Coordinating Split Rendering and Video Transport for Immersive Applications

Abstract

Immersive interactive applications are characterized by real-time user interaction and highly immersive content delivery. Their quality of experience (QoE) depends on visual quality, playback continuity, and interaction timeliness. These QoE dimensions are jointly constrained by network transmission capacity and computing resources. However, existing systems typically employ separate control loops to adapt video transport and rendering, resulting in limited cross-layer coordination and resource contention. This paper presents the Coordinating Split Rendering and Video Transport framework (CSV) for immersive interactive applications. Based on scene dynamics, network conditions, and available system resources, CSV jointly adapts object-level rendering placement, video bitrate, and forward error correction (FEC) protection under a unified QoE objective. CSV learns the joint policy from empirical system measurements and coordinates the three control dimensions across different runtime timescales. Experimental results show that CSV reduces the average deadline-miss rate by 32.0% compared with the mean across the three baselines, while improving playback continuity and maintaining comparable visual quality.

1 Introduction

Advances in high-speed networks and edge computing are enabling interactive applications to distribute computation and media processing across resource-constrained end devices and powerful remote servers. Emerging 6G [1–4] systems are expected to further accelerate this trend. Among these applications, immersive services [5–10] are particularly demanding because they combine rich 3D content with real-time user interaction. Virtual reality (VR) [11–13] constructs fully virtual three-dimensional environments to provide immersive viewing and interactive experiences. Augmented reality (AR) [14–17] augments users' perception of the physical world by superimposing virtual information onto it. Mixed reality (MR) [18–20] further spatially registers and anchors

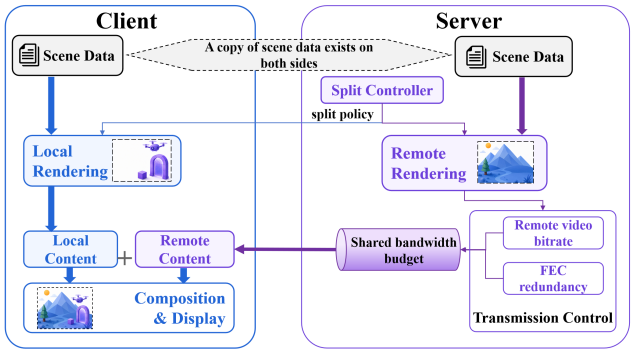


Figure 1: Split rendering and shared-bandwidth competition

virtual objects within the physical environment, enabling persistent relationships and interactions between physical and virtual entities. Collectively referred to as extended reality (XR) [21–23], these technologies have been widely applied to gaming, remote collaboration, industrial training, and digital-twin systems.

Unlike conventional video streaming, immersive XR applications must simultaneously sustain visual fidelity, smooth presentation, and timely interaction under fluctuating network conditions and varying computational resources. From a user-experience perspective, we characterize immersive QoE along three directly visible dimensions: *visual quality*, which reflects the fidelity of the rendered and composed view; *playback continuity*, which captures whether content is delivered and presented smoothly; and *interaction timeliness*, which indicates whether interaction-relevant updates arrive within the application's latency requirements. These dimensions can be measured using representative metrics, including encoding or perceptual quality, delivered frame rate and freezing behavior, and deadline-miss rate, respectively.

Rendering complex 3D scenes with high fidelity at the target frame rate requires substantial computational resources, which are often limited on lightweight XR devices. Fully local rendering is constrained by device computation and

memory, whereas fully remote rendering can exploit server-side GPUs but increases dependence on network bandwidth and latency. Split rendering [24–27] addresses this trade-off by partitioning scene objects between the local device and a remote server according to their complexity, device workload, network conditions, and interaction sensitivity. As illustrated in Figure 1, both endpoints maintain a copy of the scene state. Objects assigned locally are rendered directly on the client, whereas remotely assigned objects are rendered at the server and delivered as a video stream; the client composites the two outputs into the final view. By retaining latency-sensitive or lightweight objects locally and offloading compute-intensive or visually complex objects to the server, split rendering balances device load, interaction responsiveness, and visual quality.

The remote-rendered portion, however, is delivered as a latency-sensitive video stream and is vulnerable to packet loss. NACK-based retransmissions can recover some lost packets, but their effectiveness is bounded by round-trip time and strict presentation deadlines. FEC [28–31] can recover packet losses without waiting for retransmissions, but its redundant packets consume additional transmission capacity on the shared bottleneck link. Under a shared bandwidth constraint, this overhead couples FEC with rendering placement: offloading more objects increases the complexity and bitrate demand of the remote stream, while stronger FEC reduces the capacity available to encoded media. Independently selecting rendering placement and FEC can therefore create congestion, reduce media quality, and increase frame deadline misses. Prior work has improved end-to-end QoE in interactive and immersive applications through latency reduction [32–34], adaptive bitrate control [35–39], FEC-based reliability adaptation [40, 41], and joint media–transport control [42–49]. However, to the best of our knowledge, none jointly optimizes split-rendering placement and FEC-based protection for immersive applications. As a result, existing approaches cannot explicitly coordinate rendering demand, media bitrate, and protection overhead under a shared bandwidth budget.

To address these challenges, this paper presents CSV (Coordinating Split Rendering and Video Transport), a real-time cross-layer coordination framework for immersive interactive applications. CSV jointly adapts per-object rendering placement, video bitrate, and FEC redundancy based on scene dynamics, network conditions, and available system resources. It explicitly captures the shared-resource coupling between rendering and reliable video delivery, and employs a multi-timescale closed-loop mechanism to make responsive transport adjustments without causing unnecessary rendering oscillations. The main contributions are summarized as follows:

- **Cross-layer resource coupling formulation.** We formulate split rendering, media encoding, and reliable transport as a joint decision problem subject to shared bandwidth, computation, and latency constraints. The formulation captures how rendering placement affects encoding and delivery demands, as well as the resource contention between media traffic and reliability-protection overhead.
- **Multi-timescale closed-loop coordination.** We design an action-masked, multi-timescale mechanism that updates different decisions at their respective control intervals and aligns runtime feedback with the joint configuration actually in effect, thereby balancing transport responsiveness with rendering stability.
- **System implementation and evaluation.** We implement a CSV prototype atop Unity and WebRTC and evaluate it across diverse network conditions and user-motion patterns. CSV maintains competitive visual quality and playback continuity under steady network conditions while providing pronounced gains under burst loss. It reduces the average deadline-miss rate by 32.0% relative to the mean across the three baselines while improving frame delivery and reducing freezing under burst loss.

Overall, CSV demonstrates that jointly coordinating rendering placement and video transport can effectively improve end-to-end QoE under shared resource constraints, particularly in bandwidth-constrained and dynamically impaired network conditions.

2 Background and motivation

Immersive interactive applications can employ local, remote, or split rendering architectures [24, 25, 27, 50]. In local rendering, the end device performs the rendering workload, providing low interaction latency but being limited by local compute and memory resources. In remote rendering, rendering is offloaded to a powerful server and the rendered output is streamed back to the client, enabling higher-fidelity rendering at the cost of additional network latency and bandwidth dependence. Split rendering combines the two by distributing rendering tasks between the client and server. Such architectures are particularly relevant to interactive XR scenes and complex 3D applications, including medical visualization, architectural models, manufacturing applications, and volumetric content. In remote and split-rendered immersive applications, user interaction, rendering, and media delivery form a tightly coupled real-time feedback loop. Unlike conventional video streaming that primarily focuses on one-way content delivery, these applications continuously capture user inputs, such as head poses, gestures, or viewpoint changes, update the scene, and generate corresponding rendering outputs. Remotely rendered content is then encoded, optionally protected against packet losses through mechanisms such as FEC, and delivered over the network under congestion control.

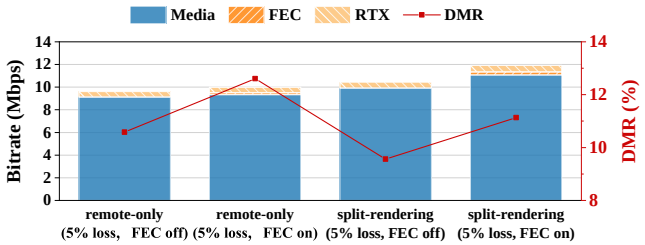


Figure 2: Rendering–FEC tradeoffs under 5% packet loss

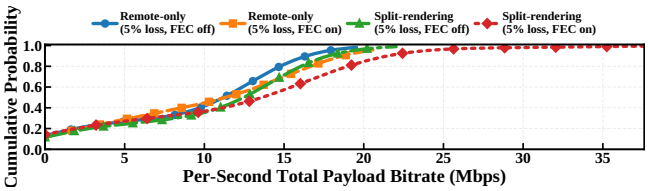


Figure 3: CDF of per-second total payload bitrate under 5% packet loss

At the receiver, the content is recovered, decoded, and composed with locally rendered elements when necessary before being displayed to the user. RenderFusion, for example, dynamically assigns scene objects to local or remote rendering according to device and network conditions. This end-to-end pipeline determines whether interactive content can be delivered within its latency budget. Consequently, QoE depends on the joint outcome of visual quality, playback continuity, and interaction timeliness, all of which are jointly affected by rendering and transport behavior rather than by any individual component alone.

