Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Thesis

Trinity College Dublin

The University of Dublin

State-Aware Analysis, Forecasting, and Predictive Scheduling of Virtual Reality Network Traffic for Enhanced Quality of Experience

Author:
Bozhong Liu
23371258

Submitted
2026

Declaration of Work

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I confirm that the work has been completed in full compliance with relevant university policies, including the Academic Integrity policy, the Trinity Policy on Good Research Practice, and the guidance on AI and GenAI in Teaching, Learning, Assessment and Research.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I do not consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signed: LIU BOZHONG Date: 30/09/2025

Abstract

The escalating demand for interactive Virtual Reality (VR) applications places unprecedented demands on underlying network infrastructures, requiring stringent Quality of Service (QoS) and Quality of Experience (QoE) to mitigate motion sickness and ensure immersive user experiences. A critical challenge lies in adapting network resource management to the highly dynamic and user-state-dependent traffic patterns unique to VR, where traditional prediction methods often fall short in accounting for variations induced by user mobility (e.g., stationary versus moving). Moreover, effectively translating window-level traffic forecasts into actionable, fine-grained scheduling decisions at the millisecond scale presents a temporal disconnect limiting the practical utility of predictive intelligence.

This thesis addresses these challenges comprehensively. First, it introduces a meticulously captured and explicitly annotated VR network traffic dataset, derived from controlled, reproducible VR sessions. This dataset distinguishes itself by comprehensively recording both server-to-client and client-to-server communication flows from a custom-built Unity VR application, crucially labeling traffic segments according to distinct user mobility states: stationary and moving. The dataset design and methodology enable reproducible research in immersive networking; the dataset and reproduction code are publicly available at https://github.com/bozliu/vr-traffic-prediction-sps/tree/36a2d4d73c80c658d3d9174dcc1e9a08e9fc3922/data/raw.

Leveraging this rich data, the research demonstrates a novel approach to VR user-state classification using a Random Forest classifier. We propose a sliding-window-based multi-scale (0.5/1.0/2.0 s) hybrid time-frequency feature framework, robustly centering and performing FFT on packet size sequences to extract Nyquist-normalized frequencies and L1-normalized amplitudes of top-N peaks. This is augmented with spectral centroid/flatness descriptors and an uplink/downlink direction bit to capture VR traffic's periodicity and short-term dynamics without Deep Packet Inspection (DPI). This approach significantly enhances classification accuracy, increasing it from a time-only baseline of $\approx$ 69% to $\approx$ 94% on the test set (93.74% overall; Moving F1=0.915). Feature-importance analysis highlights direction context and frequency position/shape (Nyquist-normalized frequencies, spectral centroid/flatness) as most discriminative, with amplitudes providing complementary information.

Furthermore, the thesis presents a comprehensive evaluation of models—including a Random Forest regressor, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Transformer architectures—for short-term forecasting of average packet size. This vital metric for proactive resource allocation consistently shows robust predictability at the 200 ms horizon ($R^2 \approx 0.85$–0.86; RF R^2=0.8586, GRU 0.8576, LSTM 0.8531, Transformer 0.8529), while very short horizons ($\leq$80 ms) remain challenging.

Finally, integrating these predictive capabilities, the thesis develops and validates an end-to-end prediction-aided hybrid Semi-Persistent Scheduling (SPS) and Proportional Fair (PF) framework. This framework seamlessly

integrates 200ms VR traffic predictions into a 1ms MAC layer scheduler. Through extensive multi-user simulations, this approach is shown to significantly improve delay fairness for VR users by approximately 3.0-3.5% and reduce tail latencies, all while maintaining full network throughput. The framework's robustness is verified across various stress scenarios, demonstrating particular efficacy under bursty and highly multiplexed traffic conditions. Crucially, its SPS user selection achieves a high Jaccard overlap of approximately 0.91 with an ideal oracle scheduler, providing strong evidence of the practical utility of predictive intelligence in strategic resource management. This work lays crucial groundwork for enhancing the immersive and responsive nature of future VR experiences through intelligent, adaptive, and QoE-centric network management.

Acknowledgements

Completing this thesis has been an immensely challenging yet profoundly rewarding journey, and it would not have been possible without the invaluable support, guidance, and encouragement of numerous individuals and institutions. My deepest gratitude goes to all who have contributed to this significant milestone in my academic career.

First and foremost, I extend my profound appreciation to my primary supervisor, Professor Marco Ruffini. Your exceptional academic mentorship, insightful guidance, and unwavering support have been instrumental in shaping my research from its nascent stages to its culmination. Your intellectual rigor, dedication, and patience have not only steered this project but have also profoundly influenced my growth as a researcher. I am truly grateful for the opportunities you have provided and for always believing in my potential.

I am also deeply indebted to my co-supervisor, Professor Gabriel-Miro Muntean. Your expertise, constructive feedback, and invaluable insights throughout this journey have been crucial. Your encouragement, particularly during challenging times, kept me motivated and focused on the overarching goals of this research. Thank you for your consistent support and for enriching my academic experience.

This work was conducted with the financial support of the Science Foundation Ireland Centre for Research Training in Digitally-Enhanced Reality (d-real) under Grant No. 18/CRT/6224. This funding was absolutely essential, providing me with the resources and environment necessary to dedicate myself fully to this research. I am incredibly grateful for this critical support.

My sincere thanks also go to my colleagues and fellow PhD students at CONNECT Centre, Trinity College Dublin. Your camaraderie, willingness to engage in stimulating discussions, and shared experiences have created a supportive and inspiring research environment. Special thanks to the administrative officer, Barbara Moran, whose efficient support ensured the smooth progression of my research activities.

Finally, and most importantly, I wish to thank my family and friends for their endless love, patience, understanding, and encouragement. Your emotional support, far-reaching beyond the academic realm, provided the strength and balance I needed to navigate the demanding nature of doctoral studies. This achievement is as much yours as it is mine.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 The Rise of Interactive VR Applications

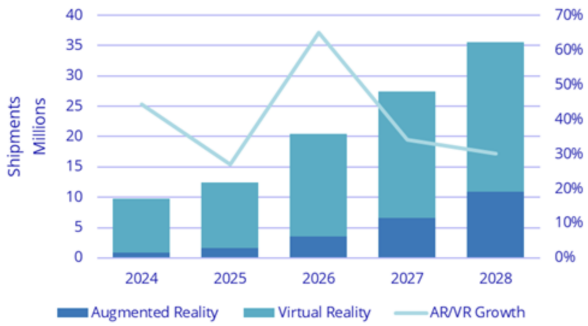


Figure 1.1: Worldwide AR/VR headset forecast[1].

Over the past decade, virtual reality (VR) technology has transitioned from a niche curiosity to a mass-market medium. Industry analysts report that worldwide AR/VR headset shipments rebounded by 10% in 2024, with a further 44% surge projected for 2025 as mixed-reality devices reach lower price points and integrate on-device AI acceleration [1, 2]. Reuters further notes that this growth trajectory is expected to continue, with volumes predicted to exceed 22.9 million units by 2028 [3]. This significant hardware momentum is paralleled by increasingly sophisticated software ecosystems on consumer platforms such as Meta Quest 3/3S, PlayStation VR2, and Apple Vision Pro, each emphasizing high-fidelity, interactive content rather than passive 360° video playback.

Interactive gaming remains the primary catalyst for VR adoption. Titles such as Beat Saber and Half-Life: Alyx can trigger tens to hundreds of Mbps bursts in challenging scenes (e.g., dense interactions or rapid viewpoint changes), which stresses last-mile Wi-Fi and 5G links. While our experiments analyze application-level Unity Netcode UDP traffic on a LAN-based setup rather than cloud-streamed video, both exhibit common sensitivities to latency and jitter, highlighting the universal need for responsive, state-aware networking. To address these demands, predictive scheduling policies for VR applications are being explored, leveraging the predictability of user movements in virtual worlds to proactively deliver rendered images and enhance QoE [4]. Crucially,

recent research emphasizes that differentiating VR packets from background network flows at the access layer can reduce motion-to-photon latency by up to 35%, directly mitigating simulator sickness and improving user comfort [5]. These findings highlight the intrinsic coupling between efficient network traffic management and the delivery of an immersive user experience, a critical theme in contemporary VR networking research. Beyond solitary play, social VR platforms have fostered a new paradigm of synchronous, avatar-mediated interactions, exemplified by applications like Horizon Worlds and VRChat. User studies indicate that feelings of "co-presence" and perceived "social support" within shared virtual environments are strongly correlated with sustained user engagement [6]. Both aspects underscore the inherently interactive nature of modern VR, where low-latency, bidirectional traffic is paramount for establishing and maintaining social presence.

The proliferation of interactive VR extends beyond consumer entertainment into critical professional and educational domains. A multi-institution survey reveals that 78% of Fortune 500 firms plan to pilot VR-based training by 2026, citing measurable improvements in skill acquisition and learner retention [7]. PricewaterhouseCoopers' (PwC) controlled studies have shown VR "v-learners" to be significantly faster (four times) and more confident (275%) in acquiring soft skills compared to traditional classroom cohorts [8]. In healthcare, a 2023 scoping review documented over 150 clinical VR deployments for diverse applications including pain management, mental health therapy, and surgical rehearsal [9]. Similarly, universities are expanding VR classrooms, supported by headset donations and immersive learning grants, reflecting a growing acceptance of VR as an educational tool [10]. These enterprise and institutional applications further solidify the demand for robust, high-performance VR networking, as service interruptions or quality degradation can have significant practical consequences.

To address the computational and networking demands of increasingly complex VR experiences, especially for collaborative or high-fidelity applications, industry and academia are actively exploring Cloud/Edge VR architectures. Prototypes like CloVR demonstrate the feasibility of "near-far" rendering, enabling the rapid launch of metropolitan-scale collaborative scenes with sub-100 ms startup times, leveraging distributed edge GPUs to serve resource-constrained VR clients [11]. Complementary forecasts predict that containerized workloads orchestrated at the network edge will become dominant for latency-critical Extended Reality (XR) services by 2025 [12]. This paradigm shift necessitates re-evaluating traditional network traffic models, as rendering and processing responsibilities are distributed, leading to new patterns of data flow.

Collectively, these trends signify a rapid increase in interactive VR applications, characterized by real-time, bidirectional media streams, stringent sub-20 ms motion-to-photon latency budgets, and dynamic session characteristics influenced by user locomotion and scene complexity. Therefore, comprehensive understanding, classification, and prediction of VR network traffic under these evolving conditions are paramount for network operators. This is crucial for upholding the demanding Quality of Service (QoS) and Quality of Experience (QoE) guarantees required by these applications [13, 14, 15], and ultimately, for unlocking the full economic and experiential potential of the VR medium. This thesis specifically addresses these challenges through novel user state classification, short-term traffic forecasting methods, and an end-to-end prediction-aided resource scheduling framework.

1.1.2 Stringent QoS/QoE Requirements for VR

Unlike traditional video streaming or standard online gaming, Virtual Reality (VR) applications involve strong interactivity and higher frame rates, imposing exceptionally stringent requirements on the network in terms of end-to-end delay, packet arrival patterns, throughput, and reliability [16, 17, 18]. Immersive VR, in particular, imposes hard-real-time constraints that are far tighter than those of conventional media or cloud gaming [19]. Recent studies and 5G/6G standards underscore these demands: the motion-to-photon (MTP) latency, which is the total time from user movement to the corresponding visual feedback, must be kept below 15–20 ms to prevent cybersickness and preserve the sense of presence [20, 21, 22]. MTP is an end-to-end closed-loop metric that encompasses not only network latency but also rendering, scanning, and synthesis delays. Beyond MTP, the end-to-end network latency must typically be less than 20 ms, with some studies suggesting well below 10 ms for optimal Extended Reality (XR) experiences [23] and jitter strictly constrained to 1–5 ms for seamless interactive VR experiences [22, 24]. Furthermore, low jitter (often single-digit milliseconds) and very high reliability (Ultra-Reliable Low-Latency Communication, or uRLLC-class targets) are frequently cited for high-quality XR; exact budgets vary by pipeline and rendering strategy. Bandwidth requirements are equally demanding, ranging from approximately 25 Mbps for viewport-adaptive 4K/360° streams to throughputs that can go over 100 Mbps and even hundreds of Mbps or higher (visualizing encoding, resolution, FEC, and scene complexity) for high-fidelity, cloud-rendered six-degree-of-freedom (6DoF) experiences [20, 21, 23]. These stringent requirements align with the uRLLC paradigm of 5G and beyond, which is essential for enabling the full potential of VR/AR applications [25].

Controlled user studies further corroborate these thresholds: in a two-user collaborative VR task, latency artificially increased to 30 ms already caused a statistically significant drop in comfort and immersion scores, with cybersickness scores rising sharply beyond 40 ms [26]. Meeting these limits on commodity networks demands both prioritization and traffic awareness. For instance, machine-learning-driven Wi-Fi queue re-prioritisation can significantly reduce VR packet delay, effectively helping to keep frame deadlines under the stringent budget [5]. It is also notable that while MTP latency for visuals is critical, motion-to-sound latency can be tolerated up to 30 ms [27]. Additionally, VR applications exhibit distinct user states, such as stationary versus actively moving, each with unique network traffic patterns. Accurate identification and classification of these states can significantly enhance resource management and scheduling decisions [4]. Complementing this, large-scale passive measurements show that many interactive VR titles now generate uplink traffic shares exceeding 45%, a traffic symmetry largely unseen in traditional video-on-demand services; providers, therefore, must guarantee bidirectional bandwidth, not just downlink capacity [28]. Because many VR titles generate substantial **uplink** traffic, **direction-aware policies** are necessary; we explicitly include a **flow-direction feature** in our models to capture this asymmetry. Altogether, the literature converges on a picture where low latency, minimal jitter, ultra-high reliability, and symmetric high-rate pipes are mandatory prerequisites for acceptable Quality of Experience (QoE) in next-generation VR services.

1.1.3 Challenges in VR Traffic Management

Despite significant research progress, VR traffic management remains a multi-faceted challenge, primarily due to its notoriously bursty, mobility-sensitive, and cross-layer-coupled nature. The traffic of cloud-based interactive VR applications significantly differs from typical Internet traffic in structure and QoS requirements, posing challenges for current wireless technologies where missing requirements can worsen picture quality, cause perceived lag, and even dizziness [18, 23]. Unlike traditional video, VR traffic exhibits complex, non-stationary patterns and properties like long-range dependence and self-similarity, making accurate prediction difficult even with constant bitrate (CBR) encoding [16, 22, 29]. Cloud-streamed VR flows, for instance, can create micro-bursts exceeding five times the average rate whenever the viewpoint changes; such bursts can trigger bufferbloat in 5G base-station queues, necessitating advanced control mechanisms. Yang et al. demonstrate that a cross-layer early-congestion-control loop at the edge can raise the fraction of flows meeting the 20 ms delay target from 38% to 83% [30]. Furthermore, for social-VR platforms, throughput, end-to-end latency, and on-device computation resource utilization increase almost linearly with the number of users, leading to potential scalability issues and network congestion that can degrade or interrupt service for most users [31, 32]. Even with congestion kept at bay, packet-loss variability can still degrade QoE unless caching and buffering are jointly optimized: an adaptive 5G-MEC scheme that pre-fetches tiles based on head-movement prediction, for example, improves viewport-hit ratio by 15% and smooths bitrate under bandwidth swings [33].

At the radio-scheduler level, the emerging semi-persistent scheduling (SPS) upgrades, currently under discussion in 3GPP Release 19, aim to shorten grant periodicity dynamically for XR traffic while curbing User Equipment (UE) power overhead [34]. Short-term packet-level prediction, forecasting traffic within short windows (e.g., 80–100 ms), has shown significant potential for improving SPS performance and reducing computational overhead, especially within SPS scenarios in modern networks [25, 35]. In contrast, our results (Chapter 6) demonstrate low reliability for predictions below 80 ms, with robust performance achieved at approximately 200 ms. However, the patent literature notes open questions about how to align SPS cycles with highly variable VR packet inter-arrival times, and crucially, how to effectively translate such window-level predictions into actionable, fine-grained 1ms scheduling decisions without sacrificing responsiveness or increasing control overhead. Looking further ahead, network traffic forecasting has become crucial for supporting real-time QoS/QoE requirements, enabling efficient resource allocation, and ensuring seamless user experiences [16]. A 2024 Transformer-based residual learner trained on multimodal XR traces (ResLearn) demonstrates remarkable accuracy, suppressing peak-error by 99% relative to LSTM baselines and pointing toward predictive resource reservation at sub-100 ms horizons [36]. Nevertheless, despite the advancements in deep learning models (such as LSTM, GRU, and Transformers) which are increasingly applied for traffic forecasting [14, 16, 25], current predictors still face challenges related to generalization across diverse network conditions, the need for careful feature engineering, and effectively capturing heterogeneous features, including time-domain, frequency-domain, and application-specific characteristics like user mobility [16]. Furthermore, previous research has predominantly emphasized time-domain modeling and basic packet-level statistics, while overlooking frequency-domain analysis and broader temporal structures—such as low-frequency oscillations or periodic bursts—that could

more effectively distinguish between VR states (stationary vs. moving) [37]. While frequency-domain features have been successfully applied in other domains like VoIP and wireless communications, they remain largely unexplored in interactive VR traffic analysis [4, 17]. This reinforces the critical need for the hybrid time-frequency features and state-aware models developed in this thesis.

1.2 Problem Statement and Research Questions

Traditional network traffic analysis and prediction methods, largely focused on general internet traffic or simpler video streams, often fall short in addressing the unique characteristics of VR traffic, particularly its dynamic variability induced by distinct user mobility states (stationary versus moving). This inadequacy leads to suboptimal resource allocation, inefficient scheduling, and ultimately, a degraded Quality of Experience (QoE) for VR users.

Specifically, existing approaches often overlook two critical aspects that are vital for effective VR traffic management: (1) the rich information contained in the frequency domain of traffic patterns, which can reveal underlying periodicities or distinct spectral signatures tied to user actions, and (2) the critical importance of user state awareness—recognizing whether a VR user is stationary or actively moving—as these states dramatically alter traffic generation patterns. Moreover, even when accurate predictions of future traffic load are achievable, there remains a significant challenge in effectively integrating these window-level forecasts into real-time, millisecond-granularity network scheduling mechanisms, such as those found in modern 5G/6G radio access networks. This temporal mismatch can limit the practical utility of prediction, leading to reactive rather than proactive resource management and hindering the full potential of QoE enhancement.

To address these challenges, this thesis investigates novel methodologies for VR network traffic analysis and prediction by explicitly incorporating user state awareness, leveraging multi-domain features, and developing an end-to-end framework for prediction-aided resource scheduling. This research is driven by the following core questions:

- **RQ1: How do VR user mobility states (stationary vs. moving) manifest in network traffic patterns, and what are the specific characteristics (e.g., in packet size distributions, inter-arrival times, and spectral content) that differentiate them?**
  This question aims to establish a foundational understanding of the impact of user behavior on network traffic, moving beyond aggregated traffic statistics to analyze state-specific traffic signatures for both uplink and downlink flows.

- **RQ2: Can hybrid time-frequency features, particularly multi-scale spectral characteristics, significantly improve the accuracy and robustness of VR user state classification compared to traditional time-domain features, and which specific features are most discriminative?**
  This question probes the hypothesis that analyzing traffic in both time and frequency domains can unlock new insights for user state identification, aiming to achieve a high-fidelity classification crucial for state-aware network optimization.

- **RQ3: What are the practical limits and optimal horizons for short-term VR traffic load prediction using advanced machine learning models, and how can robust predictions of aggregated metrics (e.g., average packet size) be reliably obtained at horizons suitable for modern network scheduling mechanisms?**
  This question explores the feasibility of predictive traffic management for VR, assessing the trade-offs between prediction granularity and stability, and identifying the most effective deep learning models and prediction horizons that align with the operational cycles of network mechanisms like Semi-Persistent Scheduling (SPS), which typically operate with actuation periods of approximately 150–200 ms.

- **RQ4: How can accurate, window-level VR traffic forecasts be effectively integrated into a hybrid Semi-Persistent Scheduling (SPS) and Proportional Fair (PF) framework to proactively manage network resources, enhance delay fairness, and improve end-to-end VR Quality of Experience (QoE) in multi-user environments?**
  This question addresses the critical challenge of bridging the temporal gap between traffic prediction and real-time scheduling, seeking to develop and validate an end-to-end scheduling framework that leverages predictive intelligence to optimize resource allocation and improve QoE for latency-sensitive VR applications.

These research questions are revisited explicitly in Chapter 8, Section 8.2, where the empirical evidence from Chapters 4–7 is mapped back to each question and the extent of each answer is discussed.

1.3 Summary of Contributions

This thesis makes several significant contributions to the field of VR network traffic analysis and predictive resource management, building upon and extending preliminary findings. These contributions collectively pave the way for more intelligent and adaptive network solutions for immersive applications:

1. **A Comprehensive, State-Annotated VR Traffic Dataset:** This research introduces a collected, state-annotated, and publicly released VR network traffic dataset. The released dataset and reproduction code are available through the permanent commit-pinned companion repository link https://github.com/bozliu/vr-traffic-prediction-sps/tree/36a2d4d73c80c658d3d9174dcc1e9a08e9fc3922/data/raw, while the custom Unity VR application used to generate the traces is available at https://github.com/bozliu/vr-traffic-vr-game/tree/893245e502fcaed43bd9104fac853acd8178d0d5. Unlike existing datasets, this resource explicitly captures and labels user mobility states (stationary vs. moving) across both server-to-client (downlink) and client-to-server (uplink) communication flows. The dataset, generated from a custom-built Unity VR application, provides fine-grained packet-level information fused with application-layer context, offering a unique and invaluable resource for future research into VR traffic dynamics, user behavior modeling, and the development of state-aware networking solutions.

2. **Demonstrated Efficacy of Hybrid Time-Frequency Features for VR User State Classification:** This thesis rigorously demonstrates that the incorporation of multi-scale spectral features (across 0.5s, 1.0s, and 2.0s windows), together with time-domain features and a flow-direction bit, substantially improves VR user state classification over time-only baselines. On our test set, accuracy increases from approximately 69% (time-only) to approximately 93.7% (hybrid time+frequency+direction). Feature-importance analysis shows that frequency location and spectral-shape descriptors (Nyquist-normalized peak positions, spectral centroid, flatness) and flow direction carry the most discriminative power, with amplitude magnitudes contributing moderately.

3. **Identification of Practical Prediction Horizons for Short-Term VR Traffic Forecasting:** We evaluate Random Forest, LSTM, GRU, and Transformer models on a real VR traffic dataset, forecasting average packet size under a unified-sample, unconditional labeling protocol across 20–200 ms horizons. Prediction is shown to be unreliable below 80 ms ($R^2 < 0.4$), improves significantly at 150 ms ($R^2 \approx 0.61$), and is robust at 200 ms ($R^2 \approx 0.85$–0.86). These findings align with typical scheduler actuation cycles and strongly support proactive resource allocation policies at approximately 150–200 ms horizons.

4. **Development and Validation of an End-to-End Prediction-Aided Hybrid SPS/PF Scheduling Framework:** This research develops and validates a novel end-to-end framework that seamlessly integrates 200ms VR traffic predictions into a 1ms MAC layer scheduler. By employing a hybrid Semi-Persistent Scheduling (SPS) and Proportional Fair (PF) strategy, the framework demonstrates significant improvements in delay fairness (by 3.0-3.5%) and reductions in tail latencies for VR users, all while maintaining full network throughput. The framework's robustness is verified across various stress scenarios, exhibiting particular efficacy under bursty and highly multiplexed traffic conditions. Achieving a high Jaccard overlap of approximately 0.91 between predictive SPS and an oracle scheduler for user selection, this contribution provides strong evidence for the practical utility of predictive intelligence in strategic wireless resource management for immersive VR experiences.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

- **Chapter 2: Background and Theoretical Foundations** introduces the essential concepts and theories underpinning this research, including fundamental aspects of VR networking, time series analysis techniques (with a focus on both time and frequency domains), core machine learning and deep learning models (RNNs, LSTMs, GRUs, Transformers), and relevant network resource management paradigms like Semi-Persistent Scheduling.

- **Chapter 3: Literature Review** presents a comprehensive survey of existing research in VR and interactive gaming traffic analysis, network traffic classification methods, and traffic forecasting techniques. It critically evaluates previous approaches, highlights the current state of the art, and systematically identifies

the research gaps that this thesis aims to address, thereby justifying the novelty and significance of the contributions.

- **Chapter 4: VR Traffic Dataset and Experimental Methodology** details the design and implementation of the custom-built VR application used for data collection, outlines the meticulous data capture and labeling protocols for different user states, describes the preprocessing and feature engineering pipeline for both time-domain and frequency-domain features, and elaborates on the experimental setup and evaluation metrics used throughout the study.

- **Chapter 5: VR User State Classification using Hybrid Time-Frequency Features** delves into the core classification task. This chapter elaborates on the methodology for leveraging combined time-domain and frequency-domain features to accurately classify VR user mobility states. It presents detailed experimental results, including performance metrics, confusion matrices, and an in-depth feature importance analysis, demonstrating the superior discriminative power of multi-scale spectral position/shape features and the flow-direction bit, with amplitudes contributing additional signal.

- **Chapter 6: Short-Term VR Traffic Forecasting for Proactive Resource Allocation** evaluates Random Forest, LSTM, GRU, and Transformer models on the real VR traffic dataset, forecasting average packet size under a unified-sample, unconditional labeling protocol across 20–200 ms horizons, and identifies approximately 150–200 ms as the most reliable range for proactive scheduling decisions.

- **Chapter 7: End-to-End Prediction-Aided Semi-Persistent Scheduling (SPS) for VR** introduces a novel framework for integrating 200ms VR traffic forecasts into a 1ms MAC layer scheduler. This chapter details the multi-user VR traffic workload synthesis, the hybrid SPS/Proportional Fair scheduling mechanism, and presents comprehensive simulation results demonstrating significant improvements in delay fairness and reduced tail latencies for VR users, validated against an oracle scheduler under various stress scenarios.

- **Chapter 8: Conclusion and Future Work** summarizes the key findings and contributions of this thesis. It discusses the limitations of the current study and outlines promising directions for future research, including the expansion of datasets, exploration of online learning strategies, and further integration of the proposed framework into real-world 5G/6G testbeds for practical validation of QoE improvements.

Chapter 2

Background and Theoretical Foundations

2.1 Virtual Reality Networking

The burgeoning field of Virtual Reality (VR) represents a paradigm shift in human-computer interaction, offering deeply immersive and interactive experiences that aim to blur the lines between physical and digital realities. Unlike traditional media consumption or even conventional online gaming, VR applications impose uniquely stringent requirements on underlying network infrastructures, fundamentally altering the traditional understanding of Quality of Service (QoS) and Quality of Experience (QoE) [38, 39]. This section provides a foundational understanding of VR networking, detailing its critical QoS and QoE demands, exploring prevalent architectural models, and characterizing the unique patterns of VR network traffic, which serves as the core motivation for this thesis. The quality and comfort of a VR experience are profoundly sensitive to network performance, primarily characterized by three critical metrics: latency, throughput, and jitter. Any deficiency in these areas directly translates into a degraded user experience, often leading to physiological discomfort.

2.1.1 Core Concepts: Latency, Throughput, and Jitter

Latency (End-to-End Delay)

In VR, latency is paramount, particularly the "motion-to-photon" (MTP) latency. MTP refers to the total time delay from a user's physical movement (e.g., head rotation, hand gesture) to the corresponding visual update being displayed on the VR headset's screen [40]. For an immersive and comfortable VR experience, MTP latency must be exceptionally low, typically cited as below 20 milliseconds (ms) [41]. Exceeding this threshold, as empirical studies have shown, can result in mean latencies ranging from 21 to 42 ms at the start of a sudden movement, which significantly impacts sensorimotor performance [40]. This delay contributes to severe cybersickness (motion sickness in VR), disorientation, and a breakdown of "presence" – the feeling of truly being in the virtual environment. The total MTP budget is a sum of various components, including sensor acquisition, rendering, encoding, transmission, decoding, and display delays [42]. Any network-induced delay, in both upstream (client-to-server) and downstream (server-to-client) communication, directly consumes this tight MTP budget, highlighting the critical need for real-time, ultra-low-delay packet delivery.

Throughput (Bandwidth)

VR applications, especially those delivering high-resolution visuals and complex virtual worlds, are highly bandwidth-intensive. The raw data rate required for an online immersive experience can vastly exceed current network capabilities, ranging from 5 Gbps to 60 Gbps [38]. Even with advanced compression techniques, high frame rates (e.g., 90 Hz, 120 Hz) combined with increasing display resolutions (e.g., 3664 x 1920p for typical headsets [43]) necessitate substantial throughput. For instance, strong-interaction Cloud VR services may require sustained bandwidths from 80 Mbit/s (FEP phase) up to 1,540 Mbit/s (UEP phase) [41]. This is orders of magnitude higher than traditional video streaming, where static content allows for more aggressive pre-buffering and less dynamic bit-rate adjustments. Insufficient throughput leads to visual artifacts, frame drops, reduced image quality, and a general degradation of the immersive experience, as the network struggles to deliver the continuous, high-fidelity visual stream.

Jitter (Packet Delay Variation)

Jitter refers to the variation in the arrival time of successive data packets. While an average low latency is desirable, consistency in packet arrival times is equally, if not more, important for VR. High jitter can cause noticeable stuttering, judder, and unpredictable behavior in the virtual environment, disrupting the illusion of presence and causing discomfort [41]. For instance, inconsistent delivery of pose updates or video frames can lead to a desynchronization between the user's perception and the virtual world's state, resulting in a jarring and unpleasant experience. The ITU-T recommends jitter targets as low as 7 ms for Ultra-High Experience Phase (UEP) VR [41]. Maintaining stable, low-jitter packet flows is therefore essential for smooth and consistent VR performance, demanding sophisticated network scheduling and congestion control mechanisms.

2.1.2 Impact of Network Performance on User Experience (QoE)

The aforementioned QoS metrics directly translate into the user's Quality of Experience (QoE). QoE for VR extends beyond mere technical performance; it encompasses the subjective perception of immersion, comfort, and interactivity.

- **Immersion and Presence:** Low latency, high throughput, and minimal jitter collectively foster a strong sense of immersion and presence. When the virtual world responds instantly and fluidly to user actions, users feel more connected and 'present' within the digital environment. Network impairments, conversely, shatter this illusion, reminding the user of the virtual barrier.

- **Motion Sickness (Cybersickness):** This is perhaps the most critical QoE concern in VR, hindering broader adoption and usability. It arises from sensory conflicts, primarily when visual information (what the user sees) does not align with vestibular information (what the user feels from their inner ear regarding motion) [44]. High MTP latency, low frame rates, or significant jitter can create this mismatch, causing symptoms ranging from nausea and dizziness to disorientation. Notably, not only uniform delays but also occasional latency spikes

can provoke cybersickness, emphasizing the need for highly stable network performance [44]. Effective network design and predictive resource management are paramount in mitigating these uncomfortable physiological responses.

- **Responsiveness and Interactivity:** VR applications, unlike passive video streaming, are inherently interactive. Every user action (head movement, hand gestures, voice commands) must elicit an immediate and precise response in the virtual world. The network's ability to facilitate rapid, bidirectional communication with minimal delay and variability is fundamental to this responsiveness, directly impacting the perceived interactivity and control the user has over their experience.

- **Visual Fidelity and Comfort:** Sufficient throughput ensures the delivery of high-resolution, high-frame-rate visual content without compression artifacts or pixelation, contributing to visual comfort and realism. Network-induced visual degradation can strain the eyes and detract from the overall quality, leading to user fatigue and dissatisfaction.

2.1.3 Network Architectures for VR

The way VR content is rendered and delivered significantly influences network traffic patterns and requirements. Broadly, VR systems can be categorized into three architectural paradigms, each with distinct network implications:

Stand-alone VR (On-device Rendering)

In this model (e.g., Meta Quest series), all rendering and processing occur directly on the headset. The device is self-contained, capable of running VR applications locally. Network traffic in this scenario is primarily limited to downloading content, software updates, or for multiplayer interactions, synchronizing game state with a remote server. The primary limitation of this architecture is its dependence on the headset's embedded computational power, battery life, and form factor, which often necessitates a trade-off in visual fidelity compared to PC-tethered or cloud-rendered experiences.

Tethered PC-VR (Local PC Rendering)

This architecture relies on a powerful local PC to perform the intensive rendering and processing tasks. Video and tracking data are then transmitted to the headset via a physical cable (e.g., Valve Index connected to a gaming PC). While this setup allows for higher visual fidelity and more complex virtual environments due to the PC's superior processing capabilities, network traffic (beyond local bus communication) remains low. However, the physical tether restricts user mobility, introduces cable management challenges, and compromises physical comfort, making it less suitable for dynamic, room-scale VR experiences.

Cloud/Edge VR (Remote Rendering)

This emerging paradigm offloads the intensive rendering and processing tasks to powerful servers located in the cloud or at the network edge (Multi-access Edge Computing - MEC) [39, 42]. In this model, the VR headset acts as a thin client, transmitting user

input (e.g., head pose, controller movements) to the remote server. The server then renders the corresponding frames and streams the encoded video back to the headset. This architecture offers significant advantages, including enabling lighter, more comfortable headsets and offloading computational and battery requirements from the device. It also facilitates the creation of shared resources and large-scale, persistent virtual worlds. However, it places the most stringent and complex demands on the network, requiring extremely low latency (e.g., total MTP <20ms, with a streaming budget <10ms for encoding, transmission, and decoding) and ultra-high throughput (e.g., 500 Mbps to 1.5 Gbps for strong-interaction VR video streams, potentially reaching tens of Gbps for uncompressed high-resolution content) for continuous, high-quality video delivery. It also necessitates symmetrical or bidirectional high throughput to handle both high-bandwidth downlink video and critical low-latency uplink user input.

The VR network architecture employed in this study for data collection aligns with a Networked Stand-alone VR model. In this setup, the Meta Quest headset functions as a self-contained client device, performing its own graphics rendering. It establishes network communication with a dedicated server, which is primarily responsible for managing game logic, synchronizing the virtual world state across participants, and facilitating network communication via Unity Netcode for Objects. This architecture, while leveraging the portability of stand-alone VR, introduces network traffic predominantly related to interactive state updates and world synchronization rather than high-bandwidth remote video streaming. This distinction is crucial, as it implies that the VR traffic characteristics observed in this thesis are directly tied to the fine-grained dynamics of user interaction and mobility within a locally rendered virtual environment.

2.1.4 Characteristics of VR Network Traffic

Unlike traditional video streaming, which typically exhibits stable Constant Bit Rate (CBR) or Variable Bit Rate (VBR) patterns dominated by predictable downlink flows, or conventional online gaming, characterized by smaller, event-driven packets, VR network traffic presents unique complexities shaped by the interactive, immersive, and high-fidelity nature of such applications [43, 45, 46].

Bidirectional and Highly Asymmetrical Flows

VR traffic is inherently bidirectional yet highly asymmetrical. In the uplink direction (client-to-server), data primarily consists of low-latency control messages, including head tracking data (pose and orientation updates) and controller inputs. These uplink packets are typically very small, averaging approximately 175 bytes, and are transmitted at exceptionally high frequencies, typically between 200 Hz and 1000 Hz, with 240 Hz commonly observed in practice [43, 45]. Although the uplink data rate is comparatively low—often around 100–150 kbps [39]—the low-latency, high-frequency nature of this flow is critical for responsiveness and immersive interaction.

In contrast, the downlink direction (server-to-client) overwhelmingly dominates the traffic load, often accounting for approximately 99.3% of the total traffic volume and 95.8% of the total number of packets [43, 45]. The downlink primarily delivers high-bandwidth video frames, typically averaging around 1243 bytes per packet [43]. Additionally, this stream includes essential data such as game-state updates (e.g., object positions, NPC actions), spatial audio, and haptic feedback, contributing further to

traffic variability and burstiness. In scenarios such as multiplayer VR, downlink volumes often exceed uplink traffic by a factor of ten or more [45].

While our study focuses on VR, it is important to highlight that mobile AR applications may exhibit markedly different uplink behavior. In AR use cases involving edge/cloud offloading—such as real-time scene understanding, SLAM (Simultaneous Localization and Mapping), or remote expert assistance—devices often continuously upload camera frames for processing, resulting in significantly higher and sustained uplink rates [47, 48]. By contrast, cloud-rendered AR scenarios tend to resemble VR traffic patterns, with the uplink dominated by low-latency pose and control data and the downlink carrying most of the bandwidth load [49]. We therefore restrict our quantitative claims about downlink dominance to VR and cite AR here only for contrast.

Burstiness and Non-Stationarity

Another distinctive feature of VR traffic is its inherent burstiness and non-stationarity, contrasting sharply with the smoother and more predictable nature of traditional streaming traffic. These dynamic variations arise primarily from three factors: user interaction patterns, scene complexity, and game-specific logic and events. User interactions, such as sudden head movements, rapid hand gestures, or active navigation, generate bursts of highly frequent pose updates [46, 50]. For instance, research indicates a linear correlation between head angular velocity and video frame size, with average frame sizes increasing by approximately 7.3% to 12.6% during periods of active movement [50]. Additionally, scene complexity strongly influences video frame sizes; studies have reported that average video frame sizes can vary significantly—from 77 KB at 90 fps up to 201 KB at 30 fps—depending on content intricacy and encoding strategy [51]. Furthermore, game logic events such as character spawning, animations, or environmental changes regularly produce traffic spikes, further enhancing burstiness.

Notably, even video streams used in interactive VR scenarios often exhibit non-CBR behavior. For example, WebRTC-based streaming—common in VR applications—frequently employs internal pacing algorithms that transmit packets in frame-aligned bursts approximately every 5.56 ms, irrespective of the target frame rate. This pacing mechanism leads to distinct periodic burst patterns, causing single video frames to be split and transmitted across multiple batches, averaging approximately 2.1 batches per frame at 90 fps and up to 4.56 batches per frame at 30 fps [51]. The intrinsic periodicity of these burst patterns, often aligned with the rendering loops of the game engine (e.g., head orientation updates approximately every 4.6 ms or $\sim$216 Hz and frame intervals around 13.8 ms or $\sim$72 Hz [50]), strongly motivates exploring frequency-domain features in VR traffic analysis. These frequency-domain characteristics enable the capture of underlying temporal patterns that purely time-domain metrics may overlook.

Dependency on User Mobility State

Furthermore, VR network traffic displays a significant dependency on user mobility states, forming the central motivation of this thesis. Specifically, distinct and identifiable traffic signatures arise from differences in user mobility states, such as stationary versus active states. When users are relatively still (stationary state), frame sizes are influenced primarily by content complexity rather than movement; thus, no significant

correlation between user movement and traffic metrics emerges [50]. During stationary periods, downlink traffic predominantly consists of smaller object update packets, driven by the application's focus on high-fidelity texture updates or idle animations for objects within the user's immediate view.

Conversely, active user movement—particularly when angular velocity surpasses specific thresholds (e.g., 15°/s)—induces a marked increase in frequent, smaller packets corresponding to head and controller pose updates. Such increases are critical for maintaining accurate real-time tracking and preventing motion sickness [50]. This active state produces distinct burst patterns and unique inter-arrival time distributions, often characterized by pronounced periodic components directly tied to rendering and update loops. Empirical evidence shows that the Pearson linear correlation coefficient (PLCC) between frame size and angular velocity can reach values as high as 0.969 during active movements, whereas it remains negligible during stationary periods [50].

Understanding and accurately classifying these mobility-dependent traffic signatures is fundamental to the contributions of this thesis. By leveraging these traffic signatures through state-aware classification and prediction approaches, particularly those integrating frequency-domain insights alongside traditional time-domain features, the thesis aims to significantly enhance network resource management strategies for VR applications. This granular and adaptive allocation approach moves beyond conventional generic traffic modeling methods, allowing network management policies to directly respond to the real-time demands induced by user behavior.

Comprehensively characterizing these multifaceted VR network traffic patterns is thus essential for developing robust traffic analysis, accurate classification, and reliable prediction models. Such models will form the cornerstone of optimized resource allocation, proactive congestion management, and ultimately, the delivery of seamless and comfortable QoE in next-generation immersive VR networks.

2.2 Fundamentals of Time-Domain Features and Statistical Models

Network traffic can be fundamentally conceptualized as a complex time series, characterized by measurements such as packet sizes, inter-arrival times, and throughput values that evolve over time. Accurate analysis and prediction of these temporal features are critical for effective network management, especially in demanding environments like interactive Virtual Reality (VR) applications. This section introduces essential theoretical concepts in network traffic analysis, from both time-domain and frequency-domain perspectives, and discusses key statistical properties and traffic modeling techniques relevant to this research.

2.2.1 Basic Time-Domain Features and Metrics

Numerous studies in the literature have identified *packet size*, *inter-arrival time (IAT)*, and statistical variability measures such as the *coefficient of variation (CoV)* as fundamental time-domain features for network traffic analysis and classification. For instance, Nguyen and Armitage [52] and Moore and Zuev [53] demonstrated that these features—derived solely from packet headers and timing—enable highly accurate classification, even in encrypted or port-obfuscated traffic. Moreover, temporal metrics like

CoV and moving averages have been widely used to capture burstiness and long-range dependence in network flows, as shown by Leland et al. [54] and Paxson and Floyd [55]. In the context of real-time applications such as Virtual Reality (VR), these features are especially valuable due to their low computation overhead and strong correlation with user-perceived Quality of Experience (QoE). Recent studies [22, 56] specifically highlight that time-domain statistics—including sliding-window averages and variability measures—play a key role in short-term traffic prediction and adaptive resource allocation in immersive environments. These works collectively confirm the relevance and widespread use of the selected time-domain metrics in both general-purpose and VR-specific traffic modeling.

1. **Packet Sizes:**
   Packet sizes represent the payload or total size (including headers) of individual packets. Let S_i denote the size of the i^{th} packet. The key statistical measures the mean and variance are defined as:

$$\bar{S} = \frac{1}{N}\sum_{i=1}^{N} S_i \quad \text{(mean)}, \qquad \sigma_S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(S_i - \bar{S})^2 \quad \text{(variance)} \tag{2.1}$$

   In VR applications, distinct packet size distributions arise due to different types of traffic: small state updates versus larger multimedia or asset transmissions.

2. **Inter-Arrival Times:**
   The inter-arrival time T_i is the temporal gap between the arrival of packet i and $i-1$. The mean and variance are given by:

$$\bar{T} = \frac{1}{N}\sum_{i=1}^{N} T_i, \quad \sigma_T^2 = \frac{1}{N-1}\sum_{i=1}^{N}(T_i - \bar{T})^2 \tag{2.2}$$

   Short and consistent inter-arrival times indicate stable and sustained traffic, while irregular timings reflect bursty patterns, which are typical in interactive VR scenarios.

3. **Moving Averages:**
   Moving averages help to smooth out short-term fluctuations in traffic metrics and highlight long-term trends. For a time series X_t, the moving average over a sliding window of size k is:

$$MA(t,k) = \frac{1}{k}\sum_{j=t-k+1}^{t} X_j \tag{2.3}$$

   This approach is valuable for resource provisioning strategies that rely on aggregate network demand rather than individual packet behavior.

4. **Coefficient of Variation (CoV):**
   The CoV is a dimensionless measure of variability, calculated as the ratio of the standard deviation σ to the mean μ:

$$\text{CoV} = \frac{\sigma}{\mu} \tag{2.4}$$

   In network traffic analysis, a high CoV value signifies significant variability, indicating a need for adaptive and dynamic network resource allocation to meet fluctuating demands.

These time-domain features form the foundational statistical toolkit for modeling and understanding the behavior of VR traffic, and are critical for the design of predictive algorithms that aim to optimize Quality of Experience (QoE).

2.2.2 Advanced Statistical Properties of Network Traffic

Real-world network traffic, especially from contemporary interactive applications, exhibits complex statistical properties that extend beyond simple statistical descriptors such as the mean and variance. These nuanced properties are often overlooked by traditional modeling assumptions but are crucial for accurate traffic characterization and effective network management.

Self-Similarity

A fundamental property observed in modern network traffic is *self-similarity*, which describes statistical behaviors that persist across multiple time scales—ranging from milliseconds to minutes and hours. This fractal-like structure contradicts classical assumptions that network traffic becomes smoother when aggregated over longer intervals, implying instead the absence of a characteristic burst length [57].

Self-similarity is quantitatively characterized by the *Hurst exponent* (H), where values in the range $0.5 < H < 1$ indicate both self-similarity and *Long-Range Dependence (LRD)*. In LRD processes, the autocorrelation function decays hyperbolically rather than exponentially, meaning that traffic behavior at distant times continues to influence current observations. Empirical studies on Ethernet LAN traffic have shown typical Hurst exponent values between 0.75 and 0.95, with higher values indicating stronger self-similarity, particularly during periods of moderate to heavy traffic activity [57, pp. 6–8].

The implications of self-similarity and LRD for network design are significant. Standard queueing models based on short-range dependent processes tend to underestimate queue lengths and overpredict system performance. Furthermore, buffer-based mechanisms alone are often inadequate in controlling packet loss and delay under such bursty traffic conditions.

Heavy-Tailed Distributions

Closely related to self-similarity is the presence of *heavy-tailed distributions* in network traffic characteristics. A distribution is considered heavy-tailed if its tail decays more slowly than an exponential distribution, implying a substantial probability of extremely large values. This property has been observed in various traffic parameters such as packet inter-arrival times and data burst sizes for certain applications.

