no worse than WebRTC while still adapting to content complexity.

4.17 Summary

This chapter presented ARCADE, an adaptive cloud-gaming framework that jointly optimizes bandwidth and compute while preserving quality. By coupling a complexity-aware offline RL controller with a pluggable SR backend, ARCADE lowers rendering resolution and encoder bitrate when conditions permit. Crucially, this reduces server-side resource usage through *two* distinct mechanisms acting together: rendering the frame at a lower resolution directly reduces GPU rendering work, while encoding the smaller frame at a lower bitrate reduces compression cost and transmission bandwidth. The server therefore saves resources both in rendering and in compression, rather than trading one for the other.

The key findings of this chapter are:

1. ARCADE achieves **up to 50% bandwidth savings** compared to baseline WebRTC while maintaining target VMAF quality.
2. Server-side compute savings of **up to 62% CPU and 41% GPU** when operating in renderer-controlled mode.
3. Under bandwidth-constrained scenarios (8.5 Mbps), ARCADE improves perceived quality by **33–38%** compared to WebRTC.
4. User study with 15 participants confirms these savings do not degrade experience, with ARCADE sessions rated equal or better in MOS.
5. The modular SR backend design allows incorporating any real-time upscaler, demonstrated with bicubic, RT4KSR, and GameSR.

These reductions translate to lower GPU provisioning and CDN costs, or more concurrent users per server. A promising future direction is pairing real-time no-reference quality metrics with on-device online RL, letting ARCADE refine its policy from live sessions while preserving privacy. The next chapter extends these ideas to virtual reality cloud gaming (Chapter 5), where bandwidth constraints are even more severe and latency requirements more stringent.

Chapter 5

GameSRVR: Joint Super-Resolution and Colorization for Cloud VR Gaming

This chapter reuses content and material from the following publication with permission:

G. Bakhtiariazad, H. U. Rehman, S. Shirmohammadi, and M. Hefeeda, “iSR: Super-resolution for Immersive Cloud VR Gaming Platforms,” accepted at ACM Multimedia Systems Conference (MMSys’26), Hong Kong, 2026.

The GameSRVR extension presented in this chapter will be submitted to ACM Transactions on Multimedia (TOMM).

5.1 Introduction

As introduced in Chapter 1, Virtual Reality (VR) cloud gaming faces even more severe constraints than flat-screen streaming: stereoscopic rendering at 4K+ resolution and 90–144 Hz [130], with motion-to-photon latency below 20 ms [131]. The global VR gaming market was valued at 32.5B USD in 2024 and is projected to reach 109.6B USD by 2030 [132], yet delivering these experiences over the network remains a major challenge: streaming stereoscopic frames at high resolution and frame rate demands substantial bandwidth that far exceeds conventional 2D cloud gaming [133, 134].

This chapter addresses the third research question of this thesis: *How can we extend neural upsampling techniques to address the unique challenges of stereoscopic VR cloud gaming, including stereo redundancy and ultra-low latency requirements?*

To keep the two systems distinct, this chapter is organized in two clearly separated parts and presents them in that order. **We first cover the iSR study in full**—a stereo-redundancy approach developed in collaboration (the prior work credited above)—before introducing our own extension. iSR transmits one view in full color and the other in monochrome, and reconstructs the missing chrominance and resolution on the client through *separate*, sequential colorization and super-resolution stages. It demonstrated that this asymmetric transmission reduces VR streaming bandwidth by up to 56% while preserving perceptual quality. Its principal limitation, which motivates the remainder of the chapter, is that this two-stage *sequential* pipeline incurs 73.61 ms of inference at $4\times$ scaling, exceeding real-time VR budgets. **We then present GameSRVR**, the primary contribution of this chapter, which builds on iSR’s stereo-redundancy insight but replaces its sequential pipeline with a unified model that *jointly* performs

stereo-aware colorization and super-resolution in a single forward pass. GameSRVR retains iSR’s bandwidth advantage while reducing inference time to ~ 11 ms, a $6.7\times$ speedup, with end-to-end cloud-streaming latency near 20 ms at 2K across 8–24 Mbps, enabling practical real-time $4\times$ upscaling for cloud VR gaming. To make this separation explicit: Sections 5.4 cover iSR end to end, after which Section 5.5 onward is devoted to GameSRVR.

The main contributions of this chapter, all relating to the GameSRVR extension rather than the iSR motivating study, are:

1. **GameSRVR**, a joint colorization-and-SR model that unifies lightweight stereo feature extraction, half-resolution attention-based color transfer, and GameSR-based upsampling (Chapter 3) into a single end-to-end architecture.
2. **Architectural optimizations** including depthwise-separable convolutions, norm-free inference, and half-resolution attention that reduce inference from 73.61 ms to ~ 11 ms at $4\times$ scaling while preserving perceptual quality.
3. **A joint training strategy** combining SR, colorization, pre-reconstruction, and Charbonnier losses with optional knowledge distillation and a dynamic colorization weight schedule.
4. A **high-resolution VR gaming dataset** collected from popular games (Beat Saber, Pavlov VR, VRChat) to simulate realistic streaming scenarios.

Bandwidth Challenges in Cloud VR Gaming

5.1.1 Measurement Study

To assess the bandwidth requirements of VR cloud gaming platforms, we conducted a measurement study using the Shadow PC VR cloud gaming platform. The experimental setup is illustrated in Figure 5.1. The cloud servers were located in Portland, Oregon, while the player was located in a North America-based lab. The player used a Meta Quest 3 headset connected via a 5 GHz Wi-Fi network with approximately 500 Mbps bandwidth. We recorded three one-minute sessions for each of three VR games (Beat Saber, Pavlov VR, and VRChat), streamed at 4K stereo resolution, and computed per-game distributions of instantaneous bitrate.

The resulting bandwidth usage distributions are shown in Figure 5.2. All three games require bitrates between 72–78 Mbps, demonstrating the high bandwidth challenge facing cloud VR gaming platforms. These results align with prior experiments reported in [135, 136].

5.2 VR System Requirements and Limitations of Existing Approaches

Modern HMDs render 4K–8K per eye at 90–144 Hz with motion-to-photon latency under 20 ms [137, 138]. A single stereo frame at 4K per eye comprises approximately 16 million pixels, and sustaining 90 Hz stereo playback requires over 2.8 billion pixels per second, making VR one of the most bandwidth-intensive streaming scenarios.

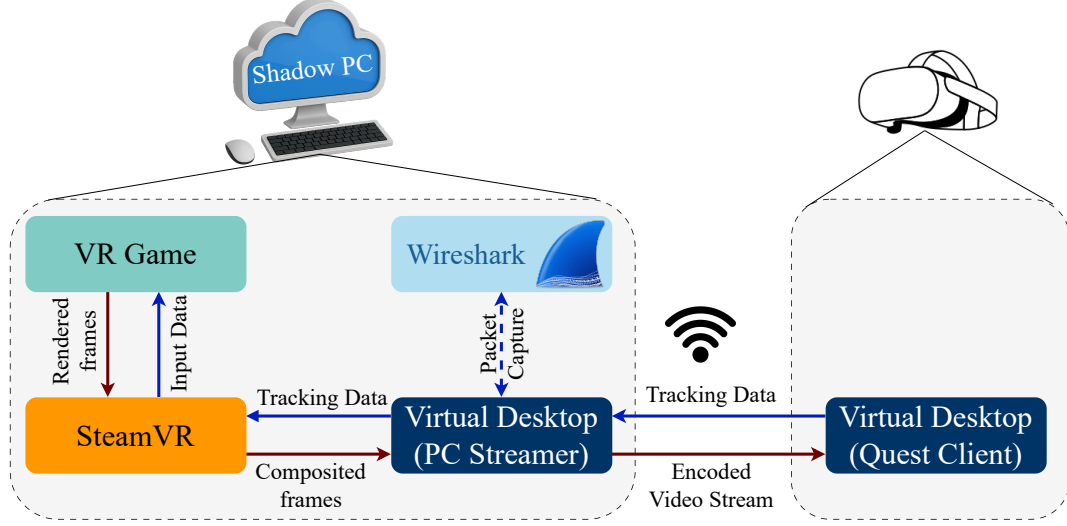


Figure 5.1: Experimental setup for measuring bandwidth usage in cloud VR gaming.

Existing approaches to mitigate this burden fall short for interactive VR cloud gaming. Foveated rendering leverages non-uniform visual acuity [139] but requires precise eye-tracking hardware [140] and may produce artifacts during rapid gaze shifts [141]. SR methods designed for 2D [142], stereoscopic [143], or 360° video [85, 144] are too slow for interactive use, CVHSSR [24], the state-of-the-art VR SR model, requires ~ 3 seconds per stereo pair at $4\times$ scaling. As discussed in Chapter 2, commercial upscalers (DLSS, FSR, XeSS) require engine-internal data unavailable to cloud gaming clients [145].

5.3 Design Goals and Key Insight

5.3.1 Design Goals

Our VR cloud gaming SR system must: (i) achieve substantial bandwidth reduction without compromising perceived quality; (ii) maintain inference latency below 20 ms per stereo pair at $4\times$ scaling to meet VR’s motion-to-photon budget; and (iii) operate engine-agnostically, requiring no access to game source code, renderer data structures, or metadata, enabling deployment across both modern and legacy VR titles.

5.3.2 Key Insight: Stereo Redundancy

A key inefficiency in VR streaming is the duplication of chrominance information across stereo views. While luminance encodes the structural differences needed for depth perception, chroma is nearly identical between left and right views. Transmitting both views in full color wastes bandwidth without improving perceptual quality.

This observation opens a new design direction: transmit one view in color and the other in grayscale, reconstructing the missing chroma on the client-side. This insight underpins the design of both iSR and GameSRVR, which integrate colorization and super-resolution to reduce bandwidth requirements while preserving high visual fidelity.

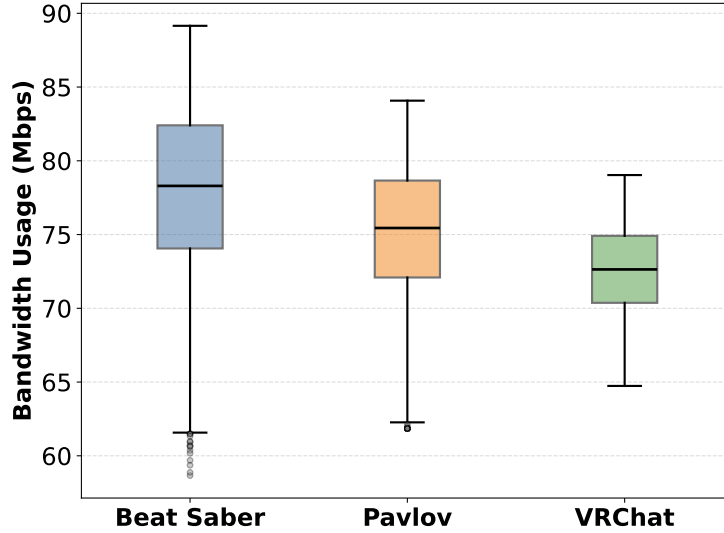


Figure 5.2: Bandwidth usage distribution for three VR games.

Table 5.1: Potential bandwidth savings from sending chrominance components of only one view for various chroma subsampling formats.

Right View + Left View	Drop in Pixel Cost	Bandwidth Saving
YUV444 + YUV400	6.0 $\rightarrow$ 4.0	33.3%
YUV422 + YUV400	4.0 $\rightarrow$ 3.0	25.0%
YUV420 + YUV400	3.0 $\rightarrow$ 2.5	16.7%
YUV444 + YUV444	6.0 $\rightarrow$ 6.0	0.0%

5.3.3 Potential Bandwidth Savings

Table 5.1 quantifies the potential bandwidth savings from transmitting chrominance for only one view under standard chroma subsampling formats.

These are raw sample counts; in practice, codecs further benefit from the lower entropy of monochrome frames, often exceeding these theoretical savings.

5.4 iSR: Motivating Study

Before presenting GameSRVR, we summarize the iSR study, which established the feasibility of stereo-redundancy-based bandwidth reduction for VR cloud gaming and motivated the joint architecture developed in this chapter.

5.4.1 System Overview

Figure 5.3 provides an overview of iSR. On the server side, the scene renderer produces stereo frames at low resolution. The left view is decolorized (retaining only luminance), and both views are compressed before streaming. On the client side, a *colorization module* reconstructs missing color in the left view using information from the right view, followed by a separate *super-resolution module* that upsamples both views to high

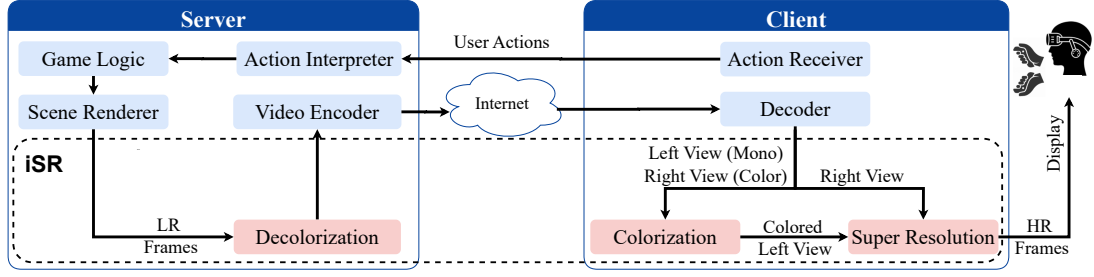


Figure 5.3: Overview of the iSR system for reducing required bandwidth in VR cloud gaming platforms. The left view is transmitted in monochrome and reconstructed to full color on the client, while the right view retains color throughout the pipeline.

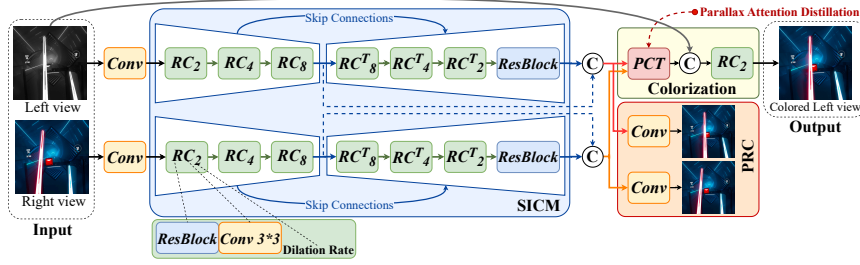


Figure 5.4: Architecture of the lightweight colorization module used in iSR, derived from SiCNet and optimized for interactive VR cloud gaming.

resolution. The operation is summarized as:

$$(L_{LR}^{\text{Mono}}, R_{LR}^{\text{Color}}) \xrightarrow{\text{iSR}} (\hat{L}_{HR}^{\text{Color}}, \hat{R}_{HR}^{\text{Color}}) \quad (5.1)$$

The colorization module was derived from SiCNet [146], optimized for VR gaming through module pruning (removing Stereo Fusion and Consistency modules), depthwise-separable convolutions, and post-training quantization (Figure 5.4). The SR module employed GameSR (Chapter 3). This pipeline integrates seamlessly with commercial cloud VR platforms (CloudXR, Shadow PC, Virtual Desktop) without modifications.