However, these components share limited network and computation resources during runtime, making their control decisions inherently coupled. Increasing the amount of remote rendering can reduce the computation burden on end devices, but it also converts more rendering outputs into media traffic that must traverse the shared network. Similarly, FEC can improve resilience to packet losses, but its redundant packets consume transmission capacity that would otherwise be available to source media and retransmissions. More importantly, such contention is not fully captured by long-term average bitrate. Even when average traffic remains below the nominal link capacity, short-term traffic bursts from media encoding, FEC, and retransmissions can transiently exceed the available bandwidth, causing queue buildup, packet loss, and delayed frame delivery. Therefore, independently optimizing rendering placement, media adaptation, and loss protection can lead to conflicting resource demands, motivating their joint coordination under a unified objective.

To quantify this coupling, we compare remote-only rendering with a fixed split configuration that renders 18 lightweight objects locally and 12 complex objects remotely. WebRTC is configured with a 30 Mbps `maxBitrate`, a 15 Mbps bottleneck, a 20 ms RTT, 3840×1080 video at 30 FPS, and 5% packet loss. Interaction timeliness is measured by the deadline-miss rate (DMR), defined as the fraction of eligible remotely rendered frames that are either lost during network delivery or delivered after the 100-ms deadline. We use a 100-ms deadline following prior interactive streaming and cloud gaming studies [34, 43, 51]. As shown in Figure 2, enabling FEC increases the average total payload bitrate, including media, RTX, and FEC traffic, from 9.61 to 9.99 Mbps for remote-only rendering and from 10.44 to 11.92 Mbps for split rendering. Meanwhile, DMR increases from 10.58% to 12.61% and from 9.56% to 11.13%, respectively. These results show that additional loss protection does not necessarily improve deadline compliance when FEC traffic competes with media and RTX traffic over the shared bottleneck.

While Figure 2 reports the run-level average payload bitrate, Figure 3 further captures its short-term variation by plotting the empirical CDF of the per-second total payload bitrate derived from consecutive sender-side telemetry samples. Although the run-level average remains below the 15 Mbps bottleneck, the per-second bitrate frequently exceeds this capacity, especially when FEC is enabled under split rendering. Such short-term traffic bursts can cause queue buildup and delayed frame delivery, helping explain the increased DMR.

3 CSV Overview

Rendering placement, video bitrate, and FEC protection are tightly coupled through shared network and computation resources, making independent adaptation potentially inefficient. To coordinate these decisions under a unified QoE objective, we design **CSV** (*Coordinating Split Rendering and Video Transport*), a cross-layer runtime coordination framework for immersive interactive applications.

As shown in Figure 4, CSV consists of four main components: the Runtime Context, the Unified CSV Controller, the Runtime Execution Plane, and Runtime Measurements & QoE Feedback. The Runtime Context collects network conditions, media and playback statistics, system-resource status, object-level rendering features, and the execution state of previously applied controls, and organizes them into a unified cross-layer state. Based on this state, the Unified CSV Controller uses a learned Joint Actor, i.e., a single policy network that first encodes the shared system-level and object-level information and then jointly produces bitrate, FEC, and rendering-placement decisions through dedicated output branches. The resulting coordinated action proposal is

$$\mathbf{a}_t^{\text{prop}} = [R_t, \rho_t, \mathbf{m}_t] \tag{1}$$

where R_t denotes the bitrate-control proposal, ρ_t denotes the FEC redundancy decision, and $\mathbf{m}_t \in \{0,1\}^N$ denotes the object-level rendering-placement vector. A larger ρ_t provides stronger packet-loss protection at the cost of additional transmission overhead, while $\rho_t = 0$ disables FEC protection. For

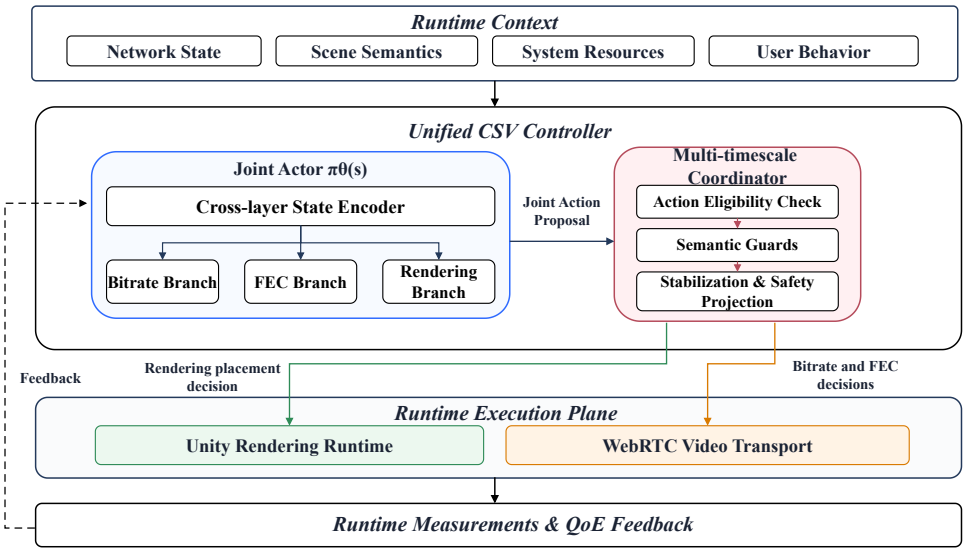


Figure 4: System overview of CSV

object i, $m_{i,t} = 0$ denotes local rendering and $m_{i,t} = 1$ denotes remote rendering. These three decisions are produced by dedicated bitrate, FEC, and rendering-placement branches of the same joint Actor rather than by independent controllers.

CSV learns the joint Actor using **Support-Constrained Critic-Guided Policy Learning (SCPL)** from controlled real-system measurements. SCPL first learns a component-wise QoE response model that characterizes how joint bitrate, FEC, and rendering configurations affect application-level outcomes. It then evaluates structured candidate actions within a support-constrained space grounded in empirical measurements and runtime feasibility constraints to construct teacher decisions, and finally distills these decisions into the lightweight joint Actor used at runtime. This design limits unsupported extrapolation and avoids unrestricted online exploration while capturing the resource coupling among the three control dimensions.

At runtime, the trained Actor maps the current cross-layer state to a joint action proposal, which is further processed by a multi-timescale runtime coordinator according to the dynamics and execution constraints of different control dimensions. The resulting rendering decisions are applied through the Unity rendering runtime, while bitrate and FEC decisions are mapped to the corresponding WebRTC control interfaces, with native bandwidth estimation, GCC, pacing, and retransmission mechanisms preserved. CSV continuously collects feedback associated with the effective configuration and incorporates it into subsequent states, forming a runtime loop of measurement, inference, and execution. These aligned measurements are also logged for subsequent offline policy refinement.

4 Design of SCPL-Based Joint Control

This section explains how CSV maps runtime system conditions to coordinated bitrate, FEC, and rendering-placement decisions. We first define the state representation and QoE objective used by the controller. We then describe SCPL, which learns a joint policy from empirical system measurements and distills it into a lightweight Actor for runtime inference. Finally, we present the multi-timescale coordination mechanism that converts Actor proposals into executable actions while accounting for the different dynamics and constraints of the three control dimensions.

4.1 Cross-Layer State Representation

To generate coordinated bitrate, FEC, and rendering-placement decisions, CSV first organizes the runtime context into a unified cross-layer state. This state captures shared system conditions, including network, media, playback, and rendering load, together with object-level characteristics required for fine-grained rendering placement.

System-level and object-level observations are separated because the control dimensions operate at different granularities: bitrate and FEC are system-level decisions, whereas rendering placement is determined for individual objects. Flattening these heterogeneous observations into a single vector would obscure this structure and make it difficult to naturally accommodate a varying number of active objects. CSV therefore adopts a hierarchical representation that separates shared system context from per-object information while exposing both to the Joint Actor.

At control instant t, CSV represents the system state as

$$s_t = (\mathbf{G}_t, \mathbf{O}_t) \tag{2}$$

where $\mathbf{G}_t$ denotes the system-level state, describing network conditions, media and playback behavior, rendering load, and the execution context of previously applied controls. $\mathbf{O}_t = \{\mathbf{o}_{i,t}\}_{i=1}^{N_t}$ denotes the object-level state, where N_t is the number of active rendering objects and $\mathbf{o}_{i,t}$ characterizes object i in terms of rendering cost, visual importance, dynamics, remote-rendering availability, and current placement. This representation allows the joint Actor to capture shared resource coupling across bitrate, FEC, and rendering decisions while preserving the object-level heterogeneity required for fine-grained placement.