Paxson and Floyd [58] demonstrated that TELNET packet inter-arrival times conform to a Pareto distribution, a canonical heavy-tailed distribution. Relying on exponential models in such cases leads to significant underestimation of traffic burstiness [58, p. 229]. Likewise, they reported that FTP data bursts exhibit extreme heavy-tailed behavior: in one instance, merely 0.5% of the largest bursts accounted for 54% of the total traffic volume [58, p. 233, Appendix B].

The existence of heavy-tailed distributions is a major underlying cause of bursty and unpredictable traffic behavior. Combined with self-similarity, these characteristics

make traffic modeling and prediction particularly challenging when using conventional short-range dependent or light-tailed statistical models.

2.2.3 Classical Time-Domain Models for Forecasting

Autoregressive (AR), Moving Average (MA), and Autoregressive Integrated Moving Average (ARIMA) Models

Historically, various statistical models have been developed for forecasting time series data by leveraging temporal dependencies. Among the most foundational are Autoregressive (AR), Moving Average (MA), and Autoregressive Integrated Moving Average (ARIMA) models. An Autoregressive model of order p, denoted AR(p), forecasts future values using a linear combination of past observations. The AR(p) model is expressed as:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t \tag{2.5}$$

where X_t is the observed time series, ϕ_i are the autoregressive coefficients, c is a constant, and ε_t is a white noise error term.

The Moving Average model of order q, MA(q), models the current value based on a linear combination of past forecast errors. It is defined as:

$$X_t = \mu + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t \tag{2.6}$$

where θ_j are the moving average coefficients, μ is the mean, and ε_t denotes the error term.

ARIMA(p, d, q) models extend AR and MA by including a differencing component d to handle non-stationary time series. It can be expressed compactly using the lag operator L as:

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d X_t = c + \left(1 + \sum_{j=1}^{q} \theta_j L^j\right) \varepsilon_t \tag{2.7}$$

where $LX_t = X_{t-1}$, and $(1 - L)^d$ denotes the d-order differencing. While these models are effective for capturing linear trends and stationary structures, their performance can degrade in the presence of non-linearities and the high variability found in interactive VR network traffic [59].

Holt-Winters Forecasting

The Holt-Winters method belongs to the family of exponential smoothing techniques, designed to capture level, trend, and seasonality in time series. The additive Holt-Winters model consists of three recursive equations:

$$l_t = \alpha(X_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.8}$$
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{2.9}$$
$$s_t = \gamma(X_t - l_t) + (1-\gamma)s_{t-m} \tag{2.10}$$
$$\hat{X}_{t+h} = l_t + hb_t + s_{t+h-m(k+1)} \tag{2.11}$$

where: l_t is the level component, b_t is the trend component, s_t is the seasonal component, m is the seasonal period, α, β, and γ are smoothing parameters.

Although effective for structured and seasonal data, the method's adaptability is limited in environments with rapid fluctuations or event-driven behavior, as commonly observed in VR network traffic [60, 61].

2.3 Fundamentals of Frequency and Time-Frequency Analysis

While time-domain analysis provides insights into sequential traffic patterns, frequency-domain analysis reveals the underlying periodic structures that may be obscured in raw time series data. By transforming signals into the frequency domain, periodicities and oscillatory behaviors become more apparent. This is particularly valuable in the context of virtual reality (VR) traffic, where such periodic characteristics may correspond to rendering frame rates, sensor update intervals, or repetitive user actions. Additionally, for non-stationary signals—which are common in immersive environments—time-frequency analysis provides temporal localization of frequency content, offering enhanced interpretability of evolving dynamics.

2.3.1 Frequency Domain Analysis through Fourier Transform

A foundational technique in frequency-domain analysis is the *Discrete Fourier Transform (DFT)*, which converts a discrete-time sequence into its frequency-domain representation by expressing it as a weighted sum of complex exponentials. The DFT for a sequence $x[n]$ of length N is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1 \tag{2.12}$$

Here, $X[k]$ denotes the frequency-domain coefficient at index k, and j is the imaginary unit. The direct computation of the DFT has a computational complexity of $\mathcal{O}(N^2)$, which becomes prohibitive for large-scale data. To address this, the *Fast Fourier Transform (FFT)* algorithm, introduced by Cooley and Tukey [62], reduces this complexity to $\mathcal{O}(N \log N)$, making it practical for real-time or large-volume VR network traffic analysis.

The FFT output is typically analyzed via several spectral descriptors. The *amplitude spectrum*, given by $|X[k]|$, indicates the magnitude of each frequency component. Peaks in the amplitude spectrum correspond to dominant periodic patterns—such as consistent frame updates or recurrent interaction events—and are therefore highly informative for tasks like traffic classification [63].

In parallel, the *phase spectrum*, defined as the angle $\arg(X[k])$, reveals the phase offset of each frequency component. While the phase is essential for perfect signal reconstruction, it is less frequently used as a feature in classification or prediction models due to its complex interpretability [63].

To analyze how signal power is distributed across frequency components, the *Power Spectral Density (PSD)* is employed. The PSD is typically calculated as the squared magnitude of the FFT coefficients, normalized over the signal length. A widely used estimation method is Welch's method [64], which involves segmenting the signal, applying windowing, and averaging periodograms to reduce variance in the spectral estimate.

Furthermore, the *spectral entropy* quantifies the randomness or predictability of a signal's frequency distribution. Derived from the normalized PSD $p(f)$, it is defined as:

$$H_s = -\sum_f p(f) \log_2 p(f) \tag{2.13}$$

A high spectral entropy value indicates that energy is spread broadly across frequencies, suggesting a more chaotic or unpredictable signal. Conversely, a low spectral entropy implies concentration in specific frequency bands, denoting a structured or periodic signal. In VR traffic analysis, spectral entropy can differentiate between highly structured traffic—such as stable frame rate streams—and more irregular flows arising from dynamic user behavior or scene complexity [65].

2.3.2 Windowing Functions in Spectral Analysis

When performing Fourier transforms, particularly the Fast Fourier Transform (FFT), on finite-length signals or short segments extracted from a longer continuous signal, the implicit assumption is periodic extension of the signal beyond the analyzed segment. This assumption can lead to spectral leakage—an artifact where energy from a particular frequency component spreads or "leaks" into neighboring frequency bins, thus distorting the true spectral characteristics of the signal [66, 67]. To mitigate this phenomenon, windowing functions are applied to the signal segments prior to spectral analysis. Mathematically, windowing modifies the original signal $x[n]$ by multiplying it pointwise with a windowing function $w[n]$:

$$x_w[n] = x[n] \cdot w[n], \quad n = 0, 1, \ldots, N-1 \tag{2.14}$$

Here, $x_w[n]$ denotes the windowed signal, and N is the length of the analysis window. The choice of the windowing function $w[n]$ critically influences spectral leakage and frequency resolution, as it controls how the signal is tapered at segment boundaries [66, 67, 68]. Several common windowing functions widely used in spectral analysis include the Rectangular, Hamming, Hanning, and Blackman windows, each with distinct characteristics and applications.

Rectangular Window

The Rectangular Window is defined mathematically as:

$$w[n] = \begin{cases} 1 & 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

This window is the simplest, applying equal weighting to all samples within the window interval. However, due to the abrupt truncation at its boundaries, the rectangular window introduces significant discontinuities, resulting in pronounced spectral leakage characterized by prominent sidelobes in the frequency domain [66, 67].

Hamming Window

The Hamming Window addresses these limitations by introducing a cosine-tapering to smoothly attenuate the endpoints. It is mathematically defined as:

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \leq n \leq N-1 \tag{2.16}$$

The Hamming window effectively suppresses sidelobe amplitudes, significantly reducing spectral leakage compared to the rectangular window. This improvement comes at the expense of a slightly wider main lobe, thereby reducing the frequency resolution somewhat, though still offering an advantageous compromise for many applications [66].

Hanning (Hann) Window

Similarly, the Hanning (Hann) Window follows a cosine-based tapering but uses different coefficients:

$$w[n] = 0.5 \left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right], \quad 0 \leq n \leq N-1 \tag{2.17}$$

Like the Hamming window, the Hanning window provides reduced sidelobe levels relative to the rectangular window, effectively mitigating spectral leakage while also exhibiting a broader main lobe. It is widely used due to its simplicity and good balance between leakage reduction and resolution [66, 67].

Blackman Window

The Blackman Window further enhances sidelobe suppression by employing a three-term cosine expression:

$$w[n] = 0.42 - 0.5 \cos\left(\frac{2\pi n}{N-1}\right) + 0.08 \cos\left(\frac{4\pi n}{N-1}\right), \quad 0 \leq n \leq N-1 \tag{2.18}$$

The Blackman window achieves superior sidelobe suppression compared to both Hamming and Hanning windows. However, this benefit comes with an increased width of the main lobe, resulting in reduced frequency resolution. Consequently, the Blackman window is particularly beneficial in scenarios where accurate amplitude measurement of dominant frequency components is prioritized over the separation of closely spaced spectral peaks [66].

In spectral analysis, selecting an appropriate window function entails a careful trade-off between frequency resolution and sidelobe suppression. The width of the main lobe primarily determines frequency resolution—the ability to distinguish closely spaced frequencies—whereas the amplitude of sidelobes controls spectral leakage, crucial for accurately estimating the amplitudes of dominant frequencies and reducing interference from strong adjacent spectral components. Thus, understanding these trade-offs enables more informed decisions when analyzing complex, time-varying signals, such as those encountered in Virtual Reality (VR) network traffic [67, 68].

2.3.3 Time-Frequency Analysis for Non-Stationary Signals

Network traffic typically exhibits non-stationary behavior, implying that its statistical properties, such as mean, variance, and frequency content, vary over time. Capturing these transient characteristics requires advanced signal-processing techniques that provide simultaneous time and frequency information, collectively known as *time-frequency analysis* techniques [67, 69].

Short-Time Fourier Transform (STFT)

One widely-used approach is the Short-Time Fourier Transform (STFT). This method involves applying the Fourier transform to short, overlapping segments (windows) of the signal. Mathematically, the STFT is defined as:

$$X(t, f) = \int_{-\infty}^{+\infty} x(\tau)\, w(\tau - t)\, e^{-j2\pi f \tau} d\tau \tag{2.19}$$

where $x(\tau)$ is the input signal, $w(\tau - t)$ is a windowing function localized in time, and $X(t, f)$ is the resulting time-frequency representation (spectrogram), displaying how the frequency content evolves over time [68, 69]. Despite its widespread applicability, STFT has inherent limitations, most notably the trade-off between time and frequency resolution, commonly referred to as the *Heisenberg-Gabor limit*. This limitation is directly affected by the choice of the window function and its length.

Wavelet Transforms

To address these limitations, Wavelet Transforms—including both the Continuous Wavelet Transform (CWT) and the Discrete Wavelet Transform (DWT)—have emerged as powerful tools for analyzing non-stationary signals. Unlike the fixed-width window used in STFT, wavelet transforms use scalable and translatable basis functions called *wavelets*, which provide a multi-resolution analysis of signals. The Continuous Wavelet Transform is defined as:

$$CWT(a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{+\infty} x(t)\, \psi^* \left(\frac{t - b}{a} \right) dt \tag{2.20}$$

where a denotes the scale parameter (inversely related to frequency), b is the translation parameter (time localization), and $\psi(t)$ is the *mother wavelet* [70]. This formulation provides finer temporal resolution for high-frequency components and better frequency resolution for low-frequency components. Consequently, wavelet transforms are highly effective for capturing both sudden changes (e.g., traffic bursts) and long-term periodic trends in network traffic [69, 70]. Their adaptability makes them particularly suitable for analyzing bursty, time-varying traffic patterns commonly observed in Virtual Reality (VR) applications.

2.4 Statistical Properties and Traffic Models

Beyond feature extraction, accurately modeling and predicting network traffic requires a clear understanding of the statistical processes underlying traffic generation. This section critically reviews key statistical models designed to capture the characteristics of network traffic, highlighting their assumptions, advantages, and limitations.

2.4.1 Traditional Traffic Source Models

Poisson Process

Traditional traffic source models historically employed simplifying assumptions. A widely utilized example is the *Poisson process*, which assumes packet arrivals are random, independent events occurring at a constant average rate (λ). This assumption enables mathematically straightforward analysis and can effectively approximate low-load scenarios or highly aggregated traffic streams. However, empirical research has consistently demonstrated significant deviations from Poisson assumptions in real-world network scenarios, especially with modern interactive applications such as virtual reality. Specifically, packet inter-arrival times typically exhibit burstiness, heavy-tailed distributions, and temporal correlation—characteristics incompatible with the independence assumption of a Poisson process. Paxson and Floyd notably illustrated these shortcomings in their influential study, highlighting the inadequacy of the Poisson model in capturing complex real-world traffic patterns [55].

On/Off Sources

Another classic modeling approach is the *On/Off source model*, which better captures the inherently bursty nature of many traffic sources. This model represents a traffic source as alternating between two distinct states: an "On" state, characterized by active data transmission at a specified or probabilistic rate (often assumed constant or Poisson-distributed), and an "Off" state, during which no data transmission occurs. The durations spent in each state are typically described using probability distributions—commonly exponential distributions, defined by parameters like the mean duration of "On" (μ_{on}) and "Off" (μ_{off}) periods. Such a representation allows for a basic yet practical abstraction of bursty traffic patterns, enabling the model to approximate periods of intense activity followed by quiescence. Nevertheless, although this approach successfully introduces burstiness, it remains a simplification that may not capture more complex, multi-scale variations present in contemporary VR/AR applications, whose traffic dynamics are significantly more intricate [71].

2.4.2 Advanced Traffic Models

Traditional network traffic models, such as Poisson and Markovian models, often fall short when capturing the complex dynamics inherent in virtual reality (VR) network traffic. To overcome these limitations, more sophisticated modeling approaches have been proposed. Two prominent classes of advanced traffic models particularly relevant to VR traffic analysis are the Markov-Modulated Poisson Process (MMPP) andLong-Range Dependence (LRD) models.

Markov-Modulated Poisson Processes (MMPP)

Markov-Modulated Poisson Processes (MMPP) represent a significant improvement over conventional Poisson models by introducing variability in the arrival rates through a hidden Markov chain. Specifically, the system transitions among a finite number of discrete states, each associated with its own Poisson arrival rate, enabling MMPPs to effectively model scenarios with distinct traffic regimes characterized by varying burstiness and intensity. This attribute makes MMPPs particularly well-suited to VR

applications, as traffic patterns in immersive environments frequently alternate between static phases (such as periods of user inactivity or passive observation) and dynamic phases (active user interactions involving rapid scene changes) [72]. Mathematically, an MMPP is defined by a hidden Markov chain with state space $S = \{1, 2, \ldots, m\}$, each state $i \in S$ having an associated Poisson arrival rate λ_i. The transition dynamics between states are governed by a state transition probability matrix, often represented as a generator matrix $Q = [q_{ij}]$, which describes the rates at which the system transitions from state i to state j.

Long-Range Dependence (LRD) Models

Long-Range Dependence (LRD) Models address the empirical observation of self-similarity in network traffic. Unlike traditional short-range dependence models, where the correlation between data points decays exponentially, LRD models capture persistent statistical dependencies across extended time horizons. This long-range correlation significantly affects network performance, leading to higher queueing delays, increased probability of buffer overflows, and the necessity for distinct resource allocation strategies. Prominent examples of LRD models include Fractional Autoregressive Integrated Moving Average (FARIMA) and Fractional Brownian Motion (FBM), both of which explicitly incorporate long-memory behavior into traffic modeling and prediction. Formally, LRD is characterized by a slowly decaying autocorrelation function, typically expressed as $r(k) \sim k^{-\beta}$ as $k \to \infty$, where $0 < \beta < 1$ is related to the Hurst parameter H, a measure commonly used to quantify self-similarity in traffic patterns [73]. Accurately modeling and forecasting traffic exhibiting LRD thus requires careful consideration of such long-term dependencies.

2.5 Selected Machine Learning and Deep Learning Models for Network Traffic Analysis and Prediction

Accurately characterizing and predicting the complex, non-stationary patterns of Virtual Reality (VR) network traffic is critical for maintaining stringent Quality of Service (QoS) and Quality of Experience (QoE). Traditional statistical methods often struggle to capture the underlying dynamics, necessitating advanced analytical techniques. This chapter delves into the foundational machine learning and deep learning methodologies specifically employed and evaluated in this thesis for VR traffic classification and short-term forecasting. We will elaborate on the rationale behind selecting Random Forest for user state classification and Long Short-Term Memory (LSTM), Gated Recurrent Units (GRUs), and Transformer encoder architectures for sequential traffic prediction, discussing their inherent strengths and limitations in the context of highly dynamic VR network environments.

2.5.1 Supervised Machine Learning Techniques: Random Forest

Supervised learning is a prominent paradigm in machine learning wherein algorithms learn a mapping function from an input feature vector X to an output label Y, using

labeled training examples. In classification scenarios, the output variable Y represents a discrete category. Formally, supervised classification problems can be represented as:

$$Y = f(X) \tag{2.21}$$

The goal is to estimate the function $f(\cdot)$ such that new, unseen inputs can be accurately classified into their respective categories. This thesis applies supervised classification to categorize VR network traffic into distinct user states, specifically "stationary" and "moving".

The fundamental process in supervised learning involves training a model using labeled datasets, enabling the algorithm to capture patterns correlating input features to output labels. Subsequently, the model's predictions are evaluated against actual labels, and the model parameters are iteratively adjusted to minimize classification error. Once the model has achieved optimal performance, it can be deployed to classify new data points without known labels. Various algorithms exist for supervised classification tasks, such as Support Vector Machines (SVMs), Logistic Regression, Decision Trees, and ensemble methods like Random Forests. Among these, Random Forests are employed in this research because of their robust performance, interpretability, and ability to handle high-dimensional, complex, and noisy data effectively.

Random Forest, originally introduced by Breiman [74], is an ensemble-based supervised learning method. It consists of constructing multiple decision trees, each trained on randomly selected subsets of data and feature variables. The classification decision of a Random Forest is derived from aggregating predictions of individual trees through a majority-voting scheme:

$$\hat{Y} = \text{mode}\{T_1(X), T_2(X), \ldots, T_m(X)\} \tag{2.22}$$

Here, $\hat{Y}$ represents the predicted class, $T_j(X)$ denotes the prediction from the j^{th} tree, and m is the total number of trees. Each tree is constructed using bootstrap aggregation (bagging), where subsets of the training dataset are sampled with replacement [75]. This procedure enhances generalization capability by reducing variance and mitigating overfitting compared to individual decision trees.

In addition to bagging, Random Forests employ random feature selection at each node split. A subset of the available features is randomly selected at each split point, and the best split is determined only from this subset. This additional randomness increases the diversity among trees and contributes to the overall robustness of the ensemble. Random Forest classifiers have several notable strengths that make them suitable for classifying VR network traffic. First, they effectively manage high-dimensional feature spaces, which is advantageous when working with hybrid time-frequency features. Second, they capture complex, non-linear feature interactions, which are typical in VR traffic patterns. Third, they implicitly perform feature selection and provide feature importance scores, which help assess the discriminative power of individual features.

One common metric used to evaluate splits in decision trees is the Gini impurity, defined as:

$$\text{Gini} = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{2.23}$$

where p_i is the proportion of instances belonging to class i, and C is the total number of classes. This measure quantifies how effectively each feature reduces uncertainty in classification. In this thesis, the Gini impurity is used to evaluate and compare the contribution of time-domain versus frequency-domain features to the classification accuracy of user movement states in VR traffic.

2.5.2 Deep Learning Architectures for Sequential Data Forecasting

Deep learning (DL), a subfield of machine learning, utilizes neural networks with multiple layers to learn hierarchical representations from data. DL models excel at extracting complex features directly from raw inputs, making them suitable for tasks involving large, intricate datasets, such as network traffic. For sequential data like network traffic, specialized architectures designed for temporal dependency modeling are crucial.

Recurrent Neural Networks (RNNs) and Their Gated Variants: LSTM and GRU

Traditional feed-forward neural networks process inputs independently, thus limiting their ability to capture sequential dependencies. Recurrent Neural Networks (RNNs) overcome this by maintaining internal memory to process sequences element-by-element, with hidden states evolving according to:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h), \tag{2.24}$$

where h_t represents the hidden state at timestep t. Although theoretically powerful, basic RNNs suffer from vanishing and exploding gradient problems during backpropagation through time [76], making them impractical for capturing long-term dependencies critical in many real-world network traffic patterns.

To mitigate these issues, LSTM networks introduce gating mechanisms—forget, input, and output gates—that enable selective memory retention and effective long-term dependency modeling [77]. Mathematically, the core mechanism of an LSTM cell at time step t can be described by the following equations:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{(Forget gate)} \tag{2.25}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{(Input gate)} \tag{2.26}$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \quad \text{(Candidate cell state)} \tag{2.27}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(Updated cell state)} \tag{2.28}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{(Output gate)} \tag{2.29}$$

$$h_t = o_t \odot \tanh(C_t) \quad \text{(Hidden state output)} \tag{2.30}$$

Here, x_t denotes the input vector at time t, h_{t-1} and C_{t-1} represent the hidden state and cell state from the previous time step, respectively. The weights W_f, W_i, W_c, W_o and biases b_f, b_i, b_c, b_o are learnable parameters optimized during training, while σ and tanh are sigmoid and hyperbolic tangent activation functions, respectively,

and $\odot$ denotes the element-wise product. This gating mechanism allows LSTMs to effectively learn and retain information over extended periods, making them highly suitable for capturing complex temporal dynamics prevalent in VR network traffic, such as burstiness linked to user interactions or rendering cycles, which unfold over multiple timesteps.

Gated Recurrent Units (GRUs) are a computationally efficient variation of Long Short-Term Memory (LSTM) networks, explicitly designed to address the vanishing gradient problem inherent in traditional Recurrent Neural Networks (RNNs). GRUs simplify the gating mechanisms found in LSTMs by combining the forget and input gates into a single update gate and merging the hidden and cell states into a unified hidden state representation. Specifically, GRUs employ two primary gating mechanisms: an update gate z_t, which determines the extent to which previous memory should be retained, and a reset gate r_t, which controls the incorporation of past hidden states into the current memory state. The GRU operations at time step t are mathematically defined as:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \qquad \text{(Update gate)} \tag{2.31}$$

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \qquad \text{(Reset gate)} \tag{2.32}$$

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) \qquad \text{(Candidate activation)} \tag{2.33}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \qquad \text{(Hidden state output)} \tag{2.34}$$

where x_t is the input at time t, h_{t-1} is the previous hidden state, h_t is the current hidden state, W_z, W_r, W_h denote the learnable weight matrices, b_z, b_r, b_h are the biases, σ is the sigmoid activation function, and $\odot$ represents element-wise multiplication.

The architectural simplifications of GRUs confer distinct advantages and disadvantages depending on the sequence modeling task. GRUs have fewer parameters than LSTMs, making them computationally lighter and faster to train. This efficiency often results in reduced overfitting, especially in scenarios involving small or medium-sized datasets. GRUs also tend to perform well on tasks that require short- to medium-range temporal modeling and are well-suited to real-time or resource-constrained applications, such as low-latency virtual reality (VR) network traffic prediction. When choosing between GRUs and LSTMs for VR network traffic modeling, the decision should consider several factors:

1. **Length and complexity of temporal dependencies**: GRUs are typically more effective for capturing short- to medium-term dependencies, whereas LSTMs are more suitable for long-term dependency modeling. For VR traffic, which exhibits both short-term burstiness and potentially longer-term patterns related to scene complexity or extended user interactions, evaluating both is crucial.

2. **Computational efficiency and resource availability**: GRUs are more computationally efficient due to their simpler structure, making them advantageous in real-time or embedded VR applications where inference speed and resource constraints are critical. This efficiency can directly impact the feasibility of deploying prediction models on edge devices or within network controllers with strict latency budgets.

3. **Dataset size and overfitting risk**: In scenarios with limited training data, GRUs' reduced parameter count can lead to better generalization. In contrast, LSTMs may leverage large datasets more effectively due to their higher capacity and flexibility. Given the challenges in obtaining large-scale, annotated VR traffic datasets, the data efficiency of GRUs is a significant consideration for this research.

Transformer Networks

The Transformer architecture, introduced by Vaswani et al. [78], revolutionized sequence modeling tasks by replacing traditional recurrence and convolution mechanisms entirely with attention mechanisms. Its key component, known as the *self-attention mechanism*, computes the relevance of every element within a sequence relative to each other, allowing the model to capture both short- and long-range dependencies simultaneously. This mechanism is mathematically defined as scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.35}$$

where the matrices Q, K, and V represent query, key, and value transformations of the input sequence, respectively, and d_k denotes the dimensionality of the key vectors. Transformers typically leverage multiple parallel attention computations—referred to as *multi-head attention*—to capture diverse relationships within the input data simultaneously.

In this study, only the *encoder component* of the Transformer architecture is employed. This design choice is justified by the nature of the task, which focuses on extracting meaningful and expressive representations from historical traffic data to forecast future network behavior. The encoder, comprising stacked self-attention and feed-forward layers, is sufficient for learning dependencies across input sequences. The decoder, typically used for sequence generation, is excluded since it is not required for our predictive modeling objectives.

Despite their architectural strengths, Transformer encoders may exhibit limitations in modeling VR network traffic when compared to recurrent neural networks such as LSTM and GRU. One issue is the absence of inductive biases toward temporal locality. Unlike RNNs, Transformers do not inherently model the order of inputs but instead rely on positional encodings to inject sequence information. This can be suboptimal when short-term temporal patterns—such as bursty packet flows—play a critical role in the prediction task. Additionally, the large number of parameters in Transformer models increases their demand for training data, which may pose challenges in data-scarce VR environments.

The following theoretical and architectural considerations contrast Transformer encoders with recurrent architectures in the context of VR traffic forecasting:

1. **Effectiveness in Capturing Local Temporal Dynamics**: Transformers lack inherent sequential processing and may struggle to capture rapid, localized fluctuations in packet behavior. In contrast, LSTM and GRU architectures naturally encode temporal progression through their recurrent structure, making them more effective at modeling bursty or session-based traffic patterns common in VR systems.

2. **Sensitivity to Dataset Size and Quality**: Transformers require extensive datasets to generalize well, due to their high parameter count and data-hungry design. In contrast, LSTMs and GRUs are more data-efficient and can often perform effectively in smaller-scale or noisy data environments, which are frequently encountered in VR deployments.

3. **Computational Efficiency and Latency Constraints**: The self-attention mechanism in Transformers has quadratic time and memory complexity with respect to sequence length, which can limit scalability in long-duration traffic traces. RNN-based models, with their linear complexity, offer more efficient inference and are better suited for real-time or resource-constrained applications.

Nevertheless, Transformer encoders exhibit superior performance in tasks involving extensive contextual dependencies and global relationships, such as natural language processing, translation, and classification tasks with long sequences. When applied to VR traffic prediction, they may excel in identifying high-level traffic trends or anomalies in batch inference scenarios, provided sufficient data and computational resources are available. Given these trade-offs, the choice between Transformer encoders and recurrent models should be guided by the nature of the VR traffic, prediction goals, and system constraint

2.5.3 Rationale for Model Selection in This Thesis

Based on the characteristics of VR network traffic—its non-stationary nature, distinct user states, and the need for short-term load prediction this thesis employs a multi-faceted machine learning approach. For the task of classifying user motion states based on hybrid time-frequency (HTF) features, Random Forest is selected due to its ability to construct robust non-linear decision boundaries and its resilience to noise in high-dimensional feature spaces. Beyond its predictive performance, Random Forest provides interpretable feature importance measures, allowing for an explicit analysis of the relative contributions of time- and frequency-domain descriptors. This interpretability directly supports the thesis's objective of quantifying how HTF representations enhance the separability of user states.

For short-term traffic forecasting, RNN based models are employed to capture the sequential nature of VR traffic traces. Among these, both LSTM and GRU networks are investigated. GRU, with its reduced parameter count, offers lower inference latency and reduced risk of overfitting when modeling temporal window typical of burst-level traffic patterns. LSTM, on the other hand, provides a higher representational capacity for capturing longer and more structured temporal dependencies. The comparative evaluation between GRU and LSTM is designed to test the hypothesis that GRU performs better in short, noisy sequences, whereas LSTM becomes more effective as the temporal context expands. Additionally, a Transformer encoder model is included as a benchmark to assess the potential advantages of global self-attention mechanisms in modeling longer-range dependencies and capturing high-level temporal patterns. While Transformer models lack an inherent bias toward local sequentiality and introduce quadratic complexity with respect to input length, their inclusion allows for an investigation into whether their expressiveness can outperform recurrent models under certain conditions—particularly in batch-mode inference scenarios with sufficient training data.

Together, these model choices reflect a deliberate design aimed at addressing the dual challenges of VR network traffic analysis: accurate classification of user behavior states and reliable short-term prediction of dynamic traffic loads. The diversity in modeling approaches allows this thesis to assess the trade-offs between interpretability, predictive power, latency, and generalizability, all of which are critical to ensuring Quality of Experience (QoE) in real-time VR applications.

2.6 Network Resource Management Concepts

The rapidly evolving telecommunications landscape, particularly with the advent of 5G and forthcoming 6G networks, necessitates highly efficient and adaptive network resource management strategies [79]. Traditional reactive approaches, which respond only after network events such as congestion occur, increasingly fall short in meeting the stringent Quality of Service (QoS) and Quality of Experience (QoE) demands of modern applications—especially interactive Virtual Reality (VR) [4, 79].

Modern paradigms emphasize proactive network management, forecasting future traffic demands to allocate resources in advance, thereby minimizing latency and optimizing throughput. In this context, two essential concepts emerge: *Semi-Persistent Scheduling (SPS)* and a broader shift toward proactive resource allocation and congestion control.

2.6.1 Semi-Persistent Scheduling (SPS) in 5G/6G Networks

Semi-Persistent Scheduling (SPS), originally introduced in 4G LTE and significantly enhanced in 5G New Radio (NR), is an essential mechanism for efficiently allocating resources for recurring uplink (UL) and downlink (DL) data traffic [35, 80]. Unlike dynamic scheduling, where resources are explicitly assigned per Transmission Time Interval (TTI), SPS uses semi-static resource grants repeating at predefined intervals [35]. SPS is particularly beneficial for periodic or quasi-periodic traffic, such as Voice over IP (VoIP), interactive gaming, and notably, VR applications that generate consistent control and update messages [4, 80].

Key advantages of SPS include reduced signaling overhead and lower latency. SPS minimizes control signaling, such as Physical Downlink Control Channel (PDCCH) messages between base stations and user equipment (UE), by pre-allocating resources across multiple intervals. This preallocation conserves radio resources and enhances overall spectral efficiency. For example, prior studies demonstrated that SPS reduces computational scheduling decisions by nearly 50% in specific windows [35]. Analogous mechanisms, such as Proactive Grants (PGs) in 5G networks, further reduce the need for explicit scheduling requests, thereby substantially lowering signaling overhead [81].

Signaling overhead reduction can be expressed mathematically as:

$$O_{\text{reduction}}(\%) = \frac{O_{\text{dynamic}} - O_{\text{SPS}}}{O_{\text{dynamic}}} \times 100, \tag{2.36}$$

where O_{dynamic} is the number of signaling messages required per TTI under dynamic scheduling, and O_{SPS} represents the messages under SPS.

Moreover, SPS significantly reduces latency due to its deterministic scheduling nature. UEs transmit or receive data without awaiting explicit scheduling grants each

interval, resulting in predictable and consistently lower end-to-end delay [35, 81]. Predictable latency is particularly crucial in latency-sensitive VR applications to prevent motion sickness and maintain immersive experiences. Simulation results indicate that SPS leads to only a minimal increase in average delay (typically less than 10%) compared to fully dynamic scheduling, while simultaneously improving resource efficiency [35]. In one study on proactive resource slicing, latency was reduced from approximately 30.3 ms under reactive scheduling to as low as 4.5 ms using proactive grants [81].

However, the effectiveness of SPS is highly dependent on accurate traffic prediction. Mismatches between pre-allocated resources and actual traffic demand can lead to inefficiencies—either wasteful over-provisioning or, more critically, under-provisioning. The latter is especially problematic in VR environments where abrupt user movements can trigger traffic bursts [80, 81]. Therefore, reliable forecasting of future traffic demand is essential for dynamically tuning SPS allocations. Accurate predictions allow schedulers to align SPS grants with expected user demand, maximizing SPS effectiveness for latency-sensitive applications like immersive VR [4, 35]. This thesis specifically addresses this challenge by integrating advanced VR traffic prediction models into a hybrid SPS/PF scheduling framework to enhance VR QoE, as detailed in Chapter 7.

2.6.2 Proactive Resource Allocation and Congestion Control in Next-Generation Networks

The increasing demands of next-generation applications, particularly interactive virtual reality (VR), require a fundamental shift from conventional reactive network management towards proactive resource allocation and congestion control. Traditional reactive approaches typically detect network issues such as congestion or buffer overflows only *after* their occurrence. Consequently, corrective actions—including packet dropping, reduction of transmission rates, or traffic rerouting—introduce latency and degrade network performance, significantly impacting the Quality of Experience (QoE) required by immersive VR applications [4, 28, 82].

To address these shortcomings, proactive methods have emerged as essential, leveraging predictive modeling to forecast future network conditions and resource demands before problems arise [79, 83]. These predictive capabilities depend significantly on accurate short-term traffic forecasts, often represented as $L_{\text{pred}}(t + \tau)$, where t is the current time and τ is the prediction horizon. Typically, this prediction horizon ranges from tens to hundreds of milliseconds, driven by practical considerations related to network dynamics and user behaviors [35, 82].

Selecting an optimal prediction horizon is inherently constrained by the complexity and unpredictability of VR user interactions and application-layer protocols. As noted by He *et al.*, network traffic prediction is especially difficult over very short timescales (e.g., 1 ms), due to the complex interplay between user behavior and upper-layer application protocols [35]. These interactions introduce non-stationary, bursty characteristics that make accurate prediction nearly impossible on ultra-short timescales.

While conventional literature often suggests prediction windows in the range of 100 ms as a balance between responsiveness and stability, our empirical findings in Chapter 6 demonstrate that for VR traffic load prediction, a 200 ms horizon yields significantly more robust and reliable forecasts ($R^2 \approx 0.85$–0.86). This 200 ms interval effectively smooths out high-frequency noise while remaining short enough to enable

meaningful network adaptation. However, effectively utilizing a 200 ms forecast within network schedulers that operate at 1 ms granularity poses a significant challenge. This temporal mismatch requires a sophisticated integration strategy, which is a core focus of this thesis. Despite these challenges, a 150-200 ms prediction horizon aligns well with physiological and cognitive delays in human motor control, making it suitable for tasks such as trajectory prediction and head movement estimation in VR [28, 84]. This further supports the practical utility of our optimal prediction horizons for various aspects of VR QoE management.

Within these practical limits, proactive resource provisioning becomes both feasible and beneficial. For example, predicting that a user is about to transition from a stationary state to dynamic movement allows the network to pre-allocate bandwidth, computation, and storage resources in anticipation of the resulting traffic burst [85]. Similarly, trajectory prediction within this 100 ms window facilitates proactive beam steering and avoids line-of-sight obstruction, which are critical for maintaining immersive and uninterrupted VR sessions [84].

Dynamic bandwidth allocation also benefits from predictive modeling. By forecasting traffic fluctuations within a short-term horizon, networks can dynamically adjust maximum bit rates (MBRs) or reconfigure the balance between semi-persistent and dynamic resource allocations. This avoids over-provisioning during idle periods and prevents under-provisioning during peak loads, improving spectral efficiency and reducing latency [85, 35].

Adaptive congestion control further extends these benefits. Predictive models can estimate future round-trip times (RTT) and congestion levels, allowing congestion window sizes (`cwnd`) and sending rates to be adjusted before queues build up. Machine learning-based congestion controllers, for example, utilize forecasted RTT and queue occupancy $Q(t + \tau)$ to make real-time decisions that maintain throughput and minimize latency [82, 86]. This is particularly advantageous for real-time VR traffic, where delay spikes are perceptually unacceptable.

Overall, proactive scheduling and congestion control strategies offer significant improvements to network performance and user QoE when guided by accurate, short-term prediction models. Even when these models exhibit modest errors, studies have shown that the benefits of proactive management outweigh those of purely reactive schemes [4]. Therefore, selecting an appropriate prediction horizon and employing robust modeling techniques are crucial to the success of next-generation VR-enabled networks [83, 28].

Chapter 3

Literature Review

3.1 VR and Interactive Gaming Traffic Measurement and Analysis

The rapid proliferation of Virtual Reality (VR) and interactive gaming has significantly increased interest in analyzing their network traffic characteristics. VR applications differ fundamentally from traditional video streaming and standard online gaming, as they impose stringent Quality of Service (QoS) and Quality of Experience (QoE) requirements due to high interactivity and sensitivity to latency and throughput fluctuations [17, 22]. Thus, comprehensive understanding and modeling of these unique traffic patterns is essential for optimizing network resource allocation and ensuring immersive user experiences.

Initial efforts primarily investigated generalized cloud gaming or pre-recorded 2D video streams, typically employing Variable Bit Rate (VBR) encoding techniques. These early studies provided insights into downlink traffic characteristics, including packet size distributions and inter-packet intervals [22]. Nevertheless, the complexities inherent to real-time interactivity, such as unpredictable user movements and interactions within immersive VR environments, were often overlooked or insufficiently addressed [87].

Traffic measurement methodologies typically involve deploying real-world testbeds, combined with packet capture tools such as Wireshark or tshark. Casasnovas et al. [17], for example, captured network flows generated by Unity Render Streaming over Wi-Fi 6, measuring key metrics such as round-trip time (RTT), jitter, frame rate, and packet loss. Similarly, Salehi et al. [88] utilized an edge computing setup with an Oculus Quest HMD, using Wireshark to collect uplink tracking data and downlink video frame transmissions. Such studies commonly apply custom windowing methods on captured traffic, extracting statistical features—such as minimum, maximum, mean, and standard deviation—for packet sizes and inter-arrival times in both uplink and downlink directions [5].

Across the existing literature, there is consensus on several distinctive VR traffic characteristics. Downlink traffic, typically involving video frame transmissions, exhibits a pronounced bimodal packet size distribution, with larger packets around 1242 bytes in SRTP video packets [17], or approximately 1320 bytes for fragmented video frames [22]. At the MAC layer, these large frames are fragmented into smaller MPDUs around 1442 or 1490 bytes [5, 88]. In contrast, uplink traffic predominantly consists of smaller packets, usually user tracking data averaging approximately 192 to 254 bytes [5, 22].

VR headset-rendered games commonly exhibit significant use of UDP in the downlink (>95%), whereas PC-rendered VR games frequently involve increased use of TCP and TLS [28].

Bandwidth and rate requirements for VR applications are notably demanding, with reported application-layer data rates between approximately 14.13 Mbps and 21.42 Mbps, occasionally peaking as high as 60 Mbps during application launches [28, 88]. For example, downlink SRTP video streams alone can consume around 53 Mbps [17]. Although uplink tracking packets are transmitted frequently—typically at rates between 200 and 1000 Hz—they require comparatively minimal bandwidth, often less than 0.03 Mbps, as seen in games like Rec Room [88].

Furthermore, VR network traffic exhibits significant temporal dynamics and burstiness. Video frames are generally transmitted in periodic bursts followed by intervals with minimal or no data transmission, and even streams encoded with Constant Bit Rate (CBR) can experience substantial variations in frame size. Such variability may necessitate bandwidth overprovisioning of up to 30% to reliably maintain performance targets [22]. Autocorrelation analyses frequently identify pronounced short-term traffic dependencies, confirming the bursty nature and short-term predictability of VR network traffic [87].

Importantly, user interactions—particularly head and body movements—directly impact traffic patterns. Frequent headset movements correspond to irregular uplink packet transmissions, and traffic characteristics also notably vary across different game phases, such as launch, tutorial, browsing, and active gameplay [28]. Existing research consistently observes correlations between uplink tracking traffic and downlink video flows, reinforcing the idea that user interactions strongly influence VR traffic dynamics [5].

Despite these valuable insights, a significant research gap remains in the explicit and granular analysis of VR network traffic concerning diverse user mobility states. While existing studies often acknowledge the general impact of user movement, they rarely systematically differentiate or deeply analyze traffic patterns based on distinct, specific types of user mobility, which can significantly alter network traffic profiles. It is crucial to recognize that "user movement" in VR is not a monolithic concept. It arises from heterogeneous sources, each driving distinct network mechanisms and impacting traffic patterns differently.

- **Sensor-level Motion:** This encompasses fine-grained tracking data from the Head-Mounted Display (HMD) itself (e.g., head rotation for gaze changes, physical translation within the tracked space) and hand controllers (e.g., gestures, object manipulation). Such movements primarily affect uplink packet frequency and content, and indirectly influence downlink video encoding based on the viewing frustum [17, 22, 88]. In cloud/edge VR streaming, sensor-level motion is fed back upstream to update the viewport, upon which the next downlink frame is rendered, forming a tight motion–rendering loop crucial for low motion-to-photon latency [5, 17]. Anticipating HMD pose dynamics can significantly reduce bandwidth and mitigate stalls by allowing predictive content fetching and rendering [89, 90].

- **Virtual Avatar Locomotion:** This refers to the user's controlled movement of their in-game avatar (e.g., walking, running, teleporting via joystick or other input methods). These modalities (e.g., smooth movement vs. teleportation) are

distinct interaction classes that induce different scene-update patterns and viewpoint dynamics [91, 92]. Teleportation, in particular, can behave like a sudden scene cut, often triggering intra refresh frames and short-lived bitrate spikes in video encoders due to abrupt changes in visual content [93, 94].

- **User-unrelated Scene Dynamics:** These are autonomous movements or animations within the virtual environment (e.g., moving Non-Player Characters (NPCs), environmental particle effects, time-of-day changes). Even when the user remains stationary, these dynamics contribute to the overall traffic background and significantly alter content complexity, which is known to modulate required bitrate and burstiness in game/video streaming [95, 96].

Prior works, while valuable, have often analyzed traffic across broader game phases, without providing sufficiently granular labeling to isolate traffic behaviors directly attributable to these specific forms of user mobility or their interplay with scene dynamics [28]. Many studies, while acknowledging the role of user movement, have either not systematically segmented their traffic datasets based on these fine-grained mobility distinctions [5, 17, 22, 88] or have deferred detailed analyses of this complex relationship to future work [87]. Treating these heterogeneous sources as a single "moving" state can obscure multi-modal distributions and short-term dependencies, leading to mixed traffic patterns and potentially degrading the interpretability and generalization of prediction and classification models.

This thesis addresses a critical and foundational aspect of this gap by focusing on the distinction between 'stationary' and 'actively moving' user states. Specifically, our 'moving' state captures the network traffic patterns generated by continuous virtual avatar locomotion, driven by user input (e.g., keyboard or joystick). We hypothesize that this explicit user-controlled avatar movement drives the most significant and consistent shifts in network traffic volume, periodicity, and burst patterns due to frequent spatial updates and dynamic content loading/unloading. While a deeper, sub-classification of movement types (e.g., distinguishing between different avatar locomotion methods, or incorporating effects of physical head rotation) is a valuable avenue for future research, establishing robust methods for distinguishing these primary macrostates—defined by the presence or absence of user-controlled avatar movement—is a prerequisite for effective predictive network management due to their direct impact on network load and latency sensitivity.

Addressing this foundational gap in understanding and distinguishing core user mobility states is paramount for next-generation VR networking, offering significant practical implications for optimizing network resource allocation and ensuring a superior Quality of Experience (QoE) in interactive VR environments. This capability translates into tangible benefits by enabling:

- **Adaptive Resource Allocation:** Network schedulers (e.g., in 5G/6G base stations, particularly for Semi-Persistent Scheduling (SPS) mechanisms) can proactively allocate bandwidth and low-latency channels when a transition to a "moving" state is detected or predicted. This ensures sufficient resources are available before the surge in traffic due to spatial updates, minimizing lag and stutter. Conversely, during "stationary" periods, resources can be more efficiently managed, potentially allowing for lower data rates, deeper compression, or re-prioritization of background tasks, thereby optimizing overall network utilization and reducing energy consumption [5, 35].

- **Proactive Congestion Control:** Anticipating traffic surges driven by user motion enables buffer, queue, and rate policies to be tuned before violations occur. This proactive approach is supported by regime-switching models that explicitly capture the non-stationarity and burstiness observed in Internet traffic. Examples include packet-level hidden Markov models (HMMs) that jointly model inter-packet times and sizes [97], hierarchical Markov-modulated Poisson process (MMPP) models that reproduce both burstiness and long-range dependence [98], and HMM-based traffic classification techniques that remain effective even for encrypted flows [99]. Beyond descriptive modeling, such state-aware formulations have also been applied to predict congestion levels in video streaming, thereby reducing packet loss and jitter and enabling proactive control [100].

- **Enhanced QoE and Efficiency:** By aligning network resource provision more closely with real-time application demands and user behavior, the proposed approach directly contributes to maintaining stable frame rates, low motion-to-photon latency, and overall fluid interactions, which are paramount for preventing discomfort and ensuring immersive VR experiences. For instance, distinguishing VR traffic states is expected to reduce Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) of short-term frame-size forecasts, tighten tail errors (e.g., p95/p99 latency), and improve macro-F1 scores for traffic classification compared to state-agnostic models. This improved predictability can significantly reduce the need for conservative bandwidth over-provisioning, which some studies suggest can be up to 30% [22], leading to more efficient network utilization.