5.4.2 Key Results

The iSR study established several important findings. At $4\times$ scaling, the system reached *watchable* quality ($\text{VMAF} \geq 70$) at only 8 Mbps compared to 18 Mbps for the baseline, a 56% bitrate saving. Perceptual quality was high across all tested VR games, as shown in Table 5.2. User studies confirmed high perceived quality with MOS scores of 4.88/5 (Beat Saber) and 4.75/5 (Pavlov VR) at $4\times$ scaling. The colorization module achieved imperceptible color differences ($\Delta E_{2000} < 2.3$) for over 85% of reconstructed frames.

5.4.3 Limitation: Sequential Pipeline Latency

Despite its strong quality results, iSR’s *two-stage sequential* design, colorize first, then super-resolve, introduces a critical bottleneck. At $4\times$ scaling (the most practical configuration for quality), the combined inference time is 73.61 ms per stereo pair (Table 5.3), well above the 20 ms motion-to-photon budget required by VR headsets. While higher scaling factors ($\times 8$, $\times 10$) operate on smaller inputs and achieve lower latency, they sacrifice quality significantly.

Table 5.2: Perceptual quality of iSR across three VR games and all scaling factors.

Scale	Beat Saber			Pavlov VR			VRChat		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
$\times 4$	39.50	0.9835	0.0128	40.23	0.9789	0.0154	37.18	0.9701	0.0283
$\times 6$	34.52	0.9659	0.0581	36.95	0.9598	0.0642	33.40	0.9565	0.0580
$\times 8$	34.49	0.9674	0.0472	35.72	0.9531	0.0718	32.70	0.9490	0.0705
$\times 10$	34.46	0.9562	0.0559	34.98	0.9447	0.0823	32.30	0.9408	0.0820

Table 5.3: iSR inference time across scaling factors. Only $\times 8$ and $\times 10$ meet the 20 ms VR budget, but at significant quality cost.

Scale	Time (ms)	Beat Saber PSNR (dB)
$\times 4$	73.61	39.50
$\times 6$	32.92	34.52
$\times 8$	24.38	34.49
$\times 10$	17.68	34.46

Table 5.4 provides the full runtime and memory profile of iSR across all scales. As the upscale factor increases, inference becomes both faster and lighter in memory because larger scales operate on proportionally smaller inputs. At $\times 6$, iSR runs at 32.9 ms, within the 50 ms threshold generally considered acceptable for VR. The $\times 8$ and $\times 10$ configurations further reduce runtime to 24.4 ms and 17.7 ms, with the latter meeting the stricter 20 ms requirement to prevent motion sickness.

This trade-off between quality and latency motivates the design of GameSRVR: a unified architecture that achieves real-time $4\times$ upscaling by jointly performing colorization and super-resolution, eliminating the sequential bottleneck while preserving high perceptual quality.

5.5 GameSRVR: Joint Colorization and Super-Resolution

GameSRVR replaces iSR’s two-stage pipeline with a single end-to-end model that jointly performs stereo feature extraction, attention-based color transfer, and neural super-resolution. The architecture comprises three tightly integrated components: (i) a **FastSICModule** for lightweight stereo feature extraction and pre-reconstruction, (ii) a **FastColorization** module that performs attention-based color transfer at half resolution, and (iii) a **GameSR-based upsampler** that super-resolves both views using shared weights. Figure 5.5 illustrates the complete pipeline.

5.5.1 FastSICModule: Lightweight Stereo Feature Extraction

The original SICModule in iSR uses a three-level encoder-decoder with dilated convolutions and instance normalization, incurring substantial overhead. GameSRVR replaces it with a **FastSICModule** applying three simplifications: (i) *depthwise-separable convolutions* factoring all spatial convolutions into depthwise and pointwise stages, reducing

Table 5.4: Inference time and memory usage of iSR across scaling factors.

Scale	Time per frame (ms)	Memory (GB)
$\times 4$	73.61	2.310
$\times 6$	32.92	0.932
$\times 8$	24.38	0.624
$\times 10$	17.68	0.538

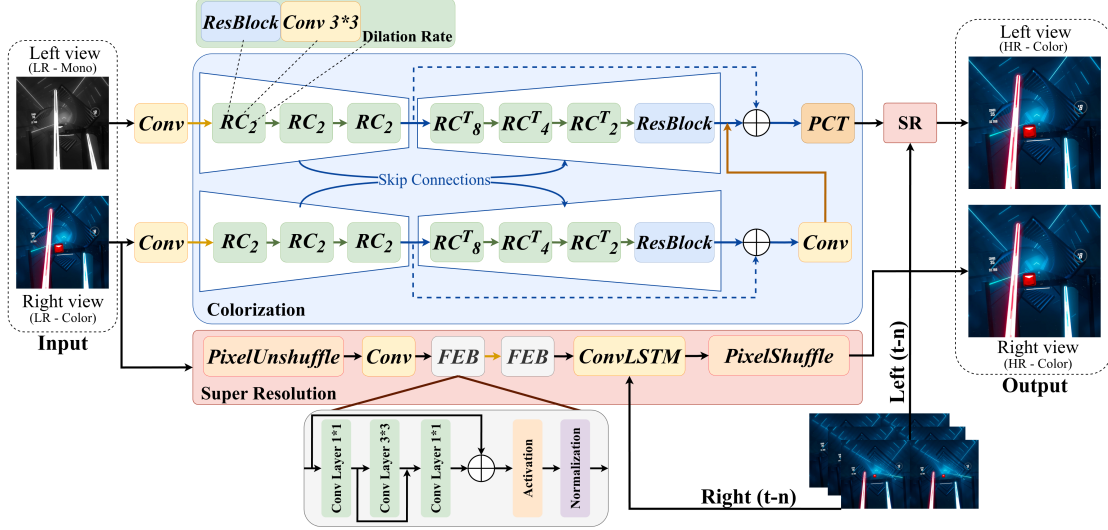


Figure 5.5: GameSRVR pipeline. The left (monochrome) and right (color) LR views are processed jointly: the FastSICModule extracts multi-scale stereo features, FastColorization transfers color from right to left at half resolution, and the GameSR-based upsampler super-resolves both views to HR. The right view bypasses colorization and is directly super-resolved.

MACs by $\sim k^2 \times$; (ii) a *shallow two-level encoder-decoder* with lightweight skip fusion using learned blending parameters; and (iii) *norm-free inference*, replacing all LayerNorm/GroupNorm with identity at test time to eliminate expensive per-sample statistics for high-resolution VR frames. The FastSICModule processes both views through shared weights and outputs multi-scale features along with pre-reconstruction RGB estimates.

5.5.2 FastColorization: Half-Resolution Attention

Stereo color transfer is the most expensive component due to its dense $W \times W$ attention map per scanline ($\mathcal{O}(B \cdot H \cdot W^2 \cdot C)$ at full resolution). GameSRVR introduces **FastColorization** with two optimizations: (i) *pooled attention*, feature maps are average-pooled by $2 \times$ before computing attention, reducing cost by $4 \times$, then upsampled back after color transfer; and (ii) a *streamlined architecture* replacing dual independent projection paths with a single shared projection and simplifying the refinement stage to a single HinBlock fused with a 3×3 convolution. The module performs attention-based color warping from the right (color) view to the left (monochrome) view, producing a colored left view at LR for upsampling.

5.5.3 GameSR-Based Upsampler

Both the colorized left view and the original right view are super-resolved using a single **GameSR head** with shared weights (Chapter 3), producing $4\times$ upscaled output through PixelShuffle. The right view bypasses colorization entirely, as it retains full color throughout the pipeline.

5.5.4 Joint Training Strategy

GameSRVR is trained end-to-end with a composite loss function that supervises all three stages simultaneously:

Super-Resolution Loss ($\mathcal{L}_{\text{SR}}$). L1 loss between the predicted HR outputs and ground-truth HR frames for both left and right views:

$$\mathcal{L}_{\text{SR}} = \|L_{\text{SR}} - L_{\text{HR}}^{\text{GT}}\|_1 + \|R_{\text{SR}} - R_{\text{HR}}^{\text{GT}}\|_1 \quad (5.2)$$

Colorization Loss ($\mathcal{L}_{\text{col}}$). L1 loss between the predicted LR colorized left view and its ground-truth color version. A **dynamic weight schedule** starts with weight λ_{col} at full strength during early training (to establish color transfer), then linearly decays to zero over a configurable number of epochs, allowing the SR loss to dominate in later stages:

$$\lambda_{\text{col}}(e) = \begin{cases} \lambda_0 & \text{if } e < e_{\text{start}} \\ \lambda_0 \cdot \left(1 - \frac{e - e_{\text{start}}}{e_{\text{span}}}\right) & \text{if } e_{\text{start}} \leq e < e_{\text{start}} + e_{\text{span}} \\ 0 & \text{otherwise} \end{cases} \quad (5.3)$$

Pre-Reconstruction Loss ($\mathcal{L}_{\text{pre}}$). L1 loss on the LR RGB estimates produced by the FastSICModule, supervised against bicubic-downsampled HR ground truth. This auxiliary loss stabilizes early training by providing gradient signal to the feature extractor before the SR head has converged.

Perceptual Loss ($\mathcal{L}_{\text{perc}}$). Charbonnier loss $\rho(x) = \sqrt{x^2 + \varepsilon^2}$ ($\varepsilon = 10^{-3}$) applied to both SR outputs, which acts as a smooth L1 surrogate that is more robust to outliers.

Knowledge Distillation (Optional). When a pretrained teacher model (StereoWarp) is available, an additional L1 loss aligns the student’s attention map M with the teacher’s attention map M_t . The distillation weight decays exponentially over training to allow the student to specialize.

The total loss is:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{SR}} + \lambda_{\text{col}}(e) \cdot \mathcal{L}_{\text{col}} + \lambda_3 \mathcal{L}_{\text{pre}} + \lambda_p \mathcal{L}_{\text{perc}} + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}} \quad (5.4)$$

where $\lambda_1 = 1.0$, $\lambda_3 = 0.05$, and λ_p is configurable. Training uses AdamW with a multi-step learning rate schedule.

Table 5.5: Quality and inference comparison between iSR (two-stage sequential) and GameSRVR (joint model) at $4\times$ scaling. GameSRVR reduces inference from 73.61 ms to ~ 11 ms while maintaining comparable quality.

Game	Model	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	Inference (ms) $\downarrow$
Beat Saber	iSR (sequential)	39.50	0.9835	0.0128	73.61
	GameSRVR (joint)	38.47	0.9851	0.0237	11.0
Pavlov VR	iSR (sequential)	40.23	0.9789	0.0154	73.61
	GameSRVR (joint)	40.14	0.9791	0.0539	11.0

5.6 Experimental Setup

5.6.1 VR Gaming Dataset

We use the VR gaming dataset collected for the iSR study, comprising three games: **Beat Saber** (fast-paced rhythm game with color-critical gameplay), **Pavlov VR** (competitive FPS with complex lighting and viewpoint shifts), and **VRChat** (social platform with diverse user-generated visual styles). Since no public VR stereo gaming dataset existed, recordings were captured on a Meta Quest 3 at native resolution ($4,128 \times 2,208$ pixels) via the built-in screen recorder.

5.6.2 Experimental Configuration

All experiments were conducted on a Linux workstation (Ubuntu 22.04) with an Intel Xeon w7-2495X (32 GB RAM) and NVIDIA RTX A4000 GPU (20 GB VRAM). Implementation used PyTorch 2.5.1 accelerated with CUDA 11.8 and cuDNN 9.1. GameSRVR was trained end-to-end with AdamW optimizer. We fix the scaling factor to $4\times$ for all GameSRVR experiments, as this represents the practical operating point balancing quality and bandwidth savings established by the iSR rate-distortion analysis.

5.7 Evaluation

5.7.1 Quality Comparison: GameSRVR vs. iSR

Table 5.5 compares GameSRVR against iSR at $4\times$ scaling on the same test sets. GameSRVR achieves comparable perceptual quality while delivering a $6.7\times$ inference speedup.

On Beat Saber, GameSRVR trades ~ 1 dB PSNR (38.47 vs. 39.50) for a slight SSIM improvement (0.9851 vs. 0.9835). On Pavlov VR, the gap is negligible: 0.09 dB PSNR with virtually identical SSIM. LPIPS is slightly higher for GameSRVR due to colorization simplifications, but these differences are unlikely to be perceptible at VR frame rates. The critical result is that GameSRVR processes a stereo pair in ~ 11 ms, well within the 20 ms budget, whereas iSR’s 73.61 ms is unsuitable for real-time VR.

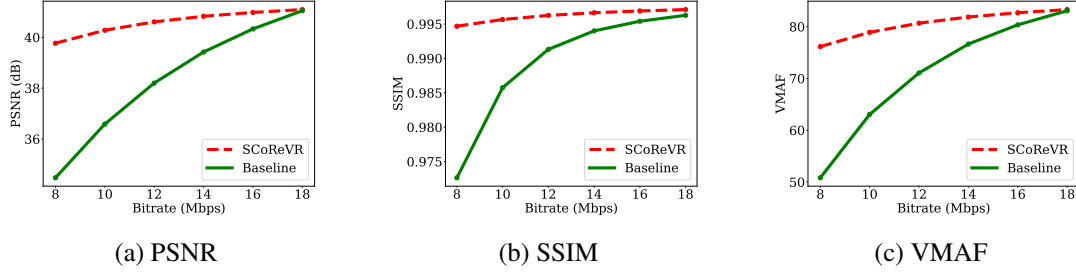


Figure 5.6: Rate-distortion comparison of iSR for $4\times$ upscaling against the baseline across different bitrates.

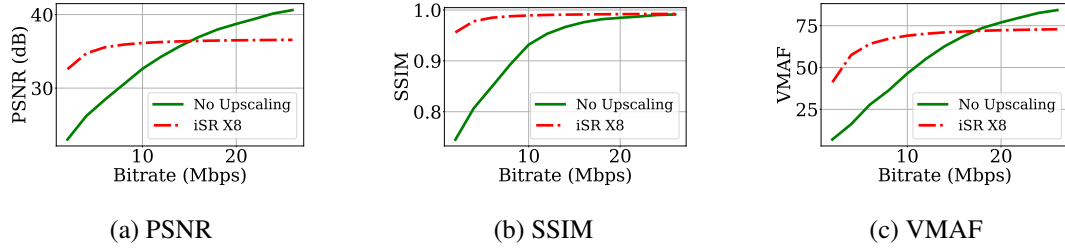


Figure 5.7: Rate-distortion comparison of iSR for $8\times$ upscaling against the baseline across different bitrates.