4.2 Unified QoE Objective

This subsection formalizes the QoE dimensions introduced earlier as a unified objective for evaluating joint bitrate, FEC, and rendering-placement decisions. CSV considers three user-visible dimensions: visual quality, playback continuity, and interaction timeliness. Accordingly, CSV defines the profile-dependent utility as

$$U_t^{(p)} = w_q^{(p)} Q_t - w_c^{(p)} P_t^{\text{cont}} - w_f^{(p)} P_t^{\text{freeze}} - w_d^{(p)} P_t^{\text{deadline}}, p \in \mathcal{P} \tag{3}$$

where Q_t denotes normalized visual quality, while P_t^{cont}, P_t^{freeze}, and P_t^{deadline} denote penalties for frame-continuity degradation, playback freezing, and deadline violations, respectively. A larger Q_t indicates better quality, whereas larger penalty values indicate more severe QoE degradation. The quality term is derived from codec-level quantization statistics. Playback continuity is characterized by two complementary terms: P_t^{cont} captures sustained reductions in effective client-side frame rate, while P_t^{freeze} captures explicit playback stalls during which visual updates temporarily stop. The continuity penalty is obtained by normalizing the effective client-side frame rate between predefined good and bad operating points, while the freeze penalty is derived from the fraction of valid observation time affected by playback stalls. The deadline penalty captures interaction timeliness and is obtained by normalizing the deadline-miss rate (DMR), defined as

$$\text{DMR}_t = \frac{N_t^{\text{late}} + N_t^{\text{netloss}}}{N_t^{\text{eligible}}} \tag{4}$$

where N_t^{late} is the number of frames delivered after the application deadline, N_t^{netloss} is the number of frames classified as lost because of network delivery failure, and N_t^{eligible} is the number of frames eligible for deadline evaluation. Frames that remain undelivered for non-network reasons, as well as presentation-side skips, are tracked separately and are not counted as network losses. All penalty terms are bounded to $[0,1]$.

To accommodate different service priorities, CSV maintains a small catalog of predefined QoE profiles, denoted by $\mathcal{P}$. Each profile $p \in \mathcal{P}$ specifies a normalized weight vector $\mathbf{w}^{(p)} = [w_q^{(p)}, w_c^{(p)}, w_f^{(p)}, w_d^{(p)}]$ that determines the relative importance of the QoE dimensions. The interaction-oriented profile emphasizes continuous frame delivery and deadline compliance, whereas the visual-quality-oriented profile places greater importance on the fidelity of the rendered output. At session setup, the user or application can either select a profile directly or provide a natural-language request to an optional LLM-based intent parser. The parser runs only at session setup and maps the request to a profile identifier in $\mathcal{P}$; it cannot generate new weights or modify predefined profiles. For invalid or ambiguous requests, CSV falls back to a predefined default profile. CSV then activates the corresponding pretrained Actor for the session.

4.3 Learning a Joint Policy from Empirical Support

Given the cross-layer state and unified QoE objective, CSV learns a joint policy that maps the current system state to coordinated decisions over video bitrate, FEC redundancy, and object-level rendering placement. The resulting action space couples continuous transport controls with a combinatorial placement decision, making exhaustive exploration of all joint configurations impractical. CSV therefore adopts **Support-Constrained Critic-Guided Policy Learning (SCPL)**, which learns this policy from empirical measurements in three stages: QoE response modeling, support-constrained teacher construction, and joint policy distillation. Specifically, SCPL estimates the QoE outcomes of joint configurations, identifies high-utility decisions within an empirically supported and feasible candidate space, and distills these decisions into a compact Actor for runtime inference.

Empirical QoE response modeling. SCPL first learns how joint rendering and transport decisions affect application-level QoE under different operating conditions. To obtain empirical supervision, CSV performs controlled offline measurements on our local testbed, forming a set of measured samples:

$$\mathcal{D} = \left\{(\mathbf{e}_j, \widetilde{\mathbf{a}}_j, \mathbf{y}_j)\right\}_{j=1}^{M} \tag{5}$$

where M is the number of measured samples, $\mathbf{e}_j = h(s_j)$ is a compact representation of the operating condition of sample j, $\widetilde{\mathbf{a}}_j$ denotes the corresponding effective joint configuration, and $\mathbf{y}_j = [Q_j, P_j^{\text{cont}}, P_j^{\text{freeze}}, P_j^{\text{deadline}}]$ contains the measured QoE outcomes. The operating-condition representation summarizes network, media, playback, rendering, and execution context, while the joint configuration contains bitrate and FEC together with a compact description of rendering placement. For rendering, CSV summarizes the remote-rendered object

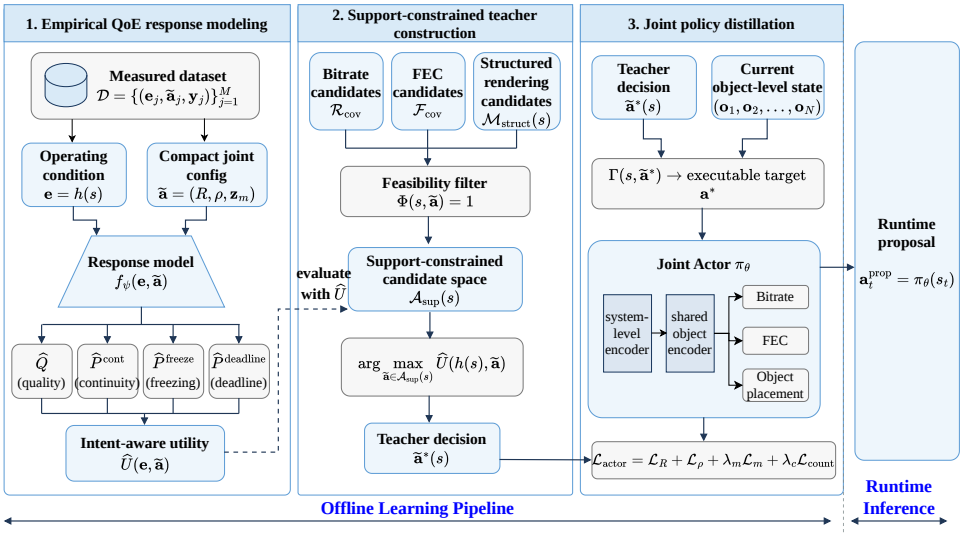


Figure 5: Support-constrained critic-guided policy learning

set using aggregate characteristics that capture its computation and transmission demand, thereby keeping the response representation compact while preserving the resource impact of rendering placement.

Using these measurements, SCPL learns a component-wise QoE response model:

$$\left[\widehat{Q}, \widehat{P}^{\text{cont}}, \widehat{P}^{\text{freeze}}, \widehat{P}^{\text{deadline}}\right] = f_{\psi}(\mathbf{e}, \widetilde{\mathbf{a}}) \tag{6}$$

where the four outputs correspond to the predicted encoding quality, frame-continuity degradation, playback freezing, and deadline violation under a candidate joint configuration. These predictions are evaluated using the unified QoE objective defined in Section 4.2, yielding a predicted utility $\widehat{U}(\mathbf{e}, \widetilde{\mathbf{a}})$. This response model provides a common basis for comparing alternative bitrate, FEC, and rendering configurations under the same operating condition and serves as the empirical critic for the subsequent policy-improvement stage.

Support-constrained teacher construction. Given the learned QoE response model, SCPL next constructs an improved joint decision for each training state. Since the joint action space spans bitrate, FEC, and object-level rendering placement, directly searching all possible configurations is impractical. SCPL therefore generates a finite set of structured candidates under the current system condition. Bitrate and FEC candidates are drawn from control ranges covered by real-system measurements, while rendering candidates are constructed from the object-level state using characteristics such as rendering cost, visual importance, and scene dynamics. These rendering candidates capture different amounts and compositions of remote rendering and are combined with bitrate and FEC choices to form candidate joint configurations.

Let $\widetilde{\mathbf{a}} = (R, \rho, \mathbf{z}_m)$ denote a candidate joint configuration. The support-constrained candidate space is defined as

$$\mathcal{A}_{\text{sup}}(s) = \left\{ \widetilde{\mathbf{a}} \in \mathcal{R}_{\text{cov}} \times \mathcal{F}_{\text{cov}} \times \mathcal{M}_{\text{struct}}(s) \mid \Phi(s, \widetilde{\mathbf{a}}) = 1 \right\} \tag{7}$$

where $\mathcal{R}_{\text{cov}}$ and $\mathcal{F}_{\text{cov}}$ denote the empirically covered bitrate and FEC candidates, $\mathcal{M}_{\text{struct}}(s)$ denotes the structured rendering candidates under state s, and $\Phi(\cdot)$ filters joint configurations according to the current resource and execution constraints.

SCPL then evaluates each candidate in $\mathcal{A}_{\text{sup}}(s)$ using the learned QoE response model and selects the candidate with the highest predicted utility as the teacher decision:

$$\widetilde{\mathbf{a}}^{*}(s) = \arg\max_{\widetilde{\mathbf{a}} \in \mathcal{A}_{\text{sup}}(s)} \widehat{U}\left(h(s), \widetilde{\mathbf{a}}\right) \tag{8}$$

In this way, SCPL constructs the teacher by combining empirically grounded transport choices, structured rendering alternatives, and current system constraints, and then using the QoE response model to identify the most promising joint configuration. This allows policy improvement to explore meaningful combinations of bitrate, FEC, and rendering placement while remaining grounded in measured system behavior.

Joint policy distillation. For each predefined QoE profile $p \in \mathcal{P}$ introduced in Section 4.2, the offline teacher uses the corresponding QoE weights to identify a preferable joint configuration for each training state through structured candidate generation and response-model evaluation. For clarity, we omit the profile index p in the following formulation. Repeating this procedure at runtime would introduce unnecessary decision overhead and is therefore unsuitable for real-time

control. SCPL instead distills the teacher decisions into a lightweight joint Actor π_θ, parameterized by θ, which directly maps the cross-layer state to a coordinated control proposal.