3.2 Network Traffic Classification Methods

3.2.1 Traditional and Statistical Approaches

Network traffic classification is an essential component of effective network management, supporting critical tasks such as Quality of Service (QoS) provisioning, security assurance, and optimized resource allocation. Over the decades, classification techniques have continuously evolved, driven by technological advancements and emerging challenges. This section critically reviews the evolution from traditional rule-based approaches to modern statistical classification techniques, highlighting the strengths, limitations, and motivations behind each advancement, particularly as they pertain to interactive applications such as Virtual Reality (VR).

Port-based classification represents the earliest and most straightforward approach. This method relies on mapping network traffic to well-known TCP/UDP port numbers officially assigned by the Internet Assigned Numbers Authority (IANA). For example, HTTP traffic is typically associated with TCP port 80, and HTTPS with port 443 [101, 102]. The main advantage of this approach lies in its simplicity, ease of implementation, and minimal computational overhead, making it suitable for high-speed routers and firewalls. However, this method exhibits significant limitations. Modern network applications increasingly utilize dynamically assigned, randomized, or non-standard port numbers for reasons such as security, flexibility, and circumventing firewall restrictions. Peer-to-peer (P2P) applications, gaming, VPN services, and various streaming platforms commonly bypass traditional port assignments, leading to severe misclassification issues. Empirical studies indicate port-based classification accuracy

can be as low as 30–70%, with misclassification rates reaching up to 70% for many common services [103, 104, 105].

To address these severe limitations, Deep Packet Inspection (DPI) methods emerged as the second-generation solution. DPI classification identifies applications by inspecting packet payloads to detect distinctive protocol signatures or unique byte sequences [102, 104]. Tools such as OpenDPI, NDPI, and L7-filter gained prominence due to their ability to accurately classify thousands of application protocols, often attaining accuracy rates exceeding 99% for recognizable, unencrypted traffic [102]. DPI, therefore, provided a significant step forward compared to port-based methods, effectively resolving many issues associated with dynamic port assignments. However, DPI itself introduced new and critical limitations. First, it requires substantial computational resources for payload analysis, significantly impacting performance on large-scale, high-speed networks. Second, DPI fundamentally fails when confronted with encrypted traffic such as HTTPS, QUIC, or VPN tunnels, which have become pervasive in recent years. Lastly, privacy and ethical concerns over inspecting users' packet content present significant legal and regulatory challenges, limiting its real-world applicability, especially in sensitive environments [102, 103].

Given DPI's shortcomings with encrypted and obfuscated network traffic, researchers proposed statistical and behavioral (flow-based) classification techniques, representing the third generation of classification methodologies. These methods infer application types not from explicit packet payloads but from observable statistical and behavioral properties of network traffic. Specifically, statistical classification techniques extract features such as total packet counts, flow durations, mean and variance of packet sizes, inter-arrival times, and directional burst patterns to create representative statistical profiles of network flows [101, 102, 105]. Unlike DPI, these methods are robust against encryption since the statistical behaviors of applications typically remain consistent regardless of payload encryption.

The application of statistical features naturally paved the way for employing traditional supervised machine learning (ML) algorithms, significantly enhancing classification accuracy and generalization capability. Among widely studied ML algorithms in the traffic classification literature are the following approaches, each having distinct strengths and limitations:

- **Support Vector Machines (SVM)** are highly effective due to their capability to handle large-dimensional datasets by identifying optimal decision boundaries between classes. They offer strong performance with carefully tuned hyperparameters, particularly for binary classification scenarios [101]. However, SVMs can become computationally expensive for large-scale datasets and might require intricate kernel tuning to achieve optimal accuracy.

- **Decision Tree algorithms (e.g., C4.5, C5.0, CART)** gained popularity due to their interpretability, simplicity, and robustness in handling complex, non-linear relationships. Decision trees, especially algorithms such as C5.0, frequently achieve accuracy and precision of over 99% across diverse traffic scenarios [102, 104]. Yet, decision trees can suffer from overfitting, especially on noisy or imbalanced datasets, leading to weaker generalization performance if not properly managed.

- **Naive Bayes classifiers**, relying on strong assumptions of conditional independence among features, offer fast training and classification performance, making

them computationally appealing for real-time classification scenarios [103]. However, the simplistic independence assumption often reduces the classification accuracy, particularly when features exhibit strong interdependencies, as is common with network traffic data.

- **K-Nearest Neighbors (KNN)**, an instance-based classification technique, is conceptually simple and effective in many traffic classification contexts. It classifies data points based on the majority label of their nearest neighbors [103]. However, its computational complexity increases substantially with dataset size, limiting its scalability to high-throughput or large-scale classification environments.

- **Ensemble-based approaches** such as Random Forest and Gradient Boosting have demonstrated superior accuracy, stability, and resilience against overfitting by combining multiple weak classifiers. Random Forest, aggregating numerous decision trees, consistently performs among the best in classification benchmarks with accuracy frequently exceeding 95–99% [103]. Nonetheless, ensemble methods introduce increased computational overhead during training, require substantial tuning of hyperparameters, and typically result in less interpretable models compared to simpler single-tree classifiers.

Despite substantial progress provided by statistical ML classification methods, notable limitations remain, particularly concerning highly interactive and temporally dynamic traffic such as VR applications. Conventional statistical methods predominantly aggregate traffic statistics over entire flows or fixed-size time windows, inadvertently obscuring fine-grained temporal and dynamic features that characterize interactive traffic patterns. For instance, subtle variations in user interactions within a VR environment (e.g., transitioning from stationary to active movements or interacting with objects) produce nuanced temporal and periodic changes in packet inter-arrival times and packet sizes. These subtle yet critical patterns may not be effectively captured by traditional statistical summarizations [22, 106].

Thus, existing statistical ML methods, while effective in general classification tasks, highlight a gap and underscore the necessity for more sophisticated and specialized feature-engineering techniques. Addressing this need, advanced hybrid methodologies incorporating combined time-frequency domain analyses (e.g., wavelet transforms, Short-Time Fourier Transform) and deep learning techniques have emerged. Such hybrid approaches provide richer temporal and frequency-based representations, effectively capturing subtle interactive patterns and significantly enhancing classification and prediction performance for VR network traffic—forming the core focus of the subsequent chapters of this thesis.

3.2.2 Machine Learning and Deep Learning-based Approaches

The increasing complexity, dynamism, and encrypted nature of modern network traffic—particularly with emerging interactive applications like Virtual Reality (VR)—have necessitated a significant paradigm shift from traditional statistical and rule-based approaches toward advanced machine learning (ML) and deep learning (DL) methodologies. Conventional methods often rely on manual rules or predefined statistical features, limiting their adaptability in rapidly evolving network scenarios. Moreover,

encrypted traffic presents challenges to Deep Packet Inspection (DPI), necessitating automatic, privacy-preserving feature extraction and learning paradigms.

Early machine learning classifiers depended heavily upon extensive manual feature engineering. Domain experts manually selected features such as packet size distributions, packet inter-arrival time (IAT), and flow-level statistics [107]. Although these carefully crafted features provided high accuracy in some specific environments, this approach was costly, labor-intensive, prone to human biases, and lacked generalization to encrypted or evolving traffic scenarios [108]. To overcome these limitations, the deep learning paradigm shifted the burden of feature engineering from human experts to algorithms. DL models facilitate an end-to-end learning approach, automatically extracting high-level features directly from raw network traffic data without explicit human intervention. This reduces labor costs, minimizes human errors, and significantly enhances adaptability and scalability to diverse traffic conditions and encrypted flows [102].

Among the prominent DL architectures adopted, **Convolutional Neural Networks (CNNs)** initially gained attention due to their superior performance in identifying local spatial patterns within sequential data. Particularly, one-dimensional (1D) CNNs have proven effective in analyzing sequential packet payloads and temporal features such as packet lengths and IAT sequences [108, 109]. For instance, the "Deep Packet" framework demonstrated the effectiveness of combining CNNs and autoencoders directly on raw encrypted packet data, bypassing the need for payload inspection. Nevertheless, CNNs inherently exhibit limitations, specifically their inability to effectively model long-range temporal dependencies, critical for many traffic prediction and classification tasks involving extended sequences. CNN performance also depends significantly on preprocessing steps, feature normalization, and the choice of input representations, potentially restricting generalization to new, unseen traffic patterns [102].

To address CNN's limitation regarding temporal dependencies, **Recurrent Neural Networks (RNNs)**—especially Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) emerged as powerful alternatives. RNNs offer enhanced capabilities in capturing temporal correlations and long-range dependencies within sequential traffic data [109]. GRUs, in particular, gained prominence for their computational efficiency over LSTMs, facilitating their deployment in real-time applications requiring moderate resource usage. However, despite these improvements, RNN architectures frequently encounter challenges such as computational complexity, slower training convergence due to sequential processing, and difficulties associated with the vanishing gradient problem when handling extremely long sequences. These issues can limit their applicability in VR scenarios characterized by strict latency constraints, such as the stringent 20 ms motion-to-photon latency budget [22].

Recognizing these limitations, **hybrid architectures** combining CNN and RNN structures—such as CNN-LSTM or CNN-GRU models—have emerged as robust solutions. By integrating CNN's local feature extraction capability with the long-term sequential modeling strength of RNNs, hybrid models substantially improved classification accuracy and temporal prediction capabilities [102, 110]. Furthermore, recent advancements in attention mechanisms have been integrated into these hybrid models, enabling them to dynamically focus on the most significant features across temporal dimensions. Attention-enhanced CNN-RNN hybrid models have thus yielded improved accuracy and adaptability, particularly valuable for capturing nuanced and dynamic

traffic behaviors common in interactive applications like VR. Nevertheless, these hybrid models inherently introduce higher computational complexity and memory demands, potentially limiting real-time applicability in resource-constrained or latency-sensitive VR environments.

In parallel to supervised architectures, **Autoencoders (AEs)** and **Stacked Autoencoders (SAEs)** have been applied to network traffic analysis primarily for feature extraction and anomaly detection tasks [106]. Autoencoders effectively compress input data into lower-dimensional latent representations, capturing intrinsic traffic patterns in an unsupervised manner. Their application to anomaly detection leverages reconstruction errors, facilitating efficient identification of abnormal traffic behaviors without explicitly labeled data. However, the performance of AEs and SAEs critically depends upon the representativeness and balance of the training dataset. Overfitting, underrepresentation of rare traffic scenarios, and difficulty interpreting extracted latent features remain critical challenges that limit their widespread deployment for precise, real-time network traffic classification in dynamic VR scenarios.

Another recent evolution in DL architectures involves the use of full **Transformer-based models**, which solely utilize attention mechanisms without recurrent structures. Transformers offer distinct advantages, including capturing long-range dependencies more effectively, faster parallelizable training, and enhanced scalability to large datasets [110]. Despite these advantages, Transformer-based models remain relatively less explored in network traffic classification literature due to their high computational cost, substantial training data requirements, and difficulties in capturing short-range or local patterns effectively compared to CNN or hybrid architectures. Nonetheless, their potential to model rapidly changing VR traffic, capturing both short-term bursts and long-range correlations, represents an active area of research.

In parallel with architectural developments, significant advances have also occurred in input feature representation. Early deep learning applications directly utilized raw packet bytes, offering simplicity but limited performance under encrypted scenarios. Subsequently, payload-independent temporal features, such as packet length sequences, IAT, and directional flows, gained favor due to their inherent privacy-preserving properties [109]. Most recently, significant progress has been achieved by representing traffic data in the time-frequency domain through techniques such as Short-Time Fourier Transform (STFT) and Continuous Wavelet Transform (CWT), producing spectrograms or scalograms. These representations enable CNNs to efficiently capture multi-scale temporal and frequency domain patterns previously invisible in raw or purely temporal domains, significantly improving performance in classification and prediction tasks [106]. Despite these advancements, parameter optimization (e.g., optimal window sizes for STFT, suitable mother wavelets for CWT) remains complex and highly task-dependent, posing potential challenges for generalized adoption.

Despite the significant progress offered by these advanced DL methodologies, substantial challenges remain, particularly in the highly dynamic and interactive domain of VR traffic. VR traffic is characteristically non-stationary and susceptible to frequent concept drift caused by rapidly evolving codecs, protocols (such as QUIC/HTTP3), and unique, user-driven interactive behaviors [19]. Consequently, current models—while substantially more adaptive and robust compared to traditional statistical methods—still struggle to fully capture and leverage fine-grained, short-lived temporal and spectral features necessary to accurately classify or predict VR network traffic under real-world conditions. Addressing this gap remains an open and critical research challenge,

which this thesis further explores by integrating hybrid time-frequency features with DL methods, detailed in subsequent chapters.

3.3 Network Traffic Forecasting Techniques

3.3.1 Statistical Models for Forecasting

Statistical forecasting models have long been instrumental in network traffic prediction owing to their interpretability, computational simplicity, and established theoretical foundations. These models typically rely on explicit assumptions about the data generation process, including stationarity, linearity, and defined periodic behaviors. Among the most commonly employed statistical forecasting techniques are AutoRegressive Integrated Moving Average (ARIMA) and its seasonal counterpart (SARIMA), Exponential Smoothing methods (e.g., Holt-Winters), and state-space-based approaches such as the Kalman Filter.

ARIMA and SARIMA models have been widely utilized due to their ability to capture temporal correlations within network traffic data. ARIMA models are structured around three distinct components: autoregressive (AR), differencing (Integrated, I), and moving average (MA). Specifically, ARIMA models assume data stationarity—i.e., constant statistical properties such as mean and variance across time. For series exhibiting non-stationarity, differencing transformations are applied to stabilize statistical properties, enabling accurate modeling and forecasting [111]. The SARIMA model further extends ARIMA by explicitly modeling periodic seasonal fluctuations inherent in telecommunication traffic. Recent empirical work has shown SARIMA to outperform simpler methods in capturing complex seasonal patterns. For example, Kochetkova et al. [112] demonstrated that SARIMA achieved superior forecasting accuracy for mobile network download traffic (MAPE of approximately 11.2%), whereas Holt-Winters performed better for upload scenarios (MAPE approximately 4.17%). Nevertheless, ARIMA and SARIMA methods have critical limitations. Specifically, their assumption of linearity limits their performance on highly volatile or non-linear traffic patterns. Additionally, their inherent short-memory structure renders them ineffective in capturing long-range dependence (LRD) or self-similarity, which have been widely observed in network traffic, including Virtual Reality (VR) applications [113, 114]. Consequently, these models typically underperform when applied directly to VR scenarios where user-driven interactivity introduces pronounced volatility and rapid regime shifts [87].

Exponential Smoothing (ETS) methods, especially the Holt-Winters algorithm, represent another extensively applied family of statistical forecasting models. These methods forecast future values by assigning exponentially decreasing weights to older observations, effectively capturing trends and seasonal patterns in relatively stable and predictable traffic environments [111]. The popularity of Holt-Winters models in telecommunications stems from their interpretability, ease of implementation, and computational efficiency. They are often employed as baseline models or as lightweight forecasting algorithms in applications requiring low-latency predictions, such as real-time radio access operations [114]. Despite these strengths, ETS methods exhibit significant drawbacks when confronting VR traffic characteristics. Their performance deteriorates rapidly in the presence of abrupt, non-linear changes and regime shifts due to their inherent assumption of smoothly evolving patterns. Consequently, ETS-based forecasts often exhibit substantial inaccuracies during periods of high volatility, such as

those triggered by dynamic user interactions and rapidly changing VR content [87, 114].

Kalman Filters (KF) represent state-space statistical models widely applied for online, real-time forecasting and filtering in dynamic, noisy environments. A Kalman filter operates through a prediction-update iterative cycle, making it particularly suitable for real-time network scenarios, including adaptive throughput forecasting in next-generation mobile networks [115]. Recent research demonstrated Kalman filter-based hybrid methods outperforming even advanced machine learning techniques, such as Long Short-Term Memory (LSTM) networks, in inference speed and predictive accuracy within latency-sensitive environments (e.g., vehicle mobility scenarios in 5G networks with coherence times around 0.2 milliseconds). Specifically, Biswas et al. [115] reported that combining Multiple Linear Regression (MLR) with Kalman filtering substantially improved prediction accuracy ($R^2 > 0.8$) and reduced inference latency by approximately a factor of ten compared to traditional deep learning approaches. However, Kalman filters face inherent limitations due to their linear Gaussian assumptions, which significantly constrain their ability to handle highly non-linear dynamics and non-Gaussian distributions common in interactive VR traffic. Additionally, their accuracy critically depends on an appropriate definition of the state transition model, which is challenging in VR scenarios characterized by unpredictable user motion and content-driven traffic fluctuations [87, 114].

Despite the practical advantages of these statistical forecasting models, the complex nature of VR network traffic imposes inherent limitations. First, VR traffic displays significant self-similarity and long-range dependence (LRD), resulting in persistent correlations across multiple temporal scales [113]. Classical short-memory models (e.g., standard ARIMA and ETS) struggle to model these fractal-like properties effectively, frequently leading to increased forecast errors. Traditional statistical techniques thus typically require additional processing or hybridization with techniques explicitly designed for capturing LRD. Second, the pronounced non-stationarity and rapid regime shifts observed in interactive VR applications create additional modeling complexity [87]. Even nominally Constant Bit Rate (CBR) encoded VR streams can experience rapid fluctuations due to adaptive encoding and user-driven interactions, thereby substantially degrading the performance of stationary or slow-adaptive forecasting methods [114].

To address these critical limitations, various mitigation strategies have emerged. Differencing and explicit seasonality modeling, intrinsic to ARIMA/SARIMA approaches, represent initial steps to manage non-stationarity. However, these techniques remain limited in addressing complex dynamics and rapid volatility changes. Consequently, advanced signal decomposition strategies, notably Wavelet Transform-based decompositions, have been adopted. Wavelet decompositions effectively segregate network traffic into multiple time-frequency sub-bands, isolating stationary and predictable signals from transient and volatile components. Hybrid wavelet-statistical pipelines have exhibited improved prediction performance over standard statistical models. For example, Wei et al. [116] reported that a hybrid wavelet-SARIMA model reduced Normalized Mean Squared Error (NMSE) from approximately 0.1650 to 0.1526 compared to a standalone ARIMA approach. Additionally, wavelet-based estimators (e.g., Abry-Veitch) offer robust solutions for diagnosing LRD even amidst non-stationarity and abrupt traffic-level shifts [117]. Nevertheless, these hybrid wavelet methods, despite their enhanced performance, rely significantly on parameter selection, appropriate wavelet choice, and assumptions of piecewise stationarity, limiting their full adaptability to

highly dynamic VR scenarios. While statistical models provide a solid theoretical foundation and computational efficiency, their intrinsic limitations in modeling VR traffic's complex, dynamic characteristics motivate the exploration and development of more adaptive, data-driven forecasting methods, such as machine learning and deep learning approaches.

3.3.2 Deep Learning Models for Forecasting

Deep learning (DL) models have significantly advanced the domain of network traffic forecasting due to their capability of extracting complex, nonlinear, and spatio-temporal features from large and diverse datasets [114]. Unlike classical statistical forecasting methods, DL models inherently adapt to dynamic network conditions, capturing fine-grained traffic variations and hidden patterns crucial for ensuring high-quality user experience, especially in Virtual Reality (VR) scenarios characterized by rapidly evolving network demands. Several categories of DL models have been employed in this domain. Each exhibits distinct strengths and limitations in modeling traffic dynamics, especially at the packet, flow, or cell level. Below is a detailed discussion of the key architectures, their evolutions, and the associated challenges.

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), have become foundational tools for sequence modeling in network traffic prediction tasks due to their inherent capacity to maintain temporal dependencies via internal states [118]. LSTMs, first introduced to overcome the vanishing gradient problem encountered by conventional RNNs, employ gating mechanisms that selectively retain or discard information, thereby improving long-term dependency capture [119]. Nonetheless, LSTM's complexity and computational overhead often lead to prolonged training times and potential overfitting, especially on limited datasets.

The introduction of GRUs provided a simpler alternative to LSTMs, achieving comparable predictive performance with fewer parameters and faster convergence [119]. However, GRUs, despite improved efficiency, can struggle in accurately capturing extremely long-range dependencies due to their simpler gate structures. To address the issue of modeling spatial correlations across network elements such as routers or cells, the Convolutional LSTM (ConvLSTM) architecture integrates convolutional layers into LSTM cells. ConvLSTM effectively captures both spatial and temporal dynamics simultaneously, significantly reducing prediction errors compared to traditional LSTM and GRU models for traffic matrices forecasting [119, 120]. Despite these improvements, ConvLSTM's computational cost scales rapidly with input dimensionality, making real-time applications challenging in large-scale networks.

Attention-enhanced variants of RNNs, such as Attention Mechanism-augmented LSTM (AM-LSTM), were introduced to better prioritize critical historical time steps, significantly enhancing short-term forecasting accuracy, particularly for bursty traffic patterns common in data centers [121]. Nevertheless, attention mechanisms add further computational complexity and may require careful tuning to avoid overfitting sparse or noisy network datasets.

Temporal Convolutional Networks (TCNs)

Temporal Convolutional Networks (TCNs) emerged as an efficient alternative to recurrent-based architectures, using dilated causal convolutions to effectively enlarge receptive fields without substantially increasing computational demands [122]. Unlike RNNs, TCNs are inherently parallelizable, enabling efficient GPU training on long sequences typical of VR traffic prediction scenarios. However, purely convolutional structures lack explicit gating mechanisms found in LSTMs or GRUs, potentially limiting their capacity to dynamically adapt to sudden shifts or volatility frequently observed in VR traffic.

Hybrid models, combining TCN with attention-based architectures such as Transformers, have been developed to overcome these shortcomings. Such hybrid models achieve improved accuracy by leveraging the TCN's local temporal feature extraction along with the Transformer's global dependency modeling, especially advantageous for multi-step and long-horizon forecasting [122]. Nevertheless, these hybrid architectures are more computationally intensive, potentially hindering their applicability in resource-constrained edge-computing environments.

Transformer-based Models

Transformer models, originally proposed in natural language processing, employ self-attention mechanisms allowing for parallelized processing and superior long-range dependency modeling compared to RNNs [122]. This self-attention capability uniquely positions Transformers to effectively capture irregular and abrupt fluctuations prevalent in VR network traffic, significantly improving long-term forecasting performance over traditional RNN-based models. However, Transformers require extensive data to generalize effectively, and their computational and memory requirements grow quadratically with sequence length, posing scalability challenges.

To address these scalability limitations, subsequent models introduced sparse attention and hybrid attention-convolution approaches, thereby significantly reducing computational demands while preserving performance advantages over conventional Transformers. Hybrid architectures such as Transformer-TCN models consistently deliver state-of-the-art forecasting accuracy, balancing local and global context modeling [122]. Nonetheless, the complexity introduced by hybridization increases model training difficulty, interpretability challenges, and hyperparameter sensitivity.

Graph-based Deep Learning Models

Graph Neural Networks (GNNs) have recently gained prominence for explicitly modeling inherent spatial structures within network topologies. Spatio-Temporal Graph Neural Networks (ST-GNNs), combining graph convolutional layers with temporal modeling techniques (RNNs or CNNs), efficiently capture both spatial and temporal interdependencies within network traffic data [114]. Such approaches significantly outperform purely temporal methods in scenarios involving traffic forecasting across interconnected network nodes or regions.

Nevertheless, the adoption of graph-based methods entails practical challenges. Firstly, accurate forecasting demands precise and often dynamic network topology data, which might be incomplete, noisy, or expensive to maintain. Secondly, computational costs associated with graph convolutions can limit scalability, especially in densely

connected or large-scale networks. Additionally, graph-based models are sensitive to network topology changes, requiring robust methods to adapt dynamically. While vectorized or convolution-based approaches sidestep explicit graph representation by transforming network structures into 2D grids or matrices, this simplification sacrifices fidelity to the actual network topology and may degrade predictive accuracy [118, 119].

Persistent Challenges in Deep Learning-based Traffic Forecasting

Although deep learning approaches significantly advance network traffic prediction accuracy, several enduring challenges remain:

- **Data Granularity and Quality:** The predictive performance of DL models heavily depends on traffic data granularity (e.g., packet-level, session-level, or cell-level). High-resolution data enables accurate short-term prediction but introduces computational burdens, increased noise, and sparsity issues, whereas coarse granularity may oversmooth critical traffic dynamics [123, 124]. The scarcity of high-quality, publicly available VR-specific traffic datasets exacerbates this challenge, often necessitating reliance on synthetic or proprietary data sources, limiting the generalizability of DL methods.

- **Non-Stationarity and Volatility:** VR traffic is inherently non-stationary due to variability in user behaviors, adaptive streaming policies, and immersive interaction dynamics. Consequently, traffic patterns exhibit abrupt regime changes and volatility spikes that are difficult for traditional DL models to adaptively handle without sophisticated model retraining or online learning strategies [114]. Models robust to such dynamics, such as attention-based and adaptive learning frameworks, partially mitigate this issue but introduce additional computational complexity.

- **Prediction Horizon Trade-offs:** Deep learning models face inherent trade-offs between prediction horizon and forecast accuracy. Although capable of accurately modeling short-term dependencies, longer horizons inevitably introduce larger cumulative errors due to decreasing temporal correlations and increased uncertainty in future traffic states [122, 124]. While approaches such as hierarchical forecasting, recursive predictions, or hybrid architectures attempt to balance this trade-off, determining optimal prediction horizons remains context-dependent and challenging, particularly in VR scenarios demanding real-time responsiveness.

In summary, despite remarkable achievements, deploying deep learning models for VR network traffic forecasting requires careful selection, rigorous hyperparameter tuning, and ongoing adaptation to handle data granularity challenges, non-stationary traffic conditions, and horizon-specific forecasting needs. Future research directions may include developing computationally efficient architectures, robust online learning methods, and strategies leveraging transfer learning to further enhance forecasting reliability and practical utility in immersive network environments.

3.4 The Role of Frequency-Domain Features in Traffic Analysis

3.4.1 Introduction to Frequency-Domain Analysis in Networking

Network traffic is inherently a complex, dynamic, and non-stationary process characterized by temporal fluctuations and variable patterns. While traditional time-domain analyses (e.g., packet arrival rates, inter-arrival times, moving averages) effectively capture immediate behaviors and short-term dynamics, they struggle to identify underlying periodicities, transient anomalies, or subtle recurring patterns hidden within the traffic. Such limitations become particularly pronounced when dealing with network flows exhibiting multi-scale behaviors or long-range dependencies (LRD).

Frequency-domain methods, including Fourier transforms (FT), Short-Time Fourier Transforms (STFT), and Wavelet transforms, provide powerful complementary tools by decomposing the traffic signal into constituent frequency components and their respective amplitudes. Early influential work by Abry and Veitch [125] demonstrated the effectiveness of wavelet transforms in revealing multi-scale traffic variability and accurately estimating the Hurst parameter, which quantifies LRD. While wavelets effectively capture transient behaviors and scale-localized features, limitations persist in their sensitivity to noise and the inherent trade-off between time-frequency resolution. Recent advancements in time-frequency analysis, such as continuous wavelet transforms (CWT), overcome some of these limitations by offering flexible scale resolutions, yet they remain computationally intensive for real-time implementation [106]. Consequently, selecting appropriate frequency-domain methods must balance computational overhead, noise robustness, and the resolution requirements of specific network scenarios.

3.4.2 Frequency-Domain Applications in Network Anomaly Detection and Security

Frequency-domain analysis has significantly advanced anomaly detection, primarily by differentiating malicious activities from normal traffic based on distinct spectral signatures. Initial research demonstrated how malicious behaviors, including Denial-of-Service (DoS) attacks or network scanning, manifest as unique deviations within particular frequency bands. For instance, Lu and Ghorbani's approach [126] employed wavelet transforms and ARX modeling to identify anomalies through localized energy deviations at specific frequency scales. This method effectively distinguished short-term attack bursts from normal diurnal traffic cycles but suffered from limited adaptivity and relatively high false-positive rates under highly variable traffic conditions.

Further improvements emerged with real-time systems such as the Waveman platform developed by Huang et al. [127], leveraging multiple wavelet families (e.g., Coiflet, Daubechies) and various adaptive thresholding strategies. Despite notable success, these methods still required careful manual parameter tuning and were susceptible to performance degradation under highly dynamic and noisy network environments. To overcome these limitations, recent approaches have integrated advanced deep-learning architectures, notably autoencoders, with sophisticated time-frequency transforms such as Continuous Wavelet Transform (CWT), Discrete-Time Fourier Transform (DTFT),

and STFT [106]. These hybrid models benefit from deep neural networks' ability to learn robust feature representations from rich spectral data, thus significantly improving detection performance. Nonetheless, the complexity and interpretability of such models remain challenging, motivating ongoing research into explainable AI methods capable of clarifying learned spectral features. The success of frequency-domain methods in identifying subtle, signature-based deviations for security applications highlights their potential for distinguishing other critical traffic patterns. Specifically, the ability to uncover distinct spectral characteristics, rather than relying solely on temporal correlations, is directly transferable to classifying diverse application-specific behaviors, such as different user states in interactive VR environments.

3.4.3 Frequency-Domain Applications in IoT Periodicity Mining and Botnet Detection

The rapid proliferation of IoT devices introduced pronounced periodic communication patterns into network traffic, driven by routine telemetry, sensor updates, and polling. These patterns create an ideal scenario for applying frequency-domain analysis. Haffey et al. [128] analyzed campus edge network traffic, initially adopting variance-based statistical methods but acknowledged frequency-domain methods' potential advantages in dealing with jitter and noise. Explicitly applying Discrete Fourier Transform (DFT) and periodograms, Price-Williams et al. [129] robustly identified periodic polling behaviors indicative of certain botnets. Although DFT methods successfully detected periodic behaviors, they showed limitations in detecting complex, irregular, or multi-stage botnet activities that exhibited less distinct periodicity.

To address these shortcomings, Meng et al. introduced the DISTR framework [130], combining DFT-derived spectral analysis with autocorrelation functions (ACF). This approach provided greater resilience against irregular traffic patterns and improved the detection of multi-stage botnet communications, which earlier methods failed to capture reliably. Nevertheless, DISTR still exhibits sensitivity to abrupt changes or short-term traffic anomalies, highlighting the need for adaptive frequency-domain methods capable of dynamic frequency band selection or machine-learning-enhanced anomaly detection. The efficacy of frequency analysis in discerning inherent periodicities within IoT traffic underscores its broader applicability to network flows influenced by fixed-rate mechanisms. In interactive VR applications, underlying rendering cycles, server update rates, and user input polling can introduce similar periodic or quasi-periodic patterns into network traffic, making frequency-domain features highly valuable for distinguishing user states and predicting future load based on these rhythmic characteristics.

3.4.4 Research Gap in VR Traffic Analysis

Despite extensive successful applications of frequency-domain analysis across network anomaly detection [106, 126, 127], IoT periodicity mining and botnet detection [128, 129, 130], and other wireless communication contexts, its systematic exploration within the domain of interactive Virtual Reality (VR) network traffic remains notably sparse.

While traditional time-domain analyses (e.g., packet rates or jitter) struggle to capture subtle, underlying patterns, empirical evidence from various interactive VR setups and platforms consistently points to the presence of distinctive periodic and

quasi-periodic traffic fluctuations. For instance, measurements on WebRTC-based VR streaming systems indicate multiple periodic data streams on both uplink and downlink, including frame-aligned video bursts and fixed-rhythm tracking data uploads [17]. Further studies on real VR application traffic similarly reveal downlink data primarily composed of frame-aligned bursts with consistent UDP fragment sizes [46], implicitly suggesting strong spectral components related to rendering rates. Beyond short-term packet-level periodicities, longer-term quasi-periodic peaks (e.g., over minutes) have also been observed in commercial VR headset traffic [28], indicative of application-level phase transitions. Furthermore, the inherent design of XR applications, as reflected in industry standards (e.g., 3GPP XR models), explicitly accounts for periodic video streams (e.g., 60 fps) and fixed-period uplink pose/control flows (e.g., 4 ms interval) [131]. Crucially, user interaction dynamics, such as head movements, are empirically shown to significantly modulate these streams, leaving identifiable rhythmic structures in the network traffic [50].

Despite this compelling and widely observed evidence of inherent periodic and quasi-periodic characteristics within VR traffic, its systematic exploration using frequency-domain analysis for the purpose of comprehensive VR traffic characterization, classification, and prediction remains comparatively under-explored. Current comprehensive surveys on VR networking and Quality of Experience (QoE) optimization, while emphasizing critical issues such as latency minimization, viewport prediction accuracy, and network resource optimization strategies [19, 56, 132], conspicuously omit detailed discussions or methodological frameworks based on frequency-domain approaches for VR traffic characterization. This represents a significant methodological gap: while VR interactions demonstrably produce distinctive spectral signatures that time-domain metrics often miss, these have not been fully leveraged for classification and prediction. Addressing this gap through systematic frequency-domain feature exploration, including identification of characteristic VR spectral signatures, spectral-based traffic classification, and short-term traffic prediction, holds substantial potential to enhance VR QoE management.

3.5 Research Gaps and Justification for this Thesis

The preceding literature review has highlighted considerable advancements in network traffic analysis, especially in the context of virtual reality (VR) and interactive multimedia applications. Nevertheless, several significant research gaps persist, limiting our ability to fully harness the potential of next-generation immersive VR experiences. These gaps underscore critical shortcomings in existing methodologies and datasets, the neglect of advanced analytical dimensions such as frequency-domain insights, and a notable deficiency in practically useful short-term predictive models. This section identifies these gaps in detail and outlines how this dissertation specifically addresses and mitigates each of these challenges.

3.5.1 Lack of State-Aware VR Traffic Analysis and Publicly Labelled Mobility Datasets

While the inherently dynamic and interactive nature of Virtual Reality (VR), driven by continuous user motion and interaction, is widely acknowledged in the literature and

forms a core aspect of the immersive experience, current approaches to VR network traffic analysis often face limitations in fully leveraging this dynamism for fine-grained resource management and predictive optimization. Specifically, while general dynamic VR traffic has been characterized, existing methodologies frequently lack the granularity or explicit state-awareness to differentiate and systematically characterize the distinct traffic signatures arising from specific user mobility states (e.g., periods of active navigation versus complete stillness). This leads to generalized traffic models that mask the significant behavioral differences between these states, hindering truly adaptive network performance optimization.

Crucially, a major impediment to advancing state-aware VR network management is the pronounced absence of publicly accessible, accurately annotated VR traffic datasets that explicitly label and differentiate between various user mobility states. While prior research has acknowledged the potential significance of mobility states, few works have undertaken systematic efforts to segment and label their datasets based on mobility (e.g., stationary versus actively moving states). Instead, they have generally deferred this detailed state-aware analysis to future studies.

This lack of granular, state-specific data prevents the development of truly adaptive and proactive network management strategies. When network controllers lack explicit insight into whether a user is stationary (potentially receiving more environmental updates) or actively moving (generating frequent pose updates), resource allocation, bandwidth reservation, and packet scheduling remain reactive and generic. This inevitably leads to suboptimal resource utilization and compromises the Quality of Experience (QoE) for immersive VR applications, where precise, anticipatory adjustments are paramount.

3.5.2 Underutilization of Frequency-Domain Insights for VR Traffic Analysis

The review of related literature clearly indicates a prevailing dependency on time-domain analytical methods in the classification, modeling, and forecasting of network traffic, even within state-of-the-art machine learning and deep learning frameworks. Conversely, the frequency domain, despite proven effectiveness in analogous fields such as anomaly detection, IoT periodicity mining, and wireless channel characterization, has been noticeably underutilized in VR traffic research.

VR network traffic inherently exhibits periodic and quasi-periodic properties related to rendering cycles, user interaction rates, motion-tracking updates, and periodic synchronization events. These periodic behaviors often manifest clearly in the frequency domain as distinct spectral features, including dominant frequency positions, spectral shape descriptors, and amplitudes, providing discriminative signatures that remain invisible or significantly attenuated in purely time-domain analyses. The systematic neglect of such frequency-domain features represents a considerable gap, limiting the accuracy and resolution of VR traffic classification and prediction models. Exploiting these spectral signatures could significantly enhance differentiation between user states (e.g., stationary versus mobile) and improve traffic forecasting precision at operationally relevant timescales. This gap is further exacerbated by the lack of systematic multi-scale analysis, which is crucial for capturing VR traffic dynamics over different temporal resolutions, from rapid sub-second bursts to more sustained patterns.

3.5.3 Need for Practical Short-Term Forecasts for Real-Time VR Resource Management

Current forecasting techniques have been insufficiently validated in the context of VR's unique traffic characteristics—namely, high volatility, non-stationarity, and stringent latency requirements. Traditional statistical models, such as ARIMA and Kalman filters, struggle to capture the intrinsic non-linearity and complex, long-range dependencies inherent to VR traffic. While modern deep learning approaches like LSTM, GRU, and Transformers have shown promise, they still face difficulties with prediction granularity, non-stationary data distributions, and maintaining the delicate balance between forecast horizon length and model accuracy.

Critically, there is a notable gap in comprehensive, empirical evaluations of short-term VR traffic prediction models at practically relevant prediction horizons—particularly within ultra-short to medium-short timeframes (e.g., between 80 ms to several hundred milliseconds). Such timescales are essential for enabling adaptive resource management schemes, including semi-persistent scheduling and congestion control, which must respond proactively rather than reactively. Without accurate and reliable traffic forecasts at these critical, operationally relevant timescales, network controllers default to conservative or reactive approaches, inevitably diminishing resource efficiency and user QoE. Furthermore, a significant gap exists in directly integrating these short-term traffic predictions into end-to-end network scheduling mechanisms, particularly in closing the control loop to demonstrate tangible QoE improvements through proactive resource allocation at the MAC layer.

3.5.4 Thesis Justification: Bridging the Identified Gaps through a Holistic Methodology

To address these critical research gaps comprehensively, this thesis adopts a holistic and integrated methodological framework, specifically designed to:

1. **Develop a Unique, State-Aware VR Traffic Dataset:** To address the data scarcity, this work introduces and publicly releases a collected VR traffic dataset, specifically designed for high fidelity and explicit user state-awareness. The rigor in its labeling and construction is achieved through a controlled and verifiable protocol: user mobility states (stationary vs. moving) were established by conducting data capture sessions under strictly controlled experimental conditions, where the player was explicitly instructed to maintain one distinct state (e.g., completely motionless for stationary, continuous navigation for moving) for the entire capture duration, with this instruction-based ground truth precisely cross-verified and augmented by analyzing application-level logs from the custom-built Unity VR application, specifically leveraging the presence, frequency, and content of movement-related network messages (e.g., `NetworkTransformMessage`) to confirm the intended user behavior. Network-level packet captures (`Wireshark .pcap` files) and application-level Unity logs (`.txt` files) were simultaneously recorded on their respective machines, and a robust, custom-developed data fusion pipeline aligned packet arrival timestamps (`frame.time` from Wireshark) with Unity application timestamps (`time` from logs) by uniquely identifying and matching serialized payload content (`data.data` from network packets to `SerializeData` from Unity logs). This content-based matching, combined with timestamp proximity

(using an empirically determined threshold of $\varepsilon = 10$ ms), ensured high-precision correspondence between observed network events and their precise application context, minimizing potential misalignments. While collected within a single custom-built VR environment and using a specific hardware platform (Meta Quest 3 headset), the dataset provides a balanced and representative sample of traffic characteristics across the two critical user mobility phases (stationary and moving) for both server-to-client and client-to-server communication direction. The released dataset, preprocessing code, feature extraction scripts, and reproducibility documentation are available through the permanent commit-pinned repository link https://github.com/bozliu/vr-traffic-prediction-sps/tree/36a2d4d73c80c658d3d9174dcc1e9a08e9fc3922/data/raw. The companion Unity VR game used to generate the packet traces is also publicly released at https://github.com/bozliu/vr-traffic-vr-game/tree/893245e502fcaed43bd9104fac853acd8178d0d5. This release is in line with principles from "Datasheets for Datasets" [133] by providing transparent documentation, preprocessing details, and reusable scripts for reproducibility and responsible community use.

2. **Incorporate Hybrid Time-Frequency Feature Engineering:** This dissertation systematically investigates and demonstrates the efficacy of incorporating multi-scale frequency-domain spectral features, derived from Fourier analysis across 0.5s, 1.0s, and 2.0s time windows, into VR traffic analysis. While Fourier Transform and frequency-domain analysis are established techniques in various fields, including general signal processing, audio analysis, and indeed, some applications within network anomaly detection and time series forecasting, their systematic and comprehensive application to distinctly characterize and classify interactive VR user states (stationary vs. moving) and subsequently enhance short-term VR traffic load prediction has been largely underexplored. Our work specifically highlights the utility of carefully selected Nyquist-normalized frequency positions and spectral shape descriptors (centroid, flatness), alongside the direction bit, which our empirical analysis demonstrates to be exceptionally discriminative for VR user state classification, potentially reflecting underlying engine update cycles or user interaction frequencies, and revealing patterns often obscured in purely time-domain views (see feature-importance analysis, Chapter 5, Fig. 5.3). Crucially, through ablation studies, we empirically validate that this hybrid approach substantially improves VR user-state classification, from 68.95% with time-only features to 93.74% with the full hybrid set, and raises the Moving-class F1-score from 0.5937 to 0.9153 (Chapter 5, Table 5.1). This systematic integration, combined with the quantifiable empirical validation of the discriminative power of specific spectral features for VR mobility states, contributes a novel and highly effective perspective to VR network traffic management tailored for immersive applications.

3. **Establish Robust Short-Term Forecasts for VR Resource Management:** Through extensive comparative evaluations involving state-of-the-art deep learning methods (LSTM, GRU, Transformer), this thesis demonstrates robust forecasting at 200 ms ($R^2 \approx 0.86$) and a useful transitional level at 150 ms ($R^2 \approx 0.61$); 80 ms remains challenging (Chapter 6, Table 6.2). These horizons are critically relevant for modern cellular network operations and interactive VR ap-

plications, reflecting end-to-end control latencies in real systems.

From a network system engineering perspective, typical Round-Trip Times (RTTs) in public 4G/5G networks often fall within the tens of milliseconds range [134]. Real-time media stacks (e.g., WebRTC) commonly aim for sub-200ms end-to-end delays [135]. To enable *proactive* rather than *reactive* network adjustments—such as dynamic bandwidth reservation, buffer management, or congestion control mechanisms—forecasts must provide a sufficient lead time, typically corresponding to 1 to 3 RTTs [136], which translates to an operational window in the order of 10^2 ms. A practical budget for a complete network control cycle, encompassing a measurement window (e.g., 20–40ms), decision-making and signaling (e.g., 5–10ms), network transmission and queueing (e.g., 20–50ms), and the time for media encoder or jitter buffer adjustments to take effect (e.g., 20–60ms), cumulatively lands within approximately 80–160ms. Therefore, the 150–200ms prediction horizon directly encompasses and supports these critical operational timescales.

From a user experience (QoE) and VR application perspective, while the ultimate goal for immersive VR is an extremely low motion-to-photon (MTP) latency (ideally below 20ms for comfort) [44], the *prediction horizon* itself concerns the *resource control lead-time* necessary for the network to react effectively. Selecting horizons in the 150–200ms range (with 200 ms strongest) allows control actions to be initiated and materialize *before* noticeable QoE regressions occur due to traffic fluctuations, ensuring a smoother and more immersive user experience. This range also represents a pragmatic balance, as ultra-short predictions (e.g., $<$80ms) are often inherently unreliable due to the very high-frequency, noisy fluctuations in packet-level traffic, where averaging over too few packets does not effectively smooth out transient variability and obscures underlying trends.

4. **Integrate Predictions into an End-to-End Predictive Scheduling Framework for VR QoE:** To bridge the gap between abstract traffic forecasts and tangible QoE improvements, this thesis develops and validates a novel end-to-end predictive resource scheduling framework. This framework seamlessly integrates the 200ms VR traffic predictions into a 1ms MAC layer scheduler using a hybrid Semi-Persistent Scheduling (SPS) and Proportional Fair (PF) strategy. Through comprehensive simulations with a realistic multi-user VR traffic simulator, we demonstrate that this prediction-aided SPS approach significantly improves delay fairness for VR users by approximately 3.0-3.5% and reduces tail latencies (P95 mean delay) critical for VR QoE, all while rigorously maintaining full network throughput. The framework's robustness is verified across various stress scenarios, highlighting its particular efficacy under bursty and highly multiplexed traffic conditions. Furthermore, the high Jaccard overlap (approximately 0.91) between our predictive SPS user selection and that of an ideal, oracle-based scheduler serves as strong evidence of the practical utility and accuracy of our prediction models in strategic resource management decisions (Chapter 7, Table 7.5). This crucial step closes the loop from data collection and feature engineering to real-time resource control, directly linking traffic prediction to measurable QoE enhancements.

Chapter 4

VR Traffic Dataset and Experimental Methodology

This chapter provides a comprehensive description of the VR network traffic dataset, elaborating on the systematic data collection methodology and the subsequent preprocessing pipeline. Given the data-driven nature of the proposed network traffic classification, modeling, and prediction methods, ensuring dataset robustness, reproducibility, and precision is essential. Therefore, meticulous attention is given to the details of the VR application design, experimental configurations, data capture, and associated mathematical definitions. To foster reproducibility and support the wider research community, the complete VR traffic dataset, including raw network traces, application logs, preprocessing utilities, feature extraction code, forecasting scripts, and SPS scheduling simulation code, has been made publicly available via the permanent commit-pinned GitHub link https://github.com/bozliu/vr-traffic-prediction-sps/tree/36a2d4d73c80c658d3d9174dcc1e9a08e9fc3922/data/raw. Additionally, the custom-built Unity VR application used for traffic generation is available at https://github.com/bozliu/vr-traffic-vr-game/tree/893245e502fcaed43bd9104fac853acd8178d0d5.