5.7.2 Rate-Distortion Analysis

Figures 5.6 and 5.7 present the iSR rate-distortion curves at $4\times$ and $8\times$ scaling. Following prior work [96, 127, 128], we categorize quality by VMAF: *watchable* ≥ 70 , *good* ≥ 80 , *excellent* ≥ 90 . At $4\times$, iSR reaches watchable quality at 8 Mbps vs. 18 Mbps baseline (56% savings) and good quality at 12 Mbps vs. 24 Mbps (50% savings). At $8\times$, watchable quality is achieved at 12 Mbps but plateaus around VMAF 73, serving as a viable fallback under severe bandwidth constraints. Since GameSRVR preserves comparable quality at $4\times$ (Table 5.5), similar rate-distortion behavior is expected.

5.7.3 End-to-End Latency

To quantify latency in a realistic setup, we used GameLab [147] (Chapter 6) streaming 2K stereo content. Figure 5.8 reports the end-to-end latency breakdown for iSR at $\times 8$ scaling. Across 8–24 Mbps, iSR’s total latency (~ 34 ms) remains within 1–2 ms of the WebRTC baseline (~ 33 ms) despite performing stereo colorization and $8\times$ SR on the client. The latency budget is redistributed: iSR’s encode+network component shrinks by $2.5\text{--}3\times$ due to lower-resolution transmission, while client-side processing grows correspondingly.

Using the same GameLab setup, we additionally evaluate GameSRVR end-to-end latency with the updated measurements in `My_Main_GameSRVR.csv`. Figure 5.9 shows that across 8, 16, and 24 Mbps, baseline WebRTC records total end-to-end latencies of 33.95, 32.33, and 32.59 ms, while GameSRVR achieves 20.10, 20.05, and 20.08 ms, respectively. This corresponds to a 38–41% reduction in total latency (39.1% on average). The main gain comes from reduced encode+network delay: 20.30/18.37/19.00 ms

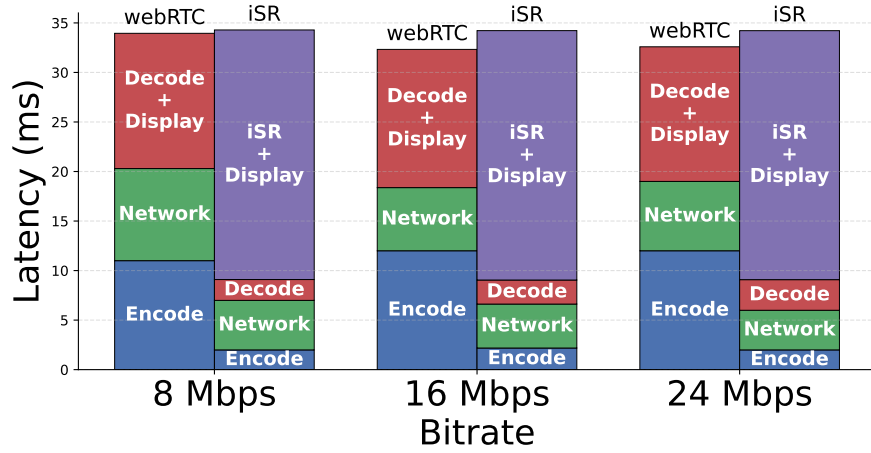


Figure 5.8: End-to-end latency breakdown across bitrates for WebRTC and iSR.

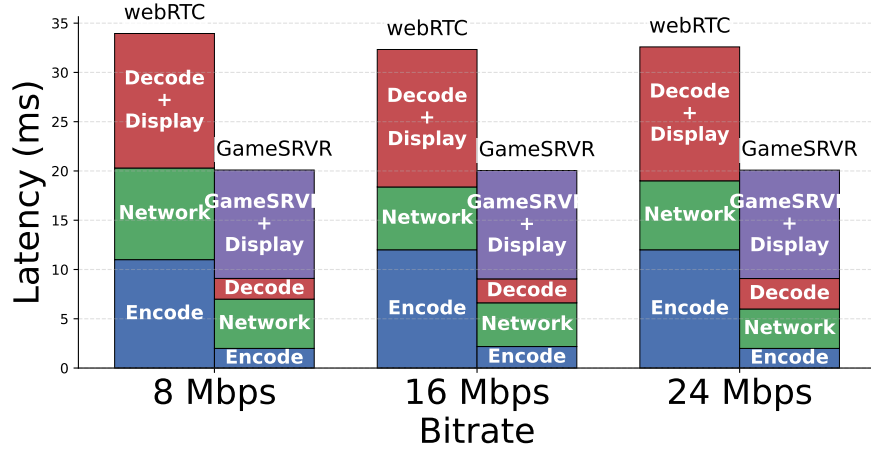


Figure 5.9: End-to-end latency breakdown across bitrates for WebRTC and GameSRVR (2K stereo streaming, GameLab).

(WebRTC) versus 6.99/6.62/5.99 ms (GameSRVR), while client processing remains approximately 11 ms.

5.7.4 User Study

We conducted a user study with **8 participants** (37.5% experienced, 37.5% occasional, 25% non-gamers), each viewing iSR-upscaled clips from all three games across all scaling factors.

The results in Table 5.6 demonstrate consistently high perceptual quality across all games and scale factors. At $\times 4$ scale, the operating point used by GameSRVR, MOS scores were **4.88/5** for Beat Saber, **4.75/5** for Pavlov VR, and **4.25/5** for VRChat. Since GameSRVR achieves comparable objective quality to iSR at $4\times$ (Table 5.5), these subjective scores are expected to be representative of GameSRVR’s perceptual quality as well.

Table 5.6: MOS for iSR across all VR games and scale factors.

Game	$\times 4$	$\times 6$	$\times 8$	$\times 10$
Beat Saber	4.88	4.50	4.00	3.75
Pavlov VR	4.75	4.38	3.88	3.63
VRChat	4.25	3.88	3.63	3.13

Table 5.7: Colorization results ($\times 4$ upscaling).

Model	Inf (ms) $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
iSR	66.89	39.78	0.983	0.007
SiCNet	–	21.03	0.526	0.120

5.7.5 Comparison with State-of-the-Art

Colorization Context. Traditional colorization relies on user input [148] or CNN-based hallucination [149, 150], neither suitable for real-time VR. Reference-based approaches [151] are more promising since one stereo view can guide the other, but prior work assumes unrelated scenes, motivating the stereo-aware colorization in iSR and GameSRVR.

Colorization Comparison. Table 5.7 compares against SiCNet [146], the state-of-the-art stereo colorization method. We evaluated a pre-trained SiCNet model directly on our gaming content. While SiCNet delivers strong performance on natural scenes, it degrades substantially on gaming assets, with PSNR dropping to 21.03 and SSIM to 0.526. This performance gap arises because SiCNet was trained primarily on natural stereo photographs, where color and depth relationships follow real-world statistics; gaming content contains synthetic lighting, sharp edges, and non-photorealistic textures that differ significantly. Regarding inference cost, we were unable to execute SiCNet on our largest GPU (24 GB VRAM, NVIDIA RTX 3090) due to excessive memory demand, and CPU inference required approximately 8 seconds per frame, highlighting the impracticality of direct deployment.

Colorization Ablation. Tables 5.8 and 5.9 report the impact of architectural modifications and feature channel width on colorization quality across all scaling factors. Each optimization (module pruning, depthwise-separable convolutions, quantization) progressively reduces inference time and memory with only marginal quality degradation, and these trends hold across all scales. For example, at $\times 4$, the full optimization pipeline reduces inference from 123 ms to 66.8 ms while losing only 0.98 dB PSNR; at $\times 8$, the same modifications reduce inference from 35.3 ms to 19.7 ms with 0.75 dB PSNR loss. For channel width, we select different widths depending on the scale: for $\times 4$ and $\times 6$, 32 channels provide strong reconstruction at substantially lower latency; for $\times 8$ and $\times 10$, 48 channels give a better trade-off at higher compression ratios.

Super-Resolution Comparison. Table 5.10 compares our SR module against CVHSSR [24] (state-of-the-art VR SR), EDSR [152], and SwinIR [153] at $4\times$ scaling, along with our results across all scaling factors. Our approach achieves the best quality

Table 5.8: Impact of cumulative SiCNet modifications across all scales.

Scale	Variant	PSNR $\uparrow$	SSIM $\uparrow$	Inf (ms) $\downarrow$	Mem (GB) $\downarrow$
$\times 4$	Baseline	41.10	0.9869	123.0	3.0
	– SFM/SCM	40.50	0.9863	114.2	2.7
	+ Depthwise Conv	40.28	0.9846	109.5	2.3
	+ Quantization	40.12	0.9823	66.8	2.0
$\times 6$	Baseline	39.70	0.9852	41.0	1.298
	– SFM/SCM	39.49	0.9848	35.9	1.128
	+ Depthwise Conv	38.22	0.9822	34.3	0.932
	+ Quantization	38.71	0.9821	22.9	0.778
$\times 8$	Baseline	40.56	0.9852	35.3	0.724
	– SFM/SCM	40.16	0.9848	32.2	0.648
	+ Depthwise Conv	39.99	0.9822	26.3	0.550
	+ Quantization	39.81	0.9821	19.7	0.488
$\times 10$	Baseline	34.92	0.9713	6.5	0.566
	– SFM/SCM	34.05	0.9710	6.0	0.508
	+ Depthwise Conv	33.89	0.9705	5.7	0.458
	+ Quantization	33.71	0.9705	5.3	0.396

(44.06 dB PSNR) at real-time speed (12.87 ms), whereas CVHSSR requires ~ 3 seconds per frame. At higher scales ($6\times$ – $10\times$), inference time decreases further due to smaller inputs.

Figure 5.10 provides a visual comparison of iSR reconstructions against ground truth for representative left-eye frames from all three VR games at $8\times$ bandwidth reduction. The upscaled frames are nearly indistinguishable from the ground truth, confirming the high perceptual quality of the stereo-redundancy approach.

5.7.6 Perceptual Color Difference

To directly assess perceptual color fidelity beyond signal-based metrics, we use ΔE_{2000} [154], a widely adopted metric in imaging and color science that measures perceptual differences between two colors in the CIELAB space. Unlike PSNR or SSIM, which quantify pixel-wise similarity or structural consistency, ΔE_{2000} is designed to model the nonlinear characteristics of human color perception, small variations in some hues can be highly noticeable even if PSNR and SSIM remain high. Lower values indicate closer color matches, with $\Delta E < 2.3$ considered imperceptible and $\Delta E < 5.1$ generally regarded as acceptable in visual applications.

Figure 5.11 reports the cumulative distribution of ΔE_{2000} across all scale factors. Over 85% of reconstructed frames achieve $\Delta E < 2.3$ and more than 98% remain below $\Delta E < 5.1$, confirming perceptually faithful colorization even under aggressive down-sampling.

5.7.7 Colorization and Upscaling Trade-off

This section analyzes the relationship between visual quality, bitrate savings, and computational efficiency. Table 5.10 shows that iSR preserves strong visual quality across all scales: at $\times 4$, it achieves the highest fidelity (PSNR ≈ 44 , SSIM > 0.99), while

Table 5.9: Impact of feature channel width across all scales. Bold indicates the selected configuration per scale.

Scale	#Ch	PSNR $\uparrow$	SSIM $\uparrow$	Inf (ms) $\downarrow$	Mem (GB) $\downarrow$
$\times 4$	16	35.90	0.9751	38.91	2.016
	32	40.12	0.9823	66.61	2.340
	48	41.43	0.9844	106.28	2.720
$\times 6$	16	34.12	0.9722	15.70	0.778
	32	38.71	0.9821	22.88	0.932
	48	40.28	0.9846	34.68	1.128
	64	41.28	0.9857	48.39	1.298
$\times 8$	16	32.15	0.9646	9.44	0.488
	32	38.62	0.9811	12.93	0.550
	48	39.81	0.9832	19.70	0.648
	64	35.34	0.9797	25.40	0.724
$\times 10$	16	34.21	0.9712	6.85	0.396
	32	37.00	0.9804	8.66	0.458
	48	39.18	0.9830	10.53	0.508
	64	39.65	0.9837	13.51	0.566

higher scales trade quality for efficiency. Notably, $\times 8$ and $\times 10$ still deliver SSIM above 0.96 with runtimes below 14 ms and 10 ms respectively, making them attractive for low-bandwidth or latency-sensitive scenarios. The bitrate–quality trade-off is evident in the rate-distortion curves (Figures 5.6 and 5.7): larger scaling factors substantially reduce required bitrate but also lead to more rapid quality saturation, whereas $\times 4$ provides smoother quality gains at higher bitrates, making it preferable when bandwidth is limited but visual fidelity remains important. Combined with colorization, which removes redundant chroma information, iSR achieves complementary spatial and chroma bandwidth reductions, colorization eliminates inter-view redundancy while SR counteracts quality loss from downsampling, making the approach practical for real-world cloud VR deployment.

5.8 Summary

This chapter presented GameSRVR, a joint colorization-and-super-resolution model for cloud VR gaming. Building on iSR’s stereo-redundancy insight (56% bandwidth savings), GameSRVR replaces the sequential two-stage pipeline with a unified architecture featuring a FastSICModule (shallow depthwise-separable encoder-decoder with norm-free inference), FastColorization (half-resolution attention reducing the $\mathcal{O}(W^2)$ cost by $4\times$), and a joint training strategy with dynamic loss scheduling. These innovations reduce inference from 73.61 ms to ~ 11 ms, a $6.7\times$ speedup, and in end-to-end 2K cloud streaming reduce total latency to ~ 20.1 ms (about 39% lower than baseline WebRTC across 8–24 Mbps), enabling practical real-time $4\times$ VR upscaling. The next chapter presents GameLab, the research infrastructure that enabled evaluation of all systems in this thesis.

Table 5.10: Super-resolution comparison at $4\times$ and results across all scales.