For each training state s, the teacher produces a compact joint decision $\widetilde{\mathbf{a}}^*$ containing the selected bitrate, FEC, and structured rendering decision. Since the teacher operates on a compact action representation while runtime execution requires explicit object-wise placement decisions, CSV converts the teacher output into an executable supervision target:

$$\mathbf{a}^* = \Gamma(s, \widetilde{\mathbf{a}}^*) \tag{9}$$

where $\Gamma(\cdot)$ bridges the compact action representation used during offline policy improvement and the full action representation required at runtime. It preserves the bitrate and FEC decisions selected by the teacher and instantiates the structured rendering decision into an object-wise placement vector according to the current object-level state. The resulting supervision target therefore specifies both the overall scale of remote rendering and the specific objects selected for remote execution.

The architecture of π_θ follows the hierarchical structure of the cross-layer state. A system-level encoder extracts the shared network, media, playback, rendering-load, and execution context, while a shared object encoder processes the features of individual rendering objects. The bitrate and FEC branches generate their decisions from the system-level context, whereas the rendering branch combines the shared system context with object-level representations to produce object-wise placement decisions.

The Actor is trained using the following joint distillation objective:

$$\mathcal{L}_{\text{actor}} = \mathcal{L}_R + \mathcal{L}_\rho + \lambda_m \mathcal{L}_m + \lambda_c \mathcal{L}_{\text{count}} \tag{10}$$

where $\mathcal{L}_R$ and $\mathcal{L}_\rho$ supervise the bitrate and FEC decisions, respectively, and $\mathcal{L}_m$ supervises the object-wise rendering placement. In addition, $\mathcal{L}_{\text{count}}$ constrains the aggregate amount of remote rendering to remain consistent with the teacher decision. The object-level placement loss and the aggregate count constraint are complementary: the former preserves which objects are selected, while the latter preserves the overall remote-rendering scale preferred by the teacher.

Profile-specific policy catalog. The component-wise QoE response model is shared across all predefined profiles because, for a given state and action, the predicted quality, continuity, freezing, and deadline outcomes do not depend on the selected QoE weights. For each profile $p \in \mathcal{P}$, SCPL applies the corresponding weight vector $\mathbf{w}^{(p)}$ to construct profile-specific teacher decisions and distills them into a lightweight Actor π_{θ_p}. The resulting Actors form a policy catalog and use the same state representation, action space, architecture, and runtime safety mechanisms. At session setup, CSV activates the Actor associated with the selected profile. This design requires neither online retraining nor language-model inference within the per-decision control loop.

After distillation, runtime control requires neither response-model evaluation nor candidate search. Given the selected profile p and current cross-layer state, the corresponding Actor directly produces a deterministic joint action proposal:

$$\mathbf{a}_t^{\text{prop}} = \pi_{\theta_p}(s_t) \tag{11}$$

which is subsequently processed by the runtime coordination layer described in the following section.

4.4 Multi-Timescale Runtime Coordination

This subsection describes how CSV coordinates joint bitrate, FEC, and rendering-placement decisions across different control timescales. These controls differ in their dynamics and reconfiguration costs: FEC should react promptly to packet-loss variations, bitrate adaptation requires time for bandwidth estimation and for encoder adjustments to take effect, and rendering placement incurs a higher reconfiguration cost because it changes object execution and composition. A single synchronous update rate is therefore unsuitable. CSV instead assigns each control dimension an independent update interval and coordinates their execution asynchronously.

Let T_R, T_ρ, and T_m denote the minimum update intervals for bitrate, FEC, and rendering placement, respectively. For each control dimension $j \in \{R, \rho, m\}$, CSV determines its update eligibility according to both the elapsed time since its previous effective update and the current system state:

$$M_{j,t} = \mathbb{I}[\Delta t_{j,t} \geq T_j] \cdot \mathcal{E}_j(s_t) \tag{12}$$

where $\Delta t_{j,t}$ is the elapsed time since dimension j last became effective, and $\mathcal{E}_j(s_t)$ denotes its state-dependent eligibility. If $M_{j,t} = 0$, the dimension retains its current effective configuration; otherwise, the new policy proposal is allowed to proceed. This enables the three control dimensions to evolve independently at their respective timescales without requiring synchronized updates.

Eligible actions are further processed by semantic guards and temporal stabilization. FEC adaptation is constrained by the current loss condition and transmission headroom, with hysteresis, holding intervals, and switching budgets used to suppress oscillation. Bitrate updates are bounded by the available transmission budget and the allowed change between consecutive decisions, while rendering updates limit the number of object-placement changes within each control cycle. Let $\bar{a}_{j,t}^{\text{prop}}$ denote the proposal after these checks. The scheduled action is

$$a_{j,t}^{\text{sched}} = M_{j,t}\bar{a}_{j,t}^{\text{prop}} + (1 - M_{j,t})\, a_{j,t-1}^{\text{eff}} \tag{13}$$

The resulting joint action is then converted into an executable command through safety projection:

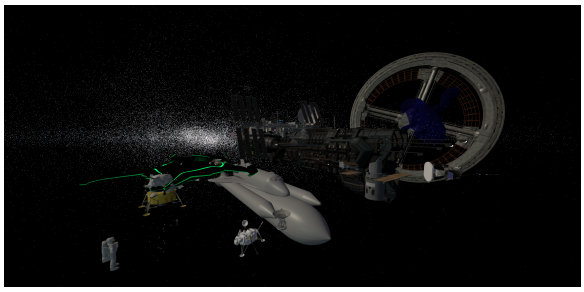


Figure 6: Space-themed immersive scene

$$\mathbf{a}_t^{\text{cmd}} = \mathcal{P}\left(\mathbf{a}_t^{\text{sched}}, s_t\right) \tag{14}$$

where $\mathcal{P}(\cdot)$ enforces admissible operating ranges, bounded reconfiguration magnitude, and resource feasibility for bitrate, FEC, and rendering placement. This stage only enforces execution feasibility and does not perform an additional QoE optimization. Under persistent performance degradation, the coordinator may further restrict disruptive updates or temporarily adjust control bounds to facilitate recovery.

Because a submitted command may not take effect immediately, CSV distinguishes the policy proposal $\mathbf{a}_t^{\text{prop}}$, the executable command $\mathbf{a}_t^{\text{cmd}}$, and the effective configuration $\mathbf{a}_t^{\text{eff}}$ observed after execution. The effective configuration and the elapsed time since each control dimension became effective are incorporated into the subsequent state, ensuring that future decisions are based on the configuration actually realized by the system.

5 Implementation

This section describes how the design in Fig. 4 is instantiated on top of Unity/RenderFusion and WebRTC. We implement the cross-layer state collection, joint-policy inference, multi-timescale runtime coordination, action execution, and feedback alignment described in Section 4 while preserving the native rendering and transport pipelines.

Cross-layer state and policy inference. The Runtime Context is constructed from WebRTC and application-level instrumentation. WebRTC provides network, transport, and encoding statistics, while the application runtime provides rendering measurements, presentation events, and object-level state. These observations are maintained in latest-value caches and organized into the system-level and object-level representations defined in Section 4.1. At session setup, the selected QoE profile defined in Section 4.2 activates the corresponding pretrained Actor learned through the SCPL pipeline in Section 4.3. During runtime, the Actor consumes the current cross-layer state and directly produces bitrate, FEC, and object-level rendering-placement proposals. Telemetry collection and policy inference execute outside the rendering and packet-processing paths to avoid blocking latency-sensitive operations.

Runtime coordination and action execution. The multi-timescale coordinator described in Section 4.4 processes Actor proposals according to update eligibility, stabilization, and feasibility constraints before issuing executable commands. The Runtime Execution Plane in Fig. 4 then maps these commands to the underlying systems. Rendering-placement decisions are mapped to Unity scene objects and applied through RenderFusion, while bitrate and FEC decisions are mapped to the corresponding WebRTC controls, including media bitrate control and the native ULPFEC path. Remote-rendered output is delivered through WebRTC and composited with locally rendered content at the client. Native GCC, pacing, and NACK/RTX remain active. The prototype supports 30 controllable rendering objects, while codec, resolution, and target frame rate are configured as experimental parameters.

Feedback alignment and offline measurements. Because submitted commands may take effect asynchronously, the implementation records policy proposals, issued commands, effective configurations, and their readback timestamps separately. Command identifiers associate each proposal with its execution result, while non-blocking queues transfer commands across runtime components. High-frequency observations may be coalesced, whereas configuration transitions and application events are retained in timestamped logs. These records realize the feedback alignment described in Section 4.4, ensuring that subsequent states reflect the configuration actually in effect. The same aligned measurements, including media bitrate, FEC overhead, retransmission traffic, and QoE outcomes defined in Section 4.2, are also used to construct the empirical samples for SCPL's offline response modeling and policy distillation.