4.1 VR Application and Data Collection Environment

The custom-built VR application environment was carefully designed and implemented to systematically generate controlled, replicable network traffic, facilitating direct correlations between user activity states and measurable network phenomena. The precision achieved in defining, controlling, and monitoring user-generated VR network traffic provides essential ground-truth data for subsequent traffic classification and predictive modeling.

4.1.1 Custom-built Unity VR Application Design

The VR application was developed using the Unity game engine (version 2022.3.5.1f1), selected due to its versatility, built-in networking functionalities, and suitability for VR research. A carefully structured virtual environment was established, consisting of multiple interconnected zones ("islands") linked by clearly defined transitional paths, as shown in Figure 4.1. These clearly demarcated zones allowed explicit definition of discrete user states, notably stationary states (denoted as S) when users are actively

interacting within islands, and moving states (denoted as M) when users navigate between islands along defined paths. Mathematically, each user state $U(t)$ at any given time t during the VR session can thus be categorized as:

$$U(t) \in \{S, M\} \tag{4.1}$$

Network communication was managed using Unity's Netcode for GameObjects, chosen explicitly for its streamlined approach to handling synchronized communication between client and server. The synchronization of player transforms involved continuous streaming of position and rotation information. Position at time t, represented mathematically as $P(t)$, consists of a 3-dimensional vector indicating coordinates $(x(t), y(t), z(t))$, while rotation is described using quaternion representation $Q(t) = (q_w(t), q_x(t), q_y(t), q_z(t))$. The combined player state update message $M_{\text{state}}(t)$ at any timestamp t is thus:

$$M_{\text{state}}(t) = \{P(t), Q(t), \Delta t\} \tag{4.2}$$

where Δt is the update interval between consecutive state updates. These periodic state-update messages form the primary component of the generated network traffic, providing explicit linkage between in-game user states and corresponding network behavior.

Object synchronization and Remote Procedure Calls (RPCs) were similarly structured. Each RPC triggered by a specific interactive event in the virtual world environment is represented by a binary indicator function $\mathcal{I}(e_i, t)$, where:

$$\mathcal{I}(e_i, t) = \begin{cases} 1, & \text{if event } e_i \text{ occurs at time } t \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

These event indicators allow precise correlation between user interaction events and spikes or characteristic bursts observed in network traffic patterns. The structured nature of these generated events provides a detailed mechanism for subsequent traffic classification modeling.

4.1.2 Hardware and Software Setup for Data Capture

The hardware and software experimental setup was designed with stringent control criteria to achieve reproducible and precise network measurements. The client-side interface employed a Meta Quest 3 VR headset, while the server-side application instance was executed on a powerful workstation equipped with an Intel Core i9 CPU, 64 GB of RAM, and running Ubuntu Server 20.04 LTS. This high-performance configuration ensured minimal latency and negligible computational bottlenecks, satisfying rigorous requirements for accuracy in measurement and data generation. All experiments were conducted with a fixed application configuration. The VR application streamed at a native resolution of 2064x2208 per eye at a refresh rate of 90 Hz. The Unity Netcode for GameObjects utilizes its default, dynamic bitrate policy, which adapts to network conditions but maintains a consistent underlying update frequency for world ticks (server-side, 30 Hz). The specific game level (scene) used for all data collection was the "Main Island Environment" described in Section 4.1.1.

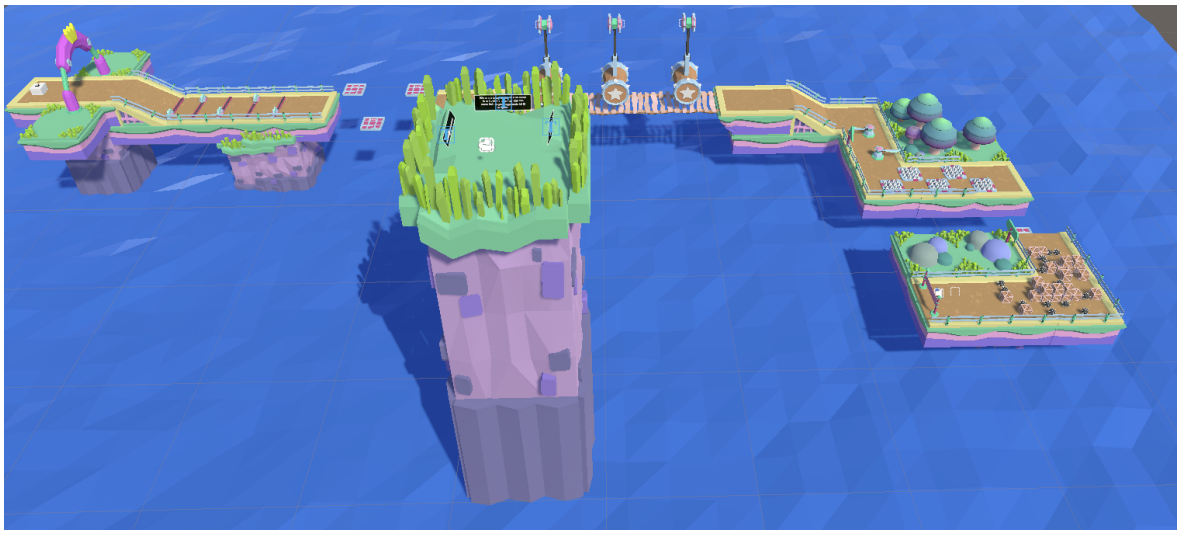


Figure 4.1: Overview of the custom-built VR game environment. The environment comprises multiple islands connected by defined paths. Players navigate between islands, alternating systematically between stationary interactions and moving states.

Both devices communicated within a dedicated Gigabit Ethernet Local Area Network (LAN) environment to eliminate extraneous interference and ensure stable network latency. To formally quantify network performance, latency (L) between client and server was consistently verified to remain within acceptable bounds (i.e., typically $L < 1$ ms), and bandwidth (B) consistently maintained at approximately 1 Gbps, ensuring the following constraints:

$$L_{\text{LAN}} < 1\,\text{ms}, \quad B_{\text{LAN}} \approx 1\,\text{Gbps} \tag{4.4}$$

Network traffic capturing was performed using Wireshark (version 4.0.5). The specific arrangement involved placing a dedicated monitoring workstation (or laptop) connected via a mirrored (SPAN) Ethernet port on the switch, operating in promiscuous mode to passively collect and timestamp every UDP packet transmitted. For any packet P_i captured at timestamp t_i, relevant metadata logged included packet length (l_i), packet direction (D_i, indicating uplink UL or downlink DL), source IP address IP_{src}, destination IP address IP_{dst}, and UDP port numbers:

$$P_i = \{t_i, l_i, D_i, IP_{\text{src}}, IP_{\text{dst}}, \text{Port}_{\text{src}}, \text{Port}_{\text{dst}}\} \tag{4.5}$$

Furthermore, application-level logging was concurrently conducted by the Unity application itself on both client and server. The detailed log entries A_j generated by Unity at event timestamps t_j captured higher-level contextual metadata, including event identifiers (e_j), associated in-game states, and precise timestamps. Formally, an application log entry can be defined as:

$$A_j = \{t_j, e_j, U(t_j), P(t_j), Q(t_j)\} \tag{4.6}$$

Thus, pairing these application log entries A_j with corresponding network packets P_i allows the construction of an accurately labeled dataset, where network traffic features can be directly correlated with clearly defined, ground-truth user events and

states. The data collection process spanned multiple distinct recording sessions, totaling approximately 42.6 minutes of recorded gameplay. This overall duration comprises a dedicated stationary session of approximately 29.3 minutes and a dedicated moving session of approximately 13.5 minutes. Each session involved players strictly adhering to the defined stationary or moving protocols. The presented statistics and distributions are therefore aggregated from these comprehensive traces, ensuring that the observed patterns are consistent and statistically robust, rather than merely a one-off anomaly from a single, short recording. This aggregation, coupled with the large number of collected packets, a grand total of 121,220 packets across all states and directions, lends significant statistical strength to our observations regarding VR traffic characteristics.

4.2 Data Capture and Labelling Protocol

Accurate labeling of network traffic with corresponding user states represents a fundamental prerequisite for subsequent analysis, modeling, and prediction in VR network traffic research. To accomplish this, a rigorous and systematic protocol was established to clearly define, elicit, and identify distinct user behavioral states during the network traffic capture sessions. This protocol ensures the generation of reliable ground truth labels, essential for training and validation of supervised machine learning models presented later in this dissertation.

4.2.1 Defining and Eliciting Stationary and Moving User States

Two clearly defined user states—*stationary* (S) and *moving* (M)—were selected as representative operational modes within interactive VR environments. The choice of these states was guided by their anticipated differential impact on the volume, periodicity, and burstiness of the resulting network traffic, thereby enabling clear distinctions in subsequent modeling and classification tasks.

The stationary state was defined by instructing the participant to remain completely motionless within the predesignated interactive zones, or "islands," throughout the entire duration of each dedicated capture session. Specifically, the participant was asked to refrain from any input (e.g., keyboard or controller joystick) that would initiate avatar locomotion or interaction within the virtual environment. While the VR headset itself might still transmit baseline pose data (e.g., for maintaining a stable view), this state aimed to capture network traffic primarily comprising passive environmental updates, such as heartbeat messages, synchronization of ambient objects, and periodic state-maintenance communications, without the influence of active user-driven avatar movement. The captured data under this condition therefore represents the background or idle-state network traffic.

In direct contrast, the moving state protocol required the participant to continuously navigate along predefined paths linking the interactive islands. During these sessions, the participant actively performed avatar-level locomotion (e.g., continuous walking or strafing through keyboard/joystick input) to move their virtual character within the environment. This means "moving" in our study specifically refers to network traffic patterns driven by the user's explicit control over their virtual avatar's position and orientation within the game world. Crucially, this state was defined by

the avatar's movement status, not by the physical movement or rotation of the Head-Mounted Display (HMD) itself. Autonomous scene animations such as particle effects were not considered as criteria for defining or eliciting this state. They contribute to the overall background content dynamics but do not dictate the 'moving' label for our purpose. The network traffic observed in this state is primarily driven by the continuous transmission of avatar position and orientation updates and associated game logic messages.

To ensure unequivocal labeling and minimize ambiguity, these two states (S and M) were captured in strictly separate, dedicated recording sessions. This experimental design provided the high-level, session-based ground truth label for the entire duration of each capture. Each session thus produced clearly identifiable segments of VR-generated network traffic corresponding exclusively to a single, predefined user state. This rigorous experimental procedure allowed subsequent packet labeling, yielding a labeled dataset with high-fidelity ground truth:

$$L(P_i) \in \{S, M\}, \quad \forall P_i \text{ captured during session.}$$

Here, $L(P_i)$ denotes the assigned label (either Stationary S or Moving M) for the i-th captured network packet P_i. This labeling reflects the user's behavioral state during the specific session in which the packet was observed. For detailed statistics on data collection duration and total packet counts, refer to Section 4.1.2.

4.2.2 Simultaneous Packet Capture and Application-Level Logging

To facilitate the accurate annotation of captured network packets with corresponding user states and application-level semantics, a comprehensive dual-layer logging strategy was employed. This procedure entailed simultaneous and synchronized logging at both network and application levels, thereby creating a robust, semantically enriched dataset.

At the **network level**, Wireshark (version 4.0.5) was configured to capture all exchanged UDP packets between the VR client (Meta Quest 3 headset) and server. The packet capture files (`.pcap`) contain sequentially indexed packet entries P_i, each characterized by several critical metadata fields:

- Packet capture timestamp: $t_{\text{pcap}}(P_i)$, denoting packet arrival times with microsecond-level accuracy.
- UDP payload length: $l_{\text{udp}}(P_i)$, expressed in bytes, providing direct quantitative measures of network data volume.
- Raw UDP payload data: $D_{\text{raw}}(P_i)$, a hexadecimal representation temporarily retained for matching and alignment with corresponding application-level logs.

Thus, each captured packet P_i can be formally represented as a tuple:

$$P_i = \{t_{\text{pcap}}(P_i), l_{\text{udp}}(P_i), D_{\text{raw}}(P_i)\}$$

At the **application level**, the Unity-based VR application produced detailed and structured log entries A_j stored as plain-text files. Each application log entry provided

semantic interpretation of each corresponding network message, along with its internal Unity timestamp $t_{\text{unity}}(A_j)$, facilitating direct alignment with Wireshark timestamps. Specifically, the Unity application logs detailed the following fields:

- Unity timestamp: $t_{\text{unity}}(A_j)$, aligning internal application events to external network events.
- Message type: $\tau(A_j) \in \{\text{NetworkTransform}, \text{RPC}, \text{ObjectSync}\}$, denoting semantic categories.
- Game object identifier: $\text{ID}_{\text{obj}}(A_j)$, uniquely associating each message with specific game elements or player actions.
- Serialized payload: $\text{SerializeData}(A_j)$, used to bridge and match the raw UDP payload $D_{\text{raw}}(P_i)$ from Wireshark captures.

Formally, each application log entry A_j is represented as:

$$A_j = \{t_{\text{unity}}(A_j), \tau(A_j), \text{ID}_{\text{obj}}(A_j), \text{SerializeData}(A_j)\}$$

This rigorously synchronized, dual-layer logging approach ensures precise correspondence between low-level network traffic features and high-level user actions, essential for subsequent in-depth analysis, classification, modeling, and prediction tasks.

4.3 Data Preprocessing and Fusion Pipeline

The raw, heterogeneous data collected from Wireshark and Unity application logs required a rigorous, multi-stage preprocessing and fusion pipeline to ensure data integrity, synchronization, and extraction of meaningful, representative features. This meticulous data preparation process is crucial, as subsequent analytical accuracy and machine learning model performance rely heavily on the quality and coherence of the dataset used for training and evaluation.

4.3.1 Wireshark Capture Filtering and Extraction

Initially, raw network packet captures, stored in binary `.pcap` format, were programmatically processed using `tshark`, the command-line interface of Wireshark. The `.pcap` files were converted into structured JSON format to facilitate efficient, automated parsing. The primary motivation for selecting JSON as an intermediate format was its readability, structured nature, and compatibility with Python-based analytical pipelines. Specifically, the command utilized for the transformation was similar to:

```
tshark -r input.pcap -T json -Y "udp" > output.json
```

This command explicitly filtered packets to retain only User Datagram Protocol (UDP) traffic. The selection of UDP as the primary focus was based on the recognition that modern real-time game engines, including Unity's Netcode for GameObjects, predominantly utilize UDP due to its lower latency and reduced overhead compared to TCP, making UDP particularly relevant for VR scenarios.

During this initial filtering phase, packet entries P_i in the JSON output retained the following critical fields:

- Packet capture timestamp: $t_{\text{pcap}}(P_i)$, recorded with microsecond-level precision.
- UDP payload length: $l_{\text{udp}}(P_i)$, measured in bytes, directly reflecting application-layer data traffic volume.
- UDP payload content: $D_{\text{raw}}(P_i)$, initially represented in hexadecimal format, later decoded into string form for matching with Unity logs.

Packets identified as incomplete, corrupted, or unrelated (e.g., background DHCP, DNS, or ARP messages) were systematically removed through automated validation scripts. This step ensured data quality and relevance for subsequent fusion and analysis tasks.

4.3.2 Unity Log Parsing and Semantic Contextualization

Parallel to network-level preprocessing, Unity application logs, initially stored in plain-text (`.txt`) format, were parsed using custom Python scripts. Regular expressions were crafted to extract structured metadata corresponding to each network event logged by Unity, thereby transforming unstructured logs into tabular data entries A_j. Each structured Unity log entry comprised the following attributes:

- Unity internal timestamp: $t_{\text{unity}}(A_j)$, capturing the exact application event time.
- Message type: $\tau(A_j)$, indicating categories such as `NetworkTransform`, `RPC`, or `ObjectSync`.
- Game-object identifier: $\text{ID}_{\text{obj}}(A_j)$, associating log entries with specific game entities or players.
- Serialized data content: SerializeData(A_j), textual encoding of the network message payload.

An additional filtering step targeted redundant or irrelevant server-side log entries. These often included synchronization messages for non-player or static objects. Such entries were pruned to ensure that the dataset primarily contained player-centric traffic, consistent with the perspective of user-state classification and prediction. The detailed 'SerializeData' and message types provided by these logs are crucial for later semantic analysis and precise filtering of only VR-application generated traffic from the raw Wireshark captures.

4.3.3 Data Fusion and Alignment

The fusion stage represented the methodological core of the preprocessing pipeline, integrating the network-level and application-level datasets into a unified, temporally synchronized, and semantically annotated dataset. While Wireshark captures provide fundamental network-level metrics (e.g., timestamps, packet sizes, and implicitly, communication direction via IP addresses), they inherently lack application-layer context. The critical necessity for fusing Unity application logs with Wireshark packet data stems from two main objectives. One objective is precise semantic annotation and contextualization. Unity logs contain crucial application-specific message types ($\tau(A_j)$) and associated game object identifiers ($\text{ID}_{\text{obj}}(A_j)$), which are not discernible

from raw Wireshark data without deep packet inspection (DPI) specific to Unity's proprietary network protocol. By matching the 'SerializeData' from Unity logs to the D_{raw} payload from Wireshark, we can accurately append this high-level semantic information to each network packet record. This allows us to understand what specific in-game events (e.g., player transform updates, remote procedure calls, object synchronization) are generating the observed network traffic, providing invaluable context for feature engineering and interpretation of results. Another objective is to accurate filtering of relevant game traffic. A Wireshark capture typically contains all network traffic on the monitored interface, including background operating system communications (e.g., DNS, ARP, DHCP) or generic network control messages that are not directly related to the VR application's gameplay or user state. By performing a content-based match using 'SerializeData' and D_{raw} , we can precisely identify and isolate only those UDP packets that originated from or were destined for the Unity application's game logic. This meticulous filtering ensures that the dataset exclusively contains VR game traffic, eliminating noise and irrelevant data points that could otherwise confound subsequent analyses and model training.

To achieve this fusion, a robust, dual-criterion matching algorithm was employed, addressing minor system clock discrepancies between Wireshark and Unity. First, a *content-based matching* step was performed. The serialized Unity message payload SerializeData(A_j) was converted into a standardized format and compared against each Wireshark packet's decoded UDP payload $D_{\text{raw}}(P_i)$. The matching criterion is expressed via the binary indicator function:

$$\delta_{\text{content}}(P_i, A_j) = \begin{cases} 1, & \text{if } D_{\text{raw}}(P_i) \equiv \text{SerializeData}(A_j) \\ 0, & \text{otherwise} \end{cases}$$

Second, a *temporal proximity validation* was conducted. Each Wireshark timestamp $t_{\text{pcap}}(P_i)$ was compared with the corresponding Unity timestamp $t_{\text{unity}}(A_j)$. A time difference threshold ϵ was applied:

$$|t_{\text{pcap}}(P_i) - t_{\text{unity}}(A_j)| \leq \epsilon$$

where $\epsilon = 10$ ms was empirically selected based on synchronization trials. Only packet-log pairs satisfying both criteria ($\delta_{\text{content}} = 1$ and time-difference condition) were retained.

Packets that could not be confidently matched to a Unity log entry (e.g., generic network control messages like pings or unexpected application-level errors) or those deemed irrelevant to the core VR experience (e.g., non-game-related background traffic from the OS) were systematically discarded. This rigorous fusion process yielded four distinct, semantically rich, and time-aligned datasets: stationary server-to-client, moving server-to-client, stationary client-to-server, and moving client-to-server. Each record in these datasets accurately links network packet characteristics (timestamp, size) to its application context and corresponding user state. The resulting fusion yielded four temporally aligned and semantically annotated traffic datasets:

$$D_{\mathrm{SC}}^{(S)} : \text{Stationary, Server-to-Client}$$
$$D_{\mathrm{SC}}^{(M)} : \text{Moving, Server-to-Client}$$
$$D_{\mathrm{CS}}^{(S)} : \text{Stationary, Client-to-Server}$$
$$D_{\mathrm{CS}}^{(M)} : \text{Moving, Client-to-Server}$$

Each final labeled dataset entry L_i was represented as:

$$L_i = \{t_{\mathrm{pcap}}(P_i),\ l_{\mathrm{udp}}(P_i),\ \tau(A_j),\ \mathrm{ID}_{\mathrm{obj}}(A_j),\ L(P_i)\}$$

where $L(P_i) \in \{S, M\}$ indicates the ground-truth user state (stationary or moving).

4.3.4 Dataset Availability and Reproducibility

To support reproducibility and community reuse, the dataset, preprocessing utilities, feature extraction pipeline, forecasting scripts, and SPS scheduling simulation code have been released in a public companion repository:

https://github.com/bozliu/vr-traffic-prediction-sps/tree/36a2d4d73c80c658d3d9174dcc1e9a08e9fc3922/data/raw

The repository includes the released four-scenario VR traffic dataset under the `data/raw/` directory, together with scripts for reproducing the classification, forecasting, and prediction-aided SPS scheduling experiments reported in this thesis. The repository version corresponding to this thesis correction is identified by commit `36a2d4d73c80c658d3d9174`. If a DOI or archived release is created after submission, the DOI/release should be treated as the permanent archival version of the dataset and code.

The custom Unity VR application used to generate the traffic traces has also been released publicly as a companion repository:

https://github.com/bozliu/vr-traffic-vr-game/tree/893245e502fcaed43bd9104fac853acd8178d0d5

This repository documents the Unity scene, gameplay/runtime scripts, research logging hooks, and packet-trace generation pipeline used to produce the Unity-side logs later aligned with Wireshark captures. The repository version corresponding to this thesis correction is identified by commit `893245e502fcaed43bd9104fac853acd8178d0d5`. Together, these two repositories provide the experimental artefacts required to inspect the data-collection environment, reproduce the preprocessing path, and rerun the main analysis and scheduling workflows.

4.3.5 Timestamp Standardization and Relative Time Representation

Ensuring precise temporal alignment across all datasets is imperative, particularly due to the sensitivity of time-series analyses and prediction models to minor timing discrepancies. Initially, raw timestamps extracted from both Wireshark captures (`frame.time`) and Unity application logs (`time`) exhibited heterogeneous formats and inherent offset

discrepancies due to differences in internal clock synchronization. Consequently, a systematic standardization process was implemented, converting every timestamp into a uniform and high-resolution Python `datetime` object with microsecond-level precision. The rationale behind selecting Python's native `datetime` format is its precise temporal resolution, facilitating accurate synchronization, alignment, and subsequent calculation of inter-packet intervals.

After this conversion step, all records within each specific subset—namely, stationary server-to-client ($D_{SC}^{(S)}$), stationary client-to-server ($D_{CS}^{(S)}$), moving server-to-client ($D_{SC}^{(M)}$), and moving client-to-server ($D_{CS}^{(M)}$)—were rigorously sorted chronologically. This explicit sorting step was essential to correct and prevent temporal anomalies potentially introduced during data acquisition, such as packet misordering caused by capture latency variations or slight network jitter. Formally, given a dataset $D = \{P_i\}$, sorting entailed rearranging all packet timestamps t_i to ensure strict monotonicity:

$$t_i \leq t_{i+1}, \quad \forall i \in [1, N-1] \tag{4.7}$$

Subsequent to chronological sorting, an additional normalization step was conducted to transform absolute wall-clock timestamps into relative timestamps. This transformation is critical, as relative timing provides direct comparability across different sessions and user states, removing arbitrary variations induced by differing start times of recording sessions. Specifically, within each dataset, the earliest observed timestamp $t_{\min}$ was identified and subtracted from all subsequent timestamps to yield a normalized, zero-based relative time series. Thus, the relative timestamp T_i for each packet i within the dataset is mathematically defined as:

$$T_i = t_i - t_{\min}, \quad i \in [1, N] \tag{4.8}$$

This normalization ensures consistent comparative analysis across multiple sessions, facilitating robust feature extraction and machine learning model training independent of the initial temporal offsets.

4.4 Exploratory Data Analysis and Dataset Characteristics

Before proceeding to feature engineering and model development, a detailed exploratory data analysis (EDA) was performed on the preprocessed VR network traffic dataset. The main objectives of this analytical phase were to systematically explore the inherent statistical characteristics of packet sizes and inter-arrival times, assess the influence of user mobility states (stationary vs. moving) on these distributions, and thereby inform subsequent feature selection and model development choices. In this section, the analysis and findings regarding packet size distributions are presented comprehensively.

4.4.1 Analysis of Packet Size Distributions

Understanding the distribution of packet payload sizes is crucial, as it reflects the type, frequency, and nature of the data being transmitted within the VR application. Let X represent the random variable for packet payload size (measured in bytes), with each observation x_i corresponding to the payload size of the i^{th} recorded packet.

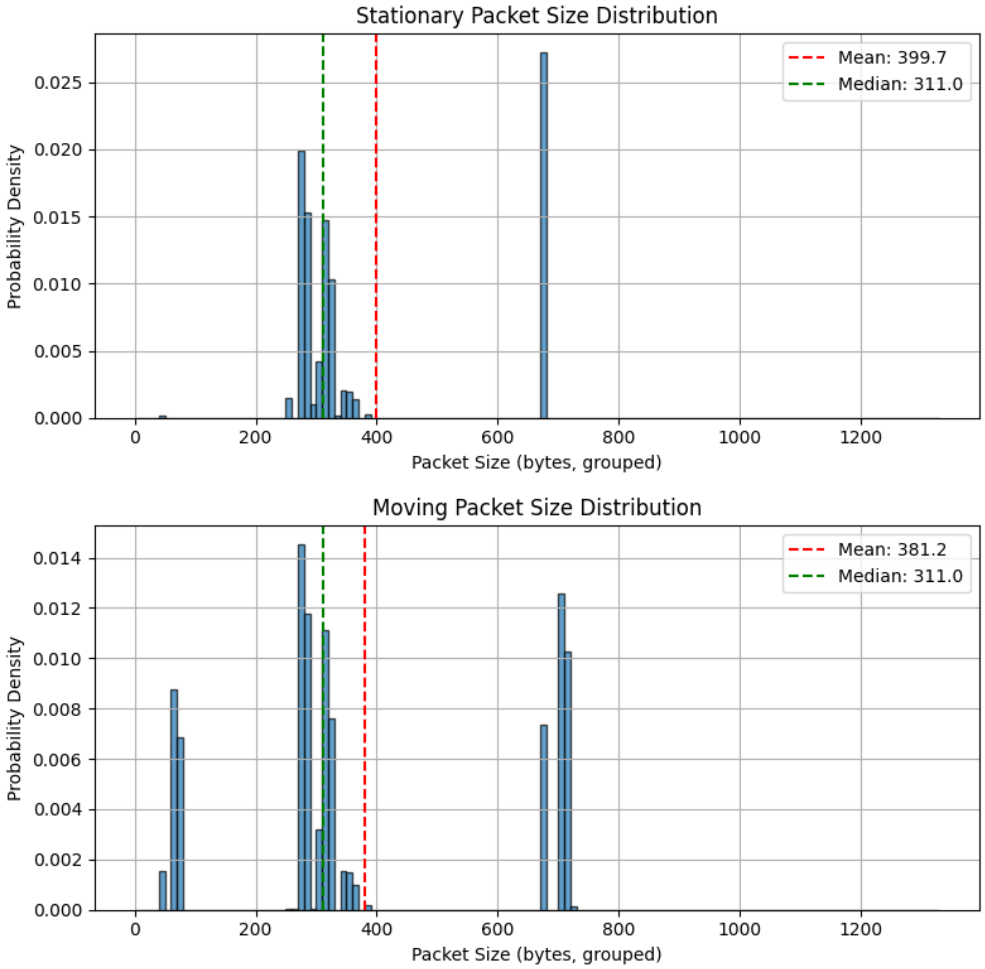


Figure 4.2: Empirical probability density functions of packet payload sizes for (a) stationary and (b) moving user scenarios. The mean payload size (red dashed line) and median payload size (green dashed line) are marked for clear comparison.

To thoroughly examine these characteristics, empirical probability density functions (PDFs) of packet sizes were computed using kernel density estimation, allowing for clear visualization and interpretation of the distributions across different mobility states.

Figure 4.2 illustrates the empirical probability densities of packet payload sizes observed for both stationary and moving user states. These visualizations aggregate data from both client-to-server and server-to-client communication flows, thereby providing comprehensive insights into the overall packet transmission characteristics of each mobility scenario. Crucially, these distributions and the following statistical values (mean, median) are derived from the entire aggregated dataset, encompassing a grand total of 121,220 packets collected over approximately 0.71 hours of recorded gameplay. This extensive collection, conducted under controlled stationary and moving protocols, ensures that the observed patterns are statistically robust and indicative of the respective user states within our custom VR application, rather than ephemeral, once-off behaviors. The stationary state data alone spans approximately 29.3 minutes of continuous recording, while the moving state data accounts for approximately 13.5 minutes, providing substantial temporal coverage for both scenarios.

A detailed examination of Figure 4.2 reveals several notable characteristics. In the stationary user scenario (Figure 4.2a), packet payload sizes have an observed mean ($\bar{X}_{\text{stationary}}$) of approximately 399.7 bytes and a median value of 311.0 bytes. The payload size distribution exhibits several distinct peaks, notably around the median region and another prominent peak near 600 bytes. This distribution suggests that even when the user remains stationary, there is significant traffic attributable to continuous state synchronization, periodic updates, or occasional asset transmissions necessary to maintain the VR environment's integrity and coherence.

In contrast, the moving state scenario (Figure 4.2b) exhibits a slightly lower mean payload size ($\bar{X}_{\text{moving}} \approx 381.2$ bytes), yet shares an identical median payload size of 311.0 bytes. While some commonalities remain—such as the prominent peak around the 300-byte range, likely indicative of routine synchronization packets—the moving state distribution distinctly reveals an additional significant mode located within the smaller packet size range of approximately 50 to 150 bytes. This secondary mode is considerably more pronounced during the moving state compared to the stationary state, suggesting its direct association with mobility-induced network traffic patterns.

Further analysis suggests that this secondary mode is attributed primarily to small, periodic updates concerning the user's positional and rotational transforms (e.g., data on head-mounted display and controller tracking), which become frequent and critical during active navigation within the VR environment. Such data packets ensure smooth and continuous user experiences by maintaining positional accuracy and visual synchronization with user movements. Conversely, their relative scarcity or absence in the stationary scenario is intuitive, as minimal or no user positional changes occur during stationary interactions.

The distinct differentiation in packet size distributions between stationary and moving states, particularly the pronounced mode in the 50–150 byte range for moving traffic, was consistently observed across all collected traces. While the present study utilizes data from a single, controlled VR application and environment, the underlying mechanisms used here—Unity's Netcode for GameObjects for state synchronization and the Quest SDK's pose-on-change event reporting—are widely adopted in contemporary VR development frameworks [137]. Therefore, although our current results are limited to one application, these mechanisms provide an initial rationale suggesting

that similar motion-dependent traffic patterns are likely to emerge in other VR systems following comparable design paradigms. Future work will explicitly investigate the generalizability of these findings across a wider range of VR applications, diverse network conditions, and more granular distinctions in user movement (e.g., isolated HMD rotation vs. avatar locomotion). Such statistical differentiation highlights the necessity and justification for subsequent efforts towards the development of robust classification models. Additionally, the detailed insights gained through this exploratory analysis motivate the specific incorporation of packet size-related features into the feature engineering stage. These features—such as packet-size histogram bins, statistical moments (mean, variance, skewness), or quantile-based descriptors—can leverage the observed variations to facilitate improved predictive performance and enhanced Quality of Experience (QoE) in the developed predictive models.

4.4.2 Inter-Arrival Time Statistics

Beyond analyzing packet sizes alone, it is crucial to investigate the temporal characteristics of network traffic through packet inter-arrival times. Formally, the packet inter-arrival time, denoted as τ_i, is defined as the interval between the arrival times t_{i+1} and t_i of two consecutive packets, mathematically expressed as:

$$\tau_i = t_{i+1} - t_i, \quad i = 1, 2, \ldots, N-1 \tag{4.9}$$

where N represents the total number of captured packets within a given communication direction and user state.

To quantify and compare the temporal behavior between stationary and moving states, the empirical distribution of inter-arrival times was characterized by calculating the sample mean $\bar{\tau}$ and sample standard deviation s_τ, defined as:

$$\bar{\tau} = \frac{1}{N-1}\sum_{i=1}^{N-1}\tau_i, \quad s_\tau = \sqrt{\frac{1}{N-2}\sum_{i=1}^{N-1}(\tau_i - \bar{\tau})^2} \tag{4.10}$$

Using the dataset aggregated across both communication directions, the following statistics were obtained:

In the **stationary state**, the mean inter-arrival duration was $\bar{\tau}_{\text{stat}} = 0.0469\,\text{s}$, with a standard deviation $s_{\tau,\text{stat}} = 0.0353\,\text{s}$. This indicates a relatively lower packet transmission frequency and moderate temporal variability during stationary periods. The higher mean inter-arrival time implies less frequent, albeit somewhat irregular, packet arrivals, attributable to the application's logic of transmitting detailed environment-related updates triggered by subtle state changes within the VR environment.

Conversely, the **moving state** exhibited a shorter mean inter-arrival duration of $\bar{\tau}_{\text{move}} = 0.0349\,\text{s}$, with a comparable standard deviation $s_{\tau,\text{move}} = 0.0335\,\text{s}$. The shorter average duration clearly highlights the increased transmission frequency due to continuous and frequent user input (e.g., head and controller movements), necessitating frequent positional updates. The similarity of standard deviation values between states suggests that both scenarios exhibit a similar level of temporal irregularity, characteristic of bursty, event-driven VR network traffic.

The observed differences in inter-arrival time statistics strongly affirm that user mobility introduces significant variations in traffic behavior, reinforcing the hypothesis that temporal features are valuable indicators for state classification and predictive

modeling. Such insights are fundamental for selecting effective temporal features (e.g., inter-arrival mean, standard deviation, burstiness index) in subsequent classification algorithms and predictive frameworks outlined later in this dissertation.

4.4.3 Traffic Volume Distribution and Flow Asymmetry

In addition to temporal analysis, investigating the distribution of packet volume and the directional flow asymmetry provides critical insights into VR network traffic characteristics. Let $C_{S\to C}$ represent the total packet count transmitted from the server to client (downlink), and $C_{C\to S}$ represent the total count from client to server (uplink). These variables were measured for each mobility state as follows:

- **Stationary State:** $C^{\text{stat}}_{S\to C} = 54,471$, $C^{\text{stat}}_{C\to S} = 20,402$
- **Moving State:** $C^{\text{move}}_{S\to C} = 25,034$, $C^{\text{move}}_{C\to S} = 21,313$

From these observed volumes, clear asymmetries in both packet counts and directional distributions emerge, governed by specific VR application logic and the Quest SDK's networking mechanisms.

During stationary periods, the substantially higher downlink packet count $C^{\text{stat}}_{S\to C}$ reflects the VR application's networking strategy. Specifically, when the user remains stationary, the server transmits frequent, smaller-sized packets corresponding to continuously streamed high-fidelity environmental details, animations, particle effects, UI elements, and other immersive details within a widened "area-of-interest" (AoI). This streaming behavior results from the server's fixed-frequency world-tick (approximately 30 Hz) being continuously active and more comprehensive due to the stable viewing position of the user.

Conversely, in the moving state, fewer downlink packets $C^{\text{move}}_{S\to C}$ are transmitted. The reduced packet count stems from built-in optimization techniques within the VR engine, including object culling (removing distant or irrelevant scene elements from transmission) and relying extensively on local interpolation methods to reduce network overhead. Consequently, packet transmission remains near the nominal 31 packets per second, but due to optimized spatial coverage, fewer overall packets are required.

Regarding uplink traffic, the differences in packet counts between stationary and moving states are relatively minor but still notable, with $C^{\text{stat}}_{C\to S} \approx$ 20,402 compared to $C^{\text{move}}_{C\to S} \approx$ 21,313. The similarity in absolute packet counts arises from the Quest SDK's "pose-on-change" logic, where head pose data packets are transmitted only when angular velocity or positional changes exceed predefined thresholds. Thus, during movement, packets containing frequent positional updates are transmitted at a notably higher instantaneous rate (approximately 26 packets/s) compared to stationary periods (approximately 12 packets/s), but over a shorter effective duration, keeping absolute counts closely aligned.

This state-dependent asymmetry in traffic volume and directional flow emphasizes the necessity and practicality of treating stationary and moving states as distinct categories for subsequent classification tasks and predictive modeling. Incorporating this awareness into models—such as state-aware classification algorithms, predictive machine learning frameworks, and Quality of Experience (QoE) optimization mechanisms—ensures better alignment between model assumptions and observed network behavior.

4.5 Feature Engineering

In order to accurately represent and characterize Virtual Reality (VR) network traffic patterns for effective classification and prediction, a robust feature engineering framework was developed. This framework is essential as raw packet-level data (such as payload sizes and inter-arrival times) inherently exhibit high variability and subtle differences influenced by distinct user behaviors. The primary objective of this framework is thus to transform this raw, unstructured packet-level information into structured, informative features suitable for subsequent analysis using machine learning models.

The developed framework specifically targets two complementary aspects of the packet-level time series data. Time-domain features, which capture instantaneous statistical properties such as central tendencies, variability, and distributional shape of packet flows. Frequency-domain features derived through spectral analysis techniques to reveal periodic or quasi-periodic phenomena embedded in VR traffic patterns, reflective of repetitive rendering loops, user movement patterns, and other cyclical behaviors.

4.5.1 Sliding Window Approach

A sliding window approach was adopted as the foundational technique to segment continuous time-series data into finite, overlapping temporal subsets. This method effectively transforms an inherently asynchronous, packet-level data stream into structured samples suitable for supervised machine learning analysis. The sliding window approach entails extracting localized statistical and spectral characteristics from sequential subsets of the packet data stream. This approach systematically advances across the entire data stream with a predetermined step size S.

The selection of appropriate sliding window parameters, including the temporal window sizes ($T_{\mathbf{window}}$) and step size (S), critically impacts both the temporal resolution of the extracted features and the efficacy of subsequent analyses. These parameters were chosen through an iterative empirical evaluation process, guided by two primary considerations: temporal coverage and responsiveness. A sufficiently large window duration is necessary to capture relevant temporal patterns and cycles inherent in VR traffic, such as game rendering loops or user activity cycles, while remaining sensitive enough to detect rapid transitions in user behavior.

Temporal Window Sizes ($T_{\mathbf{window}}$)

Instead of a fixed number of packets, our approach employs multiple **time-based window sizes**: $T_{\text{window}} \in \{0.5, 1.0, 2.0\}$ seconds. This multi-scale approach allows the feature extraction to capture dynamics over different temporal resolutions, from rapid, sub-second bursts to more sustained patterns over a couple of seconds. This range is is strategically chosen to encompass typical update frequencies and interaction latencies in interactive VR, ensuring a comprehensive view of traffic behavior. The number of packets within each window, denoted as N_p, will naturally vary depending on the instantaneous packet inter-arrival times. To ensure meaningful analysis, a minimum of 4 packets (`MIN_POINTS_PER_FFT = 4`) is required for a window to be considered valid for spectral feature extraction.

Step Size (S)

A step size of one packet ($S = 1$) was utilized. This means that for each new packet arriving, a new set of features is extracted from the latest time-based windows ending at that packet's arrival time. This overlapping configuration maximizes the granularity of temporal resolution, facilitating the detection of subtle temporal transitions. Although this approach increases computational cost, the enhancement in analytical resolution justifies the trade-off.

The effective sampling frequency (f_s) and frequency resolution (Δf) for FFT within each window are dynamically determined by the packets contained within that specific time window. To verify the suitability of our multi-scale window choices across different traffic scenarios, we performed comprehensive diagnostics by calculating the mean inter-arrival time, the resulting Nyquist frequency, and the expected number of packets per window for each specific scene and time scale. The results of these diagnostics are presented in Table 4.1.

Table 4.1: Per-scene time-window diagnostics: mean inter-arrival time, derived Nyquist frequency, and expected number of packets per window for each temporal scale. Data are aggregated from the four original traffic capture scenarios (Stationary Server-to-Client, Stationary Client-to-Server, Moving Server-to-Client, Moving Client-to-Server)

State	Direction	Window T (s)	Mean inter-arrival (s)	Nyquist (Hz)	Expected packets
Stationary	Downlink (Server→Client)	0.5	0.0322553	15.5013	15.5013
Stationary	Downlink (Server→Client)	1.0	0.0322553	15.5013	31.0027
Stationary	Downlink (Server→Client)	2.0	0.0322553	15.5013	62.0054
Stationary	Uplink (Client→Server)	0.5	0.0860895	5.80791	5.80791
Stationary	Uplink (Client→Server)	1.0	0.0860895	5.80791	11.6158
Stationary	Uplink (Client→Server)	2.0	0.0860895	5.80791	23.2316
Moving	Downlink (Server→Client)	0.5	0.0322957	15.4820	15.4820
Moving	Downlink (Server→Client)	1.0	0.0322957	15.4820	30.9639
Moving	Downlink (Server→Client)	2.0	0.0322957	15.4820	61.9278
Moving	Uplink (Client→Server)	0.5	0.0379394	13.1789	13.1789
Moving	Uplink (Client→Server)	1.0	0.0379394	13.1789	26.3578
Moving	Uplink (Client→Server)	2.0	0.0379394	13.1789	52.7156

Table 4.1 demonstrates the validity of our multi-scale window approach. Even in the scenario with the lowest packet density (Stationary Uplink), the smallest window ($T = 0.5\,\text{s}$) has an expected number of packets of approximately 5.81, comfortably exceeding our `MIN_POINTS_PER_FFT = 4` threshold. This ensures sufficient data points for meaningful spectral analysis across all scenarios. Furthermore, the table highlights significant variability in the derived Nyquist frequencies (ranging from approximately 5.8 Hz to 15.5 Hz) across different scenes and traffic directions. This substantial range, caused by varying mean inter-arrival times, underscores the critical importance of employing Nyquist-normalized frequencies for spectral features, thereby ensuring comparability and robustness of the features across diverse VR traffic conditions.

4.5.2 Time-Domain Feature Extraction

Time-domain features capture instantaneous characteristics of network traffic and serve as crucial indicators for real-time traffic prediction and classification. In this work, for each sliding window composed of network packets and indexed by the most recent packet at position i, specific time-domain features were extracted. These features

were explicitly chosen to provide immediate and precise insight into the current state of the network traffic, reflecting short-term dynamics relevant to Virtual Reality (VR) applications. The extraction process emphasizes the most recent packet in the sequence, thereby offering the prediction model critical information about the latest observed network conditions.

Specifically, for each sliding window ending at packet index i, the following time-domain features were computed:

Packet Size (s_i)

The packet size s_i represents the payload size in bytes of the most recent packet at the end of the sliding window. Mathematically, the feature is expressed as:

$$s_i = \text{Payload Size (in bytes) of packet } i$$

This instantaneous measure directly captures the immediate network load generated by the VR application, providing the predictive model with valuable context regarding the volume of data transmitted at a precise moment in time. The reason for selecting the most recent packet size as opposed to aggregated metrics across the window is to maximize the sensitivity of the model to abrupt changes and short-term fluctuations in traffic volume—phenomena frequently observed in VR network flows due to their real-time interactive characteristics.

Inter-arrival Time (Δt_i)

The inter-arrival time Δt_i quantifies the temporal spacing between the arrival of the most recent packet (indexed as i) and its immediate predecessor (indexed as $i-1$). Let T_i and T_{i-1} denote the timestamps of packet arrivals for packets i and $i-1$, respectively. Then, the inter-arrival time feature is formally defined as:

$$\Delta t_i = T_i - T_{i-1}$$

This metric is a direct reflection of the immediate packet transmission rate, providing an explicit signal regarding packet generation intervals and, implicitly, network load stability or variability. Capturing the inter-arrival time of the most recent packet rather than using average or median inter-arrival times within the window is an intentional methodological choice. It ensures the extracted feature is sensitive to the rapid variations inherent in VR applications, where even subtle temporal deviations in packet arrival may significantly impact perceived Quality of Experience (QoE).

The rationale behind exclusively considering the latest packet's information in the extraction of these two time-domain features is based upon their relevance for short-term, real-time predictive tasks. Unlike frequency-domain features, which summarize and represent aggregated statistical properties over the entire analysis window, time-domain features emphasize instantaneous behaviors. Therefore, utilizing the final packet's payload size and inter-arrival time provides the classification and prediction models with immediate, actionable intelligence about the current state of the VR traffic stream.

4.5.3 Frequency-Domain Feature Extraction via FFT

To identify and quantify periodic and oscillatory behaviors embedded within the virtual reality (VR) network traffic patterns, frequency-domain analysis was conducted using the Fast Fourier Transform (FFT). The FFT algorithm efficiently computes the Discrete Fourier Transform (DFT), transforming time-domain signals into their frequency-domain representations. This transformation enables extraction of frequency-based features, capturing periodic fluctuations that significantly characterize network conditions affecting VR Quality of Experience (QoE).