Model / Scale	Inf (ms) ↓	PSNR ↑	SSIM ↑	LPIPS ↓
<i>Comparison at $4\times$ scaling</i>				
GameSR	12.87	44.06	0.991	0.005
CVHSSR	2946	42.79	0.989	0.007
EDSR	105	39.74	0.978	0.012
SwinIR	1898	39.70	0.979	0.013
<i>GameSR model across all scales</i>				
$\times 4$	12.87	44.06	0.991	0.005
$\times 6$	19.49	36.29	0.971	—
$\times 8$	13.96	36.01	0.960	—
$\times 10$	9.94	36.71	0.973	—

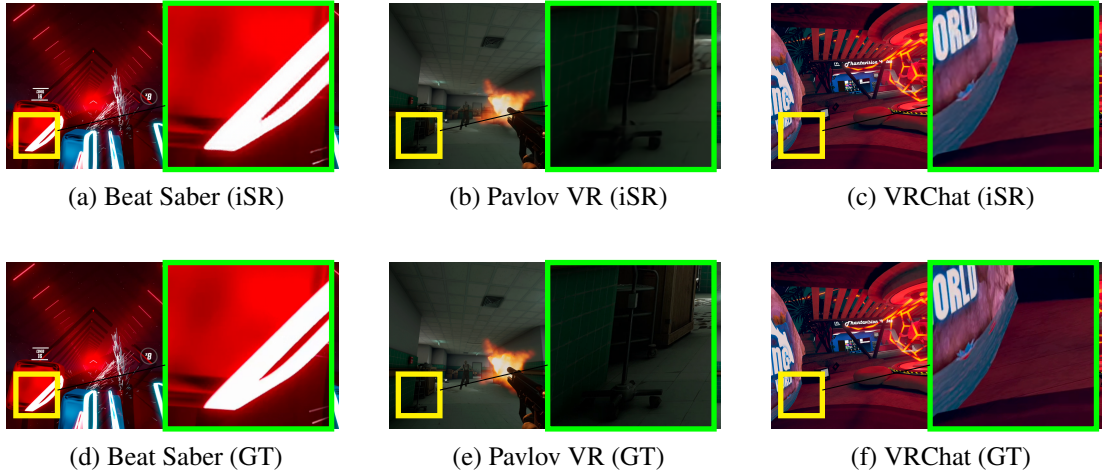
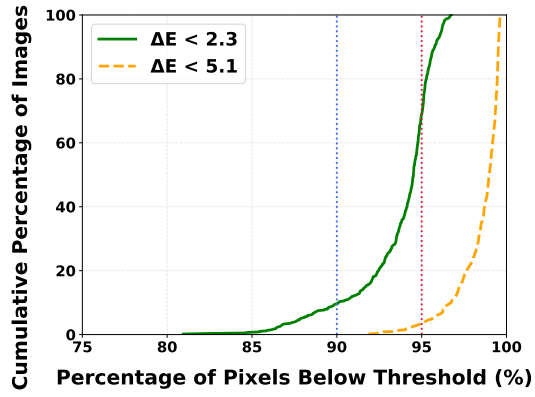
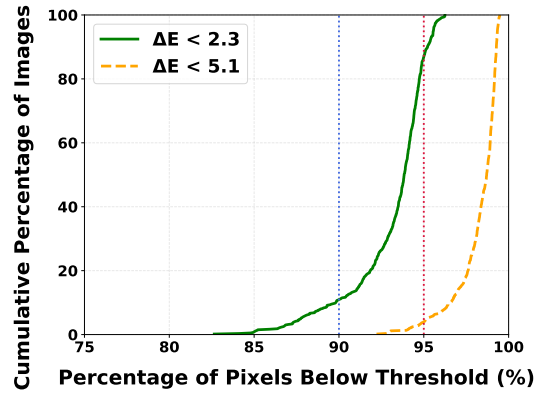


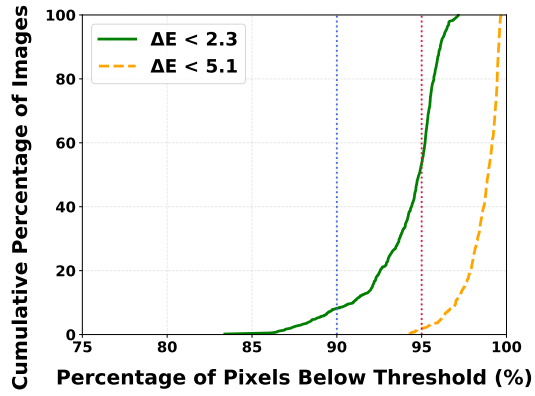
Figure 5.10: Visual quality produced by iSR with $8\times$ bandwidth reduction on sample left-eye frames from Beat Saber, Pavlov VR, and VRChat. The upscaled frames (top) are nearly indistinguishable from the ground truth (bottom).



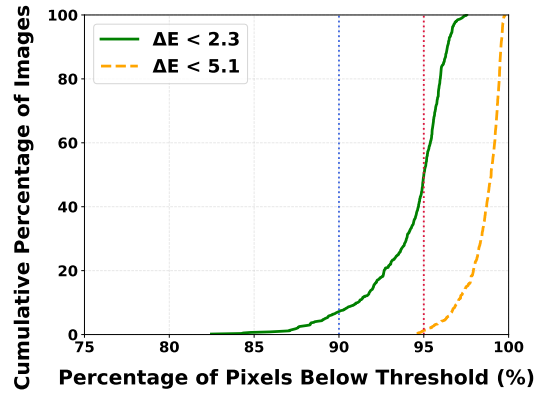
(a) Scale $\times 4$



(b) Scale $\times 6$



(c) Scale $\times 8$



(d) Scale $\times 10$

Figure 5.11: Cumulative distribution of ΔE_{2000} across all scales (from iSR evaluation).

Chapter 6

GameLab: AI-Enabled Cloud Gaming Research Testbed

This chapter reuses content and material from the following publication with permission:

H. U. Rehman, S. Shirmohammadi, I. Amer, and M. Hefeeda, “GameLab: AI-Enabled Cloud Gaming Testbed,” ACM Multimedia Systems Conference (MMSys), Hong Kong, 2026.

6.1 Introduction

The neural upsampling systems presented in this thesis, GameSR (Chapter 3), ARCADE (Chapter 4), and iSR (Chapter 5), required rigorous end-to-end evaluation under realistic network conditions. Existing cloud gaming stacks and deployment-oriented pipelines, however, are not designed for *AI-in-the-loop* systems research: researchers must implement and integrate many components while keeping the system modifiable enough to rapidly prototype new AI models.

This chapter presents **GameLab**, an open-source, Python-first, AI-enabled cloud gaming testbed built on WebRTC. Unlike existing stacks such as GamingAnywhere [25], Sunshine/Moonlight [155, 156], and Unity Render Streaming [157], GameLab provides:

1. **Programmable hook points** on both server and client for integrating AI modules (super-resolution, denoising, object detection, QoE estimation, learned rate control) without re-architecting the system.
2. **QR-based frame identity** enabling reliable full-reference quality evaluation (PSNR, SSIM, VMAF) under packet loss, reordering, and frame duplication.
3. **Unified runtime logging** of transport and application traces for online control and offline analysis.

GameLab served as the evaluation platform for all experiments in Chapters 3–5 and is publicly available at <https://github.com/haseebfazal/ai-testbed.git>.

6.2 Related Work and Positioning

Several cloud gaming stacks and deployment-oriented pipelines exist but are not designed for rapid AI-in-the-loop systems research.

Academic and Open Stacks. GamingAnywhere [25] is a widely-cited open cloud gaming system that has served as a baseline in many research prototypes. However, it primarily targets end-to-end game streaming rather than programmable experimentation; its publicly released codebase reflects a legacy toolchain (last release v0.8.0, January 2015, targeting Win32 with Visual Studio 2010), making it unsuitable for modern AI-in-the-loop workflows that require hook points for model inference, fine-grained instrumentation, and tight integration with contemporary GPU runtimes and Python training stacks.

Community Streaming Ecosystems. Sunshine/Moonlight [155, 156] have matured into practical low-latency remote-play solutions, but prioritize real-world usability over research extensibility. They generally do not expose research-grade interfaces for inserting, benchmarking, and swapping inference modules at different pipeline stages, nor do they provide standardized ground-truth alignment mechanisms for full-reference quality evaluation under realistic network conditions where packets can be lost and reordered.

Engine-Specific and Learned Pipelines. Unity Render Streaming [157] and similar engine-integrated pipelines are tied to specific game engines and unavailable for unmodified commercial titles. Content-aware and learned encoding systems (e.g., CAVE [16], DeepGame [108]) optimize individual components within tightly-coupled stacks but do not expose clean insertion points for comparing alternative AI modules on equal footing.

AI in Cloud Gaming Pipelines. A growing body of work integrates learned components into cloud gaming delivery. LevelUp [158] demonstrates edge-assisted neural enhancement under bandwidth constraints. On the quality side, gaming-oriented perceptual models span dimension-based estimators [159], no-reference deep models such as NDNetGaming [99], mobile cloud-gaming predictors such as GAMIVAL [160], and AI-based Key Quality Indicator estimation from network-side metrics [161]. Content-adaptive encoding for cloud gaming includes region-aware bit allocation [162] and selective object encoding to reduce bitrate with limited QoE impact [163]. Collectively, these efforts motivate a testbed with clean insertion points for AI vision models and learned controllers.

Positioning of GameLab. In contrast to the above, GameLab is designed to support rapid AI-in-the-loop prototyping via server-side and client-side hooks, and robust, repeatable objective evaluation in real network environments, enabling researchers to assess and compare AI modules (RL-based rate controllers, SR, denoising, object detection, QoE predictors, learned controllers) under consistent streaming conditions.

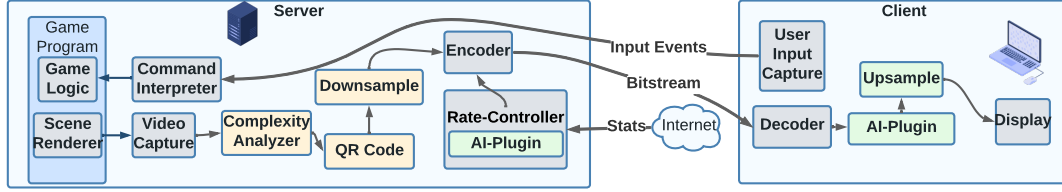


Figure 6.1: Design of GameLab. The testbed provides two Python hook points: (i) server *pre-encode* for frame analysis/modification and QR frame IDs, and (ii) client *post-decode* for pluggable AI modules such as SR and object detection. Detailed logs enable offline and online analyses.

6.3 System Architecture

Figure 6.1 provides an overview of GameLab. The testbed follows the standard cloud gaming loop (input $\rightarrow$ render/capture $\rightarrow$ encode/transport $\rightarrow$ decode/display) while adding two key capabilities: programmable hook points for AI modules and robust measurement via QR-based frame identity. GameLab runs two parallel paths: a lightweight **TCP control channel** that relays keyboard and mouse events from client to server, and a **WebRTC media channel** that streams encoded frames to the client.

6.3.1 Server-Side Pipeline

The sender (`server.py`) is implemented in Python on top of `aiortc` [147]. The server executes the game locally and captures rendered output using OS-level screen capture (MSS library [164]) rather than engine or frame-buffer hooks. This design choice avoids anti-cheat and game-integrity restrictions on injection, supports unmodified commercial games across diverse runtimes, and is broadly applicable and reproducible across setups. Captured frames are converted to BGR arrays (OpenCV-native layout, avoiding extra color-space conversions and enabling fast OpenCV/NumPy operations) before encoding. A custom `CaptureScreen` class grabs a centered region of a selected monitor at a configurable resolution and target frame rate; this stage is intentionally kept simple and scriptable so researchers can substitute alternative capture backends without touching the transport stack.

Pre-Encode Hook Points. Before encoding, GameLab exposes a generic pre-encode hook where researchers can plug in AI modules:

- **Complexity Analysis:** Computes EVCA-style spatial complexity using blockwise DCT energy on the luminance plane [165], implemented in `ComplexityAnalyzer` via `torch_dct` and a weighted DCT mask. Scores are logged per-frame or averaged over a GOP window (controlled by `-enable-complexity` and `-gop-len` flags). The design is modular and can be replaced with any other real-time complexity estimator.
- **Downsampling:** Reduces the transmitted pixel rate (e.g., $\times 2/\times 3$) before encoding to lower bandwidth demand. Downsampling is applied early in the hook so that complexity analysis and QR instrumentation operate on the resolution that will actually be transmitted (enabled via `-downsample`).

- **QR-Based Frame Identity:** Overlays a compact QR code encoding a unique frame ID and metadata (index, timestamp, resolution) for robust alignment under loss and duplication (enabled via `ENABLE_QUALITY_LOGGING`). The QR code is implemented in `QualityLogger` as a small corner patch (default top-right) with configurable size and margin; by default, we use 160 px (about 1.2% of a 1920×1080 frame) and scale it with resolution to limit occlusion. The QR region can be masked or cropped during metric computation if strict visual fidelity is required (`-{}-quality-logging`, `-{}-qr-size`, `-{}-qr-corner`).

These modules serve as reference implementations; the same pre-encode hook is generic and can host arbitrary processing (e.g., denoising, filtering, feature extraction, overlays) without changing the rest of the pipeline.

Rate Control Hook. GameLab provides a shared-memory interface that makes rate control fully customizable. Concretely, the server-side frame-processing stage (pre-encode) writes a 32-bit target bitrate into a named shared-memory region; the Python streaming process polls this value at a configurable interval and applies it to the encoder configuration. Placing the hook before the encoder lets controllers combine raw-frame signals (e.g., EVCA complexity) with transport metrics (RTT/loss/jitter), enabling plug-ins ranging from handcrafted heuristics to RL-based policies without re-plumbing the WebRTC stack. When disabled, the system falls back to default GCC behavior.

Encoding, Transport, and Network Statistics. After pre-encode processing, frames are wrapped as `VideoFrame` (bgr24) and delivered through the WebRTC pipeline. We use a modified fork of `aiortc` that (i) adds transport/statistics logging used by our measurement scripts, (ii) increases the default jitter buffer capacity to sustain high-bitrate game streams under transient delay variation, and (iii) exposes a server-side rate-control API that allows external controllers to program the encoder target bitrate at runtime. The fork also exposes optional hardware-accelerated encode/decode (e.g., NVENC/CUVID) when PyAV is built with CUDA support, while the default remains software H.264 for portability. The sender periodically queries WebRTC statistics and logs effective send bitrate via `RTCOutboundRtpStreamStats` (computed from bytesSent over time); these signals are consumed by rate-control plug-ins. We default to CPU encode/decode since CPU↔GPU transfer overhead can offset hardware-encoder speedups in our measurements.

6.3.2 Client-Side Pipeline

The receiver (`client.py`) terminates the WebRTC session, decodes video frames, and applies post-decode processing prior to display and/or saving.

Post-Decode AI Interface. Decoded frames are first converted to RGB NumPy arrays, then to a tensor-first representation (`torch.Tensor` in CHW format, float in $[0, 1]$) on GPU when available. This makes it straightforward to insert PyTorch/TensorFlow inference modules at the receiver, including super-resolution, denoising, segmentation, or QoE predictors, without leaving Python:

```
img_chw = TF.to_tensor(rgb).to(self.device).clamp(0, 1)
```



```
if self.upsample:
    img_chw = upsample_chw(img_chw, scale, mode) # swap SR here
```

Built-in Examples. The client includes reference implementations for the plug-in workflow: (i) real-time upsampling via `torch.nn.functional.interpolate` (nearest/bilinear/bicubic) as a baseline, and (ii) GPU object detection using a TorchVision SSDLite MobileNetV3 detector, executed every N frames with configurable thresholding and a top- K cap for overlay rendering. These serve as reference implementations for integrating heavier AI modules.