6 Evaluation

6.1 Evaluation Methodology

Application and testbed. We evaluate CSV on a real-system prototype built with an interactive Unity-based XR application, the RenderFusion [25] split-rendering framework, and the WebRTC media transport stack. As shown in Figure 6, the application presents a space-themed 3D scene that supports both virtual and real-world backgrounds and allows users to navigate the scene from changing viewpoints. The scene contains 30 controllable 3D objects that form the object-level rendering workload; each object can be independently assigned to local or remote rendering, enabling different split-rendering configurations under the same scene content. The detailed characteristics of these 30 objects are provided in Appendix A. The prototype implements the complete end-to-end rendering and delivery pipeline: the client renders locally

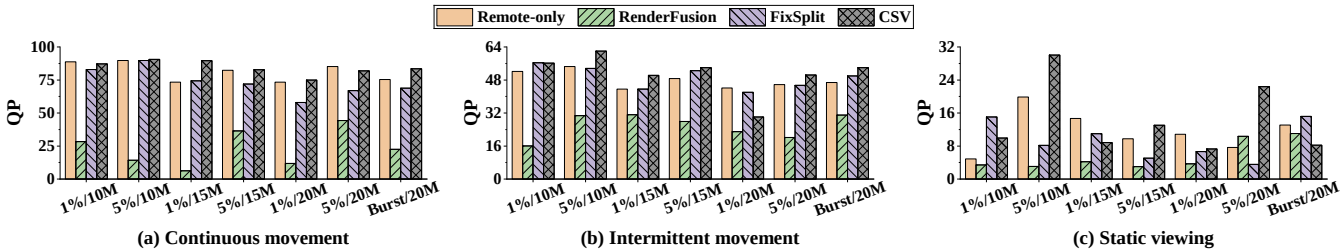


Figure 7: Encoder QP across user behaviors (lower values indicate finer quantization; x-axis: loss rate/bandwidth)

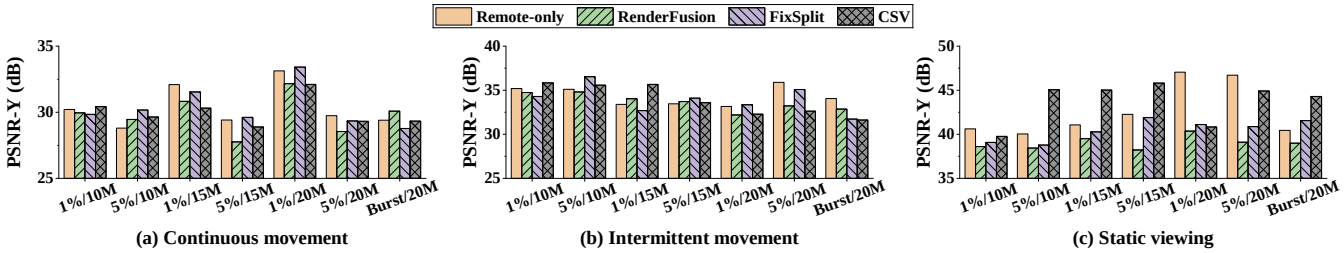


Figure 8: PSNR-Y across user behaviors (higher values indicate better visual fidelity; x-axis: loss rate/bandwidth)

assigned objects and composites them with the remote video stream received through WebRTC, while the rendering server processes remotely assigned objects and encodes and transmits the resulting content.

Our testbed consists of two physical machines interconnected through a bridged network path. The client runs 64-bit Windows 11 on an AMD Ryzen 7 5800H CPU with 8 cores, 16 threads, and a base frequency of 3.2 GHz. The rendering server runs 64-bit Windows 11 on an Intel Core i9-12900K CPU with 16 cores and 24 threads and is equipped with an NVIDIA GeForce RTX 3080 GPU with 10 GB of memory. Across all experiments, we use the same scene and rendering workload to ensure consistent comparison across rendering and transport configurations. Unless otherwise stated, the remote video stream is encoded using VP8 at 3840×1080 and 30 FPS.

Network conditions. We emulate controlled network conditions using a custom Windows impairment script implemented with PyDivert [52]. The bottleneck bandwidth is varied among 10, 15, and 20 Mbps to cover different levels of bandwidth pressure relative to the transmission demand of the immersive workload. We use a fixed RTT of 20 ms, consistent with the low-latency edge-network regime considered in prior interactive-streaming studies [51]. For steady impairments, we evaluate packet-loss rates of 1% and 5% to represent mild and more challenging loss conditions. We additionally evaluate a burst-loss condition at 20 Mbps, where the loss rate increases from 0% to 15% during two 10-s intervals at 50–60 s and 100–110 s. This setting evaluates whether CSV can adapt when network quality changes abruptly rather than remaining at a stationary operating point. Each run lasts 150 s, and each configuration is repeated five times.

Baselines. We compare CSV against three representative rendering strategies. **Remote-only** renders all eligible objects on the remote server, representing full rendering offload. **FixSplit** statically assigns the ten most computationally complex objects to the remote server and renders the remaining objects locally, representing a fixed split-rendering configuration. **RenderFusion** [25] dynamically applies its native object-level split-rendering placement. All three baselines use the same underlying WebRTC transport mechanisms, allowing the comparison to isolate the effects of rendering placement and CSV's joint coordination of rendering and transport controls.

Evaluation metrics. We evaluate CSV from three perspectives: end-to-end QoE, runtime control overhead, and system resource efficiency. End-to-end QoE is characterized by encoder quantization parameter (QP) and luminance peak signal-to-noise ratio (PSNR-Y) for visual quality, delivered FPS and cumulative freeze duration for playback continuity, and deadline-miss rate (DMR) with a 100-ms deadline for interaction timeliness. Runtime overhead is measured by Actor inference, policy refinement, action commit, and total control latency. Resource efficiency is evaluated using server CPU and GPU utilization, memory consumption, and the bandwidth allocation among media, RTX, and FEC traffic.

6.2 Interaction-Oriented QoE

Following Section 4.2, we use *"ensure responsive interaction for collaborative XR"* as a representative natural-language

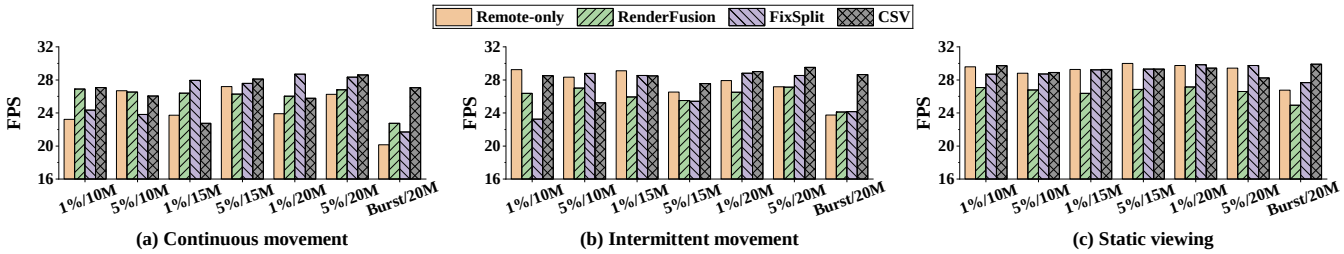


Figure 9: Delivered FPS across user behaviors (higher values indicate smoother frame delivery; x-axis: loss rate/bandwidth)

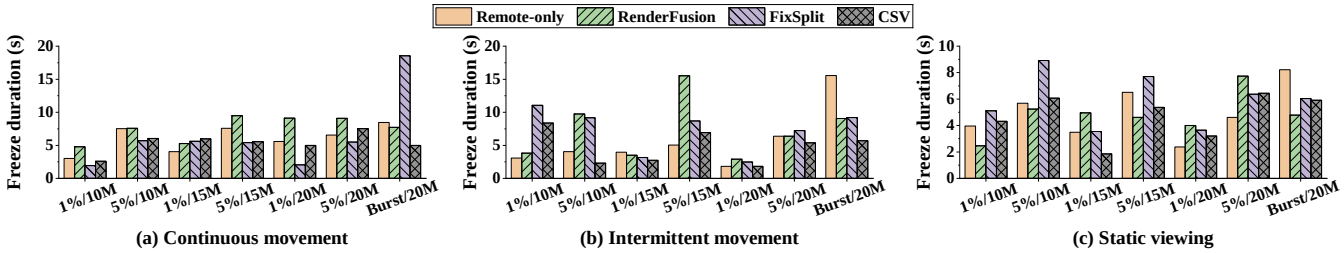


Figure 10: Freeze duration across user behaviors (lower values indicate less playback freezing; x-axis: loss rate/bandwidth)

QoE request. This request maps to the predefined interaction-oriented profile, which prioritizes timely responses and continuous frame delivery. To evaluate CSV under different levels of scene dynamics, we consider three controlled user-motion patterns: continuous movement, intermittent movement, and static viewing. These patterns respectively represent sustained viewpoint changes, periodic transitions between motion and stationary periods, and a stable viewpoint, allowing us to separate the effects of persistent motion, transient motion, and network variation without user movement. For continuous movement, the user follows a circular trajectory with a radius of 3 m at an eye height of 1.6 m, completing one revolution every 60 s at an angular velocity of 6°/s. Intermittent movement follows the same trajectory and motion speed but alternates between moving for 5 s and remaining stationary for 5 s. Static viewing keeps both the user position and viewing direction unchanged. Using the same scene and trajectory parameters across motion cases provides a controlled and repeatable workload while varying only the temporal pattern of viewpoint changes.

6.2.1 Visual Quality

We first evaluate visual quality to determine whether CSV improves rendering and transport coordination without sacrificing the fidelity of the delivered content. We use encoder quantization parameter (QP) and luminance peak signal-to-noise ratio (PSNR-Y) as complementary metrics. Lower QP indicates finer quantization, while higher PSNR-Y indicates greater luminance fidelity of the final composited output relative to the reference view.