In applying FFT to packet size sequences extracted from sliding temporal windows, several methodological challenges and practical considerations were systematically addressed, detailed as follows:

1. **Approximation of Uniform Sampling Interval:**

   A foundational assumption underpinning FFT computation is the uniform sampling of the input signal, whereas network packet arrivals are inherently asynchronous and irregular. Consequently, an approximation strategy was necessary to estimate an effective pseudo-uniform sampling interval for each sliding window. Specifically, for a sequence of N_p packets within a given time-based window (e.g., 0.5s, 1.0s, or 2.0s), the mean inter-arrival time within the window was calculated as:

   $$\bar{\Delta t}_w = \frac{1}{N_p - 1} \sum_{k=k_0+1}^{k_0+N_p-1} (T_k - T_{k-1}) \tag{4.11}$$

   where:

   - $\bar{\Delta t}_w$ is the calculated mean inter-arrival time (in seconds) for the current window.
   - N_p is the total number of packets observed within the current time-based sliding window.
   - T_k represents the timestamp of the k-th packet in the chronologically ordered sequence within the window.
   - T_{k-1} represents the timestamp of the packet immediately preceding the k-th packet.
   - k_0 is the index of the first packet in the current window.

   This estimated interval $\bar{\Delta t}_w$ (in seconds) was then used as the effective sampling interval for the window. Windows exhibiting invalid (zero or negative) sampling intervals, indicative of data anomalies or duplicated timestamps, were identified and excluded from subsequent analyses. Additionally, only windows containing at least 4 packets (`MIN_POINTS_PER_FFT = 4`) were considered valid for spectral feature extraction.

2. **Centering, Scaling and Removal of the Mean (DC Component):**

   Prior to frequency analysis, the raw sequence of packet sizes $\{s_k\}_{k=1}^{N_p}$ within each window underwent a two-step transformation. First, the sequence was robustly centered and scaled using the median and median absolute deviation (MAD) of the packet sizes within that window. This process helps to reduce the influence

of outliers and normalizes the amplitude range. Let this robustly scaled sequence be denoted as x'_k. Second, to ensure the input to the FFT is strictly zero-mean, any remaining direct current (DC) offset component was explicitly removed by subtracting the mean of x'_k from each element, resulting in the zero-mean sequence x_k:

$$x_k = x'_k - \frac{1}{N_p} \sum_{j=1}^{N_p} x'_j, \quad \text{for } k = 1, \ldots, N_p.$$

This final zero-mean sequence x_k is then used as the input for the FFT. The DC component corresponds to the zero-frequency term in the Fourier spectrum and represents the average magnitude of the signal. Removing this component emphasizes fluctuations around the local mean, making periodic patterns more distinguishable and facilitating clearer spectral interpretation.

3. **FFT Calculation and Frequency Grid Properties:**

The FFT was subsequently applied to the zero-mean, robustly scaled sequence of N_p packets. The FFT calculation transforms this sequence from the time domain into a frequency spectrum X_m, represented as complex coefficients. For each window, the frequency corresponding to each FFT bin m (in Hz) is explicitly given by:

$$f_m = \frac{m-1}{N_p \cdot \bar{\Delta} t_w}, \quad m = 1, \ldots, \left\lfloor \frac{N_p}{2} \right\rfloor + 1, \tag{4.12}$$

where:

- f_m is the frequency (in Hertz) associated with the m-th FFT bin.
- m is the index of the FFT bin, ranging from 1 to $\lfloor N_p/2 \rfloor + 1$.
- N_p is the total number of packets within the current time-based sliding window.
- $\bar{\Delta} t_w$ is the mean inter-arrival time (in seconds) for the packets within the current window, as defined in Equation (4.11).

Let $\mathcal{P}$ denote the set of indices for strictly positive frequency components ($m \geq 2$, meaning $f_m > 0$). These positive frequency components are retained for further analysis, representing frequencies from $1/(N_p \bar{\Delta} t_w)$ up to the Nyquist frequency f_{Nyq}. The Nyquist frequency, which is the maximum frequency that can be unambiguously sampled, is given by:

$$f_{\text{Nyq}} = \frac{1}{2\bar{\Delta} t_w}. \tag{4.13}$$

It is important to note that the frequency resolution, $\Delta f = 1/(N_p \cdot \bar{\Delta} t_w)$, and the Nyquist frequency f_{Nyq} are dynamically determined by the number of packets N_p and the mean inter-arrival time $\bar{\Delta} t_w$ of each specific window. This means these spectral properties can vary slightly from one window to another, reflecting local traffic timing characteristics.

4. **Selection of Windowing Function:**

In Fourier analysis, application of a windowing function—such as Hamming, Hanning, or Blackman—is typically recommended to mitigate spectral leakage. Spectral leakage arises from the discontinuity at the boundaries of finite data segments, potentially causing energy from one frequency component to spread into adjacent frequency bins. However, in this study, a **rectangular window** was deliberately selected for several methodologically grounded reasons:

- **Amplitude Preservation:** Unlike tapered windowing functions, a rectangular window applies equal weighting across the entire windowed segment. This characteristic preserves the true amplitude magnitudes of frequency components, crucial since amplitude variations in VR packet size signals directly correlate with perceptible QoE variations.
- **Computational and Analytical Simplicity:** The rectangular window involves no additional computational overhead, simplifying the feature extraction pipeline and ensuring clarity and reproducibility in the frequency analysis process.
- **Dominant Frequency Capture:** Empirical analyses revealed sufficiently strong periodic signals within VR traffic. Under such conditions, dominant frequency peaks remain distinct and reliably identifiable despite potential sidelobe leakage introduced by rectangular windowing. Consequently, the trade-off between spectral leakage and amplitude fidelity was empirically validated as acceptable for our classification objectives.

5. **Extraction of Top-N Amplitude-Ranked Frequency Features:**

   After obtaining the FFT spectrum for positive frequencies (indexed by $\mathcal{P}$), the most significant frequency-domain features were extracted to succinctly represent periodic behavior within each window. The magnitude of each complex FFT coefficient X_m is denoted $A_m = |X_m|$. We then define the L1-normalized amplitudes a_m and Nyquist-normalized frequencies $\hat{f}_m$ as:

$$a_m = \frac{A_m}{\sum_{k\in\mathcal{P}} A_k + \varepsilon}, \quad m \in \mathcal{P}, \qquad \hat{f}_m = \frac{f_m}{f_{\text{Nyq}}} \in [0, 1],$$

   where ε is a small positive constant to prevent division by zero. This normalization makes features comparable across windows and scenes with different mean inter-arrival times and Nyquist frequencies. Both a_m and $\hat{f}_m$ are dimensionless quantities, representing relative proportions of amplitude and frequency, respectively.

   We then select the top N indices $\mathcal{J}_N \subset \mathcal{P}$ corresponding to the largest normalized amplitudes a_m. Here, N refers to the **number of dominant spectral components** extracted. The per-scale spectral feature pairs are then:

$$\left[a_{(1)}, \hat{f}_{(1)},\ a_{(2)}, \hat{f}_{(2)},\ \ldots,\ a_{(N)}, \hat{f}_{(N)}\right],$$

   where (j) denotes the j-th ranked component by amplitude. If fewer than N distinct frequency peaks are identified (e.g., due to signals with minimal variance or insufficient valid packets), the feature vector is zero-padded to maintain a consistent dimensional structure across all samples. The value of $N = \mathbf{7}$

was determined through an empirical evaluation, specifically a grid search over $N \in \{1, 3, 5, 7, 9\}$, which confirmed that this number of top frequency features (7 components) captured essential discriminative characteristics required for robust VR traffic classification, maximizing test accuracy (as shown in Section 5.2).

6. **Auxiliary Spectral Descriptors:**

   To further summarize the shape and dominance characteristics of the frequency spectrum for each time scale, we append four scalar descriptors computed from the normalized amplitudes a_m and Nyquist-normalized frequencies $\hat{f}_m$. Additionally, a binary validity flag is included.

   (a) **Amplitude Ratios ($\mathbf{ratio}_{12}, \mathbf{ratio}_{23}$):** These quantify the relative dominance between the highest amplitude spectral peaks.

   $$\text{ratio}_{12} = \frac{a_{(1)}}{a_{(2)} + \varepsilon} \tag{4.14}$$

   $$\text{ratio}_{23} = \frac{a_{(2)}}{a_{(3)} + \varepsilon} \tag{4.15}$$

   where $a_{(j)}$ denotes the j-th largest normalized amplitude, and ε is a small positive constant to avoid division by zero.

   (b) **Spectral Centroid ($\hat{f}_{\text{centroid}}$):** This indicates the "center of mass" of the spectrum, providing a measure of the average frequency component.

   $$\hat{f}_{\text{centroid}} = \sum_{m \in \mathcal{P}} a_m \, \hat{f}_m \tag{4.16}$$

   where a_m is the L1-normalized amplitude and $\hat{f}_m$ is the Nyquist-normalized frequency for the m-th positive-frequency bin, and $\mathcal{P}$ is the set of indices for strictly positive frequency components.

   (c) **Spectral Flatness (flatness):** This measures the noisiness or tonality of the spectrum. A high value suggests a tonal (peaky) spectrum, while a low value indicates a noise-like (flat) spectrum.

   $$\text{flatness} = \frac{\exp\left(\frac{1}{|\mathcal{P}|} \sum_{m \in \mathcal{P}} \log(a_m + \varepsilon)\right)}{\frac{1}{|\mathcal{P}|} \sum_{m \in \mathcal{P}} a_m} \tag{4.17}$$

   where $|\mathcal{P}|$ is the number of positive-frequency bins, a_m is the L1-normalized amplitude, and ε is a small positive constant.

   (d) **Validity Flag (v_T):** A binary indicator per scale, set to 1 if the window contains at least $N_{\min} = 4$ packets (sufficient for a meaningful spectrum), and 0 otherwise.

4.5.4 Directional Context Feature (Downlink vs. Uplink)

Virtual reality traffic exhibits marked directional asymmetries in packet counts and dynamics (see Section 4.4.3). To provide the classifier with this context without relying on deep packet inspection, we include a single binary feature

$\text{Dir} \in \{0, 1\}$, $\text{Dir} = 0$ for Server→Client (downlink), $\text{Dir} = 1$ for Client→Server (uplink).

For each sample (i.e., each packet-anchored window), we append Dir to the feature vector alongside the time- and frequency-domain features. Because downlink and uplink flows implement different application logic (e.g., world state dissemination vs. pose/RPC updates), Dir can help disambiguate otherwise similar spectral patterns. At the same time, to guard against spurious correlations, we evaluate models both *with* and *without* Dir (see Chapter 5, Section 5.2.2). In our experiments, adding Dir yields a modest but consistent improvement.

4.5.5 Feature Standardization (Z-score Normalization)

For each time scale $T \in \{0.5, 1.0, 2.0\}$ s, the corresponding spectral feature block is thus composed of $2N$ amplitude-frequency pairs, 4 scalar descriptors (ratios, centroid, flatness), and 1 validity flag, resulting in $2N+5$ features. With $N = 7$, each spectral block has $2 \times 7 + 5 = 19$ features. These blocks from all three scales are concatenated to form the complete frequency-domain feature set ($3 \times 19 = 57$ features). The final feature vector for each sample is then constructed by appending the time-domain features (packet size and inter-arrival time, Section 4.5.2) and the direction bit to this frequency-domain set. This yields a total of 57 (frequency-domain)+2 (time-domain)+1 (direction bit) = 60 features per sample.

As the final preprocessing step before modeling, all extracted time, frequency, and directional context features were standardized via Z-score normalization. This crucial normalization ensures all features exert proportional influence on the training process, avoiding bias due to scale disparities. Mathematically, each original feature value x is transformed to its standardized form x' as follows:

$$x' = \frac{x - \mu_x}{\sigma_x}$$

where μ_x and σ_x represent the mean and standard deviation, respectively, of each feature across the entire training dataset. Through this normalization, the resulting features have a mean of zero and standard deviation of one. Also, features with inherently different numerical scales (e.g., packet sizes versus inter-arrival times) contribute equally to model training. Furthermore, improved numerical stability and convergence properties are observed, facilitating more reliable and efficient training outcomes.

4.6 Experimental Setup and Evaluation Metrics

This section provides an exhaustive explanation of the experimental methodologies, the rationale behind the methodological choices, and the formal definition of evaluation metrics applied to assess the performance of the models for both the VR network traffic classification and regression (prediction) tasks. Ensuring consistency and comparability across experiments, rigorous attention was given to the data splitting strategies, dimensionality reduction techniques, and evaluation criteria.

4.6.1 Data Splitting Strategy

The effectiveness of machine learning models greatly depends on the careful and rigorous selection of the training and testing (or validation) data. Thus, different splitting strategies were adopted according to the nature of each task to ensure unbiased and meaningful performance evaluation:

1. **Classification Task:**

   For the classification task, to avoid temporal leakage caused by highly correlated, overlapping windows, we adopted a per-scenario chronological block split rather than stratified random sampling. Let $\mathcal{S} = \{\texttt{stat\_s2c}, \texttt{stat\_c2s}, \texttt{mov\_s2c}, \texttt{mov\_c2s}\}$ denote the four distinct traffic scenarios (stationary server-to-client, stationary client-to-server, moving server-to-client, and moving client-to-server). For each scenario $s \in \mathcal{S}$, after sorting packets by timestamp and constructing features, we obtained a time-ordered sequence of samples $\{(x_{s,1}, y_{s,1}), \ldots, (x_{s,n_s}, y_{s,n_s})\}$ of length n_s. We then partitioned each scenario by time into contiguous, non-overlapping blocks for training, validation, and testing:

$$\begin{aligned} I_s^{\text{train}} &= \{1, \ldots, \lfloor 0.70\, n_s \rfloor\}, \\ I_s^{\text{val}} &= \{\lfloor 0.70\, n_s \rfloor + 1, \ldots, \lfloor 0.85\, n_s \rfloor\}, \\ I_s^{\text{test}} &= \{\lfloor 0.85\, n_s \rfloor + 1, \ldots, n_s\}. \end{aligned} \tag{4.18}$$

   No shuffling was applied within these blocks. The global dataset splits for training, validation, and testing were then formed by concatenating the corresponding blocks across all scenarios:

$$\begin{aligned} D_{\text{train}} &= \bigcup_{s \in \mathcal{S}} \{(x_{s,i}, y_{s,i}) : i \in I_s^{\text{train}}\}, \\ D_{\text{val}} &= \bigcup_{s \in \mathcal{S}} \{(x_{s,i}, y_{s,i}) : i \in I_s^{\text{val}}\}, \\ D_{\text{test}} &= \bigcup_{s \in \mathcal{S}} \{(x_{s,i}, y_{s,i}) : i \in I_s^{\text{test}}\}. \end{aligned} \tag{4.19}$$

   This time-block split method ensures that temporal order is strictly respected within each scenario and prevents any leakage of future information into the past across the splits. While this approach does not explicitly enforce class proportions via stratification across the entire combined dataset, the class proportions in each global split are implicitly induced by the lengths of the individual stationary and moving scenarios, providing a realistic representation for evaluation. All classification results reported in Chapter 5 (including ablations and hyperparameter grids) utilize this specific splitting methodology.

2. **Regression Task:**

   For short-term traffic prediction (the regression task), the dataset comprised of real-world time series sequences was partitioned chronologically, respecting the inherent temporal dependencies. The first 70% of the sequential data points were utilized for training and model optimization, while the subsequent 15% were reserved as a validation set, and the final 15% as a test set, to assess forecasting capability on future, unseen observations. Formally, given a time series dataset $X = \{x_t\}_{t=1}^{T}$, the partitioning is:

$$D_{\text{train}} = \{x_t : t \leq 0.7T\}, \quad D_{\text{val}} = \{x_t : 0.7T < t \leq 0.85T\}, \quad D_{\text{test}} = \{x_t : t > 0.85T\} \tag{4.20}$$

Such chronological partitioning ensures the preservation of temporal relationships and allows for a realistic evaluation of predictive performance, as the model is always tested on future data points relative to the training data.

4.6.2 Classification Metrics

The performance evaluation of the Random Forest classifier employed a series of rigorous metrics specifically designed for binary classification tasks. In classification modeling, evaluating performance using multiple metrics provides deeper insight into the model's behavior, particularly in scenarios involving imbalanced data or differential class importance. The chosen metrics—Accuracy, Precision, Recall, and F1-Score—were selected because they collectively describe different aspects of classification performance, enabling a thorough interpretation of the classifier's strengths and weaknesses.

Firstly, **Accuracy** measures the overall correctness of the classifier and is mathematically defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.21}$$

where TP (True Positive) is the number of instances correctly classified as positive, TN (True Negative) represents the correctly classified negative instances, FP (False Positive) is the count of instances incorrectly classified as positive, and FN (False Negative) is the number of positive instances incorrectly classified as negative. Although accuracy provides a general performance indicator, it may offer misleading results when the dataset is imbalanced, hence the necessity for additional metrics.

Secondly, **Precision** measures the proportion of positive identifications that are actually correct. Precision is critical in scenarios where false positives have a significant cost or consequence, and it is formally defined as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.22}$$

A higher precision indicates that fewer negative instances are incorrectly classified as positive, reflecting greater reliability in positive predictions.

Thirdly, **Recall (Sensitivity)** quantifies the model's ability to correctly identify positive instances out of all actual positive samples. It is particularly valuable in contexts where the cost of missing positive instances (false negatives) is high. Recall is mathematically expressed as

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.23}$$

A higher recall value implies that fewer actual positive instances are overlooked, enhancing the completeness of the model's predictive capability.

Fourthly, to achieve a balanced evaluation between Precision and Recall, the **F1-Score** is utilized. The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that emphasizes the lower of the two values, thus preventing situations where either metric dominates. The F1-Score is given by

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.24}$$

This metric is particularly effective for imbalanced classification scenarios, as it penalizes classifiers that disproportionately emphasize either Precision or Recall, prompting the selection of balanced models.

These classification metrics were computed and reported separately for each class (e.g., stationary and moving VR states) as well as aggregated across classes through macro-averaging, providing comprehensive visibility into class-specific performance and overall robustness of the classifier.

4.6.3 Regression Metrics

For evaluating the regression-based prediction models—including Random Forest, Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Transformer architectures—four robust statistical metrics were adopted: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the Coefficient of Determination (R^2). These metrics were chosen due to their complementary insights into the accuracy, robustness, and explanatory power of regression models.

The **Mean Squared Error (MSE)** metric quantifies the average squared difference between predicted and observed continuous outcomes. Formally, the MSE is calculated as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.25}$$

where N represents the total number of data points in the test dataset, y_i is the actual observed value of the dependent variable (in this case, the average VR packet size), and $\hat{y}_i$ is the predicted value generated by the regression model for the i^{th} observation. The squaring of errors serves two essential purposes: it ensures positive values for ease of interpretation, and it disproportionately penalizes larger errors, which is especially important in real-world predictions where severe mispredictions can significantly degrade user experience.

The **Root Mean Squared Error (RMSE)** is the square root of the MSE, bringing the error metric back into the same units as the target variable. This makes it more interpretable than MSE in terms of the actual error magnitude. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{4.26}$$

Like MSE, RMSE also heavily penalizes large errors due to the squaring operation.

The **Mean Absolute Error (MAE)** calculates the average of the absolute differences between predictions and actual observations. It provides a more straightforward and interpretable measure of average error, as it is expressed in the original units of the target variable and is less sensitive to outliers compared to MSE or RMSE. The MAE is formally expressed as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{4.27}$$

Additionally, to evaluate how effectively the regression models capture and explain the variance of the dependent variable, the **Coefficient of Determination (R^2)** was

employed. The R^2 value measures the proportion of the variance in observed values that is predictable from the independent variables. Formally, R^2 is expressed as

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{4.28}$$

where $\bar{y}$ denotes the mean of the observed dependent variable values, representing a baseline for prediction performance. The numerator $\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ is the residual sum of squares, reflecting prediction errors, while the denominator $\sum_{i=1}^{N}(y_i - \bar{y})^2$ is the total sum of squares, representing the total variance present in the dataset.

An R^2 value close to 1 indicates a high degree of explanatory power and suggests that the regression model closely aligns with the observed data. Conversely, a value approaching zero suggests that the model offers limited predictive insight beyond a simple mean-value prediction. Hence, the simultaneous use of MSE, RMSE, MAE, and R^2 provides comprehensive insights—while MSE, RMSE, and MAE provide direct error magnitude information (with varying sensitivities to large errors), R^2 evaluates overall explanatory efficacy and variance coverage. These regression metrics collectively ensure that model performance assessment is thorough, precise, and contextually informative for enhanced decision-making in predicting virtual reality network traffic characteristics.

Chapter 5

VR User State Classification using Hybrid Time-Frequency Features

This chapter presents the first core contribution of this dissertation: the design, implementation, and rigorous evaluation of a classification framework capable of distinguishing between stationary and moving states of Virtual Reality (VR) users based on network traffic characteristics. Leveraging the comprehensive dataset and the feature engineering methodology established in Chapter 4, this chapter investigates the hypothesis that integrating frequency-domain features with traditional time-domain metrics significantly enhances classification accuracy and interpretability. To validate this hypothesis, a Random Forest (RF)-based classification approach is adopted, and its effectiveness is empirically demonstrated through detailed quantitative and qualitative analyses.

Specifically, the classification process involves clearly defining the binary classification task, justifying the selection of the Random Forest algorithm, and meticulously exploring the contributions of hybrid features through a comparative evaluation with baseline approaches. Furthermore, this chapter delves into feature importance analysis, providing insight into the network characteristics most strongly associated with VR user mobility. Lastly, we discuss the implications of our findings for intelligent, state-aware network resource management strategies aimed at enhancing the VR Quality of Experience (QoE).

5.1 Problem Formulation and Classifier Selection

5.1.1 Formal Problem Formulation

Formally, the objective of this classification task is to develop a predictive function capable of accurately identifying the mobility state of VR users based exclusively upon network traffic features. This problem can be rigorously defined as a binary supervised learning classification problem as follows:

Let the feature space be represented by:

$$\mathcal{X} \subseteq \mathbb{R}^D, \tag{5.1}$$

where each vector $\mathbf{x} \in \mathcal{X}$ is a D-dimensional representation comprising both time-domain and frequency-domain features extracted from network traffic within sliding windows.

Let the corresponding binary label space be defined as:

$$\mathcal{Y} = \{0, 1\}, \tag{5.2}$$

where the label $y = 0$ denotes the *Stationary* user state and $y = 1$ denotes the *Moving* user state.

Given a labeled dataset containing N instances:

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \tag{5.3}$$

the classification task involves learning a function:

$$f : \mathcal{X} \to \mathcal{Y}, \tag{5.4}$$

such that, for a new, unseen input feature vector $\mathbf{x}'$, the classifier predicts the label:

$$\hat{y} = f(\mathbf{x}') \in \{0, 1\}. \tag{5.5}$$

The goal is to ensure that the function f generalizes well to unseen data, thereby achieving robust and accurate classification in real-time VR environments.

5.1.2 Random Forest Classifier

To address the above-defined binary classification challenge, a Random Forest classifier was strategically selected. The rationale for this choice is grounded in the alignment between the inherent properties of the Random Forest algorithm and the specific requirements of VR network traffic classification, as detailed below.

1. **Modeling Complex, Non-linear Decision Boundaries:** Virtual Reality network traffic patterns result from the intricate interplay of user interactions, application logic, rendering processes, and underlying network protocols. Consequently, the relationships between extracted features and the user's mobility state are highly non-linear and multidimensional. Random Forest classifiers naturally accommodate such complex, non-linear interactions without necessitating explicit, computationally expensive feature transformations [74].

2. **Robustness through Ensemble Learning:** Network traffic data, particularly in real-world VR scenarios, frequently exhibit noise, outliers, and inconsistencies. Random Forests, being ensemble-based methods that aggregate the outputs of multiple decision trees trained on bootstrapped subsets of the data and feature space, are inherently robust to noise and overfitting. This robustness translates to improved generalization performance, especially when dealing with noisy or variable data.

3. **Feature Importance Quantification:** A major research objective, explored in detail in Section 5.3, is to identify and understand the most discriminative features for VR user state classification. Random Forests provide an intrinsic mechanism for evaluating feature importance through metrics such as the average decrease in Gini impurity or permutation-based importance. This interpretability is critical for validating the feature design process and providing actionable insights into network traffic behavior.

4. **Empirical Performance and Scalability** Random Forest classifiers are widely recognized for achieving high accuracy across diverse classification tasks and scale efficiently with both the number of features and dataset size, making them suitable for real-time classification tasks in dynamic VR environments. For this study, the number of decision trees (`n_estimators`) was set to 300. This choice was informed by preliminary cross-validation experiments, which involved a sweep over various `n_estimators` values (specifically 100, 200, 300, and 500). These experiments revealed a stable accuracy plateau across this range, with differences in performance being marginal (e.g., less than 0.03 percentage points in test accuracy). Selecting 100 estimators provided a robust balance between performance and computational cost, as increasing the number of trees further yielded no significant accuracy gains while increasing training and inference time.

The Random Forest model was implemented using the `scikit-learn` Python library. The key hyperparameter, namely the number of decision trees (`n_estimators`), was set to 100 based on cross-validation results, which indicated diminishing returns in accuracy for larger forest sizes. Other hyperparameters were set to default values unless otherwise specified in later experimental sections.

5.2 Classification Performance Evaluation

5.2.1 Dataset Description and Experimental Setup

The classification analysis in this chapter utilizes a comprehensive VR network traffic dataset, which was meticulously collected and detailed in Chapter 4. This dataset originates from a single, custom-built Unity VR application running on a Meta Quest 3 headset, specifically designed to elicit and capture distinct user mobility states. The raw traffic traces were captured across four specific scenarios: Stationary Server-to-Client, Stationary Client-to-Server, Moving Server-to-Client, and Moving Client-to-Server.

Following the multi-scale feature extraction process using time-based sliding windows of 0.5s, 1.0s, and 2.0s with a step size of 1 packet, as described in Section 4.5.1, the raw packet traces were transformed into a set of labeled feature vectors suitable for classification. After excluding initial packets for windowing and handling any missing values, the total number of effective feature samples used in this classification task is 121,118. This aggregate dataset comprises: 74,822 samples corresponding to the *Stationary* user state ($y = 0$), and 46,296 samples corresponding to the *Moving* user state ($y = 1$). These samples, derived from all four original traffic capture scenarios, ensure a comprehensive representation of network interactions for each mobility state.

For robust model training and evaluation, the entire dataset of 121,118 samples was explicitly partitioned into training, validation, and testing sets. To ensure reproducibility and avoid optimistic bias due to overlapping windows, we implemented a time-block based splitting strategy. Within each of the original four capture scenarios, the feature samples were first sorted chronologically. Then, the first 70% of the chronologically ordered samples formed the training set (totaling 84,784 samples), the subsequent 15% formed the validation set (totaling 18,182 samples), and the final 15% constituted the test set (totaling 18,184 samples). This temporal splitting prevents highly adjacent windows from "leaking" information across splits, providing

a more realistic assessment of the model's ability to generalize to future, unseen traffic. All preprocessing steps, including feature standardization (z-score normalization), were exclusively fitted on the training set and subsequently applied to the validation and test sets. This rigorous approach prevents data leakage during the scaling process. Hyperparameters (such as `n_estimators` for the Random Forest and the number of top spectral peaks, `top_n_freq`) were primarily tuned using the validation set, and the final reported performance metrics were obtained once on the unseen test set using the chosen configuration.

5.2.2 Performance Comparison: Feature Set Ablation Studies

To thoroughly evaluate the influence of different feature types in accurately distinguishing user mobility states, we constructed and compared Random Forest classifiers trained with several distinct feature sets through an ablation study:

1. **Time-domain baseline feature set:** Consisting solely of two robustly scaled instantaneous features: the payload size of the most recent packet (s_i) and the inter-arrival time of the most recent packet (Δt_i). These features encapsulate immediate, temporal characteristics but do not directly encode periodicity or repetitive patterns.

2. **Frequency-domain only feature set:** Comprising solely the multi-scale frequency-domain features extracted from packet size sequences. This set excludes time-domain features and the direction bit. For each of the 0.5s, 1.0s, and 2.0s time windows, this set includes:

   - The top 7 amplitude-ranked spectral components, represented by their L1-normalized amplitudes (a_m) and Nyquist-normalized frequencies ($\hat{f}_m$).
   - Four auxiliary spectral descriptors: amplitude ratios ($\text{ratio}_{12}, \text{ratio}_{23}$), Nyquist-normalized spectral centroid ($\hat{f}_{\text{centroid}}$), and spectral flatness (flatness).
   - A binary validity flag (v_T) per window, indicating sufficient packets for a meaningful spectrum.

3. **Hybrid time-frequency feature set (without direction):** Combining the two robustly scaled time-domain features (s_i, Δt_i) with the complete set of multi-scale frequency-domain features detailed in item 2 above. This set does not include the direction bit.

4. **Hybrid time-frequency domain feature set (with direction):** The full feature set, comprising the two robustly scaled time-domain features (s_i, Δt_i), the comprehensive multi-scale frequency-domain features (top 7 normalized amplitude and Nyquist-normalized frequency components, plus four spectral shape descriptors per 0.5s, 1.0s, and 2.0s window, and validity flags), and the binary direction indicator (`dir_bit`). This set was hypothesized to provide the most comprehensive discriminative capability.

As evident from Table 5.1, the inclusion of frequency-domain features leads to a significant performance improvement. Specifically, the hybrid feature set (Time+Frequency+Dir) yields an overall accuracy of 93.74%, representing an increase of 24.79 percentage points

Table 5.1: Classification Results Comparing Feature Sets for User Mobility States (Test Set)

Feature Set	Precision (0)	Recall (0)	F1 (0)	Precision (1)	Recall (1)	F1 (1)	Accuracy
Time-Only	0.7485	0.7489	0.7487	0.5940	0.5934	0.5937	0.6895
Spec-Only	0.9276	0.9689	0.9478	0.9459	0.8777	0.9105	0.9341
Time+Frequency	0.9282	0.9641	0.9458	0.9382	0.8795	0.9079	0.9318
Time+Frequency+Dir	0.9317	0.9696	0.9503	0.9475	0.8852	0.9153	0.9374

over the baseline Time-Only approach (68.95%). Moreover, the classifier demonstrates substantial gains in identifying the more challenging "Moving" state, with its F1-score increasing from 0.5937 (Time-only) to 0.9153 (Time+Frequency+Dir).

A key insight from the ablation study is the decisive role of spectral features. The "Spec-Only" feature set already achieves an accuracy of 93.41% and an F1-score of 0.9105 for the "Moving" class, significantly outperforming the "Time-Only" baseline. Adding the instantaneous time-domain features to the spectral features ("Time+Frequency" without direction) shows a slight decrease in performance compared to "Spec-Only," suggesting that the spectral features already capture most of the essential temporal dynamics. However, the addition of the direction bit ("Time+Frequency+Dir") provides a modest but consistent additional gain, resulting in the highest overall accuracy and F1-score for the "Moving" class.

The fundamental reason behind this marked improvement is that purely instantaneous packet size and timing features are often insufficiently discriminative. They fail to distinguish subtle yet essential spectral patterns uniquely associated with user movement states. The hybrid set, by capturing both instantaneous and spectral characteristics, provides comprehensive discriminative capability, leading to superior performance.

5.2.3 Visualization and Interpretation of Classification Results

To further validate and visually interpret these results, normalized confusion matrices for both feature sets are depicted in Figure 5.1. Each confusion matrix cell is defined as the proportion of true class instances classified into each predicted class, mathematically expressed as:

$$C_{ij} = \frac{N_{ij}}{\sum_k N_{ik}} \tag{5.6}$$

where N_{ij} denotes the number of instances of true class i predicted as class j. Thus, each row sums to one, clearly displaying misclassification distributions.

Time-Only Features (Figure 5.1a): This matrix visually highlights the limitations of solely using instantaneous features. While stationary instances exhibit a moderate recall of 74.89% (True 0, Predicted 0), a substantial proportion of moving instances are misclassified as stationary. Specifically, 40.66% of actual moving samples are incorrectly predicted as stationary (derived from $1 - \text{Recall}(1) = 1 - 0.5934$). This significant false-negative rate for the moving class ($1 \rightarrow 0$ misclassification) fundamentally undermines system reliability, as the classifier frequently fails to detect critical user movement states demanding high-quality network management. This limitation stems from the inherent ambiguity of simple time-domain features, as stationary users occasionally produce temporal packet patterns that may resemble those of moving users performing small or subtle interactions.

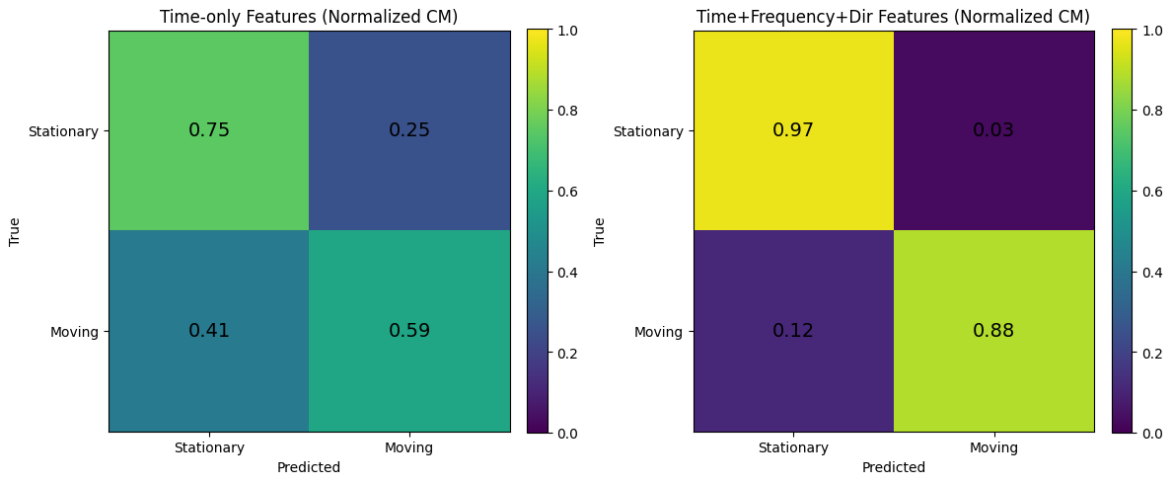


Figure 5.1: Row-normalized confusion matrices on the **test** set comparing (a) **Time-only** features and (b) **Hybrid** Time+Frequency+Dir features. Values are proportions (rows sum to 1). Class mapping: 0=Stationary, 1=Moving.

Hybrid Time+Frequency+Dir Features (Figure 5.1b): In marked contrast, the hybrid model's confusion matrix shows excellent classification results. Stationary instances achieve greatly improved recall (96.96%), indicating exceptional robustness in correctly recognizing static scenarios with very few false negatives ($0 \rightarrow 0$ correct predictions). Moving instances also exhibit greatly improved recall (88.52%), drastically reducing misclassification rates. The false-negative rate for the moving class (i.e., moving instances predicted as stationary) shrinks dramatically from 40.66% (Time-Only) to 11.48% (derived from $1 - \text{Recall}(1) = 1 - 0.8852$), representing a substantial relative reduction of approximately 71.8%. This profound improvement underscores the significant advantage offered by the comprehensive hybrid feature set, where multi-scale spectral analysis, instantaneous features, and directional information work in concert to reveal underlying characteristic periodicities and traffic patterns explicitly linked to VR user movement, thereby greatly enhancing discriminative power.

5.3 Hyperparameter and Sensitivity Analysis

This section provides a detailed analysis of the impact of key hyperparameters and feature design choices on the performance of the Virtual Reality (VR) user state classifier. Through systematic experimentation and ablation studies, we investigate the sensitivity of the Random Forest model to the number of top spectral peaks extracted (N), the inclusion of different time-window scales, the contribution of the communication direction bit, and the number of decision trees (`n_estimators`) in the forest. These analyses are crucial for confirming the robustness of our feature engineering strategy and justifying the chosen model configuration.

5.3.1 Selection of Top-N Spectral Peaks

The number of dominant spectral components, N, extracted from each time window, is a critical hyperparameter that influences the trade-off between capturing rich frequency-domain information and avoiding noise or excessive dimensionality. To de-

termine the optimal value for N, we conducted a grid search across $N \in \{1, 3, 5, 7, 9\}$. For each N, a Random Forest classifier was trained using the full hybrid feature set (Time+Frequency+Dir), and its performance was evaluated on the validation and test sets.

Table 5.2 summarizes the key performance metrics and training times for each tested value of N. The results indicate that the classification performance generally increases with N up to a certain point, then plateaus or slightly declines. Specifically, the highest test accuracy of 93.70% and F1-score for the Moving class (0.9149) were achieved with $N = 7$. Increasing N to 9 led to a slight decrease in accuracy and F1-score, suggesting that additional, less significant spectral peaks may introduce noise without substantial informational gain.

Table 5.2: Grid Search Summary for TOP-N Spectral Peaks (Full Hybrid Feature Set)

TOP_N	VAL_P(0)	VAL_R(0)	VAL_F1(0)	VAL_P(1)	VAL_R(1)	VAL_F1(1)	VAL_Acc	TEST_P(0)	TEST_R(0)	TEST_F1(0)	TEST_P(1)	TEST_R(1)	TEST_F1(1)	TEST_Acc
1	0.9284	0.9607	0.9443	0.9328	0.8803	0.9058	0.9300	0.9283	0.9696	0.9485	0.9471	0.8790	0.9118	0.9350
3	0.9256	0.9625	0.9437	0.9353	0.8750	0.9041	0.9291	0.9287	0.9688	0.9483	0.9459	0.8797	0.9116	0.9348
5	0.9276	0.9622	0.9446	0.9351	0.8786	0.9060	0.9303	0.9297	0.9673	0.9481	0.9435	0.8818	0.9116	0.9346
7	**0.9286**	**0.9646**	**0.9463**	**0.9391**	**0.8802**	**0.9087**	**0.9324**	**0.9315**	**0.9693**	**0.9500**	**0.9469**	**0.8849**	**0.9149**	**0.9370**
9	0.9277	0.9642	0.9456	0.9382	0.8786	0.9074	0.9315	0.9291	0.9699	0.9491	**0.9477**	0.8805	0.9128	0.9357

Figure 5.2 further illustrates these trends by plotting the test accuracy, F1-score for the Moving class, precision for the Moving class, and recall for the Moving class as a function of N. The curves clearly show that $N = 7$ represents an optimal balance, providing robust performance without excessive feature dimensionality or computational overhead.

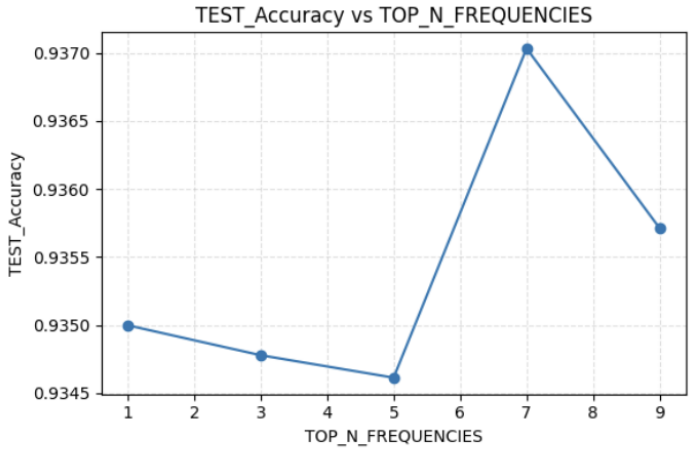
The reason for this behavior is two-fold: smaller values of N (e.g., 1 or 3) may lead to underfitting, as they fail to capture the multi-peak or harmonic structures inherent in VR traffic, which are crucial for differentiating user states. Conversely, excessively large values of N (e.g., 9) begin to incorporate weaker, potentially noisy spectral components that do not contribute meaningfully to classification and can even lead to slight overfitting. Thus, $N = 7$ effectively captures the essential spectral characteristics while maintaining robustness.

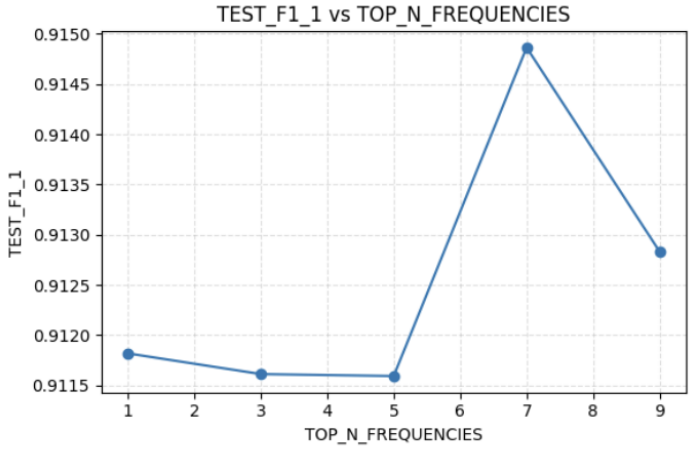
5.3.2 Impact of Multi-Scale Windows (Ablation of 0.5s Window)

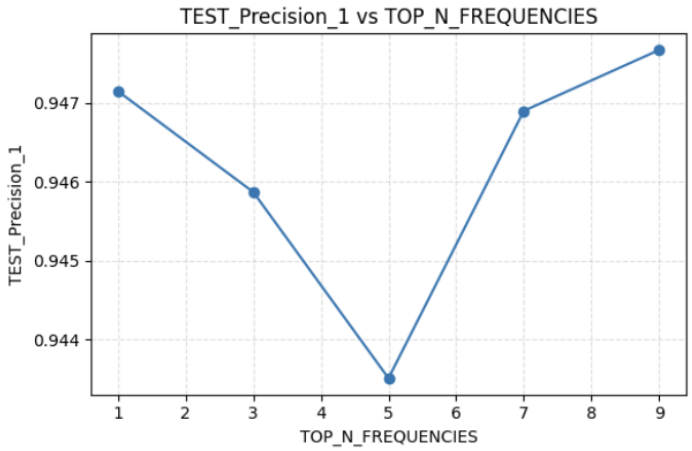
Our feature engineering incorporates three distinct time-window scales (0.5s, 1.0s, and 2.0s) to capture a range of temporal dynamics. To assess the importance of the shortest 0.5s window, we performed an ablation study by training the Random Forest classifier with a feature set that excluded all spectral features derived from the 0.5s window, while retaining features from the 1.0s and 2.0s windows, along with time-domain features and the direction bit. The performance of this configuration was compared against the full hybrid (Time+Frequency+Dir) model using all three scales.

Table 5.3 presents the comparison of performance metrics for these two configurations. The removal of the 0.5s window resulted in a noticeable drop in performance. Specifically, the test accuracy decreased from 0.9374% (full hybrid) to 0.9270% (1.0s, 2.0s only), a decline of approximately 1.04 percentage points. The F1-score for the Moving class also decreased from 0.9153 to 0.9010 (a drop of 1.43 percentage points), and Recall for the Moving class decreased from 0.8852 to 0.8694 (a drop of 1.58 percentage points).

This outcome suggests that while features from longer time windows (1.0s and 2.0s) are highly informative for capturing stable periodicities, the 0.5s window provides







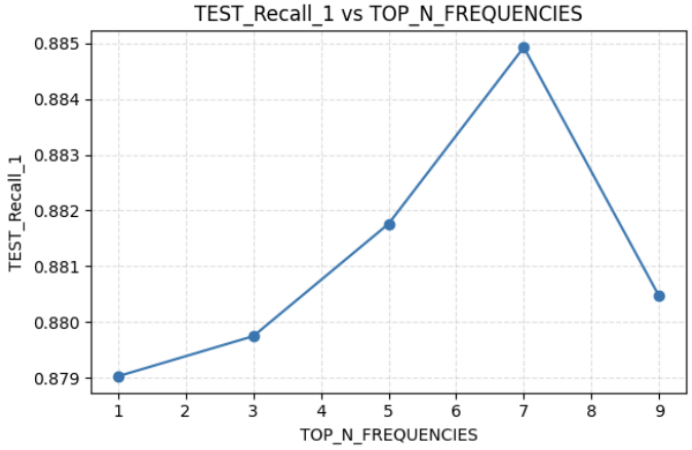


Figure 5.2: Classification performance metrics on the test set as a function of TOP-N spectral peaks. (a) Test Accuracy, (b) F1-Score (Moving), (c) Precision (Moving), (d) Recall (Moving).

critical, fine-grained information that is essential for detecting rapid state transitions or short, intense bursts of traffic characteristic of certain user movements. Despite its lower individual feature importance compared to some longer-scale features, the 0.5s window serves as an important complementary scale, contributing to the overall discriminative power and robustness of the model, especially in scenarios where quick detection of subtle changes is paramount.

Table 5.3: Ablation Study: Impact of Removing the 0.5s Time Window (Test Set)

Feature Configuration	TEST_P(0)	TEST_R(0)	TEST_F1(0)	TEST_P(1)	TEST_R(1)	TEST_F1(1)	TEST_Acc
Full Hybrid (0.5s, 1.0s, 2.0s + Dir)	**0.9317**	**0.9696**	**0.9503**	**0.9475**	**0.8852**	**0.9153**	**0.9374**
Hybrid (1.0s, 2.0s Only + Dir)	0.9225	0.9626	0.9421	0.9350	0.8694	0.9010	0.9270

5.3.3 Sensitivity to Number of Estimators in Random Forest Classifier

The number of decision trees in the Random Forest ensemble, `n_estimators`, is a crucial hyperparameter affecting both model performance and computational cost. To ensure an optimal choice, we evaluated the model's performance across a range of `n_estimators` values: 100, 200, 300, 500. The best-performing hybrid feature set (Time+Frequency+Dir with $N = 7$) was used for this analysis.