GPU-Native Visualization and Frame Dumping. For interactive experimentation, the client displays tensors directly using `cudacanvas` [166], avoiding GPU→CPU copies that would distort latency measurements. Optionally, the receiver can dump processed frames to disk (PNG) for offline evaluation; combined with the server’s QR-based frame identity, this enables robust alignment for full-reference metrics (PSNR/SSIM/VMAF) across bandwidth sweeps and real network traces.

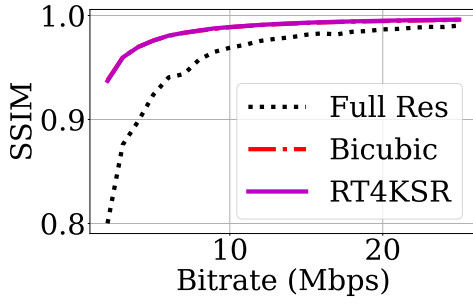
Client Transport Statistics. The client also logs receive-side transport bitrate using `RTCTransportStats` (`bytesReceived`), enabling correlation of network dynamics with post-decode AI module behavior and observed visual quality.

6.4 Configuring and Running GameLab

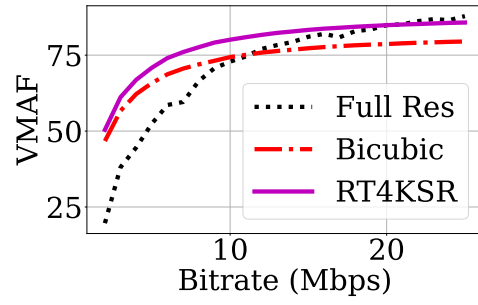
GameLab provides scriptable entry points for both endpoints: `Server/server.sh` launches the WebRTC sender (together with the C++ input server for keyboard/mouse relay), and `Client/client.sh` launches the WebRTC receiver (and the input client). Researchers can enable or disable each capability by editing a small set of flags in these scripts. Server-side flags include `DOWNSAMPLE`, `ENABLE_QUALITY_LOGGING`, `ENABLE_COMPLEXITY`, and `ENABLE_EVCA_BITRATE`; client-side flags include `UPSAMPLE`, `UPSAMPLE_MODE`, `DETECT`, and `SAVE_FRAMES`.

Reliable Full-Reference Evaluation via QR Identity. When `ENABLE_QUALITY_LOGGING=1`, the server overlays a compact QR code per frame encoding a unique identifier and metadata. If `SAVE_FRAMES=1` is enabled on the client, decoded frames can be paired with their server-side references by decoding the QR IDs offline, enabling robust PSNR/SSIM/VMAF computation even under real-time artifacts such as loss, duplication, and timestamp drift.

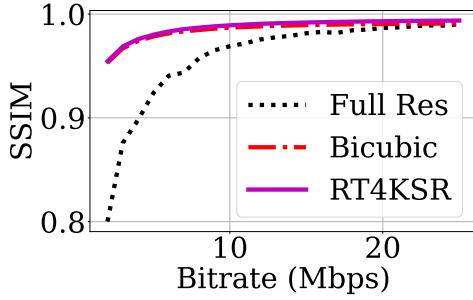
Trace Collection. Both endpoints log time-stamped transport and application signals (e.g., achieved bitrate, RTT, jitter, FPS, and module runtimes). These traces can be redirected to files for offline analysis, or consumed online by external control logic such as shared-memory rate controllers.



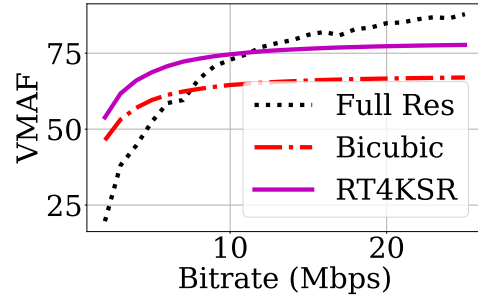
(a) CS2 (SSIM) 540p $\rightarrow$ 1080p



(b) CS2 (VMAF) 540p $\rightarrow$ 1080p



(c) CS2 (SSIM) 360p $\rightarrow$ 1080p



(d) CS2 (VMAF) 360p $\rightarrow$ 1080p

Figure 6.2: CS2 bandwidth sweep: SSIM/VMAF vs. achieved bitrate for three pipelines. Metrics use QR-aligned frames.

6.5 Demonstrations

This section demonstrates how GameLab enables AI-in-the-loop cloud gaming experiments through (i) programmable server and client hook points, (ii) robust measurement via QR-based frame identity, and (iii) unified runtime trace collection. The goal is to make it easy to swap AI modules and reproduce end-to-end sessions while collecting the signals needed for online control and offline analysis. These experiments are presented as feature demonstrations of GameLab rather than as a competition for absolute performance.

6.5.1 Swappable Video Upsamplers

We demonstrate how GameLab supports end-to-end evaluation while swapping receiver-side enhancement modules without changing capture, encoding, or transport. Figure 6.2 compares three pipelines under identical streaming conditions: (i) native 1080p streaming, (ii) server-side downsampling with client-side interpolation baseline, and (iii) the same post-decode hook populated with a neural upsampler (RT4KSR [100]).

Replacing bicubic with any PyTorch/TensorFlow SR model only changes the function applied at the post-decode hook, without affecting the rest of the streaming loop. This capability is difficult to reproduce in many existing pipelines because swapping post-decode AI modules while keeping the streaming loop fixed, and still obtaining reliable full-reference metrics via robust frame alignment, typically requires substantial re-



Figure 6.3: Client-side object detection as a post-decode plug-in. Annotated frames from CS2 and OW2 showing detector outputs computed after WebRTC decoding.

engineering and custom instrumentation. GameLab makes this a single-flag change.

6.5.2 Client-Side Analytics

GameLab can host real-time client analytics at the post-decode hook. Figure 6.3 shows object detection results from two games, illustrating how semantic signals can be produced online for ROI-aware encoding, selective enhancement, or gameplay telemetry.

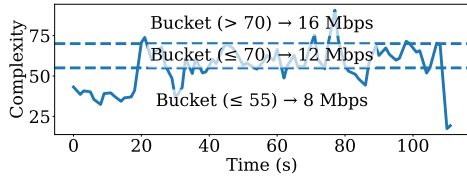
6.5.3 Complexity-Driven Rate Control

GameLab enables content-aware control by exposing the server-side EVCA complexity analyzer and shared-memory rate-control hook. The included example maps the online spatial complexity score to a simple bitrate ladder (three buckets); whenever the score crosses a threshold, the controller switches to the corresponding rung (e.g., 8/12/16 Mbps):

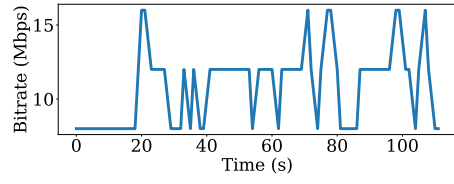
```
# ... inside CaptureScreen.recv(), after computing GOP-average
#     complexity (sc_avg)
if self._bitrate_ctl is not None:
    target_bps, desired_bps, gcc_bps = \
        self._bitrate_ctl.update_from_complexity(sc_avg)
```

Figure 6.4 shows the time series of complexity and resulting bitrate selection using this ladder.

This example is intentionally simple and is meant to demonstrate GameLab’s programmability. Many prior systems expose only the default congestion-control behavior

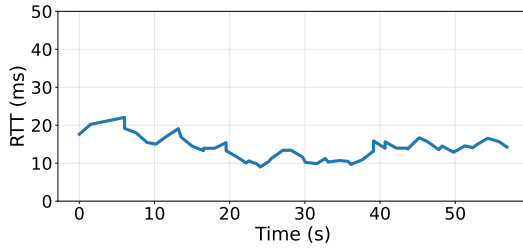


(a) EVCA complexity over time

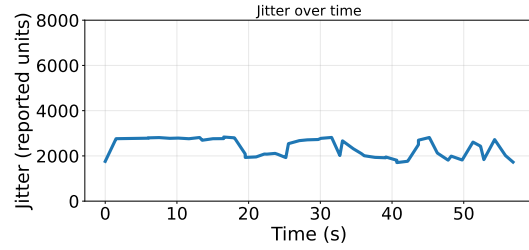


(b) Selected bitrate over time

Figure 6.4: Time-series behavior of EVCA-driven rate control: (left) estimated spatial complexity, and (right) corresponding target bitrate from the ladder.



(a) RTT over time



(b) Jitter over time

Figure 6.5: RTT and jitter traces logged from WebRTC transport statistics during CS2 bandwidth-sweep runs.

as a black box; GameLab instead makes the sender-side control loop programmable, so researchers can rapidly prototype and compare heuristic or learned controllers (e.g., the offline RL policy in Chapter 4) using the same signals and logs, without changing the system architecture.

6.5.4 Transport Logging

GameLab logs WebRTC transport statistics during each run. Figure 6.5 shows representative RTT and jitter traces, enabling researchers to jointly analyze network dynamics with quality metrics and AI-module choices.

6.6 Reproducibility and Overheads

Reproducibility. GameLab exposes its functionality through explicit, script-controlled flags and produces time-stamped logs at both endpoints, making it straightforward to rerun identical sessions while toggling a single capability (e.g., QR identity on/off, downsample factor, post-decode plug-ins, or the rate-control hook). This supports fair comparisons of alternative AI modules and control policies under consistent streaming conditions.

Overheads. The QR identity mechanism is designed to be lightweight (small overlay region per frame) while substantially improving robustness of offline alignment. The EVCA analyzer is optional and configurable (block size and GOP averaging window), incurring overhead only when enabled. For interactive experimentation, the client

displays frames via GPU-native visualization to avoid extra device transfers that would otherwise distort latency measurements.

6.7 Summary

This chapter presented GameLab, an open-source, Python-first AI-enabled cloud gaming testbed that provided the evaluation infrastructure for all neural upsampling systems in this thesis. The key features are:

1. **Server pre-encode and client post-decode hooks** for plugging in ML modules (SR, analytics, learned controllers) without re-architecting the system.
2. **QR-based frame identity** for robust alignment and full-reference quality evaluation (PSNR/SSIM/VMAF) under loss, reordering, and duplication.
3. **Programmable rate control** via shared-memory interface, enabling heuristic or RL-based controllers.
4. **Dual-channel design**: a TCP control channel for keyboard/mouse input relay and a WebRTC media channel for low-latency video streaming.
5. **Unified logging** of transport statistics (bitrate, RTT, jitter) and application metrics for online control and offline analysis.

GameLab enabled the reproducible end-to-end experiments presented throughout this thesis, including bandwidth sweeps with swappable upsamplers, user studies with real gameplay, and RL-based joint optimization evaluation in Chapter 4. The testbed is publicly available to facilitate future research on AI-driven cloud gaming systems.

Chapter 7

Conclusion and Future Work

7.1 Summary of Contributions

This thesis addressed the challenge of delivering high-quality, interactive gaming experiences over bandwidth-constrained networks through neural super-resolution. We developed a comprehensive framework spanning real-time upsampling models, joint optimization algorithms, stereo-aware VR enhancement, and research infrastructure. The following sections summarize the key contributions and findings from each component.

7.1.1 GameSR: Real-Time Neural Upsampling

Chapter 3 presented GameSR, a lightweight, engine-independent super-resolution model that achieves near-parity with commercial upscalers (DLSS, FSR) on perceptual metrics despite operating without engine integration. Through PixelUnshuffle-based feature extraction, reparameterized convolutions, and lightweight ConvLSTM temporal modeling, GameSR runs at 4.12 ms per frame, 30–60 $\times$ faster than CNN baselines and nearly 500 $\times$ faster than SwinIR. End-to-end deployment demonstrated 35–49% bandwidth savings at VMAF ≥ 90 , validated by a user study with 15 participants (MOS 4.73/5 on CS2, 4.70/5 on OW2).

7.1.2 ARCADE: Joint Optimization Framework

Chapter 4 presented ARCADE, the first framework to jointly select rendering resolution, encoding bitrate, and upscaling ratio in real time. Using offline RL (BCQ) with EVCA-based content complexity prediction, ARCADE achieves up to 50% bandwidth savings and 62%/41% CPU/GPU savings compared to baseline WebRTC. A user study with 15 participants confirmed 34–38% improvement in perceived quality under bandwidth-constrained conditions.

7.1.3 GameSRVR: Joint Super-Resolution and Colorization for Cloud VR

Chapter 5 extended neural upsampling to immersive VR. The iSR study established asymmetric color–monochrome stereo transmission, achieving up to 56% bandwidth savings (MOS 4.88/5 on Beat Saber). GameSRVR then unified colorization and super-resolution into a single forward pass, reducing inference from 73.61 ms to ~ 11 ms at

4 $\times$ scaling. Perceptual color fidelity was confirmed with 85% of frames achieving imperceptible color difference ($\Delta E_{2000} < 2.3$). Deployed on the GameLab testbed, GameSRVR achieved over 33% bandwidth savings at watchable quality and an end-to-end latency of ~ 20 ms, meeting the VR motion-to-photon budget.

7.1.4 GameLab: Research Infrastructure

Chapter 6 presented GameLab, an open-source, AI-enabled cloud gaming testbed providing programmable server/client hook points for ML module integration and QR-based frame identity for reliable full-reference quality evaluation under adverse network conditions. GameLab underpinned the evaluation of all contributions in this thesis and is publicly available at <https://github.com/haseebfazal/ai-testbed.git>.

7.2 Addressing the Research Questions

This thesis set out to address four research questions posed in Chapter 1. We now summarize how each has been answered.

RQ1: How can we design a real-time, engine-agnostic neural upsampler for cloud gaming that achieves comparable quality to engine-coupled solutions? GameSR (Chapter 3) demonstrated that combining PixelUnshuffle, reparameterized convolutions, and ConvLSTM temporal modeling yields an architecture that matches DLSS and FSR quality without engine integration, operating fast enough for real-time deployment.

RQ2: How can we jointly optimize rendering resolution, encoding bitrate, and super-resolution to maximize QoE under bandwidth constraints? ARCADE (Chapter 4) showed that offline RL with content complexity awareness can coordinate the full rendering–encoding–upsampling pipeline, yielding multiplicative savings beyond what independent optimization of each component achieves.

RQ3: How can neural upsampling be extended to VR cloud gaming while meeting stricter latency requirements? iSR and GameSRVR (Chapter 5) demonstrated that exploiting stereo color redundancy, combined with a unified colorization-and-SR architecture, enables real-time 4 $\times$ VR upscaling within the 20 ms motion-to-photon budget while substantially reducing bandwidth.