Figure 7 compares the encoder QP of Remote-only, RenderFusion, FixSplit, and CSV across the seven network conditions under continuous movement, intermittent movement, and static viewing. QP generally decreases as user motion becomes less frequent, reflecting the lower temporal complexity of remotely rendered content under more stable viewpoints. CSV does not consistently achieve the lowest QP, while RenderFusion often operates at lower QP values because its rendering placement changes the content and complexity of the remote video stream.

Figure 8 reports the PSNR-Y of the final composited output for the same schemes, network conditions, and user behaviors. Across the six steady-loss conditions, CSV achieves average PSNR-Y values of 30.12, 34.26, and 43.57 dB under continuous movement, intermittent movement, and static viewing, respectively. Under continuous and intermittent movement, CSV remains within 0.45 dB and 0.11 dB of Remote-only, respectively, while maintaining visual quality comparable to FixSplit and generally outperforming RenderFusion. Under static viewing, CSV achieves the highest average PSNR-Y, exceeding Remote-only by 0.61 dB. Under burst loss, CSV maintains comparable visual quality during continuous movement and achieves the highest PSNR-Y under static viewing, although its advantage is less pronounced during intermittent movement. Overall, these results show that CSV preserves competitive end-to-end visual quality while jointly adapting rendering placement and transport controls.

6.2.2 Playback Continuity

We evaluate playback continuity using delivered FPS and freeze duration. Delivered FPS measures the rate at which video frames are successfully received and decoded by the

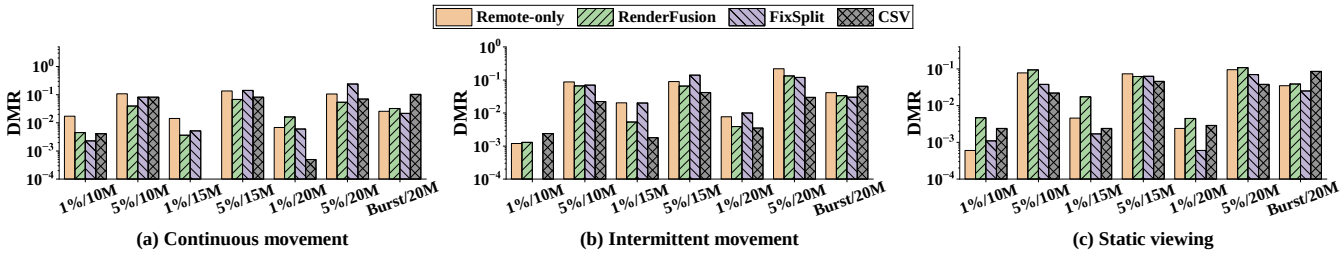


Figure 11: DMR across user behaviors (lower values indicate fewer deadline misses; x-axis: loss rate/bandwidth)

browser, while freeze duration measures the cumulative time spent in playback freezes.

As shown in Figure 9, CSV matches or remains within 1.5% of the strongest baseline in delivered FPS when averaged across the six steady-loss conditions for each user behavior. Its advantage is more pronounced under burst loss: CSV achieves the highest delivered FPS under continuous movement, intermittent movement, and static viewing, exceeding the strongest baseline for each behavior by 18.95%, 18.63%, and 8.02%, respectively. These results demonstrate CSV's ability to sustain frame delivery during abrupt network degradation.

As shown in Figure 10, CSV also improves playback continuity by reducing freezing across several motion-intensive settings. Under steady loss, CSV reduces the average freeze duration by approximately 34% relative to RenderFusion and FixSplit during intermittent movement, although Remote-only achieves a shorter duration. During continuous movement, CSV also reduces average freezing compared with Remote-only and RenderFusion, while FixSplit achieves the shortest duration. Under static viewing, CSV remains close to Remote-only and outperforms the other two baselines on average. The clearest gains occur under burst loss during continuous and intermittent movement, where CSV achieves the shortest freeze duration, reducing it by 35.58% and 37.20% relative to the strongest baseline, respectively. Overall, CSV's benefits are most evident under dynamic user movement and burst loss, where coordinated rendering and transport adaptation helps preserve playback continuity under rapidly changing network conditions.

6.2.3 Interaction Timeliness

We evaluate interaction timeliness using the deadline-miss rate (DMR) defined in Section 4.2, with a 100-ms deadline. A lower DMR indicates better deadline compliance.

As shown in Figure 11, CSV maintains a low DMR across the six steady-loss conditions, with gains varying across user behaviors. Under continuous movement, CSV reduces average DMR by 38.63% and 50.22% relative to Remote-only and FixSplit, respectively, although RenderFusion performs better. Under intermittent movement, CSV achieves the lowest DMR in five of the six conditions and reduces the average by 63.12%

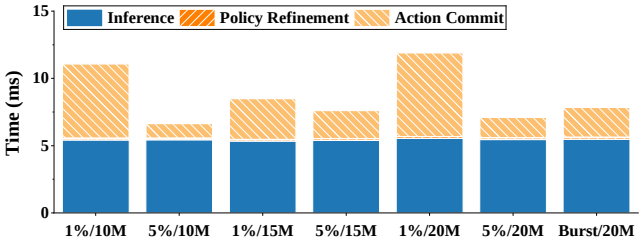


Figure 12: Runtime control latency breakdown

relative to RenderFusion, the strongest baseline. Under static viewing, CSV achieves the lowest average DMR, outperforming FixSplit by 34.80%. In particular, CSV achieves the lowest DMR under all three 5%-loss conditions for both intermittent movement and static viewing.

Under burst loss, CSV exhibits a higher DMR than the baselines while achieving the highest delivered FPS across all three user behaviors and the shortest freeze duration during continuous and intermittent movement, reflecting a trade-off between playback continuity and interaction timeliness. Across all seven network conditions and three user-motion patterns, CSV reduces the average DMR by 32.0% relative to the mean across the three baselines.

6.3 SCPL Training and Runtime Overhead

We analyze SCPL from two perspectives: offline training cost and online control latency. SCPL is trained entirely offline. Its pipeline fits empirical QoE response models from collected system measurements, searches the support-constrained action space to construct teacher decisions, and distills these decisions into a lightweight Actor. Excluding measurement collection, the complete pipeline finishes in approximately 5–10 minutes on our prototype. Empirical QoE response modeling takes 1–3 minutes, support-constrained teacher construction takes about 40 seconds, and joint policy distillation takes 1–2 minutes. These costs are incurred before deployment and do not affect runtime operation.

At runtime, Figure 12 decomposes each joint decision into inference, policy refinement, and action commit. Policy refinement denotes lightweight runtime processing that enforces update eligibility, stabilization, feasibility constraints, and safety

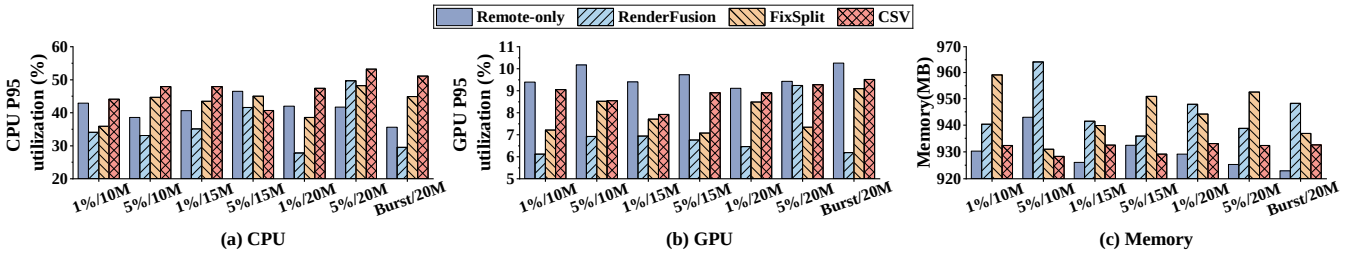


Figure 13: Server resource utilization

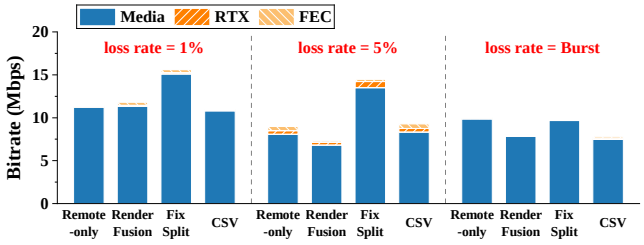


Figure 14: Bandwidth allocation

checks on the Actor proposal. Across the seven evaluated network conditions, the three stages take 5.434, 0.129, and 2.941 ms on average, respectively, yielding an average end-to-end control latency of 8.504 ms. The total latency ranges from 6.639 to 11.904 ms. Inference remains stable between 5.344 and 5.554 ms, while policy refinement adds at most 0.148 ms; most variation comes from action commit, which ranges from 1.072 to 6.217 ms as different decisions activate different runtime control paths. The average and maximum total latencies account for only 0.28% and 0.40% of the 3-s decision interval, respectively, demonstrating that CSV incurs negligible runtime control overhead at its decision timescale.