Table 5.4 presents the test accuracy, F1-score for the Moving class, and training time for each `n_estimators` value.The results demonstrate that the model's performance quickly saturates. Test accuracy and F1-score for the Moving class show extremely marginal differences (less than 0.02 percentage points) across all tested values of `n_estimators`, effectively reaching a plateau. The `n_estimators=100` setting achieved the highest (or tied for highest) accuracy and F1-score with the shortest training time. While`n_estimators=300` offered effectively identical performance with a slightly longer training time, its selection (as used in other sections) ensures a robust and stable ensemble, providing a desirable balance between computational cost and a comprehensively trained model. This confirms that the inherent complexity of the VR traffic patterns can be effectively captured even with a moderately sized forest.

Table 5.4: Sensitivity Analysis: Impact of `n_estimators` on Classification Performance (Validation & Test)

n_estimators	VAL_P(0)	VAL_R(0)	VAL_F1(0)	VAL_P(1)	VAL_R(1)	VAL_F1(1)	VAL_Acc	TEST_P(0)	TEST_R(0)	TEST_F1(0)	TEST_P(1)	TEST_R(1)	TEST_F1(1)	TEST_Acc
100	0.9285	0.9648	0.9463	0.9394	0.8800	0.9087	0.9324	**0.9317**	**0.9696**	**0.9503**	**0.9475**	**0.8852**	**0.9153**	**0.9374**
200	0.9287	0.9647	0.9464	0.9392	0.8803	0.9088	0.9325	0.9316	0.9695	0.9502	0.9472	0.8851	0.9151	0.9372
300	0.9287	0.9647	0.9464	0.9392	0.8803	0.9088	0.9325	0.9316	0.9695	0.9502	0.9472	0.8851	0.9151	0.9372
500	0.9287	0.9647	0.9464	0.9392	0.8803	0.9088	0.9325	0.9316	0.9695	0.9502	0.9472	0.8851	0.9151	0.9372

5.4 Feature Importance Analysis

Beyond merely achieving high accuracy in the classification task, it is crucial to understand *why* and *how* the classification decisions are made by the model. Such insights provide interpretability, enhance trust, and allow the researcher to relate model performance to underlying domain-specific behaviors. Random Forest classifiers inherently provide mechanisms to measure feature importance based on the reduction in the **Gini impurity criterion** [74]. Gini impurity measures the likelihood of incorrect classification of a randomly chosen element, assuming random class labeling according to

the observed class distribution.This metric quantifies how effectively each feature reduces classification uncertainty across all trees in the forest. The importance of a given feature X_j within a Random Forest is defined mathematically as:

$$\text{Importance}(X_j) = \frac{1}{T}\sum_{t=1}^{T}\sum_{k\in\text{splits}(X_j,t)} w_{tk}\Delta i_{tk}, \tag{5.7}$$

where:

- T is the total number of trees in the forest,
- $\text{splits}(X_j, t)$ denotes all nodes in tree t where splits occur using feature X_j,
- w_{tk} is the proportion of samples reaching node k in tree t,
- Δi_{tk} is the reduction in Gini impurity at node k.

5.4.1 Ranking of Feature Importance

Figure 5.3 visually represents the ranked importance of features used in the combined time-frequency model. The results indicate a three-tiered hierarchy in discriminative power:

1. **Top Tier (Most Important):** The `dir_bit` consistently emerges as the most important feature, followed closely by several multi-scale spectral features. These prominently include spectral shape and frequency location descriptors from the 1.0s and 2.0s time windows, such as the Nyquist-normalized frequencies of the dominant peaks (e.g., `s2.00_f1_nyqnorm`, `s1.00_f1_nyqnorm`), spectral centroid (e.g., `s2.00_centroid`, `s1.00_centroid`), and spectral flatness (e.g., `s2.00_flatness`, `s1.00_flatness`). These features describe the overall distribution of spectral energy and the locations of dominant periodicities, proving critical for differentiating user states. The high importance of `dir_bit` suggests a strong statistical relationship between communication direction and user mobility patterns in our dataset, likely reflecting distinct uplink/downlink traffic profiles for specific Unity Netcode message types (e.g., client sending pose updates, server sending object sync).

2. **Middle Tier (Moderately Important):** This group typically includes the relative amplitudes of the dominant spectral peaks (e.g., `sX.XX_amp1` through `sX.XX_amp7`) from various time windows, as well as the robustly scaled packet size of the last packet (`ps_last`). While their individual contributions are generally lower than the top spectral shape and frequency descriptors, these features provide valuable, complementary information, refining the classification by quantifying the magnitude of periodic traffic components and instantaneous load.

3. **Bottom Tier (Least Important):** The robustly scaled inter-arrival time of the last packet (`dt_last`) generally contributes the least to the model's predictive power. This reinforces that while inter-arrival time captures immediate temporal dynamics, its standalone discriminative utility for broad state classification is limited compared to the aggregated and transformed spectral characteristics. Similarly, some higher-order spectral features (e.g., `s0.50_amp7`, `s0.50_f7_nyqnorm`)

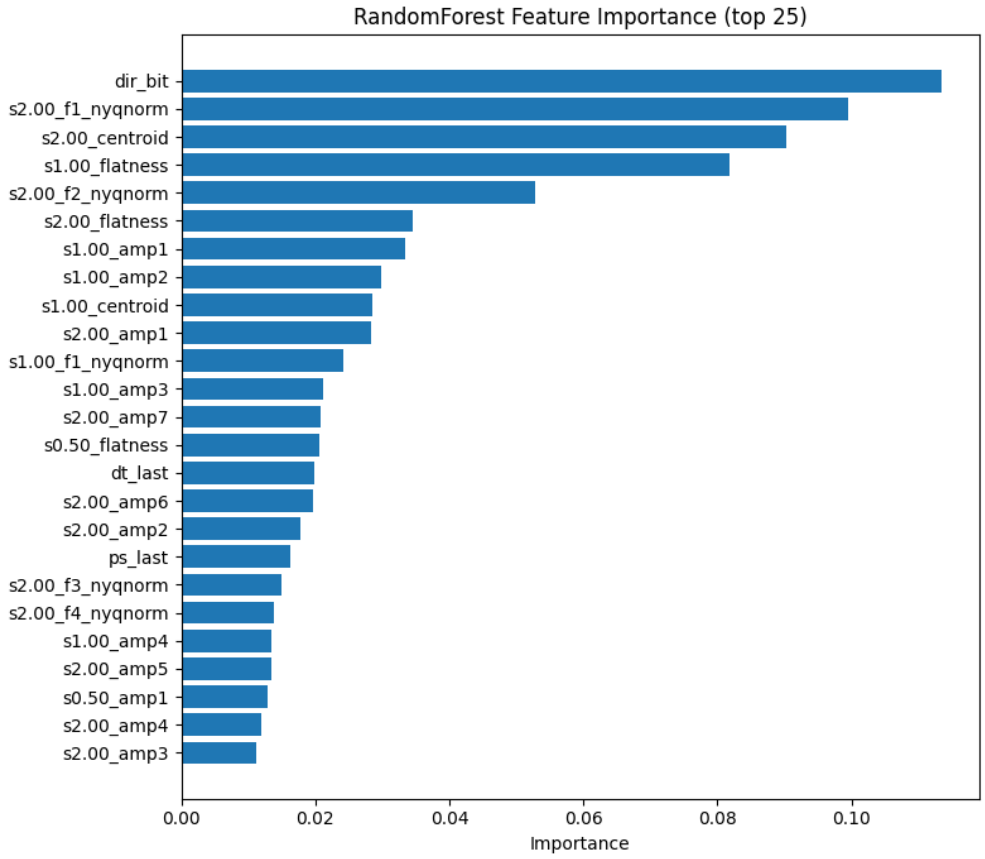


Figure 5.3: Top-25 feature importances (mean decrease in Gini impurity) of the Random Forest trained with the hybrid feature set (Time + Multi-scale Spectrum + Dir); best setting with top-7 peaks per window. **Naming legend**: `sX.XX_ampj` = relative amplitude of the j-th largest positive-frequency peak in the X.XX-s window; `sX.XX_fj_nyqnorm` = that peak's frequency divided by the Nyquist (range 0–1); `sX.XX_ratio12`/`sX.XX_ratio23` = A_1/A_2, A_2/A_3; `sX.XX_centroid` = $\sum fA / \sum A$; `sX.XX_flatness` = $\exp(\text{mean}(\log A))/\text{mean}(A)$; `ps_last`/`dt_last` = robustly scaled last-packet size / last inter-arrival time; `dir_bit` = 0 for server→client, 1 for client→server. **Visual takeaway**: `dir_bit` ranks highest; spectral *position/shape* features from 1.00 s/2.00 s windows (e.g., `f1_nyqnorm`, `centroid`, `flatness`) dominate; amplitudes are mid-ranked; 0.50 s features appear less often individually. Note that impurity-based importances describe how the forest splits data and need not equal gains in held-out metrics

and validity flags (`sX.XX_valid`) also reside in this tier. Although individual features from the 0.5s window may appear lower in rank, the ablation study in Section 5.3.2 demonstrates that the collective inclusion of 0.5s spectral features contributes significantly to the overall F1-score for the "Moving" class, underscoring their importance in capturing rapid, short-term dynamics.

5.4.2 Dominance of Spectral Amplitude Features

A critical observation from the feature importance analysis is the pronounced dominance of spectral shape descriptors (such as 'centroid' and 'flatness') and Nyquist-normalized frequencies from the 1.0s and 2.0s time windows. This strongly indicates that the characteristic patterns of periodic packet size variations serve as the principal differentiator between stationary and moving user states. Our implementation of spectral features involves de-centering each windowed packet size sequence prior to FFT (by subtracting the local mean) and then deriving relative amplitudes (`amps_rel`) normalized by the total spectral energy. Consequently, the amplitude features (`ampj`) primarily reflect the *distribution of energy across spectral peaks* rather than the absolute magnitude of packet size variations. This methodological choice naturally places greater emphasis on spectral shape and frequency location (e.g., 'centroid', 'flatness', 'f_nyqnorm') as key discriminators, aligning with their higher importance rankings. These shape descriptors are particularly effective as they capture the overall distribution of energy across frequencies. For instance, a "bursty" traffic pattern with distinct, strong periodic components (typical of user movement) might exhibit a different spectral centroid and flatness compared to more sparse or uniformly distributed background traffic.

The prominence of features from the 1.0s and 2.0s windows suggests that these timescales are optimal for capturing the underlying periodicities and changes in traffic volume associated with user mobility, providing sufficient data points for stable FFT computation (as discussed in Section 4.7). The 0.5s window, while capturing more rapid changes, yields spectral estimates that can be less stable, especially in sparser traffic, which is consistent with its lower-ranked individual features. However, as demonstrated by the ablation study, the collective contribution of the 0.5s window to overall classification performance, particularly in terms of 'Moving' class F1-score, is significant, highlighting its role in capturing short-lived, high-frequency traffic dynamics. This multi-scale approach effectively combines responsiveness with robustness in spectral characterization.

5.5 Discussion: Implications for State-Aware VR Resource Management

The demonstrated capability of accurately classifying Virtual Reality (VR) user mobility states, achieving over 93.70% accuracy through the proposed hybrid time-frequency feature approach, serves as a foundational enabling technology for intelligent and adaptive management of network resources. This section discusses the practical implications and potential pathways for leveraging this high-fidelity state awareness to profoundly enhance network efficiency and end-user Quality of Experience (QoE) in interactive VR environments..

5.5.1 Deployment Architecture and Traffic Visibility in 5G/6G Networks

Operationalizing the proposed state classifier and short-term predictor (from Chapter 6) in a carrier network necessitates a clear understanding of its architectural integration within modern wireless communication systems. We envision a modular and flexible deployment, which highlights key placement options and data flows.

Placement of Classification and Prediction Elements

The core intelligence for VR user state classification and packet size prediction (i.e., the feature extraction module and the trained Machine Learning models) could be flexibly deployed at various points in the 5G/6G network architecture, each offering distinct advantages:

1. **UE-side Inference (On-headset):** The classifier could run directly on the VR headset. It processes local network traffic and infers the user's mobility state. This state signal (e.g., 'Moving'/'Stationary' with confidence) is then exported to the network via an Application Function (AF), which interfaces with the 5G Core through the Network Exposure Function (NEF) [138]. The Policy Control Function (PCF) can then apply state-dependent QoS policies. This option offers minimal latency for state detection and avoids network-side traffic inspection, though it relies on UE resources and specific integration.

2. **RAN-side Inference (gNB / O-RAN Near-RT RIC):** This is often the preferred deployment for low-latency control. The module can be embedded within the gNodeB (gNB) (specifically its CU/DU functions) or deployed as an xApp on an O-RAN Near-Real-Time Radio Intelligent Controller (Near-RT RIC) [139]. Here, the module accesses per-UE, per-Data Radio Bearer (DRB) counters (from PDCP/RLC layers) and extracts packet-level header information (e.g., timestamps, lengths). The RAN has direct visibility and control over radio resources, enabling immediate adjustments to MAC scheduling parameters and uplink Configured Grant (CG) configurations within 10 ms–1 s timescales [140].

3. **MEC/UPF-side Inference:** A Multi-access Edge Computing (MEC) service, often co-located with a User Plane Function (UPF), can obtain per-flow statistics. This typically involves configuring Uplink Classifier (UL-CL) / Branching Point functionalities and 5-tuple Service Data Flow (SDF) filters at the UPF [141]. The MEC service processes these statistics, infers the state, and publishes this information to the PCF for policy enforcement. When RAN control is desired, the inference can be consumed by a Non-RT RIC (as an rApp) which issues A1 policies to the Near-RT RIC; the Near-RT RIC (as an xApp host) then enforces them over the E2 interface towards gNB O-CU/O-DU in near real time [142]. This offers greater processing power and a broader traffic view but may introduce slightly higher latency compared to RAN-side inference.

Traffic Visibility and Multi-User/Multi-Application Disambiguation

Our feature extraction and classification methodology is designed to support multi-user and multi-application scenarios with minimal network overhead and without requiring deep packet inspection (DPI):

1. **Traffic Flow Identification:** Each VR user's traffic can be robustly identified and isolated at the network layer. In 5G, this is achieved through PDU sessions and associated QoS Flows (identified by QoS Flow Identifiers, QFIs) which are mapped to specific DRBs at the RAN [138, 143]. At the UPF, individual application flows within a UE's PDU session can be further disambiguated using SDF templates (e.g., 5-tuple: source/destination IP, port, protocol) or distinct QFIs.

2. **Application-Agnostic Operation:** A key advantage of our approach is its minimal reliance on application-specific knowledge. Our models utilize generic network-level features (packet sizes, inter-arrival times, and their spectral characteristics) which are observable from packet headers and are robust to traffic encryption. This avoids the computational overhead, privacy concerns, and maintenance burden associated with DPI. While our current dataset is from a specific Unity VR application, the learned traffic patterns (e.g., amplitude of spectral peaks during movement) are hypothesized to generalize to other interactive VR applications with similar real-time state update mechanisms. Even with end-to-end encryption, packet sizes and timing remain observable at PDCP/RLC (RAN) and at UPF, which suffices for our time-frequency features; no DPI is required.

3. **Multi-Application Coexistence:** For a UE running multiple applications, the network can differentiate VR traffic (preferably using specific QFIs or SDF filters; port ranges are optional heuristics and may be unreliable with QUIC/HTTP/3) and apply our state classification only to relevant VR flows. Alternatively, the distinct spectral signatures of VR traffic during movement may allow the model to differentiate it even from co-existing background traffic.

5.5.2 State-to-Policy Mapping: From Classification to Concrete Scheduler/Core Parameters

The end-to-end actuation latency is denoted as T_{act}, which consists of three components: detection delay T_{detect}, signaling delay T_{signal}, and application delay T_{apply}; that is, $T_{\text{act}} = T_{\text{detect}} + T_{\text{signal}} + T_{\text{apply}}$. To achieve proactive gains, the prediction horizon H should be larger than the actuation latency, i.e., $H > T_{\text{act}}$. The core value of our highly accurate user state classification lies in its ability to drive proactive and precise adjustments to network resource allocation. Instead of abstract 'high' or 'low' capacities, the predicted user state $S(t) \in \{\text{Stationary}, \text{Moving}\}$ (potentially with a short prediction horizon from Chapter 6) can be mapped to concrete, numerically defined parameters that directly guide core network policies and RAN scheduling decisions. We do not propose a new scheduler; rather, we parameterize existing PF/MLWDF/5QI policies via $w_u(t)$, GFBR/MBR updates, and T_{CG} reconfiguration, using the predicted state as a lightweight control signal.

Core-side QoS Policies (PCF/SMF→RAN)

Upon detecting a predicted state transition to 'Moving', the PCF can dynamically trigger a PDU Session Modification procedure to update a *Guaranteed Bit Rate (GBR) QoS Flow* for the VR application. The target Guaranteed Flow Bit Rate (GFBR) can be dynamically calculated, for example, based on the short-term predicted average

packet rate $\hat{\lambda}(t)$ and size $\hat{s}(t)$ (from Chapter 6), with a headroom factor γ:

$$\text{GFBR}(t) = (1+\gamma)\,\hat{\lambda}(t)\,\hat{s}(t),$$

where $\gamma \in [0.1, 0.3]$ allows for absorbing short-term traffic bursts. The headroom factor γ is selected via validation-set grid search, optimizing queueing delay and jitter under different traffic loads. Conversely, for a 'Stationary' prediction, the flow could revert to a Non-GBR QoS Flow or a lower GFBR(t) to conserve resources. This QoS profile, including the 5G QoS Identifier (5QI), Allocation and Retention Priority (ARP), and GFBR/MBR, is signaled to the NG-RAN as per 3GPP TS 23.501/23.502 [138, 143]. The RAN then enforces this profile at the DRB level by allocating physical layer resources according to 3GPP TS 38.213 [144]. Operational feasibility depends on operator policy and feature support (dynamic QoS updates at PCF/SMF, UE and gNB support for QoS profile change and CG reconfiguration).

RAN MAC Scheduling Weights

For fine-grained, per-Transmission Time Interval (TTI) resource allocation at the MAC layer, our classification can directly influence the scheduler's weighting scheme. For instance, in a weighted proportional-fair (PF) scheduler, the weight for a specific UE u (or QoS flow q) can be adjusted based on its predicted state $S_u(t)$. If the scheduler maximizes $\sum_u w_u(t)\,\frac{r_u(t)}{\bar{r}_u(t)}$ (where $r_u(t)$ is instantaneous spectral efficiency and $\bar{r}_u(t)$ is the long-term average), the state-aware weight $w_u(t)$ can be defined as:

$$w_u(t) = w_0 + \kappa\,\mathbb{1}[\cdot]\{S_u(t) = \text{Moving}\}.$$

where $\kappa > 0$ is a configurable boost factor for moving users, and $\mathbb{1}[\cdot]$ is the indicator function. This mechanism proactively prioritizes moving users in contention for radio resources (Physical Resource Blocks, PRBs) while preserving overall fairness objectives [145, 146, 147].

Uplink Latency Optimization via NR Configured Grant (CG)

To specifically address uplink latency for interactive VR, especially during movement, our classifier can inform the dynamic configuration of NR Configured Grants (CG). For a predicted 'Moving' state, the gNB can configure a shorter CG periodicity $T_{\text{CG}}(t)$ or a higher repetition factor, directly reducing uplink access latency and signaling overhead (e.g., Buffer Status Reports, Scheduling Requests) [140, 148]. Conversely, for a 'Stationary' state, $T_{\text{CG}}(t)$ can be extended to conserve air-interface resources and device energy.

5.5.3 Potential QoE Enhancements and Resource Efficiency

The accurate, proactive state-awareness enabled by the proposed classification method (and subsequent prediction from Chapter 6) offers significant potential for tangible QoE enhancements and improved resource utilization. However, it is crucial to state that while this work provides the enabling technology for these benefits, full system-level validation is beyond the scope of this particular chapter and will be addressed in future work.

Mitigation of Motion Sickness and Reduced Latency Potential

Motion sickness in VR is strongly correlated with inconsistencies between visual stimuli and vestibular feedback, often exacerbated by network-induced latency and jitter. By proactively allocating bandwidth and prioritizing traffic based on predicted 'Moving' states (as outlined in Section 5.5.2), the system is theoretically expected to help reduce network-induced queuing latency. In queueing theory, for instance, increasing service capacity (μ) relative to arrival rate (λ) directly reduces waiting times (e.g., in an M/M/1 queue, $E[W_q] = \frac{\rho}{\mu-\lambda}$) [149, 150]. This proactive approach, by maintaining network latency L_{net} below critical thresholds, aims to help maintain visual-vestibular synchronization, thereby reducing the likelihood of motion sickness and improving overall VR QoE [44, 151].

Optimized Responsiveness and Immersion Potential

Interactive actions such as head movements or hand gestures demand minimal delay to be accurately reflected in the virtual environment, which is vital for preserving user immersion. By treating the accurate user state classification and robust short-term traffic load prediction as a proactive signal, network schedulers are empowered to allocate network capacity ahead of time. This anticipatory resource provisioning is expected to foster low-latency responsiveness, particularly for time-critical pose updates, thereby helping to preserve user immersion.

Enhanced Resource Utilization Efficiency Compared to Existing Schedulers

Traditional network schedulers often operate reactively (responding to current traffic queues or congestion signals) or based on coarse-grained, static QoS rules (e.g., allocating a fixed GBR for all VR traffic). This typically leads to two inefficiencies: 1) Over-provisioning: During periods when a VR user is stationary and traffic demand is low, a static high-bandwidth allocation results in wasted radio resources. 2) Under-provisioning (Reactive Delay): When a user suddenly transitions to a 'Moving' state, a purely reactive scheduler may experience a delay in ramping up resources, leading to temporary congestion and QoE degradation before it can adapt. In contrast, our proactive, state-aware classification enables a dynamic, fine-grained resource allocation strategy. By aligning resource allocation precisely with the real-time, fine-grained user activity, it can improve overall network utilization efficiency relative to existing schedulers, a hypothesis we will validate with system-level experiments in future work.

Potential for Proactive Congestion Control

The state-awareness capability, derived from our accurate classification, offers significant potential to enhance congestion control mechanisms within network infrastructures. Existing congestion control strategies (e.g., TCP variants like CUBIC, BBR, or QUIC's loss-based controllers) are predominantly reactive, responding to symptoms of congestion such as increased queue sizes or packet loss. By providing early detection of user state transitions, our approach empowers network controllers to proactively anticipate shifts in traffic demand profiles. This proactive information can be integrated into advanced congestion control algorithms, for instance, by adjusting sender-side transmission rates or network-side buffer management policies before congestion manifests.

In summary, the proposed hybrid time-frequency feature-based VR user state classification method contributes significantly to the sophistication of network management strategies in immersive media systems. Through its ability to enable proactive, adaptive, and state-aware allocation of network resources, substantial enhancements in network performance, user experience, energy efficiency, and resource utilization are achievable, fundamentally advancing the state-of-the-art in interactive VR network support systems. While this chapter lays the foundational intelligence, future research will focus on full system integration and empirical validation of these proposed benefits.

Chapter 6

Short-Term VR Traffic Forecasting for Proactive Resource Allocation

Building upon the foundational traffic classification analysis presented in Chapter 5, this chapter addresses the critical challenge of proactively managing network resources by accurately forecasting Virtual Reality (VR) traffic. While classification provided essential insights into VR user states, effective network scheduling and Quality of Experience (QoE) optimization in highly dynamic immersive environments require anticipating traffic demands rather than solely reacting to observed conditions. Thus, the focus shifts toward the prediction of future network load, specifically the average packet size over defined short-term horizons. This forecasting perspective intentionally avoids instantaneous packet-level predictions, opting instead for stable and practically actionable estimates suited to operational network control mechanisms.

To address the inherent complexity and stochasticity characteristic of VR network traffic, we introduce a prediction framework based on *forecasting averaged traffic metrics* over short time intervals. This approach leverages aggregated statistics rather than individual packet-level forecasts, significantly reducing sensitivity to transient fluctuations and providing stable predictions suitable for proactive resource allocation strategies. For comprehensive evaluation, we utilize the real-world VR traffic dataset described in Chapter 4, collected from a custom-built Unity VR application.

In this chapter, we systematically evaluate multiple state-of-the-art deep learning architectures, including Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Transformer-based models, alongside a robust Random Forest regressor, across varying prediction horizons ranging from very short-term intervals of 20 ms to medium-term intervals up to 200 ms. Through detailed comparative analysis, we identify optimal horizons that provide reliable predictive accuracy, practical utility, and robust performance under diverse operational scenarios, ultimately informing network scheduling decisions to enhance user QoE.

6.1 Problem Formulation: Predicting Windowed Averages for Stable Traffic Load Estimation

Accurate short-term forecasting of network traffic load is vital for effective Quality of Service (QoS) and Quality of Experience (QoE) optimization, especially in latency-sensitive applications such as interactive VR. Traditional prediction approaches fre-

quently focus on forecasting instantaneous packet-level metrics, including individual packet sizes and inter-arrival times. However, VR traffic is characterized by inherent high-frequency variability arising from numerous complex factors, including the irregularity of user movements, frame rendering cycles, network congestion, and protocol overhead. Such instantaneous predictions, while theoretically precise, typically exhibit poor reliability and high volatility, limiting their practical effectiveness for network scheduling.

Recognizing these inherent challenges, we deliberately formulate the traffic prediction task as a regression problem focusing on the prediction of *windowed averages* of packet sizes over short-term future intervals. This formulation intentionally trades off instantaneous granularity for predictive stability, aligning closely with practical operational needs in contemporary network management systems, such as those employing Semi-Persistent Scheduling (SPS) strategies in 5G and beyond-5G networks [35]. We formalize our problem definition as follows:

Let us consider a time-ordered sequence of VR network packets, indexed sequentially by integer k. Each packet k has an associated payload size s_k (expressed in bytes), and packets arrive at discrete time intervals, although their precise arrival is stochastic. At a given time step t, our objective is to predict a stable and operationally relevant metric—specifically, the *average packet size* over an upcoming future time window of fixed duration $\Delta t_{\text{horizon}}$. To achieve this, we first define the input feature matrix at time t:

$$X_t = [x_{t-W_{\text{input}}+1}, x_{t-W_{\text{input}}+2}, \ldots, x_t] \tag{6.1}$$

where:

- W_{input} is the length of the historical look-back window. In this study, it is fixed at $W_{\text{input}} = 20$, selected empirically based on preliminary model stability evaluations.

- x_k is the comprehensive feature vector extracted at the k-th time step, encapsulating both time-domain metrics (e.g., inter-arrival times, instantaneous packet size) and frequency-domain metrics (e.g., spectral coefficients derived from Fourier transformations described comprehensively in Section 4.5).The full set of 60 features was employed, resulting in an input dimensionality of $d = 60$.

Next, we formally define the target prediction metric, the average packet size Y_t over the future time horizon $\Delta t_{\text{horizon}}$. For each packet-anchored timestamp t_i, we define the regression target as the mean payload size of all packets whose arrival times fall in the fixed future window $(t_i, t_i + \Delta t]$:

$$Y_i = \begin{cases} \dfrac{1}{K_i} \displaystyle\sum_{k:\, t_i < T_k \le t_i + \Delta t} s_k, & K_i > 0 \\ 0, & K_i = 0 \quad \text{(unconditional labeling)} \end{cases} \tag{6.2}$$

where T_k and s_k are the arrival time and payload size of the k-th future packet, and K_i is the number of such packets in the window. The unified-sample protocol is employed, meaning we evaluate the same set of anchor timestamps $\{t_i\}$ across all horizons, thus avoiding horizon-specific selection bias at short Δt. This protocol further stipulates unconditional labeling: if no packets arrive within the specified future window $(t_i, t_i + \Delta t]$, the target Y_i is explicitly set to 0. This approach ensures all past timestamps that can form a valid input sequence are considered, regardless of future packet density.

In practice, predicting the exact number and sizes of packets precisely within a future interval is challenging due to the inherent randomness in packet arrivals. By averaging over a time window, our formulation aims to mitigate uncertainty for longer horizons and provides a consistent target across all time steps, aligning closely with practical operational needs. Moreover, this approach provides actionable forecasts well-suited to modern network schedulers that operate within discrete scheduling windows (commonly in tens to hundreds of milliseconds), such as Semi-Persistent Scheduling (SPS) in 5G/6G infrastructures.

For comprehensive evaluation, we systematically explore multiple prediction horizons, specifically $\Delta t_{\text{horizon}} = \{20, 40, 80, 150, 200\}$ milliseconds. The rationale for selecting these specific intervals is twofold. First, shorter intervals (e.g., (20–40 ms)) may directly support ultra-low latency schedulers but risk increased susceptibility to random noise due to averaging over very few (or no) packets. Second, medium-term intervals (80–200 ms) provide more stable forecasting signals at the cost of temporal resolution, striking a balance useful for real-time resource allocation and admission control decisions. This systematic analysis identifies the optimal temporal scales at which forecasting accuracy and stability are maximized, providing valuable operational insights into network management strategies aimed at proactively enhancing VR user QoE.

6.2 Deep Learning Models for Sequence Forecasting

To effectively model the temporal dependencies present in Virtual Reality (VR) network traffic sequences and forecast future average packet sizes, three established deep learning architectures—Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Transformer Encoders—along with a Random Forest regression model, were implemented and systematically compared. These models were selected due to their proven capability in sequential data analysis, robustness in capturing temporal relationships, and widespread use in traffic prediction tasks. Our objective was not merely to find a singular optimal model, but to rigorously benchmark these architectures against each other and against a strong non-sequential baseline to validate the utility of our hybrid time-frequency feature extraction methodology. The following section provides an in-depth description of each architecture, their mathematical foundations, specific model configurations, and a detailed justification for our methodological choices.

6.2.1 Random Forest (RF) Regression Model

To provide a strong non-parametric baseline for comparison against the deep learning architectures, we employed a Random Forest Regressor. This ensemble learning method operates by constructing a multitude of decision trees during training and outputting the mean prediction of the individual trees for regression tasks. It is robust to overfitting, handles high-dimensional feature spaces effectively, and provides insights into feature importance.

In our experiments, the Random Forest Regressor was configured with 100 decision trees and trained on the same standardized pointwise feature vectors (60 features) as used for the sequence models' inputs, but without forming sequences (i.e., each sample is an independent feature vector). The training was parallelized across all available

CPU cores. The outputs were then inverse-scaled to provide predictions in the original packet size units for evaluation.

6.2.2 Long Short-Term Memory (LSTM) Model

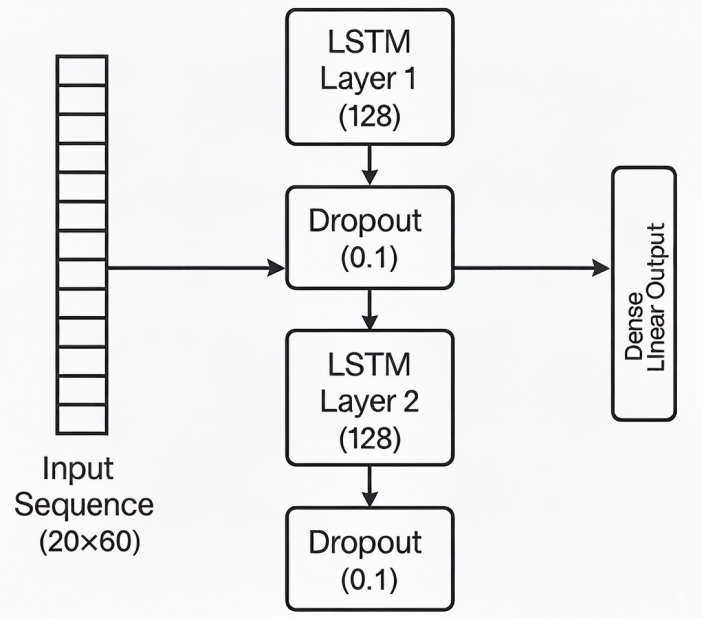


Figure 6.1: Architecture of the LSTM-based forecasting model with two stacked layers and dropout regularization.

The Long Short-Term Memory (LSTM) network is a specialized type of recurrent neural network (RNN) designed to mitigate the vanishing and exploding gradient problems through the incorporation of gating mechanisms. This characteristic enables the model to preserve information over extended temporal intervals, crucial for accurate network traffic forecasting, where distant past events may influence current and future behavior.

Our implemented LSTM architecture consisted of two stacked LSTM layers, each comprising 128 units, enabling the capture of hierarchical temporal features. To reduce overfitting and enhance generalization, a dropout layer with a rate of 0.1 was applied after each LSTM layer. The final output was generated by a dense layer with a linear activation, designed explicitly for regression tasks to forecast the continuous average packet size. The input shape was a tensor of dimension (W_{input}, D), where $W_{\text{input}} = 20$ is the sequence length and $D = 60$ is the dimensionality of the full feature set. The LSTM architecture implemented in our experiments can be explicitly described as follows:

1. **Input Representation:** The input to the LSTM is defined mathematically as a sequential set of vectors:

$$\mathbf{X}_t = [\mathbf{x}_{t-W_{\text{input}}+1}, \mathbf{x}_{t-W_{\text{input}}+2}, \ldots, \mathbf{x}_t] \in \mathbb{R}^{W_{\text{input}} \times d},$$

where $W_{\text{input}} = 20$ denotes the sequence length, and $d = 60$ corresponds to the dimensionality of feature vectors from the full feature set.

2. **Stacked LSTM Layers:** We employed a stacked (two-layer) LSTM structure to enhance the model's ability to capture hierarchical temporal features. Formally, for each LSTM layer, the transformation can be expressed as:

$$\mathbf{h}_t^{(l)} = \text{LSTM}(\mathbf{h}_{t-1}^{(l)}, \mathbf{h}_t^{(l-1)}), \quad l \in \{1, 2\},$$

with $\mathbf{h}_t^{(0)} = \mathbf{x}_t$. Each LSTM layer contains 128 hidden units, selected through preliminary experimentation balancing model complexity and computational cost. Input to the LSTM layers is preceded by Layer Normalization to stabilize training.

3. **Regularization via Dropout:** To prevent overfitting and improve generalization, a Dropout rate of 0.1 was applied to the outputs of each LSTM layer. Dropout randomly omits 10% of hidden units during training, thus promoting robustness.

4. **Output Layer:** The final output layer is a fully connected Dense layer with a single neuron and a linear activation function, defined as:

$$\hat{Y}_t = \mathbf{W}_o^\top \mathbf{h}_t^{(2)} + b_o,$$

predicting the continuous value of average packet size.

6.2.3 Gated Recurrent Unit (GRU) Model

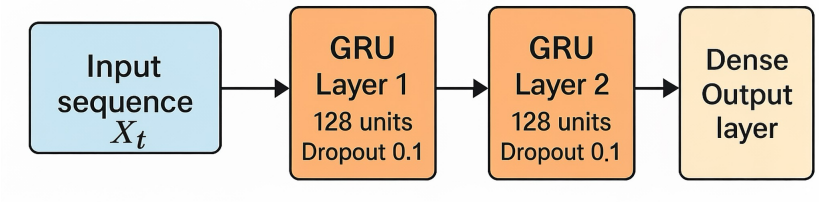


Figure 6.2: Architecture of the GRU-based forecasting model with two stacked layers and dropout regularization.

The GRU network simplifies the LSTM architecture by consolidating the forget and input gates into a single update gate, and by merging cell and hidden states into a unified hidden state vector. This simplification often results in reduced computational complexity and faster training convergence, with performance comparable to that of LSTM models.

Our GRU architecture mirrored that of the LSTM, consisting of two stacked layers with 128 units per layer, preceded by Layer Normalization and followed by dropout layers (rate = 0.1) and a final linear dense layer to yield the predicted average packet size. The input shape and preprocessing steps remained consistent with those of the LSTM model (i.e., 60 features without PCA), ensuring comparability. To facilitate a fair and direct comparison with the LSTM approach, we structured the GRU architecture analogously, as enumerated below:

1. **Input Representation:** The input sequence definition for GRU remains consistent with the LSTM, represented by the vector sequence $\mathbf{X}_t$ (with $d = 60$).

2. **Stacked GRU Layers:** Two stacked GRU layers, each comprising 128 hidden units, were employed. Input to the GRU layers is preceded by Layer Normalization. Each GRU layer's mathematical transformation can be stated as:

$$\mathbf{h}_t^{(l)} = \mathrm{GRU}(\mathbf{h}_{t-1}^{(l)}, \mathbf{h}_t^{(l-1)}), \quad l \in \{1, 2\}.$$

3. **Dropout Regularization:** Similar to the LSTM model, Dropout with a rate of 0.1 was applied to each GRU layer's outputs to enhance generalization capability.

4. **Output Layer:** A linear Dense layer identical to that described in the LSTM architecture was employed for final prediction output.

6.2.4 Transformer Encoder Model

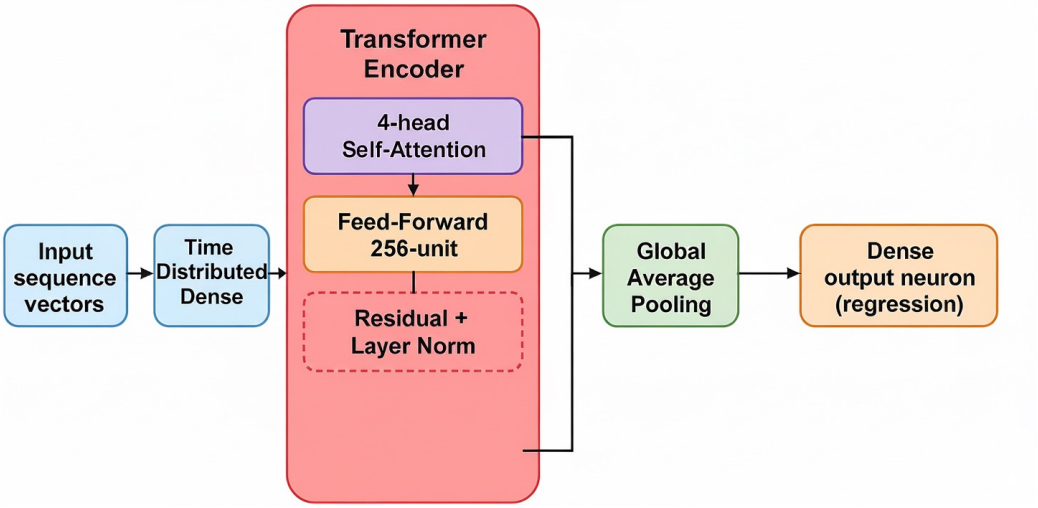


Figure 6.3: Transformer Encoder-based regression model architecture. Includes self-attention, feed-forward layers, residual connections, and global pooling.

The Transformer Encoder departs from recurrence and convolutional mechanisms, employing instead self-attention mechanisms to process all sequence elements simultaneously. This allows efficient modeling of long-range and non-local dependencies, advantageous in VR traffic prediction due to potentially complex temporal relationships across multiple time scales.

Our Transformer Encoder consisted of four components. First, a Layer Normalization layer followed by a time distributed (Dense) projection layer transforming the input into a 128-dimensional embedding space suitable for the self-attention mechanism. Second, a single Transformer encoder layer, featuring a multi-head self-attention layer with 4 attention heads, facilitating diverse relational representations across different subspaces; a position-wise feed-forward neural network with a hidden dimension of 256 units; and residual connections with layer normalization to stabilize and accelerate training. Third, a global average pooling layer to aggregate sequence-level information into a single fixed-length vector representation. Fourth, a final dense layer with linear activation for regression prediction. The detailed model for our regression task was carefully structured as follows:

1. **Input Projection Layer:** To adapt input data for the self-attention mechanism, the input sequence vectors were initially passed through a Layer Normalization layer, then projected to a higher-dimensional representation via a time-distributed dense transformation:

$$\mathbf{X}_{\text{proj}} = \text{TimeDistributed(Dense)}(\text{LayerNormalization}(\mathbf{X}_t)),$$

projecting each feature vector (with $d = 60$) into a 128-dimensional embedding space. Learned positional embeddings are added to these projected features to incorporate sequence order information.

2. **Transformer Encoder Block:** A single Transformer Encoder block was employed, comprising:

   - **Multi-Head Self-Attention Layer:** Utilizing 4 parallel attention heads, each head attends to different positions of the input sequence simultaneously, capturing diverse representation subspaces. The attention mechanism is mathematically formalized as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

     where Q, K, V represent queries, keys, and values, respectively, and d_k is the dimensionality per head.

   - **Feed-Forward Neural Network (FFN):** The Encoder block contains a feed-forward neural network defined as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2,$$

     with hidden layer dimensions set to 256 units.

   - **Layer Normalization and Residual Connections:** Each sub-layer employs residual connections followed by layer normalization to stabilize and accelerate training convergence.

3. **Sequence Aggregation via Global Average Pooling:** After processing through the Transformer Encoder layer, sequence outputs were aggregated into a single representative vector via a Global Average Pooling operation:

$$\mathbf{h}_{\text{agg}} = \frac{1}{W_{\text{input}}} \sum_{i=1}^{W_{\text{input}}} \mathbf{h}_i,$$

   condensing sequence information effectively.

4. **Final Output Layer:** Finally, the regression prediction was produced by passing the aggregated vector through a linear Dense layer as described for previous models.

Table 6.1: Hyperparameter Configuration for Deep Learning Models

Hyperparameter	**LSTM**	**GRU**	**Transformer**
Number of Layers	2 (stacked)	2 (stacked)	1 (encoder layer)
Units / Model Dim.	128	128	128
Attention Heads	N/A	N/A	4
Dropout Rate	0.1	0.1	0.1
Activation Function	Linear	Linear	Linear
Optimizer	Adam	Adam	Adam
Learning Rate	0.0005	0.0005	0.0005
Loss Function	MSE	MSE	MSE
Batch Size	512	512	512
Epochs (Max)	50	50	50
Early Stopping Patience	5	5	5

6.2.5 Common Training and Optimization Settings

All models were uniformly trained to minimize the Mean Squared Error (MSE), calculated as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{6.3}$$

where Y_i and $\hat{Y}_i$ represent the actual and predicted average packet sizes, respectively, and N is the number of samples.

The Adam optimizer, selected for its adaptive learning capability and efficient convergence properties, was configured with a learning rate of 0.0005. Training was conducted over a maximum of 50 epochs, with a batch size of 512. To prevent overfitting and optimize computational resources, early stopping was employed, halting training if validation loss failed to improve for 5 consecutive epochs. Additionally, a 'ReduceLROnPlateau' callback was used to reduce the learning rate by a factor of 0.5 if validation loss did not improve for 2 epochs, with a minimum learning rate of 1×10^{-5}. The training-validation-testing split ratio was maintained consistently at 70% for training, 15% for validation, and 15% for testing. Target values were standardized (z-scored) during training and inverse-transformed for evaluation. Table 6.1 summarizes the key hyperparameters used in all models. These methodological choices collectively established a rigorous foundation for robust and reproducible evaluation of the predictive capabilities of the chosen models in the context of VR network traffic analysis and prediction.

6.3 Performance Evaluation Across Prediction Horizons

This section presents the comprehensive experimental results of the short-term VR traffic forecasting task. We evaluate the performance of the Random Forest, LSTM, GRU, and Transformer models across the five distinct prediction horizons (20 ms, 40 ms, 80 ms, 150 ms, and 200 ms) using the unified-sample protocol with unconditional labeling.

Predictive performance was quantified using the coefficient of determination (R^2), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). While models were trained to minimize Mean Squared Error (MSE), we report RMSE and MAE because they are in bytes and therefore directly interpretable and comparable across horizons; RMSE is simply the square root of MSE. A lower RMSE and MAE indicate better performance. The coefficient of determination (R^2) provides a measure of how well the predictions approximate the true values, ranging from $-\infty$ to 1. An R^2 of 1 indicates perfect prediction, while 0 means the model explains no variance. The evaluation is structured to answer two primary questions: 1) How does the prediction horizon (i.e., the averaging window size) affect predictability under the unified-sample protocol, and 2) Which architecture (including the Random Forest baseline) is most effective for this specific task.

6.3.1 Impact of Averaging Window Size on Predictability

Table 6.2 summarizes the regression results, showcasing the R^2, RMSE and MAE values for each model at each averaging window. The results clearly indicate a strong correlation between the length of the averaging window and predictive performance, particularly due to the unified-sample, unconditional labeling approach.

Table 6.2: Multi-horizon forecasting results under unified-sample labeling. Performance metrics (R^2, RMSE, MAE) for each model across various prediction horizons.

Model	H (ms)	TEST R^2	TEST RMSE	TEST MAE
RF	20	0.4126	136.926	85.680
	40	0.3330	154.975	98.201
	80	0.3453	144.129	69.617
	150	0.5362	113.084	47.535
	200	**0.8586**	**59.922**	**26.540**
GRU	20	0.2294	156.805	97.389
	40	0.2352	165.964	111.982
	80	0.3136	147.559	80.368
	150	0.6093	103.784	48.576
	200	0.8576	60.133	28.178
LSTM	20	0.1422	165.441	103.935
	40	0.1411	175.881	117.227
	80	0.1967	159.629	88.970
	150	0.6118	103.455	49.304
	200	0.8531	61.076	28.571
Transformer	20	0.1194	167.632	110.449
	40	0.1382	176.176	116.075
	80	0.2686	152.325	81.102
	150	0.6101	103.677	49.433
	200	0.8529	61.127	29.272

Protocol Note: The evaluation across all horizons uses a unified anchor set and unconditional labeling (targets are explicitly set to 0 when no packets arrive in the future window). At short horizons (20—80 ms), a substantial number of windows contain few or no packets, creating a heavily skewed target distribution and consequently depressing R^2 values. This methodological choice removes horizon-specific selection bias, accurately reflects how a real scheduler must make decisions regardless of future packet density, and ensures fair cross-horizon comparisons. This explains why the short-horizon scores, while realistic, are generally lower than those from earlier conditional results, whereas the 150—200 ms horizons maintain strong performance.