RQ4: What infrastructure is needed to evaluate AI-driven cloud gaming systems? GameLab (Chapter 6) showed that programmable ML insertion points, QR-based frame identity for full-reference evaluation under adverse networks, and shared-memory rate control are essential components, providing the infrastructure that enabled reproducible experiments throughout this thesis.

7.3 Key Findings and Insights

Beyond the specific contributions of each chapter, this thesis yielded several cross-cutting insights:

Gaming Content is Learnable. Unlike natural video, game content is synthetic and highly structured, reusing meshes, textures, shaders, and HUD elements across sessions. This bounded, repetitive structure makes high-frequency details highly recurring and learnable from representative gameplay, enabling lightweight models to achieve quality comparable to much larger architectures trained on diverse natural images.

Engine Independence is Achievable. Commercial upscalers (DLSS, FSR) require engine integration for motion vectors and depth buffers. This thesis demonstrated that engine-independent approaches operating on decoded frames can achieve near-parity quality while enabling deployment across the entire gaming catalog, including legacy titles that will never receive engine updates.

Joint Optimization Yields Multiplicative Benefits. ARCADE showed that coordinating rendering resolution, bitrate, and upscaling decisions yields greater savings than optimizing each component independently. The RL-based controller exploits content complexity and network conditions to avoid both over-encoding simple scenes and under-encoding complex ones.

Stereo Redundancy is Substantial. The iSR study and GameSRVR demonstrated that transmitting chrominance for only one stereo view, combined with joint colorization and super-resolution, can reduce VR bandwidth requirements by over 50% while maintaining imperceptible color differences for the majority of frames.

Research Infrastructure Matters. GameLab enabled reproducible experiments across all thesis components. The QR-based frame alignment was essential for computing reliable full-reference metrics under realistic network conditions with packet loss and reordering.

7.4 Practical Applications

As outlined in Chapter 1, the techniques developed in this thesis are directly applicable to cloud gaming service providers (35–50% bandwidth and CDN cost reduction), game developers and publishers (engine-independent deployment across legacy and modern catalogs), VR and immersive platforms (56% bandwidth savings enabling wireless VR streaming), and mobile/edge cloud gaming (lightweight architectures compatible with mobile GPU/NPU hardware).

7.5 Limitations

While this thesis made significant contributions, several limitations should be acknowledged:

Per-Game Training. GameSR, ARCADE, and GameSRVR require per-game training to achieve optimal quality. While approximately 10 diverse gameplay sessions suffice, this still represents a barrier to immediate deployment on new titles. Cross-game generalization experiments showed reasonable performance when training on shooter

games and testing on other shooters, but more diverse game genres may require separate models.

Hardware Requirements. Real-time neural upsampling requires GPU compute on client devices. While we demonstrated feasibility across a range of hardware (from mobile Adreno 750 to desktop RTX 4090), devices without adequate GPU/NPU capability must fall back to classical interpolation or disable SR entirely.

Offline RL Training. ARCADE’s BCQ-based controller is trained offline on logged sessions. While this avoids degrading QoE during online exploration, it limits adaptation to network conditions or game content not represented in the training data. The behavior policy constrains what the learned policy can safely explore.

VR User Study Scale. The iSR user study included 8 participants. While results were statistically consistent and aligned with objective metrics, a larger study would strengthen confidence in the perceptual findings, particularly for higher scaling factors where quality degradation becomes more noticeable.

Game Genre Coverage. Evaluation focused primarily on first-person shooters (CS2, Overwatch 2), with limited coverage of other genres such as strategy games, platformers, or open-world RPGs. These genres differ substantially in visual complexity, rendering style, and temporal dynamics, and may require distinct training strategies or architectural modifications.

7.6 Future Research Directions

This thesis opens several promising directions for future research:

7.6.1 Cross-Game and Zero-Shot Generalization

Current models are optimized per title to maximize quality under strict real-time constraints. While approximately 10 diverse gameplay sessions suffice for training (Section 7.5), this per-game requirement limits scalability to large catalogs. A key next step is reducing adaptation cost while preserving latency guarantees:

- **Meta-learning for fast per-title adaptation:** approaches such as Model-Agnostic Meta-Learning (MAML) could pre-train on a diverse corpus of game content, producing an initialization from which a new title can be adapted with only minutes of representative gameplay rather than full retraining. The critical constraint is that adaptation must not increase inference latency beyond the frame-time budget established by the base model.
- **Genre-conditioned model families:** train shared backbone architectures with lightweight per-genre adapters (e.g., low-rank adaptation modules) for broad game categories (FPS, sports, strategy, open-world). This approach balances specialization with deployment simplicity—a single backbone is deployed once, and only small adapter weights need to be swapped or downloaded per genre. Our

cross-game generalization results (Section 3.5.6) suggest that games within the same genre share sufficient visual structure to make this viable.

- **Zero-shot robustness benchmarks:** establish a standardized cross-title evaluation protocol measuring both quality (PSNR, SSIM, VMAF, LPIPS) and inference latency across diverse genres and rendering engines. Such a benchmark would enable fair comparison of generalization approaches and identify failure modes (e.g., art styles, HUD layouts, or particle effects that degrade zero-shot performance).

7.6.2 Online Reinforcement Learning

ARCADE currently relies on offline RL because full-reference quality signals (which require access to the original uncompressed frame) are unavailable during live sessions. This limits adaptation to network conditions and game content represented in the training data. Extending to online adaptation requires two prerequisites: a low-latency reward signal and safe policy updates that never degrade the player experience below acceptable thresholds.

- **Real-time lightweight no-reference quality metric:** the central enabler for online RL is a game-specific NR quality estimator that can run at every GOP boundary with sub-millisecond inference and minimal memory overhead. Existing NR metrics such as NDNetGaming [99] and GAMIVAL are too heavy for per-frame deployment. A purpose-built metric, trained on the same gaming content used for SR models and distilled into a compact architecture (e.g., a few convolutional layers operating on downsampled feature maps), could provide a stable per-GOP reward signal that correlates well with VMAF while adding negligible latency to the pipeline. This metric would be used in tandem with online RL to close the loop between quality estimation and adaptive decision-making.
- **Hybrid offline-to-online RL:** initialize policies from the offline-trained BCQ agent (Chapter 4), then perform constrained online fine-tuning using the lightweight NR metric as the reward. Trust-region or behavior-regularized updates (e.g., CQL, Cal-QL) would prevent the policy from deviating too far from the safe offline baseline during early exploration, gradually relaxing constraints as confidence in the online reward signal increases.
- **Safety-aware exploration:** enforce hard bounds on bitrate, resolution, and model selection during online updates, ensuring that exploration never causes quality to drop below watchable thresholds ($\text{VMAF} \geq 70$) or latency to exceed frame-time budgets. Action masking, similar to ARCADE’s GCC-constrained feasible set, can restrict the online policy to a safe operating envelope.
- **Federated policy improvement:** aggregate anonymized trajectory statistics (state-action-reward tuples without raw video) across geographically distributed clients to improve policy robustness across diverse network conditions and hardware configurations without centralizing sensitive gameplay data.

7.6.3 Joint Neural Codec and Super-Resolution

Neural compression and super-resolution share the same core primitives—feature extraction, representation learning, and reconstruction—making them natural candidates

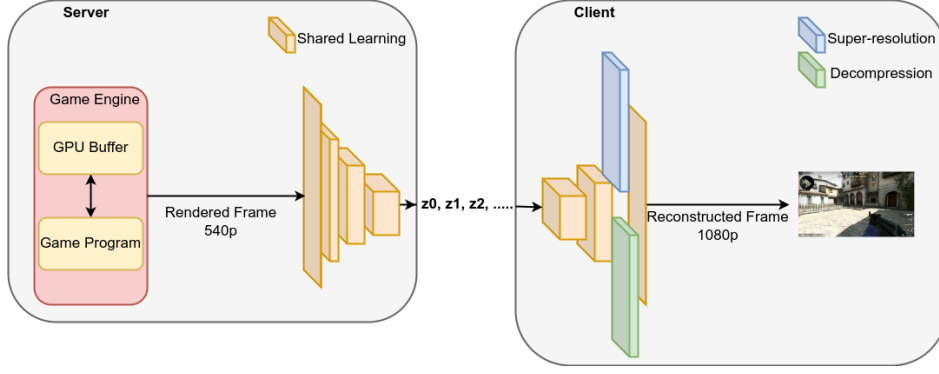


Figure 7.1: Proposed joint neural codec and super-resolution architecture. The server-side encoder receives low-resolution rendered frames and produces a compact latent representation $z_0, z_1, z_2, \dots$ that is transmitted to the client. The client-side decoder splits into a decompression branch (green) and a super-resolution branch (blue), whose outputs are fused to produce the reconstructed high-resolution frame. Shared early layers (yellow) enable feature reuse across both tasks.

for joint design. In the current pipeline (Chapters 3–4), a traditional codec (H.264) compresses rendered frames on the server, the compressed stream is transmitted and decoded on the client, and a separate SR model then upscales the decoded output. This two-stage design means the codec is unaware of the downstream upsampler, potentially discarding high-frequency features that the SR model relies on for reconstruction. A unified architecture can instead transmit compact latent features and reconstruct high-resolution output directly at the client, as illustrated in Figure 7.1.

The proposed architecture consists of three components:

- **Shared encoder (server-side):** a neural encoder receives the low-resolution rendered frame (e.g., 540p) and maps it to a compact latent representation z . The initial layers of this encoder can reuse the feature extraction primitives developed for GameSR (PixelUnshuffle, reparameterized convolutions), followed by pooling or strided convolutions for dimensionality reduction. The latent representation is quantized and entropy-coded for transmission, analogous to the bottleneck in learned image compression.
- **Task-specific decoding (client-side):** after the shared front-end, the architecture splits into two branches. A *decompression branch* reconstructs the low-resolution frame from the latent representation, while a *super-resolution branch* upscales the reconstructed features to the target resolution using sub-pixel convolution (PixelShuffle). Both branches share early decoder layers, with task-specific heads diverging only at the final stages. This hard parameter sharing reduces total client-side computation and memory compared to running independent codec and SR networks.
- **Latent-space transmission:** instead of transmitting pixel-domain video and then applying SR as a post-process, the server transmits the compact latent codes $z_0, z_1, z_2, \dots$ directly. This eliminates redundant encoding/decoding at the pixel level and allows the SR model to operate on richer feature representations rather than compression-degraded pixels, potentially improving both bandwidth efficiency and reconstruction quality.

The system is trained end-to-end with a combined loss:

$$L_{\text{combined}} = \alpha L_{\text{codec}} + \beta L_{\text{sr}} + \gamma L_{\text{rate}} \quad (7.1)$$

where L_{codec} measures reconstruction fidelity between the decoded and original low-resolution frames, L_{sr} measures upsampling quality between the super-resolved output and the high-resolution ground truth, and L_{rate} regularizes the entropy of the latent representation to control transmission bitrate. The weights α , β , γ can be adjusted to navigate the rate–distortion–complexity Pareto frontier depending on deployment constraints. In practice, the encoder must meet server-side latency requirements (operating within the rendering–encoding budget), while the combined decoder/SR must fit within the client frame-time budget established in this thesis ($\sim 4\text{--}11$ ms depending on the target application).

7.6.4 Extended Reality and Emerging Displays

GameSRVR addressed cloud VR gaming with stereoscopic content at $4\times$ scaling, but the broader XR ecosystem—including augmented reality (AR), mixed reality (MR), and next-generation headsets—introduces additional constraints that require coordinated rendering, transport, and enhancement:

- **Inter-view (disparity-based) stereo prediction:** GameSRVR exploits the *chrominance* redundancy between the left and right eye, transmitting full color for only one view. A complementary and largely orthogonal direction is to exploit the *geometric* redundancy between the two views. Because a stereo pair differs primarily by a depth-dependent horizontal disparity, most of the second view can be predicted from the first using inter-view disparity (motion) vectors, so that only a compact disparity field plus the residual occluded and disoccluded regions need to be transmitted. This is the principle behind multiview and volumetric video coding standards such as MPEG Immersive Video (MIV) and Multiview HEVC (MV-HEVC), which use inter-view prediction to avoid coding pixels that are redundant across views. Integrating learned disparity estimation and view-warping into the GameSRVR pipeline—so that the client reconstructs the second view by warping the first and refining only the residual—could reduce stereo bandwidth further still, on top of the chrominance savings already demonstrated. The key challenges are performing disparity estimation and warping within the strict VR latency budget, handling occlusions and view-dependent specular highlights that violate the warping assumption, and jointly training disparity prediction with the existing colorization-and-super-resolution objectives.
- **Foveated neural enhancement:** modern VR headsets (e.g., Meta Quest Pro, Apple Vision Pro, PlayStation VR2) include eye-tracking hardware that enables foveated rendering. A natural extension is *foveated neural enhancement*, where the SR model applies full-capacity processing only to the central gaze region while using lightweight interpolation or a smaller model variant for the periphery. This can reduce client-side SR compute while remaining perceptually transparent to the user. Key challenges include robustness to eye-tracker noise, latency between gaze measurement and model adaptation, and smooth quality transitions during fast saccades to avoid visible boundary artifacts.

- **Mixed-reality cloud gaming:** in MR scenarios, real-world camera feeds are composited with rendered game content. This requires the SR model to handle both synthetic and natural imagery within the same frame—a domain gap that current game-specific models are not trained for. A dual-domain architecture, or selective enhancement that applies game-trained SR only to rendered regions identified via segmentation masks, could address this challenge.
- **8K+ stereo streaming:** headset resolutions are increasing rapidly. At these resolutions, even moderate upscaling requires processing very large stereo frame sizes per frame. Hierarchical or tile-adaptive enhancement strategies—where compute is allocated non-uniformly based on content complexity and viewing region—are needed to scale beyond the resolutions demonstrated in this thesis.
- **VR-specific quality assessment:** standard metrics (PSNR, SSIM, VMAF) were designed for flat-screen viewing. VR introduces unique artifacts such as stereo inconsistency, temporal flicker at high refresh rates, and motion-induced discomfort. Developing VR-specific quality metrics that capture these phenomena would improve both model optimization and adaptive control for immersive applications.