6.4 System Resource Analysis

Under continuous movement, we evaluate CSV's overhead in terms of network bandwidth allocation and server resource utilization, to characterize its repair-traffic demand and computational cost.

Bandwidth allocation. Figure 14 decomposes server traffic into media, RTX, and FEC bitrate. At a 1% loss rate, CSV limits the combined RTX and FEC bitrate to 0.07, 0.07, and 0.11 Mbps under 10, 15, and 20 Mbps bottleneck bandwidths, respectively, preserving most of the transmission budget for useful media. At a 5% loss rate, CSV increases the repair bitrate to 0.58, 0.93, and 0.97 Mbps, while keeping the total sending rate within the configured bottleneck capacity. For example, under 20 Mbps bandwidth and 5% loss, CSV sustains 8.33 Mbps of media traffic, compared with 8.07 Mbps for Remote-only and 6.81 Mbps for RenderFusion. Under burst loss, CSV allocates 0.39 Mbps to RTX and FEC, higher than Remote-only, RenderFusion, and FixSplit. This increased repair traffic helps CSV maintain a higher delivered FPS and reduce freezing during movement, while also introducing the tradeoff between playback continuity and interaction timeliness discussed earlier. Overall, CSV does not impose a fixed bandwidth overhead. Instead, it dynamically adjusts repair traffic according to the observed network conditions while preserving useful media bandwidth.

Server resource utilization. We measure the 95th percentile of CPU and GPU utilization and the average memory consumption at the rendering server, as shown in Figure 13. These measurements include remote rendering, video encoding, packet recovery, telemetry collection, policy inference, and action execution, and therefore characterize the overall system footprint rather than the isolated cost of policy inference. Across the six steady-loss conditions, CSV has an average CPU P95 of 47.50%, compared with 42.05% for Remote-only, 36.90% for RenderFusion, and 42.65% for FixSplit. Thus, CSV increases CPU utilization by 4.85 to 10.60 percentage points relative to the baselines, while its maximum CPU P95 remains 53.20%. CSV's average GPU P95 is 8.77%, within the baseline range of 7.07% to 9.54%. Its average memory consumption is 931.27 MB, comparable to Remote-only (931.02 MB) and lower than RenderFusion (944.72 MB) and FixSplit (946.26 MB). Under burst loss, CSV records a CPU P95 of 51.10%, a GPU P95 of 9.51%, and an average memory footprint of 932.58 MB. These results show that CSV's runtime coordination introduces a moderate additional CPU cost, but does not create substantial GPU or memory pressure, and the server resources remain well below saturation.

7 Conclusion

This paper presents CSV, a QoE-driven framework that jointly coordinates media bitrate, FEC, and object-level rendering placement for immersive interactive applications. By explicitly accounting for the shared-bandwidth coupling between remote rendering and reliability protection, CSV improves end-to-end QoE across diverse network conditions while balancing visual quality, playback continuity, and interaction timeliness.

References

[1] Mengmeng Yang, Youyang Qu, Thilina Ranbaduge, Chandra Thapa, Nazatul Haque Sultan, Ming Ding, Hajime Suzuki, Wei Ni, Sharif Abuadbba, David Smith, et al. From 5g to 6g: A survey on security, privacy, and standardization pathways. *ACM Computing Surveys*, 58(8):1–38, 2026.

[2] Haochen Sun, Yifan Liu, Ahmed Al-Tahmeesschi, Avishek Nag, Mohadeseh Soleimanpour, Berk Canberk, Hüseyin Arslan, and Hamed Ahmadi. Advancing 6g: Survey for explainable ai on communications and network slicing. *IEEE Open Journal of the Communications Society*, 6:1372–1412, 2025.

[3] Yao Wang, Chungang Yang, Tong Li, Ying Ouyang, Xinru Mi, and Yanbo Song. A survey on intent-driven end-to-end 6g mobile communication system. *IEEE Communications Surveys & Tutorials*, 28:882–915, 2025.

[4] Cheng-Xiang Wang, Xiaohu You, Xiqi Gao, Xiuming Zhu, Zixin Li, Chuan Zhang, Haiming Wang, Yongming Huang, Yunfei Chen, Harald Haas, et al. On the road to 6g: Visions, requirements, key technologies, and testbeds. *IEEE Communications Surveys & Tutorials*, 25(2):905–974, 2023.

[5] Anlan Zhang, Chendong Wang, Yuming Hu, Ahmad Hassan, Zejun Zhang, Bo Han, Feng Qian, and Shichang Xu. Habitus: Boosting mobile immersive content delivery through full-body pose tracking and multipath networking. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1677–1695, 2024.

[6] M Eswaran, Prasad Jones Christydass Samuel, SK Karthika, Bonda Atchuta Ganesh Yuvaraju, Anilkumar Inkulu, and MVA Raju Bahubalendruni. Extended reality (xr) for industrial assembly: A state-of-the-art review toward human-centric industry 5.0. *Expert Systems with Applications*, 307:130877, 2026.

[7] Xinyi Zhang, Huaiyi Zhao, Jianer Zhou, Furong Yang, Shiyi Liu, and Gaogang Xie. Remote collaborations in metaverse under 5g mobile networks. *IEEE Communications Magazine*, 61(9):16–22, 2023.

[8] Yu Liu, Puqi Zhou, Zejun Zhang, Anlan Zhang, Bo Han, Zhenhua Li, and Feng Qian. Muv2: scaling up multi-user mobile volumetric video streaming via content hybridization and sharing. In *Proceedings of the 30th annual International Conference on mobile computing and Networking*, pages 327–341, 2024.

[9] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. Metastream: Live volumetric content capture, creation, delivery, and rendering in real time. In *Proceedings of the 29th annual international conference on mobile computing and networking*, pages 1–15, 2023.

[10] Jing Wang, Xiao Kong, Yunzhe Ni, Nian Wen, Jiaxing Zhang, Congcong Miao, and Honghao Liu. From source to solution: Tackling packet losses in large-scale cloud gaming systematically and precisely. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 1191–1206, 2026.

[11] Shibo Wang, Shusen Yang, Hailiang Li, Xiaodan Zhang, Chen Zhou, Chenren Xu, Feng Qian, Nanbin Wang, and Zongben Xu. Salientvr: Saliency-driven mobile 360-degree video streaming with gaze information. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 542–555, 2022.

[12] Dapeng Wu, Zhigang Yang, Puning Zhang, Ruyan Wang, Boran Yang, and Xinqiang Ma. Virtual-reality interpromotion technology for metaverse: A survey. *IEEE Internet of Things Journal*, 10(18):15788–15809, 2023.

[13] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 conference papers*, pages 1–1, 2024.

[14] Fabio Arena, Mario Collotta, Giovanni Pau, and Francesco Termine. An overview of augmented reality. *Computers*, 11(2):28, 2022.

[15] Jeevan S Devagiri, Sidike Paheding, Quamar Niyaz, Xiaoli Yang, and Samantha Smith. Augmented reality and artificial intelligence in industry: Trends, tools, and future challenges. *Expert Systems with Applications*, 207:118002, 2022.

[16] Sergio Barta, Raquel Gurrea, and Carlos Flavián. Augmented reality experiences: Consumer-centered augmented reality framework and research agenda. *Psychology & Marketing*, 42(2):634–650, 2025.

[17] Tara Boroushaki, Maisy Lam, Laura Dodds, Aline Eid, and Fadel Adib. Augmenting augmented reality with {Non-Line-of-Sight} perception. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1341–1358, 2023.

[18] Somaiieh Rokhsaritalemi, Abolghasem Sadeghi-Niaraki, and Soo-Mi Choi. A review on mixed reality: Current trends, challenges and prospects. *Applied Sciences*, 10(2):636, 2020.

[19] Tanh Quang Tran, Tobias Langlotz, and Holger Regenbrecht. A survey on measuring presence in mixed reality. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–38, 2024.

[20] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. {YuZu}:{Neural-Enhanced} volumetric video streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 137–154, 2022.

[21] Philipp A Rauschnabel, Reto Felix, Chris Hinsch, Hamza Shahab, and Florian Alt. What is xr? towards a framework for augmented and virtual reality. *Computers in human behavior*, 133:107289, 2022.

[22] Artur Becker and Carla M Dal Sasso Freitas. Evaluation of xr applications: A tertiary review. *ACM Computing Surveys*, 56(5):1–35, 2023.

[23] Fengqian Guo, Yuhan Zhou, Longwei Jiang, Congcong Miao, Yuxin Liu, Chenren Xu, Hancheng Lu, Chang Wen Chen, Yaxiong Xie, and Honghao Liu. {BLADE}: Adaptive {Wi-Fi} contention control for {Next-Generation}{Real-Time} communication. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 131–151, 2026.

[24] Jangwoo Son, Serhan Gül, Gurdeep Singh Bhullar, Gabriel Hege, Wieland Morgenstern, Anna Hilsmann, Thomas Ebner, Sven Bliedung, Peter Eisert, Thomas Schierl, et al. Split rendering for mixed reality: Interactive volumetric video in action. In *SIGGRAPH Asia 2020 XR*, pages 1–3. 2020.

[25] Edward Lu, Sagar Bharadwaj, Mallesham Dasari, Connor Smith, Srinivasan Seshan, and Anthony Rowe. Renderfusion: Balancing local and remote rendering for interactive 3d scenes. In *2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 312–321. IEEE, 2023.