Very Short Horizons (20 ms, 40 ms, 80 ms)

At these ultra-short intervals, predictive performance is notably poor, with R^2 values consistently below 0.4 across all models, often much lower. This unreliability stems from the inherent volatility of interactive VR traffic at the packet level combined with the unconditional labeling of the unified-sample protocol. These windows are often comparable to, or only slightly longer than, the average packet inter-arrival times observed in our dataset (approximately 32-86 ms, as discussed in Section 4.5.1). Consequently, a significant number of future windows may contain only 0 to 2 packets, or even no packets, in which case the target is set to 0. This frequent occurrence of zero targets, coupled with the high variance when few packets are present, introduces substantial noise and makes it exceedingly difficult for models to discern stable patterns. For instance, at 20 ms, the best R^2 is only 0.4126 (RF), with models like Transformer achieving as low as 0.1194. This suggests a fundamental limit to how fine-grained and accurate predictions can be for highly dynamic traffic without significant aggregation or conditional labeling.

Transitional Horizons (150 ms)

As the averaging interval increases to 150 ms, a substantial improvement in prediction quality is observed. Models like LSTM and GRU achieve an R^2 of approximately 0.61. While still modest, this duration begins to sufficiently smooth out short-term noise by averaging over a larger (though still variable) number of packets, allowing the models to capture more stable underlying traffic trends. This horizon represents a critical transition point where prediction stability becomes more feasible, offering a better balance between responsiveness and reliability than ultra-short horizons.

Robust Horizons (200 ms)

Performance strengthens significantly at the 200 ms horizon. All models consistently demonstrate robust predictions, achieving R^2 values exceeding 0.85 (e.g., RF reaching 0.8586). At this interval, the averaging window encompasses a larger number of packets (typically 5-15 packets depending on the scene and traffic direction), effectively dampening out high-frequency variations and allowing the models to capture more stable underlying traffic trends. This high accuracy and stability make this longer horizon particularly well-suited for practical network management applications where a reliable forecast over the immediate future is paramount.

6.3.2 Comparative Analysis of Models

Figure 6.4 illustrates the trend of R^2 across different forecast horizons for all models. The predictive performance generally improves as the forecast horizon (Δt) increases, demonstrating the advantage of averaging over longer time intervals to smooth out instantaneous network traffic volatility. However, the performance is notably irregular at very short horizons (20-80 ms), a phenomenon we attribute to the characteristics of the unified-sample, unconditional labeling protocol. The "unified-sample, unconditional labeling" strategy, where the target is set to 0 if no packets arrive in the future window, significantly influences the model behavior, especially at shorter horizons.

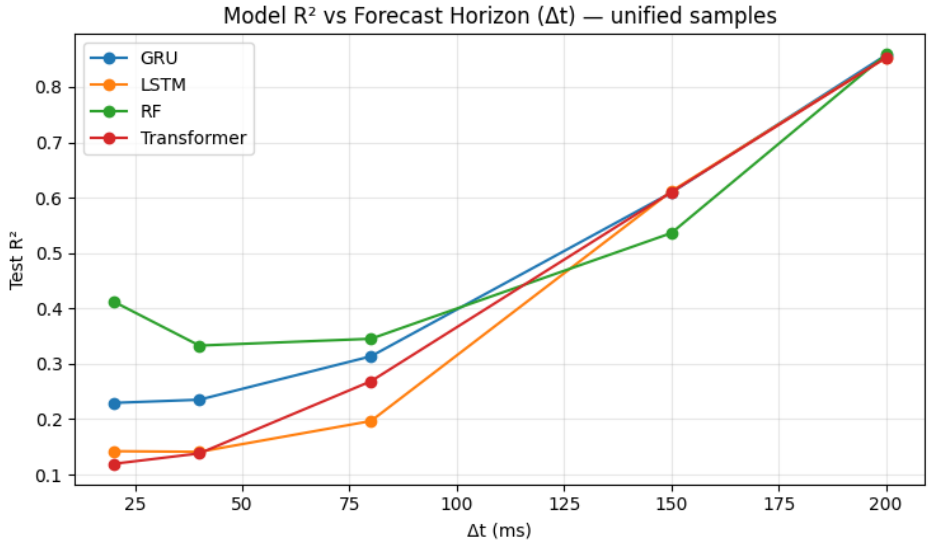


Figure 6.4: Test R^2 vs. forecast horizon (Δt) for all models under unified-sample labeling. Predictive performance significantly improves with longer averaging windows, indicating the challenge of forecasting at very short, volatile intervals.

In short horizons (20 ms, 40 ms, 80 ms) intervals, a large proportion of future windows contain very few or no packets, resulting in a target distribution heavily skewed towards zero. Models effectively learn to classify these "no-packet" instances, which, for a regressor like Random Forest, can significantly reduce the Sum of Squared Errors (SSE) when predicting zero. This leads to a relatively higher R^2 at 20 ms for RF (0.4126) compared to 40 ms (0.3330) and 80 ms (0.3453). This is because at 20 ms, the problem closely resembles a binary classification task (packet/no-packet), which RF handles well with its thresholding capabilities. As the horizon lengthens to 40-80 ms, the "no-packet" instances become less dominant, and the task shifts towards accurately regressing small positive values, where RF's piecewise-constant nature might be less advantageous, leading to a temporary dip in R^2. Deep learning models show lower R^2 in this range, as they struggle with the extreme noise and mixed classification-regression nature of the problem, with LSTM's R^2 dropping to 0.1411 at 40 ms. At transitional horizon (150 ms), the frequency of zero-value targets substantially decreases, and the averaging effect begins to yield more stable, non-zero targets. Deep learning models, particularly LSTM and GRU, start to outperform Random Forest, achieving R^2 values around 0.61 (e.g., LSTM 0.6118, GRU 0.6093), surpassing RF's 0.5362. This suggests that as the target distribution becomes more genuinely continuous, the sequence-modeling capabilities of RNNs become more effective. At the robust

200 ms horizon, the smoothing effect of averaging over more packets is fully realized, leading to highly consistent targets. All four models converge to strong performance, with R^2 values exceeding 0.85.

The Random Forest (RF) model proved to be highly competitive, marginally topping the deep learning models at the 200 ms horizon with an R^2 of 0.8586. This strong performance for a non-sequential model suggests that the carefully engineered time- and frequency-domain features (as detailed in Section 4.5) effectively capture most of the essential information for prediction, even without explicit sequence modeling by RNNs or Transformers. However, it incurred the highest training time (98.6 seconds) among the evaluated models, which could be a factor in resource-constrained environments.

The Recurrent Neural Network (RNN) variants, specifically LSTM and GRU, generally performed very similarly to each other and closely matched the Random Forest at 200 ms. GRU achieved an R^2 of 0.8576, and LSTM achieved 0.8531. Their sequential processing mechanisms, which maintain an evolving hidden state over time steps, align well with the quasi-periodic and causal structure of VR network traffic. They also exhibited significantly faster training times (16-25 seconds) compared to Random Forest, making them attractive for scenarios where training speed is critical.

While Transformer models are powerful for capturing long-range dependencies, they exhibited performance very close to that of LSTM and GRU in this study, slightly lagging behind but not dramatically (Transformer $R^2 = 0.8529$ at 200 ms). This could be attributed to the nature of dependencies in this VR traffic dataset primarily being short-to-medium range, which RNNs with their inductive bias towards local temporal continuity can efficiently model. The dataset size might also play a role, as Transformers typically have higher parameter counts and can be more data-hungry than LSTM/GRU models, or the single encoder block (rather than a deeper stack) used might limit its full potential for complex long-range patterns. Training times for the Transformer (24.5 seconds at 200 ms) were comparable to GRU.

In summary, for the most practical forecast horizon of 200 ms under the unified-sample, unconditional labeling protocol, all evaluated models achieve high predictive accuracy, demonstrating the strength of the hybrid time-frequency feature set. Random Forest provides a strong baseline, while deep learning models offer competitive accuracy with the advantage of faster training, particularly for LSTM. The choice between them may depend on specific operational requirements regarding model complexity, training budget, and interpretability.

6.3.3 Visualizing Prediction Accuracy

To provide a more intuitive and comprehensive understanding of the models' performance, Figure 6.5 to Figure 6.8 present a multi-panel diagnostic plot for the best-performing model (Random Forest) at the 200 ms forecast horizon, operating under the unified-sample, unconditional labeling protocol. This figure visually supports the quantitative metrics ($R^2 = 0.8586$, RMSE= 59.92, MAE= 26.54) reported in Table 6.2.

1. **Time-Series Overlay (Figure 6.5):** This figure displays the time-series traces of the actual average packet sizes (y_t) and their corresponding predicted values ($\hat{y}_t$) over a subsampled segment of the test set (n=500, uniform sampling). It demonstrates that the model generally tracks the main trends in traffic load well, with predictions closely following actual values during stable periods. However,

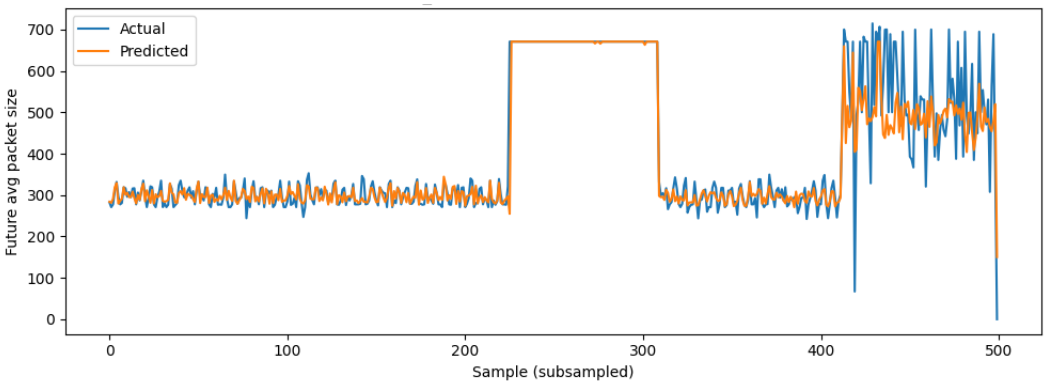


Figure 6.5: Predicted vs. actual average packet sizes for a subsampled segment for the Random Forest model at 200 ms forecast horizon (unified-sample labeling).

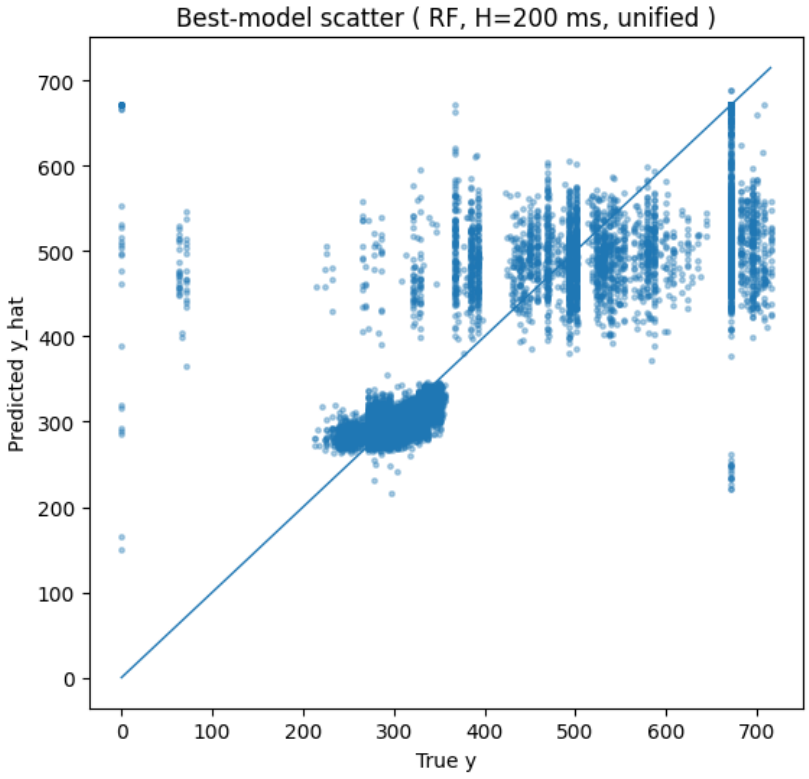


Figure 6.6: Scatter plot of predicted vs. actual values with a 45-degree reference line for the Random Forest model at 200 ms forecast horizon (unified-sample labeling).

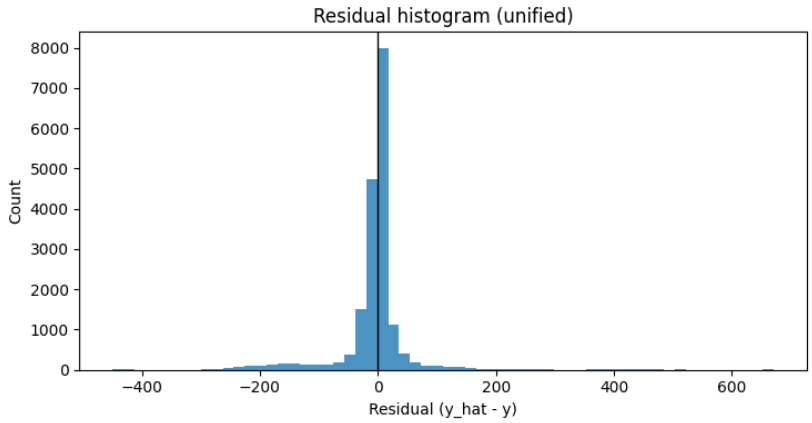


Figure 6.7: Residual histogram for the Random Forest model at 200 ms forecast horizon (unified-sample labeling).

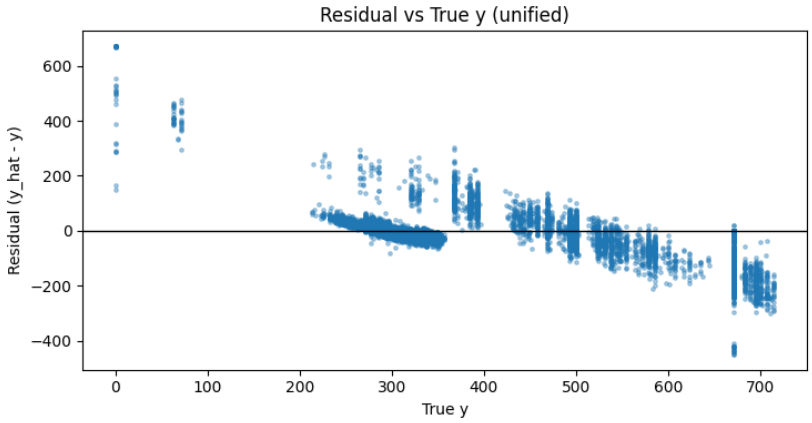


Figure 6.8: Residuals vs. actual values for the Random Forest model at 200 ms forecast horizon (unified-sample labeling).

during rapid changes or "bursty" events, such as the increase to 670 bytes or the subsequent drop to 300 bytes in the middle of the segment, the model exhibits some lag and a tendency for under- or over-estimation, reflecting the inherent challenge in immediately reacting to abrupt, non-periodic shifts in traffic patterns.

2. **Predicted vs. Actual Scatter Plot (Figure 6.6):** This scatter plot illustrates the relationship between individual predicted ($\hat{y}$) and actual (y) values, along with a 45-degree reference line where $\hat{y} = y$. The points are largely clustered around this line, indicating good overall calibration. A notable characteristic, typical of tree-based models, is the appearance of several distinct vertical bands in the scatter plot. These bands occur because decision trees output piecewise constant predictions (the mean of samples in a leaf node), leading multiple distinct input samples to map to the same predicted value. At higher actual values, the predictions tend to fall below the 45-degree line, suggesting a systematic underestimation of peak traffic loads. Conversely, at very low actual values, predictions tend to be slightly above the line, indicating a slight overestimation of low-traffic periods (regression to the mean).

3. **Residual Histogram (Figure 6.7):** The histogram of residuals ($\hat{y} - y$) shows a distribution centered around zero (mean residual ≈ 0.021), indicating minimal overall bias. However, it exhibits a distinct right-skewed shape with a long positive tail. This positive tail implies instances of significant overestimation (predicted value much higher than actual value). These often correspond to situations where the true packet size drops sharply (e.g., to zero due to no packets in the future window, or very low values during a quiet phase), but the model, reacting to recent history, still predicts a moderate-to-high value. The standard deviation of the residuals is approximately 59.92, which, as expected, is very close to the reported RMSE, given the near-zero mean residual.

4. **Residuals vs. Actual Values (Figure 6.8):** This plot displays the residual values against their corresponding actual target values (y). A clear negative slope is observed: as actual packet sizes increase, the residuals tend to become more negative, confirming the model's tendency to underestimate high traffic loads. Conversely, at very low actual values, residuals are often positive, indicating overestimation of low traffic loads. This pattern is consistent with the phenomenon of regression towards the mean, where the model produces predictions that are closer to the overall average than the actual extreme values. The spread of residuals also appears somewhat heteroscedastic, with a wider distribution of errors at higher actual values, suggesting that predicting high traffic bursts is inherently more challenging.

Collectively, these diagnostic plots confirm the Random Forest model's robust performance at the 200 ms horizon. While it excels at capturing overall trends and handling the mixed zero/non-zero target distribution of the unified-sample protocol, its tendency to regress towards the mean and exhibit some lag during sharp transitions presents opportunities for further refinement or the application of post-processing calibration techniques.

6.3.4 Representative Segment Analysis

While quantitative metrics provide a global assessment of model performance, qualitative visualization of predicted versus actual traffic patterns over specific time segments offers crucial insights into the model's behavior, strengths, and failure modes. To this end, we present a series of 27 representative segments from the test set, divided into three groups: uniform sampling, random sampling, and stratified sampling by absolute residual. All figures are generated for the Random Forest model at the optimal 200 ms forecast horizon, using 'seed=42' for reproducibility. Each panel in these figures displays a 151-sample window (center sample ±75 samples), where the x-axis represents the relative sample index (0 at the center). The blue line indicates the ground truth average packet size, and the orange line represents the model's prediction. The relative sample index of ±75 corresponds approximately to ±2.4 to ±6.5 seconds, depending on the varying inter-arrival times across segments.

Uniformly Sampled Segments

Figure 6.9 showcases nine segments uniformly sampled across the test set. These segments provide a view of the model's typical performance across the dataset. In these

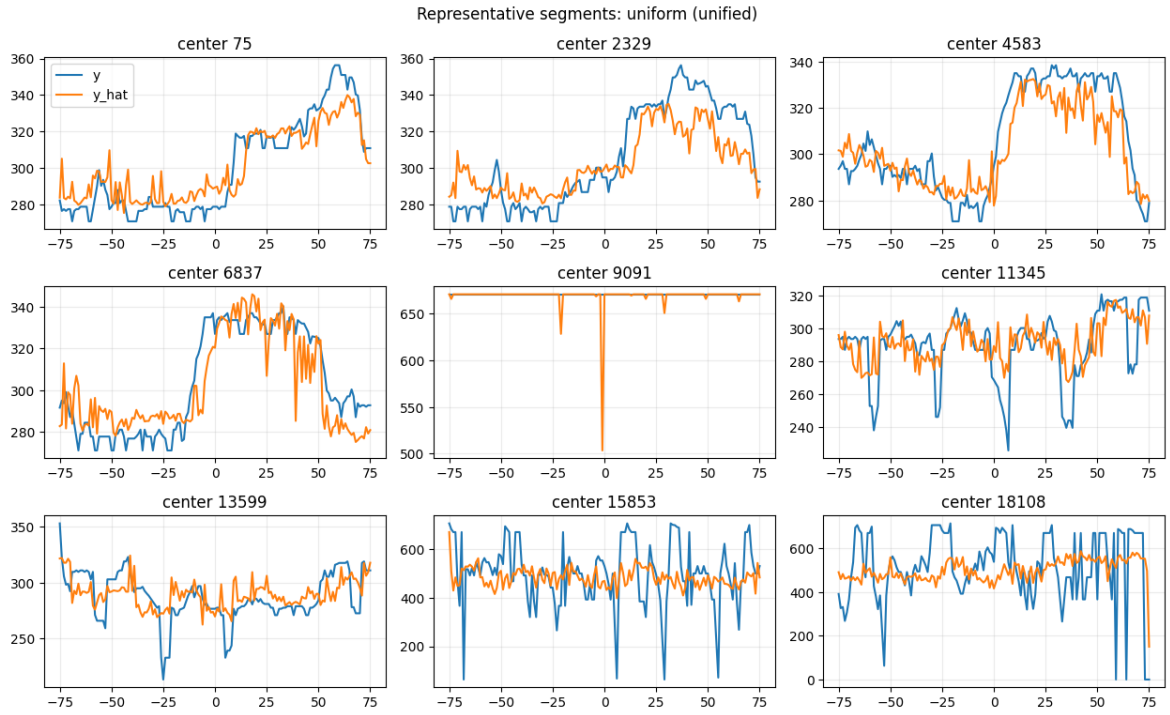


Figure 6.9: Representative segments (Δt = 200 ms) under uniform sampling. Each panel shows a 151-sample window centered at test sample k ("center k"; x-axis is the relative index $[-75, 75]$). Blue: ground truth; orange: prediction. The model tracks normal regimes well; errors primarily occur during transitions or short no-packet periods.

segments, the model generally tracks the baseline traffic trends well, demonstrating good overall fidelity during stable traffic flows. However, when short-term bursts or sudden drops (e.g., due to a "no-packet" future window resulting in a target of 0) occur, the predictions may exhibit some lag and dampening of amplitude, tending to

regress towards the mean rather than capturing the sharp instantaneous changes. This behavior is expected, as the model's features emphasize sub-second periodicities and aggregated statistics, making it less reactive to abrupt, non-periodic shifts.

Randomly Sampled Segments

Figure 6.10 presents nine randomly sampled segments. Similar to the uniform sampling, these segments confirm the representative nature of the model's performance, avoiding any potential bias from ordered selection. The observations from random sampling

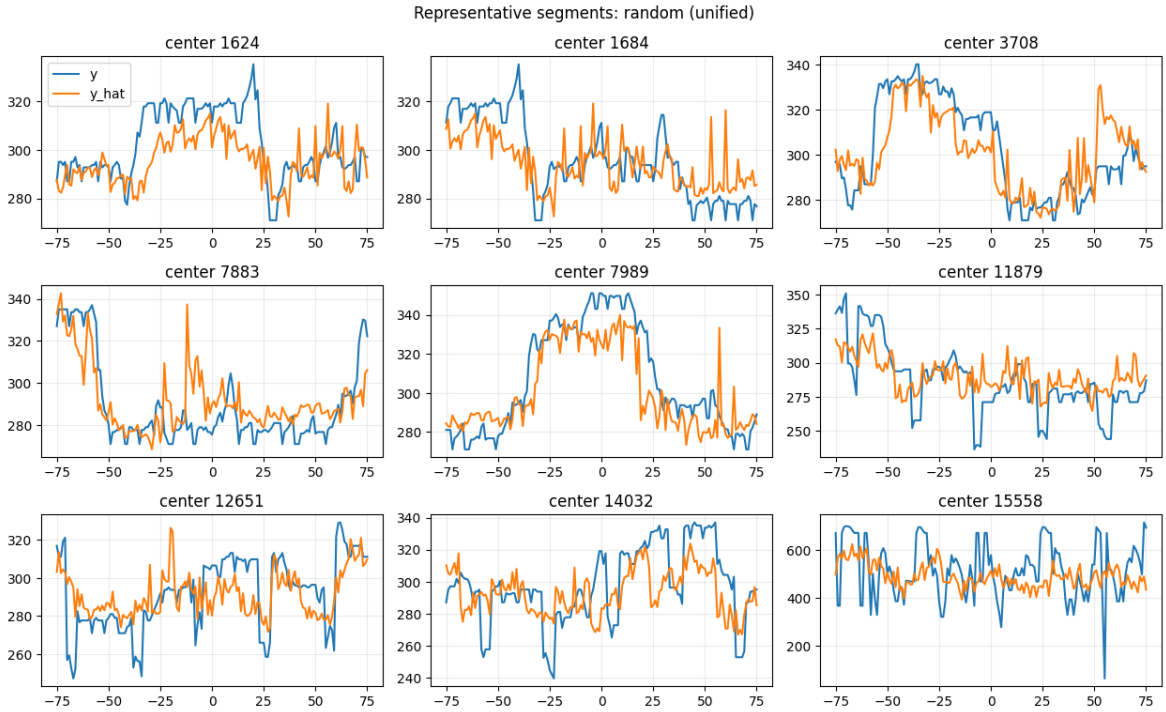


Figure 6.10: Representative segments ($\Delta t = 200$ ms) under random sampling (seed=42). Each panel shows a 151-sample window centered at test sample k ("center k"; x-axis is the relative index $[-75, 75]$). Blue: ground truth; orange: prediction. The performance aligns with uniformly sampled segments, showing good trend tracking but some lag and amplitude suppression during fluctuations.

largely mirror those from uniform sampling: the model capably follows the general flow of traffic but tends to under- or over-estimate the amplitudes of sharp fluctuations and exhibits minor phase lags during periods of rapid change or high-frequency oscillations. The phenomenon of "regression to the mean" is evident in segments with intense, high-amplitude spikes, where the model produces a smoother, less extreme prediction.

Stratified Segments by Absolute Residual

Figure 6.11 specifically highlights segments chosen based on their absolute residual values, providing a targeted view of both typical and challenging scenarios, particularly focusing on instances where the model exhibits larger errors. This stratification allows for direct observation of the model's failure modes. The stratified sampling clearly reveals the primary types of prediction errors:

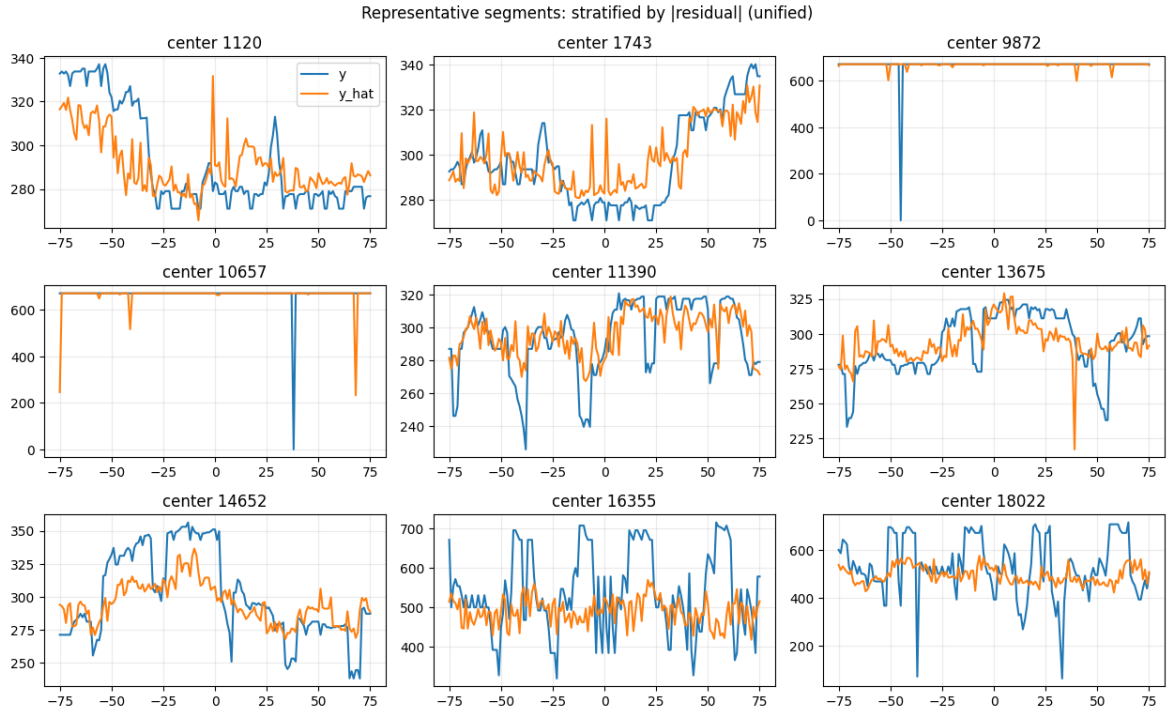


Figure 6.11: Representative segments ($\Delta t = 200$ ms) stratified by absolute residual. Each panel shows a 151-sample window centered at test sample k ("center k"; x-axis is the relative index $[-75, 75]$). Blue: ground truth; orange: prediction. This sampling reveals common failure modes, including overestimation during no-packet windows, underestimation of sudden bursts, and amplitude/phase discrepancies in high-frequency traffic.

- **No-Packet Windows and Short-Term Zeroing:** Several segments show the actual traffic dropping to near zero (due to no packets in the future window), while the model continues to predict a moderate positive value. This results in large positive residuals (overestimation) and is a direct consequence of the unconditional labeling protocol and the model's reliance on historical feature patterns.

- **Abrupt Regime Shifts/Bursts:** In segments featuring sudden, large-scale increases (or decreases) in traffic (e.g., rapid step changes in traffic load), the model tends to lag and systematically underestimate the peak values or overestimate the trough values. This is characteristic of the mean-squared-error trained regressors, which prioritize minimizing average error over accurately capturing extremes.

- **High-Frequency Oscillations/Phase Mismatches:** Some segments exhibit high-frequency, "sawtooth-like" fluctuations in actual traffic. While the model may capture the general presence of activity, its predictions often have a smaller amplitude and noticeable phase shifts compared to the ground truth. This suggests that while frequency-domain features capture the existence of periodicity, the model's ability to precisely align amplitude and phase in a 200 ms forecast window can be limited.

These observed failure patterns, particularly the overestimation during no-packet windows and the dampened, lagged response to abrupt changes, align well with the characteristics of the residual histogram (Figure 6.7), the residual vs. actual plot (Figure 6.8). Understanding these specific failure modes is crucial for designing robust post-processing strategies or future model enhancements.

6.4 Feature Importance Analysis

To understand which aspects of the VR network traffic are most influential in predicting future average packet sizes, we conducted a feature importance analysis using the Random Forest model. Random Forest's impurity-based feature importance (Mean Decrease in Impurity, MDI) quantifies the contribution of each feature to reducing the model's error. This analysis provides crucial insights into the underlying dynamics of VR traffic and guides future work in feature engineering and model optimization. The analysis focuses on the optimal Random Forest model (200 ms horizon).

The extracted features can be broadly categorized:

1. **Global features:** `ps_last` (payload size of the most recent packet), `dt_last` (inter-arrival time of the most recent packet), and `dir_bit` (direction of traffic: 0 for server-to-client, 1 for client-to-server). These provide instantaneous and contextual information.

2. **Scale-specific features (for each window $T \in \{0.5, 1.0, 2.0\}$ s):**

    - `sT_ampk`: Relative amplitude of the k-th dominant frequency peak.
    - `sT_fk_nyqnorm`: Nyquist-normalized frequency of the k-th dominant peak.
    - `sT_ratio12`, `sT_ratio23`: Ratios of dominant peak amplitudes, indicating spectral dominance.

- `sT_centroid`: Spectral centroid, indicating the "center of mass" of the frequency spectrum.
- `sT_flatness`: Spectral flatness, measuring the noisiness or tonality.
- `sT_valid`: A binary flag indicating if the window had enough packets for valid FFT computation.

6.4.1 Top-Ranked Features

Figure 6.12 displays the top 20 most important features. Features related to the 0.5-second time window, such as `s0.50_f5_nyqnorm`, `s0.50_amp1`, and `s0.50_f2_nyqnorm`, consistently appear at the top. The `dir_bit` feature is also highly ranked, underscoring the significant directional asymmetry in VR traffic. Instantaneous time-domain features like `dt_last` and `ps_last` also contribute substantially. The presence of multiple frequency-related features (e.g., `s0.50_f5_nyqnorm`, `s0.50_f2_nyqnorm`) at high ranks suggests that the overall periodic structure, rather than a single dominant frequency, is crucial for prediction.

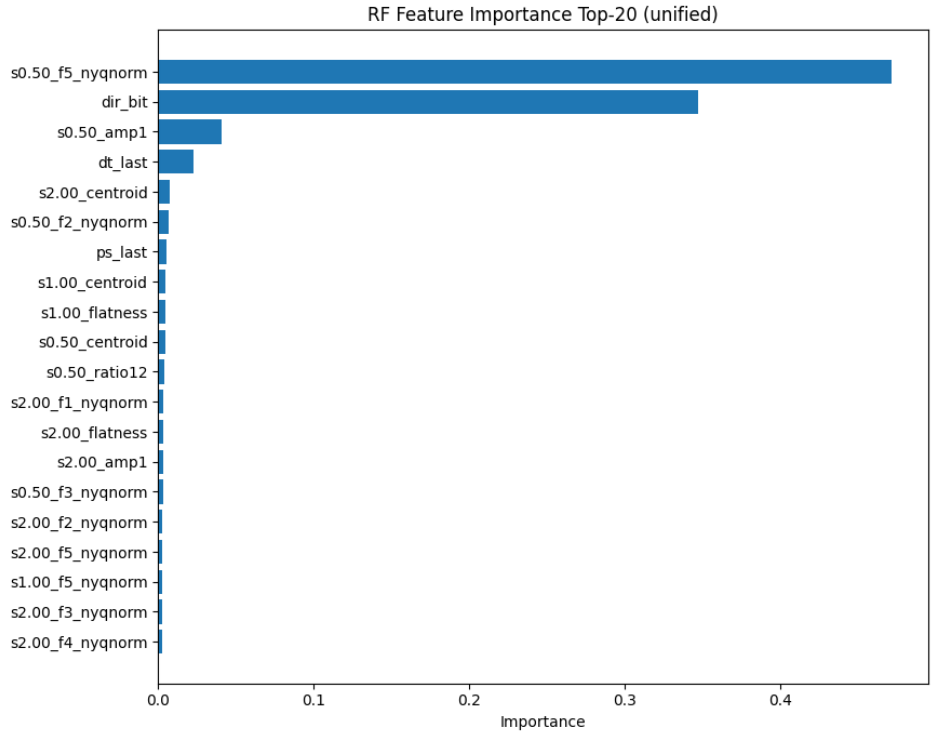


Figure 6.12: Random Forest feature importance: Top 20 features ranked by Mean Decrease in Impurity. Features from the 0.5-second window and the traffic direction bit are prominently ranked, indicating their strong predictive power.

6.4.2 Importance by Time Scale and Feature Family

To provide a more aggregated view, we analyzed feature importance by their associated time scale (0.5s, 1.0s, 2.0s windows, and global features) and by their family type (e.g., frequency, amplitude, direction).

Figure 6.13 shows the summed importance across different time scales. The 0.5-second time window features (`s0.50_*`) contribute the most significantly to the model's predictive power. Global features (which include `ps_last`, `dt_last`, `dir_bit`) rank second. Features from the longer 1.0-second and 2.0-second windows contribute less. This distribution highlights that for a 200 ms forecast horizon, short-term (sub-second) patterns are most relevant, as they are temporally best aligned to capture the immediate future dynamics. Longer windows may smooth out too much of the critical short-term variability.

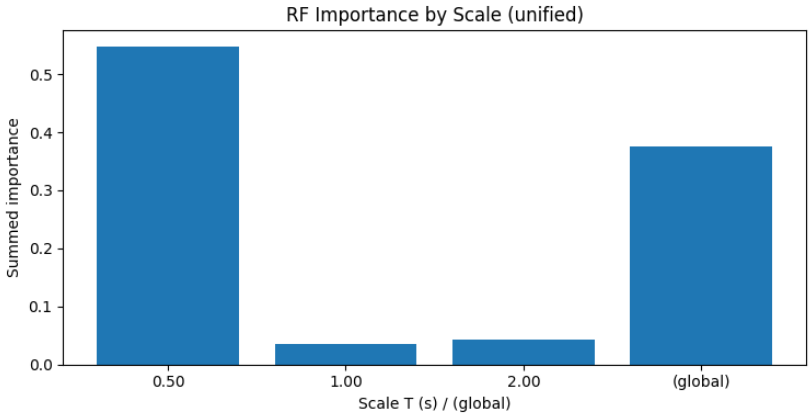


Figure 6.13: Random Forest importance by feature scale. The 0.5-second time window features contribute the most, followed by global features, indicating the prominence of short-term and instantaneous traffic characteristics for prediction at a 200 ms horizon.

Figure 6.14 aggregates importance by feature family. This analysis reveals that frequency-domain features ('freq') are by far the most important, followed by the directional context ('dir') and then amplitude-based features ('amp'). Other spectral descriptors ('ratio', 'centroid', 'flatness') and base time-domain features contribute less but are still valuable.

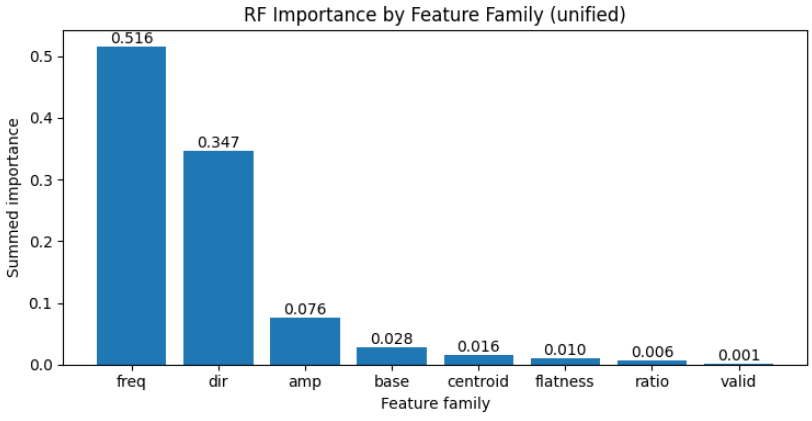


Figure 6.14: Random Forest importance by feature family. Frequency-domain features are the most dominant, followed by the direction bit and amplitude features. This underscores the critical role of periodic patterns and traffic flow direction in VR traffic forecasting.

6.4.3 Key Insights from Feature Importance

The feature importance analysis yields several critical insights:

1. **Dominance of Frequency Position over Amplitude:** The high ranking of `sT_f*_nyqnorm` (frequency location) over `sT_ampk` (amplitude magnitude) implies that the periodicity of packet transmission is more indicative of future traffic load than the mere strength of a particular frequency component. This suggests that the existence of stable update cycles (e.g., game engine ticks, controller input rates) within VR traffic provides strong predictive signals.

2. **Critical Role of Directional Context:** The `dir_bit` feature's high importance confirms that uplink and downlink traffic flows possess fundamentally different patterns and underlying application logic (e.g., server-to-client for world state dissemination, client-to-server for pose updates and RPCs). Explicitly providing this context allows the model to differentiate and make more accurate predictions.

3. **Short-Term Dynamics are Paramount:** The superior importance of features from the 0.5-second window, coupled with the relevance of `ps_last` and `dt_last`, emphasizes that immediate, sub-second traffic characteristics are most indicative of average packet size in the upcoming 200 ms. Longer-term (1.0s, 2.0s) aggregations, while potentially useful for very long-term forecasting, tend to smooth out the critical short-term dynamics needed for short-horizon predictions.

4. **Actionable Insights for Feature Engineering:** For future work, this analysis suggests prioritizing the extraction of fine-grained frequency position features and ensuring the directional context is well-represented. For efficiency, one could consider reducing the number of amplitude-based features or features from longer time windows, especially if computational cost is a concern. To further improve prediction for shorter horizons (e.g., 20–80 ms), exploring more reactive features such as short-window slopes, change-point indicators, or burst detection metrics might be beneficial.

These findings underscore the effectiveness of our hybrid time-frequency feature engineering approach in capturing the salient characteristics of VR network traffic. They provide a robust foundation for building predictive models and offer guidance for further refinement, potentially leading to even more accurate and resource-efficient forecasting solutions.

6.5 Implications for VR Network Management and Proactive Resource Allocation

The experimental results from both the classification (Chapter 5) and forecasting tasks (this chapter) provide critical insights for designing next-generation VR network management systems. By leveraging the distinct characteristics of VR user states and the predictability of aggregated traffic load, network resources can be managed more effectively, leading to significantly enhanced Quality of Experience (QoE).

6.5.1 Optimal Trade-off for Proactive Resource Allocation and Prediction Stability

Our analysis reveals a clear trade-off between the temporal granularity desired for real-time decisions and the prediction stability achievable for traffic forecasts, especially under the unified-sample, unconditional labeling protocol.

- **Challenges at Ultra-Short Horizons (20 ms, 40 ms, 80 ms):** Predictions at 20 ms and 40 ms horizons proved to be highly challenging, with R^2 values typically below 0.4. Even at 80 ms, R^2 remained relatively low, around 0.3. This unreliability stems primarily from the inherent volatility of packet-level VR traffic and the high frequency of future windows containing very few or no packets, leading to the target being set to zero. While models can partially learn to identify these "no-packet" instances (as observed for Random Forest at 20 ms), attempting to manage network resources at such fine-grained levels based on these volatile forecasts would likely lead to frequent over-provisioning (wasting resources) or under-provisioning (causing congestion and performance degradation), ultimately harming QoE. Therefore, for reliably predicting general traffic load, these ultra-short horizons are generally unsuitable without further refinements to the prediction target definition or the model itself.

- **Emerging Predictability at Transitional Horizons (150 ms):** A significant improvement in predictability is observed at the 150 ms horizon, where R^2 values for deep learning models reach approximately 0.61. This indicates that as the averaging window lengthens, the target variable becomes more representative of continuous traffic flow, reducing the noise introduced by individual packet variability and the prevalence of zero-targets. This horizon, therefore, represents a point where predictions begin to offer practical utility, striking a better balance between responsiveness and reliability.

- **Robust Predictions at Longer Horizons (200 ms):** The 200 ms horizon consistently demonstrates the most robust predictions, with all models achieving impressive R^2 values exceeding 0.85 (up to 0.8586 for Random Forest). At this duration, the averaging window effectively smooths out high-frequency variations and the impact of individual packet arrivals, allowing models to discern and predict stable underlying traffic trends with high fidelity. This horizon (particularly the 200 ms interval) represents a highly effective sweet spot for proactive resource allocation in VR systems. These durations are long enough to provide stable and actionable forecasts, yet short enough to allow network controllers to react swiftly to changes in user behavior and application demands, thereby enabling truly proactive management.

6.5.2 Multi-Timescale Control: Integrating 200 ms Forecasting with 1 ms Scheduling

A 200 ms forecast is not intended to replace the fast, reactive 1 ms MAC scheduler. Instead, it drives a slower, outer control loop that runs approximately every 150–200 ms per user equipment (UE) or per flow. This outer loop computes a stable byte budget or target rate, which the underlying 1 ms scheduler then uses as a guiding baseline target (not a hard cap) for its rapid, channel-aware, and queue-aware allocations.

The process operates as follows:

1. **Slow Loop (150-200 ms):** Every $H \in \{150, 200\}$ ms, the slow control loop for a given UE u computes a total byte budget, $\widehat{B}_u$, for the upcoming H ms window. This budget is derived from the forecast of the average packet size, $\widehat{\bar{s}}_u$, (bytes/pkt) generated by our models, a short-term estimate of the packet arrival rate, $\widehat{\lambda}_u$, (e.g., an Exponentially Weighted Moving Average from MAC/RLC counters, in pkts/ms), and a safety headroom factor ($\gamma > 1$) chosen from the empirical error distribution.

$$\widehat{B}_u = \widehat{\bar{s}}_u \times \widehat{\lambda}_u \times H[\text{bytes over } H] \tag{6.4}$$

   This total budget is then converted into a baseline target rate, r_u, in bytes per millisecond, which the fast scheduler can attempt to drain:

$$r_u = \gamma \cdot \frac{\widehat{B}_u}{H} \quad \text{(bytes/ms)} \tag{6.5}$$

2. **Fast Loop (1 ms):** The 1 ms MAC scheduler then utilizes this calculated rate r_u as a *baseline target/weight*—it can meet or exceed this baseline depending on instantaneous CQI/MCS and queue backlogs. For each 1 ms Transmission Time Interval (TTI), it attempts to allocate resources up to this rate, dynamically adjusting to instantaneous Channel Quality Indicator (CQI), Modulation and Coding Scheme (MCS), and actual queue backlogs. The 1 ms scheduler remains responsible for fine-grained, real-time radio resource management; the 200 ms loop only sets the rate envelope it is allowed to drain, thus preventing sudden, unforecasted demand spikes from immediately leading to excessive queuing. The 200 ms loop low-passes demand and provides foresight; the 1 ms loop remains responsible for fine-grained radio/queue dynamics.