7.6.5 Mobile Cloud Gaming

Mobile devices are the fastest-growing cloud gaming platform, yet they impose the strictest energy, thermal, and compute constraints. GameSR’s hardware scalability results (Section 3.5.9) demonstrated feasibility on mobile GPUs (Adreno 750), but sustained deployment requires deeper optimization:

- **NPU-optimized architectures:** modern mobile SoCs include dedicated neural processing units with substantial AI throughput (e.g., Apple Neural Engine and MediaTek Dimensity NPUs) [86, 87]. Co-designing model operators and quantization formats (INT8/INT4) specifically for these accelerators can achieve significantly lower latency and power consumption than GPU-based inference. This requires architecture search under NPU-specific operator constraints rather than adapting GPU-optimized designs.
- **Battery- and thermal-aware adaptation:** mobile gaming sessions can last hours, during which battery state and thermal conditions change substantially. Coupling the SR enhancement level with real-time device telemetry (battery percentage, SoC temperature, thermal throttling state) enables dynamic adaptation—for example, switching from a 64-filter model to a 24-filter model or disabling SR entirely when the device approaches thermal limits, preserving interactivity at the cost of gradual quality reduction.
- **5G/edge split inference:** 5G networks with multi-access edge computing (MEC) offer an alternative to purely client-side SR. A dynamic split-inference approach could run the computationally intensive feature extraction layers on the edge server (co-located with the game server) and transmit compact intermediate features to the client, where only lightweight reconstruction is performed. The split point would adapt based on network latency, available edge compute, and client device capability.

- **Graceful degradation:** rather than abrupt quality transitions when compute or power budgets are exhausted, a tiered model family (as demonstrated with GameSR’s filter-size variants) combined with smooth interpolation between model outputs can maintain perceptual continuity during adaptation events, avoiding jarring quality shifts that disrupt gameplay.

7.6.6 Perceptual Quality and User Experience

This thesis used established objective metrics (PSNR, SSIM, VMAF) supplemented by focused user studies with 8–15 participants. While these provided consistent and statistically meaningful results, a stronger next phase is to align optimization targets more directly with perceived playability and interactive quality:

- **Gaming-specific perceptual metrics:** standard image/video quality metrics do not capture gaming-specific artifacts such as HUD text illegibility, minimap distortion, crosshair blurring, texture shimmer on moving surfaces, and temporal flicker in particle effects. Developing specialized metrics—potentially through a gaming quality dataset with crowd-sourced annotations—would provide better optimization targets for SR models and more accurate reward signals for adaptive control systems like ARCADE.
- **Longitudinal QoE studies:** the user studies in this thesis evaluated short game-play clips (30–60 seconds). Longer studies spanning full gaming sessions (30–60 minutes) are needed to assess whether SR-induced artifacts accumulate perceptually over time, how neural upsampling affects player performance metrics (K/D ratio, accuracy, reaction time), and whether perceived quality adapts or degrades with sustained exposure.
- **Cross-device perceptual calibration:** the same VMAF score may correspond to different perceived quality on a 27-inch monitor, a 55-inch TV, and a VR headset due to differences in pixel density, viewing distance, and display characteristics. Establishing device-class-specific quality mappings would enable adaptive policies to make better decisions based on the client’s display type.
- **Accessibility and digital equity:** cloud gaming with neural enhancement has the potential to enable high-quality gaming experiences on low-bandwidth connections common in rural areas, developing regions, and mobile-only environments. Optimizing specifically for these constrained scenarios—prioritizing robustness at very low bitrates (2–5 Mbps) over peak quality at high bitrates—could broaden equitable access to interactive entertainment for underserved populations.

7.7 Concluding Remarks

Cloud gaming promises to democratize access to high-end gaming experiences by removing hardware barriers. However, this vision has been constrained by bandwidth requirements that exceed what many users can reliably access. This thesis demonstrated that neural super-resolution, when carefully designed for the unique characteristics of gaming content and the strict latency requirements of interactive applications, can

substantially reduce these bandwidth requirements while maintaining high perceptual quality.

GameSR showed that engine-independent upsampling is viable and can match commercial solutions. ARCADE demonstrated that joint optimization of the rendering–encoding–upsampling pipeline yields multiplicative benefits beyond optimizing individual components. GameSRVR extended these ideas to the demanding domain of stereoscopic VR, building on the iSR study to unify colorization and super-resolution into a single real-time model. GameLab provided the research infrastructure that made systematic evaluation possible.

Together, these contributions advance the state of the art in cloud gaming delivery and provide a foundation for future research on AI-driven multimedia systems. As display resolutions continue increasing, network infrastructure remains heterogeneous, and immersive applications proliferate, the techniques developed in this thesis will become increasingly relevant for delivering high-quality interactive experiences to users worldwide.

Bibliography

- [1] NVIDIA. System requirements for geforce now, 2025. Online overview.
- [2] Mark Claypool and David Finkel. The effects of latency on player performance in cloud-based games. In *2014 13th Annual Workshop on Network and Systems Support for Games*, pages 1–6. IEEE, 2014.
- [3] Michael Long and Carl Gutwin. Characterizing and Modeling the Effects of Local Latency on Game Performance and Experience. CHI PLAY '18, New York, NY, USA, 2018. Association for Computing Machinery.
- [4] Statista. Cloud gaming - worldwide, 2025. Market statistics.
- [5] Microsoft. Xbox cloud gaming, 2025. Cloud gaming service.
- [6] NVIDIA. Nvidia geforce now, 2025. Cloud gaming service.
- [7] Sony. Playstation plus premium, 2025. Cloud streaming service.
- [8] Amazon. Amazon luna, 2025. Cloud gaming service.
- [9] Netflix. Internet connection speed recommendations, 2020.
- [10] Netflix. How to control how much data Netflix uses, 2022.
- [11] Netflix. Bandwidth, data usage, and stream quality , 2022.
- [12] Igor's Lab. NVIDIA GeForce RTX 3080 Ti FE in test: Almost an RTX 3090 but with halved memory expansion for gamers. <https://www.igorslab.de/en/nvidia-geforce-rtx-3080-ti-fe-in-test-almost-an-rtx-3090-but-with-halved-memory-expansion-for-gamers/>, 2021.
- [13] Mezha. PC Power Consumption in Games: How Many Watts Does Your Computer Use? <https://mezha.media/en/articles/pc-power-consumption-in-games/>, 2024. Accessed: 2025-09-22.
- [14] Evan Mills and Nathaniel Mills. Taming the energy use of gaming computers. *Energy Efficiency*, 8:865–885, 2015.
- [15] Jason. Jerald. *The VR book : human-centered design for virtual reality*. Association for Computing Machinery ; M & C, Morgan & Claypool, 2016.
- [16] Mohamed Hegazy, Khaled Diab, Mehdi Saeedi, Boris Ivanovic, Ihab Amer, Yang Liu, Gabor Sines, and Mohamed Hefeeda. Content-aware video encoding for cloud gaming. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys '19)*, pages 60–73, 2019.

- [17] NVIDIA. Deep Learning Super Sampling (DLSS) SDK. <https://developer.nvidia.com/dlss>, 2025. Accessed: Sep. 20, 2025.
- [18] AMD. AMD Fidelity Super Resolution, 2022.
- [19] Intel Corporation. Intel® xess: Ai-enhanced gaming and graphics technology, 2022. Accessed: 2022-06-28.
- [20] NVIDIA. RTX Games, Engines, and Apps, 2025. Accessed: 2025-09-21.
- [21] AMD. AMD FidelityFX Super Resolution: Supported Games, 2025. Accessed: 2025-09-21.
- [22] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6. IEEE, 2012.
- [23] Jingyun Liang, Jiezhong Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer, 2021.
- [24] Wenbin Zou, Hongxia Gao, Liang Chen, Yunchen Zhang, Mingchao Jiang, Zhongxin Yu, and Ming Tan. Cross-View Hierarchy Network for Stereo Image Super-Resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1396–1405, Vancouver, 2023.
- [25] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. Gamin-ganywhere: An open cloud gaming system. In *Proceedings of the 4th ACM multimedia systems conference*, pages 36–47, 2013.
- [26] Microsoft. Xbox Cloud Gaming Beta, 2022.
- [27] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [28] L. Alzubaidi, J. Zhang, A. J. Humaidi, and et al. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(53), 2021.
- [29] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.
- [30] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

- [32] Wenzhe Shi, Jose Caballero, Ferenc Huszár, et al. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *CoRR*, abs/1609.05158, 2016.
- [33] Xiangyu Zhang Xiaohan Ding and others. RepVGG: Making VGG-style ConvNets Great Again. *CoRR*, abs/2101.03697, 2021.
- [34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [35] Xingjian Shi, Zhihan Gao, Lihan Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems* 28, pages 802–810, 2015.
- [36] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010.
- [37] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015.
- [38] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *CoRR*, abs/1511.04587, 2015.
- [39] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. *CoRR*, abs/1608.00367, 2016.
- [40] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017.
- [41] Christian Ledig, Lucas Theis, Huszar, et al. Photo-realistic single image super-resolution using a generative adversarial network, 2016.
- [42] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. *CoRR*, abs/1807.02758, 2018.
- [43] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. *CoRR*, abs/1704.03915, 2017.
- [44] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: enhanced super-resolution generative adversarial networks. *CoRR*, abs/1809.00219, 2018.
- [45] Prafulla Dhariwal and Alexander Nichol. Diffusion Models Beat GANs on Image Synthesis. In M Ranzato, A Beygelzimer, Y Dauphin, P S Liang, and J Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 12 2021.

- [46] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4713–4726, 2022.
- [47] Zongsheng Yue, Jianyi Wang, and Chen Change Loy. Resshift: Efficient diffusion model for image super-resolution by residual shifting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [48] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2024–2032, 2019.
- [49] Xiaotong Luo, Yuan Xie, Yulun Zhang, Yanyun Qu, Cuihua Li, and Yun Fu. Latticenet: Towards lightweight image super-resolution with lattice block. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 272–289, Cham, 2020. Springer International Publishing.
- [50] Eduard Zamfir, Marcos V. Conde, and Radu Timofte. Towards real-time 4k image super-resolution. In *CVPR Workshops*, pages 1522–1532, 2023.
- [51] Jose Caballero, Christian Ledig, et al. Real-time video super-resolution with spatio-temporal networks and motion compensation. *CoRR*, abs/1611.05250, 2016.
- [52] Longguang Wang, Yulan Guo, Li Liu, Zaiping Lin, Xinpu Deng, and Wei An. Deep video super-resolution using hr optical flow estimation. *IEEE Transactions on Image Processing*, 29:4323–4336, 2020.
- [53] Xinyi Ying, Longguang Wang, et al. Deformable 3d convolution for video super-resolution. *CoRR*, abs/2004.02803, 2020.
- [54] Xintao Wang, Kelvin C.K. Chan, Ke Yu, Chao Dong, and Chen Change Loy. EDVR: Video Restoration With Enhanced Deformable Convolutional Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR’19) Workshops*, 4 2019.
- [55] Shuwei Shi, Jinjin Gu, Liangbin Xie, Xintao Wang, Yujiu Yang, and Chao Dong. Rethinking alignment in video super-resolution transformers, 2022.
- [56] Shuwei Shi, Jinjin Gu, Liangbin Xie, Xintao Wang, Yujiu Yang, and Chao Dong. Rethinking Alignment in Video Super-Resolution Transformers, 2022.
- [57] Kelvin C. K. Chan, Xintao Wang, Ke Xu, Kelvin Chan, Tao Yu, Chao Dong, and Chen Change Loy. Basicvsr: The search for essential components in video super-resolution and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [58] NVIDIA. Deep Learning Super Sampling (DLSS) Technology - Nvidia, 2022.
- [59] Vignesh V Menon et al. Vca: Video complexity analyzer. In *MMSys*, pages 259–264, 2022.

- [60] Hadi Amirpour et al. Evca: Enhanced video complexity analyzer. In *MMSys*, pages 285–291, 2024.
- [61] NVIDIA. Internet Requirements, 2022.
- [62] NVIDIA. GeForce, 2022.
- [63] Playstation. Playstation Plus Premium, 2022.
- [64] Amazon. Luna, 2022.
- [65] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [66] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [67] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.
- [68] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.
- [69] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.
- [70] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. *The Netflix Tech Blog*, 6(2), 2016.
- [71] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. Technical report.
- [72] Greg Chance, Abanoub Ghobrial, Kevin McAreavey, Séverin Lemaignan, Tony Pipe, and Kerstin Eder. On determinism of game engines used for simulation-based autonomous vehicle verification. *IEEE Trans. Intell. Transp. Syst.*, 23(11):20538–20552, 2022.
- [73] Markus Utke, Saman Zadtootaghaj, Steven Schmidt, Sebastian Bosse, and Sebastian Möller. NDNetGaming - development of a no-reference deep CNN for gaming video quality prediction. *Multimedia Tools and Applications*, 81:3181–3203, 2022.
- [74] Dingquan Li, Tingting Jiang, and Ming Jiang. Quality Assessment of In-the-Wild Videos. In *Proceedings of the 27th ACM International Conference on Multimedia (MM ’19)*, MM ’19, pages 2351–2359, New York, NY, USA, 2019. Association for Computing Machinery.
- [75] George Suciu, Stefan Stefanescu, Cristian Beceanu, and Marian Ceaparu. WebRTC role in real-time communication and video conferencing. In *2020 Global Internet of Things Summit (GloTS)*, pages 1–6, 2020.

- [76] NVIDIA. RTX Games, Engines, and Apps. <https://www.nvidia.com/en-us/geforce/news/nvidia-rtx-games-engines-apps/>, 2025. Accessed: 2025-09-21.
- [77] AMD. AMD FidelityFX Super Resolution: Supported Games. <https://www.amd.com/en/products/graphics/technologies/fidelityfx/supported-games.html>, 2025. Accessed: 2025-09-21.
- [78] SQ Magazine. Steam statistics, 2025. Accessed: 2025-09-21.
- [79] Dario Fuoli, Shuhang Gu, and Radu Timofte. Efficient Video Super-Resolution through Recurrent Latent Space Propagation. In *IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3476–3485, 10 2019.
- [80] Benjamin Naoto Chiche et al. Stable long-term recurrent video super-resolution. In *Proceedings of the IEEE/CVF CVPR*, 2022.
- [81] Kelvin C. K. Chan, Xintao Wang, Kelvin Chan, Chao Dong, and Chen Change Loy. Basicvsr++: Improving video super-resolution with enhanced propagation and alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [82] Ling Zheng, Jinchen Zhu, Jinpeng Shi, and Shizhuang Weng. Efficient mixed transformer for single image super-resolution. *arXiv preprint arXiv:2305.11403*, 2023.
- [83] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 645–661, 2018.
- [84] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. Streaming 360-Degree Videos Using Super-Resolution. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1385–1394, Beijing, China, 2020. IEEE.
- [85] Guanjun Ye, Ge Li, Xiangtao Li, Zhiyong Gao, and Xiaoqi Sun. Neural super-resolution for omnidirectional video streaming. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 8123–8131, 2023.
- [86] Apple Inc. iPhone 16 Pro – Technical Specifications, 2024. Accessed: 2025-04-24.
- [87] MediaTek Inc. MediaTek Dimensity 9400 – Flagship 5G Agentic AI Platform, 2024. Accessed: 2025-04-24.
- [88] Sony Interactive Entertainment. PlayStation 5 Specifications, 2024. Accessed: 2025-04-24.
- [89] Microsoft. Xbox Series X – Console Specs, 2024. Accessed: 2025-04-24.
- [90] Nabajeet Barman, Saman Zadtootaghaj, Maria G. Martini, Sebastian Möller, and Sanghoon Lee. A comparative quality assessment study for gaming and non-gaming videos. In *2018 QoMEX*, pages 1–6. IEEE, 2018.