[26] Jangwoo Son, Yago Sanchez, Cornelius Hellge, and Thomas Schierl. Split rendering with l4s over 5g for latency critical interactive xr applications. *IEEE Communications Magazine*, 62(8):46–52, 2024.

[27] Zhihui Ke, Xiaobo Zhou, Yuyang Liu, Zhizhuo Pang, and Tie Qiu. Edgegaussian: Real-time free-viewpoint video for mobile vr via edge-client collaborative neural rendering. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*, pages 773–787, 2025.

[28] Yihua Cheng, Ziyi Zhang, Hanchen Li, Anton Arapin, Yue Zhang, Qizheng Zhang, Yuhan Liu, Kuntai Du, Xu Zhang, Francis Y Yan, et al. {GRACE}:{Loss-Resilient}{Real-Time} video through neural codecs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 509–531, 2024.

[29] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[30] Shiyang Huang, Gerui Lv, Yuankang Zhao, Jiaxing Zhang, Qingyue Tan, Congkai An, Huanhuan Zhang, Xinyi Zhang, Qinghua Wu, and Zhenyu Li. Breath: Adaptive protection boundary in fec encoding for mobile real-time video streaming. In *Proceedings of the ACM Web Conference 2026*, pages 5221–5230, 2026.

[31] Ke Chen, Han Wang, Shuwen Fang, Xiaotian Li, Minghao Ye, and H Jonathan Chao. Rl-afec: adaptive forward error correction for real-time video communication based on reinforcement learning. In *Proceedings of the 13th ACM Multimedia Systems Conference*, pages 96–108, 2022.

[32] Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, Aaron Yi Ding, and Jörg Ott. Nimbus: Towards latency-energy efficient task offloading for ar services. *IEEE Transactions on Cloud Computing*, 11(2):1530–1545, 2022.

[33] Håkon Medhus Fornes, Elias Hoel Birketvedt, Carsten Griwodz, Magnus Skjegstad, Michael Welzl, and Özgü Alay. Acceptable latency in predictable first-person vr cloud games. In *Proceedings of the 17th International Workshop on IMmersive Mixed and Virtual Environment Systems*, pages 58–64, 2025.

[34] Yang Li, Jiaxing Qiu, Hongyi Wang, Zhenhua Li, Feng Qian, Jing Yang, Hao Lin, Yunhao Liu, Bo Xiao, Xiaokang Qin, et al. Dissecting and streamlining the interactive loop of mobile cloud gaming. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 595–611, 2025.

[35] Sam Shippey, Suresh Srinivasan, Huu Phuoc Dang, Ehsan Aryafar, and Jacob Chakareski. An experimental evaluation of 360-degree abr video streaming over mmwave wireless links. In *2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom)*, pages 113–120. IEEE, 2024.

[36] Ali Zeynali, Mohammad H Hajiesmaili, and Ramesh K Sitaraman. Bola360: Near-optimal view and bitrate adaptation for 360-degree video streaming. In *Proceedings of the 15th ACM Multimedia Systems Conference*, pages 12–22, 2024.

[37] Zitong Li, Changqiao Xu, Chuxing Fang, Mu Wang, Shujie Yang, Yuang Cai, Lujie Zhong, and Gabriel-Miro Muntean. Joint abr and routing optimization for 360° video in sagin: Resolving the fov prediction-buffer dilemma. *IEEE Transactions on Mobile Computing*, 2026.

[38] Neil Agarwal, Rui Pan, Francis Y Yan, and Ravi Netravali. Mowgli: Passively learned rate control for {Real-Time} video. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 579–594, 2025.

[39] Tianyu Chen, Yiheng Lin, Nicolas Christianson, Zahaib Akhtar, Sharath Dharmaji, Mohammad Hajiesmaili, Adam Wierman, and Ramesh K Sitaraman. Soda: An adaptive bitrate controller for consistent high-quality video streaming. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 613–644, 2024.

[40] Zhetao Li, Fei Gui, Jinkun Geng, Dan Li, Zhibo Wang, Junfeng Li, Yang Cheng, and Usama Zafar. Dante: Enabling fov-aware adaptive fec coding for 360-degree video streaming. In *Proceedings of the 2nd Asia-Pacific Workshop on Networking*, pages 15–21, 2018.

[41] Congkai An, Huanhuan Zhang, Shibo Wang, Jingyang Kang, Anfu Zhou, Liang Liu, Huadong Ma, Zili Meng, Delei Ma, Yusheng Dong, et al. Tooth: Toward optimal balance of video {QoE} and redundancy cost by {Fine-Grained}{FEC} in cloud gaming streaming. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 635–651, 2025.

[42] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify:{Low-Latency} network video through tighter integration between a video codec and a transport protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, 2018.

[43] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Enabling high quality {Real-Time} communications with adaptive {Frame-Rate}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1429–1450, 2023.

[44] Pantea Karimi, Sadjad Fouladi, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Vidaptive: Efficient and responsive rate control for real-time video on variable networks. *arXiv preprint arXiv:2309.16869*, 2023.

[45] Sheng Cheng, Han Hu, and Xinggong Zhang. Abrf: Adaptive bitrate-fec joint control for real-time video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(9):5212–5226, 2023.

[46] Jia Pan, Anup Agarwal, Isil Diling, and Venkat Arun. Syntra: Synthesizing cross-layer controllers for low-latency video streaming. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26), Renton, WA*, 2026.

[47] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Converge: Qoe-driven multipath video conferencing over webrtc. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 637–653, 2023.

[48] Goodsol Lee, Junhong Min, Seyeon Kim, Juheon Yi, Kwang Taik Kim, Mung Chiang, Sangtae Ha, Kyunghan Lee, and Saewoong Bahk. {QCON}: Seamless {QoE-Aware} 5g streaming via {Multi-Connectivity}. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 1721–1739, 2026.

[49] Yixin Shen, Ruihua Chen, Bo Wang, Jing Chen, Haochen Zhang, Minhu Wang, Mingwei Xu, and Zili Meng. Mortise: Auto-tuning congestion control to optimize {QoE} via {Network-Aware} parameter optimization. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 2303–2321, 2026.

[50] Jin Heo, Vic Wang, Ketan Bhardwaj, and Ada Gavrilovska. Stimpack: An adaptive rendering optimization system for scalable cloud gaming. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 397–414, 2026.

[51] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 907–926, 2024.

[52] Fabio Falcinelli and Maximilian Hils. PyDivert: A python binding for WinDivert. https://github.com/ffalcinelli/pydivert. Accessed: September 15, 2026.

A Controlled Objects in the XR Scene

Table 1 summarizes the 30 controllable objects used in the space-themed XR scene. These objects span a wide range of geometric complexities and asset sizes, providing heterogeneous rendering workloads for evaluating object-level local and remote placement decisions.

Table 1: Characteristics of the 30 controllable objects in the space-themed XR scene.

#	Object	Description	Triangles	Vertices	File Size
1	maven	MAVEN Mars orbiter	1,746,023	4,214,407	9.98 MB
2	cassiopeia_remnant	Cassiopeia supernova remnant	574,678	1,677,130	9.99 MB
3	jwst	James Webb Space Telescope	493,785	508,474	4.49 MB
4	cassini	Cassini spacecraft	453,987	396,560	2.28 MB
5	emu_suit	EMU spacesuit	343,455	442,172	3.29 MB
6	iss	International Space Station	247,525	324,552	21.47 MB
7	crab_nebula	Crab Nebula	233,366	421,971	2.11 MB
8	robonaut	Robonaut	213,244	153,270	1.02 MB
9	capital_shuttle	Space shuttle	212,970	131,643	1.14 MB
10	iqto_spaceship	IQTO spaceship	200,552	134,494	7.44 MB
11	perseverance_rover	Perseverance Mars rover	199,521	138,783	4.76 MB
12	osirisrex	OSIRIS-REx spacecraft	184,407	553,221	1.22 MB
13	viking_lander	Viking lander	99,526	51,694	1.76 MB
14	lunar_module	Lunar module	97,588	64,787	700 KB
15	harbinger_cruiser	Harbinger cruiser	73,379	94,293	19.02 MB
16	space_station_3	Space station 3	68,300	113,855	21.87 MB
17	display_helmet	Display helmet	64,664	39,843	227 KB
18	parker_probe	Parker Solar Probe	39,257	39,233	433 KB
19	dsn_antenna	Deep Space Network antenna	31,842	55,637	2.77 MB
20	voyager	Voyager spacecraft	29,911	89,733	279 KB
21	station_v	Space station V	29,269	37,054	1.86 MB
22	landsat	Landsat satellite	26,738	32,223	15.59 MB
23	bennu_asteroid	Bennu asteroid	18,240	9,122	322 KB
24	juno_probe	Juno spacecraft	17,124	24,332	8.58 MB
25	need_some_space_galaxy	Galaxy background sphere	16,666*	50,000	1.34 MB
26	ingenuity_copter	Ingenuity helicopter	9,124	8,512	1.93 MB
27	pioneer	Pioneer spacecraft	8,189	8,797	1.95 MB
28	hubble	Hubble Space Telescope	7,672	23,016	1.62 MB
29	magellan	Magellan spacecraft	7,364	6,656	3.01 MB
30	nasa_astronaut	NASA astronaut	6,381	19,143	745 KB