- [91] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
- [92] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.
- [93] Avery Lee. VirtualDub: Video capture/processing utility for Windows, 2024. Accessed: 2025-04-25.
- [94] Jonathan T. Barron. A more general robust loss function. *CoRR*, abs/1701.03077, 2017.
- [95] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
- [96] Netflix Technology Blog. Vmaf: The journey continues. <https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12>, 2018. Accessed: 2025-09-22.
- [97] Sara Ghazanfari, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, and Alexandre Araujo. R-LPIPS: An adversarially robust perceptual similarity metric. *arXiv preprint arXiv:2307.15157*, 7 2023.
- [98] Greg Chance, Abanoub Ghobrial, Kevin McAreavey, Séverin Lemaignan, Tony Pipe, and Kerstin Eder. On Determinism of Game Engines Used for Simulation-Based Autonomous Vehicle Verification. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):20538–20552, 2022.
- [99] Markus Utke, Saman Zadtootaghaj, Steven Schmidt, Sebastian Bosse, and Sebastian Möller. NDNetGaming - development of a no-reference deep CNN for gaming video quality prediction. *Multimedia Tools and Applications*, 81:3181–3203, 1 2022.
- [100] Eduard Zamfir, Marcos V. Conde, and Radu Timofte. RT4KSR: Real-Time 4K Image Super-Resolution Baseline (NTIRE Real-Time 4K SR Challenge). In *Proceedings of the IEEE/CVF CVPRW*, 2023. NTIRE Real-Time 4K SR Challenge baseline.
- [101] International Telecommunication Union. ITU-T Rec. P.910 (04/2008) Subjective video quality assessment methods for multimedia applications. 11 2021.
- [102] International Telecommunication Union. Recommendation ITU-R BT.500-15: Methodologies for the subjective assessment of the quality of television images. Technical report, ITU-R, 2023.
- [103] GameLab: An Open-Source WebRTC Cloud Gaming Testbed. <https://github.com/haseebfazal/ai-testbed>. Accessed: 2026-01-11.
- [104] Tianyi Liu et al. Improving resource and energy efficiency for cloud 3d through excessive rendering reduction. In *EuroSys*, pages 317–332, 2024.
- [105] Yao Liu, Sujit Dey, and Yao Lu. Enhancing video encoding for cloud gaming using rendering information. *IEEE TCSVT*, 25(12):1960–1972, 2015.

- [106] Mehdi Semsarzadeh, Abdulsalam Yassine, and Shervin Shirmohammadi. Video encoding acceleration in cloud gaming. *IEEE TCSVT*, 25(12):1975–1987, 2015.
- [107] Ryan Shea, Jiangchuan Liu, Edith C.-H. Ngai, and Yong Cui. Cloud gaming: Architecture and performance. *IEEE Network*, 27(4):16–21, 2013.
- [108] Omar Mossad, Khaled Diab, Ihab Amer, and Mohamed Hefeeda. Deepgame: Efficient video encoding for cloud gaming. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21)*, pages 1387–1395, 2021.
- [109] Vignesh V Menon, Jingwen Zhu, Prajit T Rajendran, Hadi Amirpour, Patrick Le Callet, and Christian Timmerer. Just Noticeable Difference-Aware Per-Scene Bitrate-Laddering for Adaptive Video Streaming. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1673–1678, 2023.
- [110] Information technology — dynamic adaptive streaming over http (dash) — part 1: Media presentation description and segment formats, 2022.
- [111] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17*, pages 197–210, New York, NY, USA, 2017. Association for Computing Machinery.
- [112] Tianchi Huang, Ruibin Zhou, Chenguang Qiu, Chuang Du, Haibin Wu, Xiaofei Xing, Yao Liu, Feng Zhuang, De Li, Weinan Zhang, et al. Comyco: Quality-aware adaptive video streaming via imitation learning. *arXiv preprint arXiv:1905.01623*, 2019.
- [113] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18*, pages 44–58, New York, NY, USA, 2018. Association for Computing Machinery.
- [114] Twitch. Low latency video. Twitch Help, 2024. Accessed: 2026-03-01.
- [115] Zoom. System requirements: Windows, macos, and linux. Zoom Support, 2026. Accessed: 2026-03-01.
- [116] Sangwook Na et al. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the ACM SIGCOMM 2020 Workshop on Network-Application Integration (NetAI)*, 2020.
- [117] Timothy P. Lillicrap et al. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.
- [118] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.
- [119] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking Batch Deep Reinforcement Learning Algorithms. *CoRR*, abs/1910.01708, 2019.

- [120] Haseeb Ur Rehman, Shervin Shirmohammadi, Ihab Amer, and Mohamed Hefeeda. GameSR: Real-time super-resolution for interactive gaming. TechRxiv preprint, 2026.
- [121] BoboTiG. python-mss: Multiple screenshots module, 2024. GitHub project.
- [122] ra1nty. Dxcam: Screen capturing library for windows, 2024. GitHub project.
- [123] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th ACM Multimedia Systems Conference (MMSys '16)*, 2016.
- [124] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network*, 2021.
- [125] Netflix. VMAF: The Journey Continues, 2018.
- [126] Alaa Eddin Alchalabi and Shervin Shirmohammadi. CGCSDD: Cloud Gaming Client-Server Delay Dataset, 2021.
- [127] Yanyuan Qin, Shuai Hao, Krishna R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. Quality-aware strategies for optimizing abr video streaming qoe and reducing data usage. *MMSys '19*, page 189–200, New York, NY, USA, 2019. Association for Computing Machinery.
- [128] Elecard. Interpretation of metrics: Psnr, ssim, vmaf. https://www.elecard.com/page/article_interpretation_of_metrics, 2023. Accessed: 2025-04-23.
- [129] Robert C. Streijl, Stefan Winkler, and David S. Hands. Mean opinion score (mos) revisited. *Multimedia Systems*, 22(2):213–227, 2016.
- [130] Jiannan Ye, Xiaoxu Meng, Daiyun Guo, Cheng Shang, Haotian Mao, and Xubo Yang. Neural foveated super-resolution for real-time VR rendering. *Computer Animation and Virtual Worlds*, 35(4), 7 2024.
- [131] En Sing Wong, Nur Haliza Abdul Wahab, Faisal Saeed, and Nouf Alharbi. 360-Degree Video Bandwidth Reduction: Technique and Approaches Comprehensive Review. *Applied Sciences (Switzerland)*, 12(15), 8 2022.
- [132] Grand View Research. Virtual Reality in Gaming Market Size, Share \& Trends Analysis Report By Component, By Region, And Segment Forecasts, 2024 - 2030, 2024.
- [133] Ritik Vatsal, Shrivatsa Mishra, Rushil Thareja, Mrinmoy Chakrabarty, Ojaswa Sharma, and Jainendra Shukla. An Analysis of Physiological and Psychological Responses in Virtual Reality and Flat Screen Gaming. *IEEE Transactions on Affective Computing*, 15(3):1696–1710, 7 2024.

- [134] Sihao Zhao, Hatem Abou-Zeid, Ramy Atawia, Yoga Suhas Kuruba Manjunath, Akram Bin Sediq, and Xiao Ping Zhang. Virtual Reality Gaming on the Cloud: A Reality Check. In *2021 IEEE Global Communications Conference, GLOBECOM 2021*, pages 1–6, Madrid, Spain, 9 2021.
- [135] Miguel Casasnovas, Costas Michaelides, Marc Carrascosa-Zamacois, and Boris Bellalta. Experimental evaluation of interactive Edge/Cloud Virtual Reality gaming over Wi-Fi using unity render streaming. *Computer Networks*, 243, 2024.
- [136] Costas Michaelides, Miguel Casasnovas, Daniele Marchitelli, and Boris Bellalta. Is Wi-Fi 6 Ready for Virtual Reality Mayhem? A Case Study Using One AP and Three HMDs. In *Proceedings - 2023 IEEE Future Networks World Forum: Future Networks: Imagining the Network of the Future, FNWF 2023*, pages 1–6, Baltimore, Maryland, USA., 2023. Institute of Electrical and Electronics Engineers Inc.
- [137] Kar Long Chan, Kohei Ichikawa, Yasuhiro Watashiba, and Hajimu Iida. Cloud-Based VR Gaming: Our Vision on Improving the Accessibility of VR Gaming. In *2017 International Symposium on Ubiquitous Virtual Reality (ISUVR)*, pages 24–25, Nara, Japan, 7 2017.
- [138] Qiqi Hou, Farzad Farhadzadeh, Amir Said, Guillaume Sautiere, and Hoang Le. Low-Latency Neural Stereo Streaming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7974–7984, Seattle, Washington, USA, 6 2024.
- [139] Ahmad Alhilal, Ze Wu, Yuk Hang Tsui, and Pan Hui. FovOptix: Human Vision-Compatible Video Encoding and Adaptive Streaming in VR Cloud Gaming. In *Proceedings of ACM Multimedia Systems Conference (MMSys’24)*, pages 67–77, Bari, Italy, 4 2024.
- [140] Farhaan Ebadulla, Chiraag Mudlapur, and Gaurav BV. GazeProphet: Software-Only Gaze Prediction for VR Foveated Rendering. 8 2025.
- [141] David Petrescu, Paul A. Warren, Zahra Montazeri, Gabriel Strain, and Steve Petifer. The Effects of Visual Behavior and Ego-Movement on Foveated Rendering Performance in Virtual Reality. *SN Computer Science*, 6(4), 4 2025.
- [142] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *Proceedings of ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM’20)*, pages 107–125, 7 2020.
- [143] Wenbo Xu et al. Cvgsr: Cross-view guided stereo super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3193–3203, 2023.
- [144] Qing Jiang et al. Omnidirectional video super-resolution using deep learning. *IEEE Transactions on Multimedia*, 24:4325–4338, 2022.

- [145] Deniz Ugur, Ihah Amer, and Mohamed Hefeeda. Decoupling Video Upscaling from Rendering for Cloud Gaming. In *Proceedings of ACM Multimedia Systems Conference (MMSys'25)*, volume 1, Stellenbosch, South Africa, 3 2025.
- [146] Kaiwen Zheng, Jie Huang, Man Zhou, and Feng Zhao. Asymmetric Color Transfer with Consistent Modality Learning. Technical report, Vancouver, 6 2023.
- [147] GameLab: An ai-enabled cloud gaming testbed (source code). GitHub repository, 2026.
- [148] Jooyeol Yun, Sanghyeon Lee, Minho Park, and Jaegul Choo. iColoriT: Towards Propagating Local Hint to the Right Region in Interactive Colorization by Leveraging Vision Transformer. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1787–1796, Waikoloa, Hawaii, 7 2023.
- [149] Patricia Vitoria, Lara Raad, and Coloma Ballester. ChromaGAN: Adversarial Picture Colorization with Semantic Class Distribution. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Colorado, 3 2020.
- [150] Yifan Li, Yuhang Bai, Shuai Yang, and Jiaying Liu. COCO-LC: Colorfulness Controllable Language-based Colorization. In *MM 2024 - Proceedings of the 32nd ACM International Conference on Multimedia*, pages 10939–10947, New York, NY, USA, 10 2024. Association for Computing Machinery, Inc.
- [151] Chengyi Zou, Shuai Wan, Marc Gorriz Blanch, Luka Murn, Marta Mrak, Juil Sock, Fei Yang, and Luis Herranz. Lightweight Deep Exemplar Colorization via Semantic Attention-Guided Laplacian Pyramid. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–12, 12 2024.
- [152] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced Deep Residual Networks for Single Image Super-Resolution. *CoRR*, abs/1707.02921, 2017.
- [153] Jingyun Liang, Jiezhong Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1833–1844, 2021.
- [154] Gaurav Sharma, Wencheng Wu, and Edul N Dalal. The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations. *Col Res Appl*, 30:21–30, 2005.
- [155] LizardByte and contributors. Sunshine: Self-hosted game stream host for moonlight, 2025. Accessed: 2025-12-27.
- [156] Moonlight Game Streaming Project. Moonlight: Open-source game streaming client, 2025. Accessed: 2025-12-27.
- [157] Unity Technologies. Unity render streaming (com.unity.renderstreaming) documentation, 2025. Accessed: 2025-12-27.

- [158] Lixiang Ao and Landon P. Cox. LevelUp: A thin-cloud approach to game livestreaming, 2020. Accessed: 2025-12-27.
- [159] Saman Zadtootaghaj, Steven Schmidt, Saeed Shafiee Sabet, Sebastian Möller, and Carsten Griwodz. Quality estimation models for gaming video streaming services using perceptual video quality dimensions. In *Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20)*, pages 213–224, 2020.
- [160] Yu-Chih Chen, Avinab Saha, Chase Davis, Bo Qiu, Xiaoming Wang, Rahul Gowda, Ioannis Katsavounidis, and Alan C. Bovik. GAMIVAL: Video quality prediction on mobile cloud gaming content. *IEEE Signal Processing Letters*, 30:324–328, 2023.
- [161] Carlos Baena, O. S. Peñaherrera-Pulla, Raquel Barco, and Sergio Fortes. Measuring and estimating key quality indicators in cloud gaming services. *Computer Networks*, 231:109808, 2023.
- [162] Shakarim Soltanayev, Odysseas Zisimopoulos, Mohammad Ashraful Anam, Man Cheung Kung, Angeliki Katsenou, and Yiannis Andreopoulos. Content adaptive encoding for interactive game streaming, 2025.
- [163] Mahdi Hemmati, Abbas Javadtalab, Ali Asghar Nazari Shirehjini, Shervin Shirmohammadi, and Tarik Arici. Game as video: Bit rate reduction through adaptive object encoding. In *Proceedings of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '13)*, pages 7–12, 2013.
- [164] python-mss contributors. python-mss: Ultra fast cross-platform multiple screenshots module in pure python, 2025. Accessed: 2025-12-27.
- [165] Hadi Amirpour, Mohammad Ghasempour, Lingfeng Qu, Wassim Hamidouche, and Christian Timmerer. Evca: Enhanced video complexity analyzer. In *Proceedings of the 15th ACM Multimedia Systems Conference (MMSys '24)*, pages 285–291, 2024.
- [166] OutofAi. cudacanvas: Pytorch tensor image display in cuda (eliminating cpu transfer), 2025. Accessed: 2025-12-27.