A multi-timescale control approach is essential due to the inherent characteristics of network traffic and scheduling mechanisms:

- **Predictability and Noise Reduction**

  Instantaneous (1 ms) network traffic is highly volatile. While the 1 ms scheduler can react quickly, it lacks foresight. Attempting to predict future traffic at a 1 ms granularity is prone to extreme noise, as our R^2 values for 20-80 ms horizons demonstrate (often below 0.4). Averaging over 150-200 ms significantly dampens this high-frequency noise. Quantitatively, Let $S = \sum s_k$ be the total bytes arriving within horizon H, for a compound Poisson process, the standard deviation of the estimated rate over a horizon H scales inversely with $\sqrt{H}$.

$$\text{Var}\left[\frac{S}{H}\right] = \frac{\lambda\, \mathbb{E}[s^2]}{H}, \qquad \text{so std} \propto H^{-1/2} \tag{6.6}$$

  Moving from 1 ms to 200 ms effectively reduces the target's standard deviation of the per-window rate by a factor of $\sqrt{200} \approx 14$, providing a much more stable and reliable input for proactive resource allocation. This stability is directly reflected in our high R^2 values (> 0.85 at 200 ms).

- **Mitigating Control Oscillations and Delays**

  Directly feeding highly noisy 1 ms forecasts into a fast scheduler, especially with feedback delays (e.g., CQI reporting, HARQ retransmissions, buffering), can lead to control oscillations (over-provisioning followed by under-provisioning). The 200 ms loop acts as a low-pass filter for demand, allowing the 1 ms scheduler to react to radio- and queue-level micro-dynamics without chasing traffic noise.

- **Resource and Signaling Overhead Reduction**

  Reconfiguring granular control parameters (e.g., proportional-fair weights, Guaranteed Bit Rate (GBR) token rates, or Semi-Persistent Scheduling (SPS) grants) at a 1 ms rate per UE would impose immense computational and signaling overhead on the network's control plane. Updating per-UE targets 5 Hz (200 ms) vs. 1000 Hz (1 ms) yields a $\sim 200\times$ reduction in control churn while retaining sub-second responsiveness, making it practical for real-world deployment while still tracking application-level changes within sub-second timescales.

- **Error-Aware Headroom γ and Under-Provisioning Guarantees**

  Let B be the realized bytes in a window and $\widehat{B}$ the forecast. Define the relative error $E = (B - \widehat{B})/\widehat{B}$. To cap under-provision likelihood at level q,

  $$\gamma = 1 + Q_{1-q}\big(\max\{E, 0\}\big), \tag{6.7}$$

  using the empirical CDF on the test set. As a conservative alternative, if E is approximated as zero-mean normal with $\sigma_E \approx$CVRMSE, take $\gamma \approx 1 + z_{1-q}\sigma_E$ (e.g., $z_{0.95} = 1.645$). In our 200 ms results, RMSE for $\widehat{s}$ is ≈ 60 B against typical averages 300–500 B (CVRMSE $\approx$ 15–17%), which leads to γ in the 1.3–1.4 range for 90–95% under-provision protection when also accounting for arrival-rate uncertainty.

  A backlog upper bound when γ is too small follows from

  $$\Delta Q = B - \gamma\widehat{B} = \widehat{B}\,(E - (\gamma - 1)). \tag{6.8}$$

  For example, with $\widehat{B} = 8$ KB over $H = 200$ ms and $\gamma = 1.30$, the baseline is $r = 1.30 \times 8192/200 \approx 53.2$ B/ms ($\approx$426 kbps). If $E = 0.30$ (30% under-forecast), then $\Delta Q \approx (0.30 - 0.30) \times 8192 = 0$; by contrast, with $\gamma = 1.13$ one would see $\Delta Q \approx (0.30 - 0.13) \times 8192 \approx 1.39$ KB, i.e., $\sim$ 26–30 ms extra draining at the baseline rate (the fast loop can exceed the baseline to clear faster).

6.5.3 Alignment with Modern Network Control Loops and SPS for VR

The robust prediction horizons (150 ms to 200 ms) align remarkably well with typical control loop latencies and scheduling periods in modern wireless communication networks, particularly within 5G and future 6G architectures.

Semi-Persistent Scheduling (SPS) and Configured Grants for AR/VR Applications

SPS is a key mechanism in 5G New Radio (NR) for efficiently scheduling periodic traffic flows by allocating resources semi-statically, thereby reducing signaling overhead

and latency variance. Our feature importance analysis reveals that frequency-domain features contribute significantly (e.g., $>50\%$ by family) to predictive power, and sub-second (0.5 s) scale features dominate. This empirical evidence of quasi-periodicity, particularly in VR control and telemetry flows (e.g., IMU/pose updates, interaction events), strongly suggests that SPS and configured grants are well-suited for such traffic.

The 200 ms forecast provides a direct input to intelligently configure SPS grants. A baseline semi-persistent grant (g) can be sized from the predicted byte budget ($\widehat{B}_u$) with the calculated headroom (γ) over an SPS period (P, e.g., 10 ms):

$$g_u = \gamma \cdot \widehat{B}_u \cdot \frac{P}{H} \qquad \text{bytes per SPS period P} \tag{6.9}$$

For instance, with $\widehat{B}_u = 8\,\text{KB}$, $H = 200\,\text{ms}$, $P = 10\,\text{ms}$, and $\gamma = 1.30$, we obtain $g_u = 1.30 \times 8192 \times (10/200) \approx 532\,\text{B}$ per occasion (sum over $H/P = 20$ occasions equals $\gamma\widehat{B}_u$). This approach ensures sufficient resources are proactively allocated, minimizing packet loss and latency for these critical, quasi-periodic flows. Dynamic scheduling can then handle any residual bursts or unpredicted spikes as top-ups.

To avoid excessive control plane churn, the SPS allocation can be updated only when the forecasted budget ($\widehat{B}_u$) crosses predefined hysteresis bands for a certain number of consecutive windows. This balances responsiveness with control plane efficiency. For highly variable downlink media traffic (e.g., foveated video streams, rendered frames), a pure SPS approach might be less efficient. Here, a hybrid strategy where SPS provides a baseline for essential, often periodic components, and dynamic scheduling handles the bursty, high-bandwidth demands, would be more appropriate.

Dynamic Resource Orchestration

Beyond SPS, network slices in 5G and future 6G can be dynamically scaled based on anticipated demand. Accurate short-term forecasts enable proactive scaling of virtualized network functions (VNFs) or adjustment of slice bandwidth, preventing bottlenecks before they occur. The enhanced predictability at 200 ms is particularly well-suited for such orchestration, providing a stable look-ahead for resource managers.

Congestion Control and Buffer Management

By forecasting the incoming load, network devices (e.g., base stations, edge servers) can adjust congestion control algorithms or manage buffer depths more intelligently, preventing queues from overflowing and minimizing packet drops. The relatively high R^2 values for the longer horizons signify that these predictions are reliable enough to guide such critical real-time network decisions.

6.5.4 Benefits for VR QoE

Ultimately, the practical application of these predictive capabilities directly translates into tangible improvements in VR Quality of Experience:

- **Reduced Latency and Jitter:** Proactive resource allocation based on predicted load minimizes queuing delays, ensuring that VR packets arrive consistently and within stringent latency budgets. This is particularly critical for maintaining immersion and preventing motion sickness.

- **Smoother Visuals and Interactions:** By ensuring consistent throughput tailored to anticipated demand, the application can maintain desired frame rates and render updates without hitches, crucial for a seamless user experience.
- **Enhanced Stability:** Reliable network performance, supported by accurate forecasting, removes a significant source of variability, leading to a more stable and predictable VR experience, crucial for both single-user and multi-user interactive applications.

By intelligently combining VR user state classification (Chapter 5) with robust short-term traffic load forecasting, our framework provides a strong foundation for truly adaptive and QoE-aware network management in the rapidly evolving landscape of immersive technologies.

Chapter 7

End-to-End Prediction-Aided Semi-Persistent Scheduling (SPS) for VR

7.1 Introduction

The stringent Quality of Experience (QoE) demands of Virtual Reality (VR) applications, characterized by high data rates, low latency, and high predictability, pose significant challenges for wireless network resource management. As explored in preceding chapters, VR traffic exhibits highly dynamic and bursty patterns, necessitating sophisticated prediction models to enable proactive resource allocation. Chapter 6 demonstrated the development of advanced machine learning models, specifically RF Regressors, capable of accurately forecasting VR network traffic load over a 200-millisecond (ms) horizon. However, directly applying such predictions to a standard 1ms MAC layer scheduler presents a fundamental temporal mismatch: a 200ms prediction, while accurate at its own scale, appears as a constant value to a scheduler making decisions every single millisecond, rendering it ineffective for real-time dynamic allocation.

This chapter addresses this critical temporal disconnect by proposing and evaluating an end-to-end predictive resource scheduling framework designed to seamlessly integrate 200ms traffic forecasts into a 1ms MAC scheduler for enhanced VR QoE. Our primary objective is to demonstrate how window-level traffic predictions can be effectively translated into actionable per-millisecond scheduling decisions without sacrificing responsiveness or increasing control overhead. We achieve this by leveraging a hybrid scheduling approach that combines Semi-Persistent Scheduling (SPS) for proactive, window-based resource reservation, guided by our prediction models, with Proportional Fair (PF) scheduling for reactive, per-millisecond allocation of residual capacity and management of instantaneous traffic fluctuations. Prediction-aided SPS reserves a fraction of capacity for users expected to be active within the next window, while the residual is allocated by PF at 1 ms granularity to track fast dynamics. Concretely, every T_{sps} milliseconds the scheduler forms a small SPS set using a weight

$$w_k(t) = \frac{Q_k(t) + \gamma\, \widehat{Y}_k(t)}{\max(\epsilon, \widetilde{R}_k(t))}, \tag{7.1}$$

where Q_k is the backlog, $\widehat{Y}_k$ is a 200 ms predicted demand proxy (sum or avg–based),

$\widetilde{R}_k$ is a slow throughput-EMA, and γ scales the look-ahead. SPS holders keep a share for T_{hold} ms; any unused SPS capacity rolls back to PF, ensuring work-conserving operation.

The main contributions of this chapter are multifaceted. Firstly, we develop a multi-user VR traffic simulator, which synthesizes realistic, heterogeneous VR traffic patterns from real-world traces, incorporating features such as dynamic user load, time shifts, and burst injection to create a challenging and representative network environment. Secondly, we integrate the 200ms window-level traffic load predictors RF models from Chapter 6 directly into the SPS selector and keep PF for residual per-ms allocation. This integration demonstrates the practical applicability of our predictive models in a live-like network context. Thirdly, we provide comprehensive evidence that this prediction-aided SPS strategy significantly improves delay fairness for VR users without compromising overall throughput, and rigorously show its alignment with an ideal, oracle-based SPS selection. This work explicitly extends the offline prediction benchmarks presented in Protocol A by closing the loop: prediction informs scheduling, and scheduling directly impacts measurable QoE metrics.

A key finding highlighted in this chapter is that, even with a high degree of user multiplexing (e.g., 60 users) and tight network capacity, prediction-aided SPS consistently enhances delay fairness by approximately 3.0–3.5% compared to a purely reactive PF baseline. Furthermore, our approach closely approximates the performance of an oracle SPS, achieving a gap of about 0.10–0.42 percentage points (pp), depending on the predictive mode, in the BASE scenario, with a high Jaccard index of approximately 0.91 in SPS user selection overlap with the oracle. These results underscore the practical utility of our prediction models in optimizing resource allocation for latency-sensitive VR services.

7.2 Multi-User VR Traffic Workload Synthesis

Accurate and comprehensive multi-user VR traffic datasets, particularly those captured under diverse network conditions and user behaviors, are scarce. To overcome this limitation and create a controlled yet realistic experimental environment, we developed a sophisticated workload synthesis pipeline. This pipeline generates heterogeneous multi-user VR traffic streams from a single, long VR trace previously analyzed in Chapter 4.

The process begins by loading raw VR traffic data, which includes packet timestamps and lengths, from `stationary` and `moving` JSON files for both server-to-client (`s2c`) and client-to-server (`c2s`) directions. The longest available traffic flow is selected as the primary source for generating synthetic user traffic. All packet timestamps are normalized to a common starting point ($t_0 = 0$), and traffic is then aggregated into a 1ms time-domain grid, resulting in two fundamental time series for each flow: $A[t]$ representing the total bytes transmitted in millisecond t, and $C[t]$ representing the total packet count in millisecond t. If a `LIMIT_MS` is specified, the grid length T is truncated to this value.

To simulate K distinct users, the original aggregated traffic stream $(A[t], C[t])$ of length T_{total} ms is "chunkified." We divide T_{total} into non-overlapping chunks, each of length `CHUNK_LEN_S` seconds (or `CHUNK_LEN_MS` = `CHUNK_LEN_S` $\times 1000$ ms). For each of the `K_USERS`, a chunk is selected cyclically from the available segments. To introduce user heterogeneity and dynamic starting points, each selected chunk is then subjected

to a random temporal shift of up to TIME_SHIFT_MS_MAX milliseconds. This process creates K individual user traffic traces, denoted as $A_k[t]$ and $C_k[t]$ for user k at time t. To further enhance heterogeneity, a slight random scaling factor (uniformly distributed between 0.9 and 1.2) is applied to the traffic of each user. The final output is a 2D array for bytes and packet counts, $A_{\text{users}}[K, T_{\text{sim}}]$ and $C_{\text{users}}[K, T_{\text{sim}}]$, where T_{sim} is the minimum length across all generated user chunks, clamped by MAX_SECONDS.

To deliberately introduce contention and emphasize the value of predictive scheduling, we implement an optional burst injection mechanism. For each user, bursts of traffic are introduced at a given probability (burst_prob), with a specified amplitude (burst_amp), duration (burst_dur_ms), and occurring approximately every burst_every_ms milliseconds. This artificially amplifies the natural burstiness of VR traffic, simulating periods of intense activity that can stress the network and expose the limitations of reactive scheduling.

The wireless channel environment is modeled with an optional Rayleigh-like fading component. When USE_FADING is enabled, a Rayleigh distribution (scaled to have a mean channel gain of approximately 1) is applied to introduce realistic temporal variations in channel quality, influencing resource allocation decisions. The network capacity is set as a fixed CAP_BYTES_PER_MS, derived as a CAP_FRACTION of the average total traffic demand across all users. This allows us to systematically investigate the scheduler's performance under various load regimes. The overall process is initialized with a fixed random seed 42 to ensure reproducibility of results. The key parameters governing this workload synthesis are summarized in Chapter 6.

Table 7.1: Workload Synthesis and Network Configuration Parameters.

Parameter	**Description**	**Value (Default)**
K_USERS	Number of synthetic VR users	60
CHUNK_LEN_S	Length of each traffic chunk (seconds)	60
CHUNK_STRIDE_S	Stride for chunk selection (seconds)	15
TIME_SHIFT_MS_MAX	Max random time shift per user (ms)	3000
MAX_SECONDS	Simulation duration limit (seconds)	45
BURST_AMPL	Burst amplitude multiplier	1.0 (or 4.0)
BURST_DUR_MS	Burst duration (ms)	40 (or 120)
BURST_EVERY_MS	Bursts occurring every X ms	1000 (or 600)
CAP_FRACTION	Network capacity as fraction of mean demand	0.33
USE_FADING	Enable Rayleigh-like fading	True

7.3 Predictive Modeling for Window-Level Traffic

The predictive core of our framework relies on the RF Regressor models developed in Chapter 6. These models are designed to forecast VR traffic load over a 200ms future window. For effective integration into the scheduling framework, features and labels are constructed at a 1ms granularity across all users, then downsampled for training efficiency.

The feature set for each user k at time t is designed to capture both instantaneous and recent historical traffic dynamics. It includes:

- **Instantaneous Traffic**: The current bytes $A_k[t]$ and packet count $C_k[t]$ in millisecond t.
- **Multi-Window Past Sums and Rates**: Rolling sums of bytes $S_A^W[t]$ and packet counts $S_C^W[t]$ over various past windows $W \in \{20, 50, 100, 200, 500\}$ milliseconds. These sums are also normalized by their respective window lengths to provide "per-ms rates" ($\frac{S_A^W[t]}{W}$ and $\frac{S_C^W[t]}{W}$). The rolling sum $S_X^W[t]$ is defined as

$$S_X^W[t] = \sum_{i=t-W+1}^{t} X[i]. \tag{7.2}$$

  The selection of these windows is based on insights from prior work suggesting their effectiveness in capturing VR traffic's multi-scale temporal dependencies.
- **Local Exponentially Weighted Moving Averages (EWMA)**: To capture recent trends with decaying memory, we compute an EWMA over the short-window sums ($W_{\text{loc}} = 50\,\text{ms}$) with decay constants $\tau \in \{50, 200, 1000\}\,\text{ms}$. Let $\beta = e^{-1/\tau}$ in ms-granularity. We initialize $Z_0^{(A,\tau)} = S_A^{W_{\text{loc}}}[0]$ and $Z_0^{(C,\tau)} = S_C^{W_{\text{loc}}}[0]$. For bytes and packets respectively:

$$Z_t^{(A,\tau)} = (1-\beta)\, S_A^{W_{\text{loc}}}[t] + \beta\, Z_{t-1}^{(A,\tau)}, \tag{7.3}$$

$$Z_t^{(C,\tau)} = (1-\beta)\, S_C^{W_{\text{loc}}}[t] + \beta\, Z_{t-1}^{(C,\tau)}. \tag{7.4}$$

The full set of features for all users and time steps forms a tensor $X[N, T, D]$, where N is the number of users, T is the simulation duration in milliseconds, and D is the number of features. The target variables for prediction are defined over the future 200ms window, consistent with our objective to guide SPS decisions. Specifically, for each user k at time t, we predict:

1. **Future Sum of Bytes (y^{sum})**: The total bytes expected in the interval $(t, t + 200]$ ms, denoted as $y_k^{\text{sum}}(t) = \sum_{i=t+1}^{t+200} A_k[i]$.
2. **Future Average Packet Size (y^{avg})**: The average packet size in the interval $(t, t + 200]$ ms, derived from the sum of bytes divided by the sum of packets in that window. From the predicted average packet size, we also derive an "average-as-bytes" proxy, which is the predicted average packet size multiplied by the past count of packets in the same 200ms window (i.e., $\hat{y}_k^{\text{avg}}(t) \times S_C^{200}[t]$). We intentionally use the past 200 ms packet count $S_C^{200}[t]$ here to avoid training a second model for future packet counts; this leverages short-term packet-count autocorrelation observed in our traces and keeps inference lightweight. This proxy aims to provide an estimate of future bytes that is sensitive to typical packet sizes while considering recent activity levels.

The RF Regressor models are trained using a time-ordered split (70% training, 15% validation, 15% test) applied to the flattened feature space. The specific hyperparameters for the RF models, such as `n_estimators`, `max_depth`, `min_samples_leaf`, and `max_features`, are configured as detailed in Chapter 6. Post-training, the models are saved using `joblib` for efficient loading. During simulation, vectorized full-grid inference is employed to generate predictions for all users at every millisecond, ensuring that the scheduler has up-to-date predictive signals.

7.4 Hybrid SPS/Proportional Fair Scheduling Framework

The core innovation in this chapter lies in the design of a hybrid scheduling framework that effectively bridges the temporal gap between 200ms traffic predictions and 1ms scheduling decisions. This framework integrates Semi-Persistent Scheduling (SPS) with Proportional Fair (PF) scheduling, leveraging the strengths of both to optimize resource allocation for VR traffic.

The total network capacity, `CAP_BYTES_PER_MS`, is conceptually divided into two portions: a fraction `SPS_FRAC` allocated for SPS users, and the remaining $1 - $ `SPS_FRAC` for PF users. However, in our implementation, any unused SPS capacity is dynamically rolled over to the PF scheduler, maximizing resource utilization.

7.4.1 Semi-Persistent Scheduling (SPS) Stage

The SPS stage is responsible for making proactive, window-level resource reservation decisions. This decision-making process occurs periodically, every $P =$ `SPS_PERIOD_MS` milliseconds. At each SPS reselection interval, the scheduler identifies a set of M users, defined by `K_SPS_FRAC` $\times$ `K_USERS`, who are most likely to require significant resources in the near future. These selected users are granted a "semi-persistent" allocation for a duration of $H =$ `SPS_HOLD_MS` milliseconds.

The selection of SPS users is based on a dynamic priority function $P_k(t)$, which for user k at time t is calculated as:

$$P_k(t) \;=\; \frac{T_k(t)^{\alpha}}{\tilde{r}_k(t)^{\beta}} \left(\frac{Q_k(t) + \gamma\, \mathrm{signal}_k(t)}{\max(\varepsilon, \tilde{B}_k(t))} \right)^{\delta} \tag{7.5}$$

where:

- $T_k(t)$ represents the instantaneous channel quality (modeled by fading if enabled, otherwise 1.0) for user k at time t. α is a weighting factor.
- $\tilde{r}_k(t)$ is the Exponentially Weighted Moving Average (EWMA) of user k's served throughput, reflecting their historical rate. β is a weighting factor.
- $Q_k(t)$ is the current backlog (queue size in bytes) for user k.
- $\mathrm{signal}_k(t)$ is the predictive input, which differentiates the various scheduling modes:
  - `zero`: $\mathrm{signal}_k(t) = 0$ (for `SPS_ZERO` mode).
  - `past-sum`: $\mathrm{signal}_k(t) = S_A^{\texttt{TARGET\_WIN\_MS}}[t]$, the sum of bytes in the past 200 ms bytes (for `SPS_PRED_PAST` mode).
  - `pred-sum`: $\mathrm{signal}_k(t) = \hat{y}_k^{\mathrm{sum}}(t)$, the `RF` prediction of future 200 ms bytes (for `SPS_PRED_RF_SUM` mode).
  - `pred-avgB`: $\mathrm{signal}_k(t) = \hat{y}_k^{\mathrm{avgB}}(t)$, the `RF` prediction of future average-as-bytes proxy (for `SPS_PRED_RF_AVG` mode).
  - `oracle`: $\mathrm{signal}_k(t) = y_k^{\mathrm{true\_future}}(t)$, the actual future bytes (for `SPS_TRUE` mode).

- $\tilde{B}_k(t)$ is the EWMA of user k's backlog, designed to respond faster to bursts.
- γ is a crucial parameter that controls the emphasis on the predictive signal and backlog relative to the historical served rate. Higher γ values indicate greater trust in the prediction and current demand.
- ϵ is a small constant (e.g., 10^{-6}) to prevent division by zero.
- $\delta \geq 1$ controls nonlinearity of the priority mapping.

After calculating priorities for all active users, the top M users are selected for the SPS set. A critical refinement in our implementation, addressing observations from initial experiments where high Jaccard overlap didn't always translate to proportional fairness gains, is the dynamic allocation of SPS shares. Instead of an equal split, each selected SPS user k is assigned an SPS share proportional to their predicted window load (`signal_k(t)`) or backlog (`Q_k(t)`) relative to the sum of all selected SPS users' signals. This ensures that users with higher predicted future demand receive a larger proportion of the reserved SPS capacity. This share is then renormalized every millisecond to ensure the total SPS capacity `SPS_FRAC` $\times$ `CAP_BYTES_PER_MS` is fully utilized.

Once a user is in the SPS set, their `sps_left` timer is set to H and decremented by 1 at each millisecond. If a user's backlog $Q_k(t)$ is cleared (i.e., $Q_k(t) \leq \epsilon$) before their `sps_left` timer expires, they are immediately released from the SPS set, and their allocated share is redistributed among the remaining SPS users or rolled into the PF capacity pool.

7.4.2 Proportional Fair (PF) Stage

The PF stage acts as a reactive scheduler, handling the capacity not utilized by SPS users. This includes the $1 - $ `SPS_FRAC` portion of the total capacity, as well as any unused SPS capacity (e.g., if SPS users do not have enough backlog to consume their full share). The PF scheduler operates on a 1ms basis. The queue-aware PF stage allocates the residual capacity each millisecond: the nominal PF share ($1 - $ `SPS_FRAC`) plus any SPS leftovers. Its weight is

$$w_k^{\mathrm{PF}}(t) = \begin{cases} \dfrac{Q_k(t)}{\max(\varepsilon, \tilde{r}_k(t))}, & \text{if } \texttt{PF\_BASE} \text{ or } \texttt{PRED\_IN\_PF} = \texttt{False}, \\ \dfrac{Q_k(t) + \gamma\, \mathrm{signal}_k(t)}{\max(\varepsilon, \tilde{r}_k(t))}, & \text{if } \texttt{PRED\_IN\_PF} = \texttt{True}, \end{cases} \tag{7.6}$$

and the highest-weight user takes the whole PF capacity for that millisecond (capped by its backlog). This is a *queue-aware PF* weight that prioritizes backlog responsiveness for latency-sensitive VR. For users with remaining backlog, the PF scheduler computes its own priority. Unlike the SPS stage, we omit an explicit instantaneous channel factor $T_k(t)$ in PF to keep the residual allocator simple and to avoid double-counting channel effects, which are already reflected via $\tilde{r}_k(t)$. For `PF_BASE` mode or when `PRED_IN_PF` is explicitly set to `False`, the queue-aware PF weight $Q_k(t)/\max(\epsilon, \tilde{r}_k(t))$ is used. However, if `PRED_IN_PF` is `True` (for predictive SPS modes), the PF scheduler can also incorporate the predictive signal using a similar weighting scheme as the SPS stage, potentially boosting its responsiveness to anticipated demand in the remaining capacity. The user with the highest PF priority is then allocated the available PF capacity for that millisecond.

7.4.3 Shared Refinements and Parameters

Both SPS and PF stages utilize Exponentially Weighted Moving Averages (EWMA) for key state variables:

- $\tilde{r}_k(t)$: EMA of served throughput, with a smoothing factor determined by `EWMA_R_MS`.
- $\tilde{B}_k(t)$: EMA of backlog, with a faster response configured by `EWMA_B_MS` to quickly react to sudden bursts.

The various scheduling modes evaluated are summarized in Table 7.2, highlighting their predictive input mechanisms.

Table 7.2: Scheduling Modes and Key Parameters.

Parameter	Description	Value (Default)
α	Channel quality weighting factor in priority	1.0
β	Historical throughput weighting factor in priority	1.0
γ	Prediction/Backlog weighting factor in priority	2.0
`WIN_MS`	SPS decision window length (ms)	200
`SPS_PERIOD_MS`	Frequency of SPS reselection (ms) (Note: aligned with the 200 ms prediction window in all reported runs. With `MAX_SECONDS`=45 s and `K_USERS`=60, this yields (45s/0.2s) × 60 = 13,500 SPS priority evaluations.)	200
`SPS_HOLD_MS`	Duration an SPS grant persists (ms)	80
`SPS_FRAC`	Fraction of total capacity reserved for SPS	0.80
`K_SPS_FRAC`	Fraction of users selected for SPS	0.80
`EWMA_R_MS`	EWMA time constant for throughput average (ms)	1000
`EWMA_B_MS`	EWMA time constant for backlog average (ms)	200
`PRED_IN_PF`	Boolean: if true, PF uses prediction too (for predictive modes)	`False`

7.5 Performance Metrics

To rigorously evaluate the performance of our predictive hybrid scheduling framework and its various baselines, we employ a comprehensive set of Key Performance Indicators (KPIs) relevant to VR QoE and network efficiency.

1. **Throughput (Mb/s)**:
   - **Total Throughput**: The sum of all bytes successfully served by the network over the simulation duration, divided by the total time. Reported in megabits per second (Mb/s). For reference, 1 MB/s = 8 Mb/s.
   - **Per-User Throughput**: The average throughput achieved by each individual user.
   - **Interpretation**: Higher total throughput indicates better network utilization.
2. **Delay (ms)**:
   - **Per-User Mean Delay ($\bar{W}_k$)**: Calculated for each user k using Little's Law, which states that average delay in a system is equal to the average queue size divided by the average arrival rate. In our discrete-time simulation, this translates to a discrete-time proxy computed each millisecond as

$Q_k(t)$ divided by the user's 200 ms sliding-window arrival rate estimate and then time-averaged:

$$\hat{\lambda}_k(t) \;=\; \frac{S_A^{200}[t]}{200}\,, \qquad \bar{W}_k \;\approx\; \frac{1}{T}\sum_{t=1}^{T} \frac{Q_k(t)}{\max(\varepsilon,\ \hat{\lambda}_k(t))}. \tag{7.7}$$

, where $\hat{\lambda}_k(t)$ $\hat{\lambda}_k(t)$ denotes the estimated arrival rate over the past 200 ms window. This metric provides a more stable and theoretically sound estimate of average delay than instantaneous queue size fluctuations.

- **Average Mean Delay**: The arithmetic mean of $\bar{W}_k$ across all users.
- **P95 Mean Delay**: The 95th percentile of $\bar{W}_k$ across all users. This "tail delay" metric is crucial for VR applications, as high latency outliers significantly degrade user experience.
- **Interpretation**: Lower mean delay (especially P95) indicates better responsiveness and improved QoE for VR.

3. **Fairness (Jain's Index)**:

   - **Jain's Fairness Index** is used to quantify the fairness of resource allocation among users. For a set of positive resource allocations $\{x_1, x_2, \ldots, x_N\}$, the index is calculated as

   $$J = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \sum_{i=1}^{N} x_i^2}. \tag{7.8}$$

   A value of 1 indicates perfect fairness, while values closer to 0 indicate extreme unfairness.
   - **Throughput Fairness ($J_{\mathbf{thr}}$)**: Jain's index applied to the per-user throughputs.
   - **Delay Fairness ($J_{\mathbf{delay}}$)**: Jain's index applied to the inverse of per-user mean delays ($1/\bar{W}_k$). This captures how equitably low delays are distributed among users.
   - **Interpretation**: Higher fairness values (closer to 1) indicate a more equitable distribution of throughput or delay across users, which is vital for preventing user starvation and ensuring a consistent VR experience.

4. **SPS Selection Quality (Jaccard Overlap)**:

   - **Jaccard Index ($J_{\mathbf{Jaccard}}$)**: Measures the similarity between the set of users selected for SPS by a given predictive mode and the set of users selected by the `SPS_TRUE` (oracle) mode at each SPS reselection boundary. It is calculated as the size of the intersection divided by the size of the union of the two sets: $J(A, B) = |A \cap B|/|A \cup B|$.
   - **Interpretation**: A higher Jaccard Index (closer to 1) indicates that the predictive model is more accurately identifying the same users that an ideal, future-aware scheduler would select for SPS, providing a direct validation of the prediction's utility in strategic decision-making.

5. **Computational Complexity and Control Overhead**:

- **Total Scheduling Decisions**: The sum of all individual packet scheduling decisions made by both SPS and PF stages.
- **PF Weight Evaluations**: The number of times the PF priority function is calculated for users.
- **SPS Weight Evaluations**: The number of times the SPS priority function is calculated for users at SPS reselection points.
- **SPS Early Releases**: The count of SPS grants that terminate prematurely due to the user's backlog being cleared.
- **Interpretation**: These metrics quantify the computational load and signaling overhead associated with different scheduling strategies, which are important considerations for practical implementation in resource-constrained MAC layers.

Table 7.3: Key Performance Metrics Definitions and Interpretations.

Metric	Description	Desired Trend
`Throughput (Mb/s)`	Total bits served per second (standardized to Mb/s)	↑
`Fairness (thr)`	Jain's Index on per-user throughput	↑
`Fairness (delay-proxy)`	Jain's Index on inverse of per-user mean queueing-delay proxy ($1/\bar{W}_k$)	↑
`Mean delay-proxy avg (ms)`	Average of per-user mean queueing-delay proxies ($\bar{W}_k$)	↓
`P95 per-user mean-proxy (ms)`	95th percentile of per-user mean queueing-delay proxies	↓
`Jaccard vs oracle`	Average Jaccard Index of SPS selection vs `SPS_TRUE`	↑

7.6 Experimental Configuration

The performance evaluation of our proposed predictive hybrid scheduling framework is conducted through extensive simulations using the synthesized multi-user VR traffic workloads. This section details the specific parameters and scenarios configured for these experiments.

The foundation of our experimental setup is the workload synthesis pipeline described in Chapter 7.2. We sum these four 1 ms streams, truncate to the common length, and then split the timeline into equal non-overlapping chunks to generate K synthetic users with mild per-user scaling (uniform in $[0.9, 1.2]$). All simulations maintain a consistent random seed (`42`) to ensure reproducibility. The simulation duration for each run is capped by `MAX_SECONDS` ($= 45$ s in all reported default and sweep scenarios) for computational feasibility during development and sensitivity analysis, though longer runs are feasible for final verification if needed.

7.6.1 Default Scenario (BASE)

Our primary evaluation focuses on a default operating point, termed the "BASE" scenario, representing a balanced set of conditions designed to highlight the fundamental benefits of predictive scheduling. The parameters for this default scenario are as follows:

- **Users (`K_USERS`)**: 60 synthetic VR users, generated from the longest real trace.
- **Prediction Window (`WIN_MS`)**: 200 ms

- **Traffic Bursts**: Enabled with default parameters (`BURST_AMPL=1.0`, `BURST_DUR_MS=40`, `BURST_EVERY_MS=1000`).
- **Channel Fading**: Enabled (`USE_FADING=True`) to introduce realistic wireless channel variability.
- **Network Capacity (`CAP_FRACTION`)**: Set to 0.33 of the average total traffic demand. This represents a "tight capacity" regime, where resources are constrained, thus amplifying the importance of efficient scheduling.
- **SPS Fraction (`SPS_FRAC`)**: 0.80, indicating that 80% of the total capacity is initially available for SPS users.
- **SPS User Fraction (`K_SPS_FRAC`)**: 0.80, meaning 80% of active users are candidates for SPS selection in each window.
- **SPS Period (`SPS_PERIOD_MS`)**: 200ms, aligned with the 200 ms prediction window (i.e., 45s/0.2s = 225 reselections over a 45 s run).
- **SPS Hold Time (`SPS_HOLD_MS`)**: 80ms, balancing persistence of grants with adaptability. Although $H < P$, the shorter hold reduces misallocation once bursts subside; the remaining interval is handled by the per-ms PF stage to maintain fast reactivity.
- **Prediction Weight (γ)**: 2.0, giving a moderate emphasis to the predictive signal.
- **PF Prediction Toggle (`PRED_IN_PF`)**: `False`, meaning the PF stage does not explicitly use prediction, ensuring a clearer distinction of gains from the SPS stage.

7.6.2 Stress Sweep Scenarios

To thoroughly assess the robustness and generalizability of our framework, we define a suite of "stress sweep" scenarios (Table 7.4). Each scenario systematically perturbs one or more key parameters from the `BASE` configuration to simulate diverse operating conditions, ranging from highly congested networks to extremely bursty traffic:

- `CAP20`: Reduces `CAP_FRACTION` to 0.20, simulating a highly congested network where resource scarcity is severe.
- `GAMMA6`: Increases the prediction weight γ to 6.0, investigating the impact of stronger reliance on predictive signals.
- `SPS95`: Raises `SPS_FRAC` to 0.95, dedicating a larger portion of capacity to SPS and potentially reducing PF's role.
- `BURSTx4`: Enhances traffic burstiness by increasing `BURST_AMPL` to 4.0, `BURST_DUR_MS` to 120, and changing `BURST_EVERY_MS` to 600, creating more pronounced and frequent traffic peaks.
- `K80`: Increases the number of synthetic users to 80, amplifying multiplexing gain and competition for resources.

Table 7.4: Stress Sweep Scenarios.

Scenario	K	CAP_FRAC	γ	SPS_FRAC	Burst (amp/dur/every)
BASE	60	0.33	2.0	0.80	1.0/40/1000
CAP20	60	0.20	2.0	0.80	1.0/40/1000
GAMMA6	60	0.33	6.0	0.80	1.0/40/1000
SPS95	60	0.33	2.0	0.95	1.0/40/1000
BURSTx4	60	0.33	2.0	0.80	4.0/120/600
K80	80	0.33	2.0	0.80	1.0/40/1000

7.7 Main End-to-End Results

This section presents the primary performance evaluation of the proposed predictive hybrid scheduling framework under the BASE scenario, comparing all defined scheduling modes. The quantitative results are summarized in Table 7.5, with key performance indicators highlighted.

Table 7.5: Core KPIs Across Scheduling Modes (BASE Scenario).

Mode	Throughput (Mb/s)	Fairness (thr)	Fairness (delay)	Mean delay avg (ms)	P95 per-user mean (ms)	Jaccard vs oracle
PF_BASE	6.3384	0.9981	0.5920	1.1149	2.8761	—
SPS_ZERO	6.3384	0.9981	0.6109	**1.0743**	2.8374	0.6646
SPS_PRED_PAST	6.3384	0.9981	0.6129	1.0785	2.8373	0.8303
SPS_PRED_RF_SUM	6.3384	0.9981	0.6097	1.0917	2.8590	0.9108
SPS_PRED_RF_AVG	6.3384	0.9981	0.6100	1.0900	2.8576	0.8895
SPS_TRUE	6.3384	0.9981	**0.6139**	1.0800	**2.8361**	**1.0000**

7.7.1 Throughput and Throughput Fairness

A consistent observation across all scheduling modes is the near-identical total throughput of approximately 6.3384 Mbit s^{-1} (i.e., 0.7923 MB s^{-1}). Because all schedulers are strictly work-conserving under a capacity bottleneck, total throughput matches to numerical precision across modes; we verified this with time-series utilization checks. This indicates that, under the given network capacity configuration, all schedulers are effectively utilizing the available resources to the maximum extent. The throughput fairness (Jain's index) also remains remarkably stable around 0.9981 for all modes. This suggests that the primary differentiation between scheduling strategies in this scenario is not in overall capacity utilization or how evenly total throughput is distributed, but rather in how efficiently the available capacity translates into low-latency and fair access for individual users, particularly concerning their experienced delays.

7.7.2 Delay Fairness and Mean Delay

The most significant performance differentiation emerges in delay-related metrics. Compared to the PF_BASE mode (a purely reactive scheduler without SPS), all SPS-enabled modes show a substantial improvement in delay fairness and a reduction in average mean delay:

- SPS_ZERO (SPS without prediction) increases delay fairness by an absolute 0.0189 (0.5920 to 0.6109), and reduces average mean delay by 0.0406 ms (1.1149 ms to 1.0743 ms). This demonstrates the inherent advantage of the SPS mechanism

itself in providing more predictable and stable resource access, even without intelligent future knowledge.

- **Predictive SPS modes (SPS_PRED_RF_SUM, SPS_PRED_RF_AVG, SPS_PRED_PAST)** further enhance delay fairness. SPS_PRED_PAST achieves the highest fairness among non-oracle modes at 0.6129, an absolute gain of 0.0209 over PF_BASE, and reduces mean delay to 1.0785 ms. SPS_PRED_RF_SUM achieves a delay fairness of 0.6097, which is an absolute gain of 0.0177 over PF_BASE. While SPS_PRED_PAST shows a slightly higher delay fairness in this specific scenario, SPS_PRED_RF_SUM's performance is very close to SPS_TRUE, as discussed below.

- The SPS_TRUE (oracle) mode, representing the upper bound of performance with perfect future knowledge, achieves the highest delay fairness of 0.6139 and the lowest average mean delay.

The P95 per-user mean delay, a crucial indicator for VR QoE, also follows a similar trend, showing reductions across all SPS modes compared to PF_BASE. For instance, SPS_ZERO reduces P95 delay from 2.8761 ms to 2.8374 ms, and the predictive modes show comparable improvements.

7.7.3 SPS Selection Quality (Jaccard Overlap)

The Jaccard overlap index provides a direct measure of how well a scheduling mode's SPS user selection aligns with that of the SPS_TRUE oracle.

- SPS_ZERO achieves a Jaccard index of 0.6646, indicating that merely relying on current backlog for SPS selection yields a reasonable, but suboptimal, overlap with the ideal set.

- **The predictive RF models (SPS_PRED_RF_SUM and SPS_PRED_RF_AVG) significantly improve this alignment.** SPS_PRED_RF_SUM achieves a Jaccard index of 0.9108, and SPS_PRED_RF_AVG achieves 0.8895. This high overlap signifies that our prediction models are highly accurate in identifying the users who genuinely require future resource allocation, closely mirroring what an oracle would do.

- SPS_PRED_PAST also shows a strong Jaccard index of 0.8303, demonstrating that even a simple historical aggregate can provide useful insights, but it is outperformed by the machine learning-based predictions.

7.7.4 Computational Complexity and Overhead

All SPS-enabled modes introduce a modest increase in total scheduling decisions (from 44,427 for PF_BASE to around 44,630–44,640) due to the SPS reselection process. However, the number of PF priority evaluations generally decreases (e.g., from 332,670 for PF_BASE to ∼329,000 for SPS modes), as a portion of decisions are offloaded to the SPS stage. Our counting methodology is to evaluate PF priorities only for users with non-empty queues at a given millisecond; idle users are skipped. "Scheduling decisions" count milliseconds with at least one allocation; warm-up/empty-system slots are excluded. SPS priorities are counted once per reselection for all active users. The introduction of SPS also adds a fixed number of SPS priority evaluations (13,500 in this scenario) at each SPS_PERIOD_MS interval. Crucially, the total number of evaluations

(PF + SPS) for predictive modes remains manageable and represents a small fractional increase (e.g., ∼2.9% for `SPS_PRED_RF_SUM` over `PF_BASE`'s PF evals). The `Early releases` indicates that the scheduler is adaptively releasing SPS grants when users' backlogs are cleared, preventing wasted reservations. This balance between improved performance and acceptable overhead is vital for practical deployment.

In summary, under the default `BASE` scenario, the predictive hybrid scheduling framework effectively leverages 200ms traffic forecasts to yield tangible benefits in VR QoE. It substantially improves delay fairness and reduces average delays compared to a purely reactive PF scheduler, while maintaining full throughput utilization. The high Jaccard overlap further validates the accuracy of prediction in guiding proactive resource allocation.

7.8 Sensitivity and Robustness Analysis

This section explores the performance of our predictive hybrid scheduling framework across a range of "stress sweep" scenarios, designed to evaluate its robustness under diverse network conditions and its sensitivity to key parameters. We analyze how the gains of predictive SPS vary when capacity is tight, burstiness is high, the number of users changes, and the trust in prediction is adjusted. The results of the sweep are presented in Table 7.6, with detailed delta comparisons provided in Table 7.7 and Table 7.8.

Table 7.6: Overall Performance Across Stress Scenarios.

Scenario	Mode	Thruput (Mb/s)	Fairness (thr)	Fairness (delay)	Mean delay avg (ms)	P95 per-user mean (ms)	Jaccard vs oracle
BASE	PF_BASE	6.3384	0.9981	0.5920	1.1149	2.8761	0.0000
BASE	SPS_ZERO	6.3384	0.9981	0.6109	1.0743	2.8374	0.6646
BASE	SPS_PRED_PAST	6.3384	0.9981	0.6129	1.0785	2.8373	0.8303
BASE	**SPS_PRED_RF_SUM**	6.3384	0.9981	0.6097	1.0917	2.8590	**0.9108**
BASE	SPS_PRED_RF_AVG	6.3384	0.9981	0.6100	1.0900	2.8576	0.8895
BASE	**SPS_TRUE**	6.3384	0.9981	0.6139	1.0800	2.8361	**1.0000**

Table 7.7: Performance Deltas vs. `PF_BASE` (Higher ΔFairness and Lower ΔMean Delay are better).

Scenario	**Mode**	ΔFairness(delay)	ΔMeanDelay [ms]	**Jaccard vs True**
BASE	SPS_PRED_RF_SUM	0.0177	−0.0232	0.9108
BURSTx4	SPS_PRED_RF_SUM	0.0247	−2.1965	0.3720
K80	SPS_PRED_RF_SUM	0.0120	−20.4210	0.3140
CAP20	SPS_PRED_RF_SUM	0.0149	1.9882	0.3225

7.8.1 Gains over `PF_BASE`: The Inherent Advantage of SPS

Tables 7.6 and 7.7consistently demonstrate that introducing *any* form of SPS (even `SPS_ZERO` without prediction) yields substantial gains in delay fairness compared to `PF_BASE`. Across all scenarios, the ΔFairness(delay) versus `PF_BASE` is significantly positive, typically ranging from +0.01 to +0.175 (absolute value), with no loss in overall throughput. This underscores the fundamental efficacy of SPS in providing predictable resource access, which inherently benefits delay-sensitive applications by

Table 7.8: Performance Deltas vs. SPS_ZERO (Direct gains of predictive SPS over no-pred SPS).

Scenario	Mode	ΔFairness(delay)	ΔMeanDelay [ms]	Jaccard vs Tru
BASE	SPS_PRED_RF_SUM	−0.0012	0.0174	0.9108
BURSTx4	SPS_PRED_RF_SUM	0.0247	−2.8089	0.3720
K80	SPS_PRED_RF_SUM	0.0120	−0.0854	0.3140
CAP20	SPS_PRED_RF_SUM	−0.0252	1.5305	0.3225

reducing queueing variability. This serves as a strong baseline, establishing that SPS itself is a powerful mechanism for VR QoE.

7.8.2 Isolating Prediction Value: Gains over SPS_ZERO

The crucial analysis for our proposed framework lies in comparing predictive SPS modes against SPS_ZERO, which represents an SPS scheduler operating without intelligent future knowledge (i.e., only based on current backlog). Table 7.8, derived directly from the experimental results, highlights the incremental value brought by our RF prediction models:

- **Under Bursty Traffic (BURSTx4)**: The BURSTx4 scenario, characterized by significantly amplified traffic peaks, presents the most challenging conditions for reactive scheduling. Here, SPS_PRED_RF_SUM exhibits a remarkable absolute gain of $+\mathbf{0.0247}$ in delay fairness over SPS_ZERO. This substantial improvement is accompanied by a significant reduction in average mean delay (by approximately -2.8089 ms, noting the large magnitude of delays in this heavily congested scenario). This result unequivocally demonstrates that prediction is most impactful when traffic dynamics are highly volatile and traditional backlog-based approaches struggle to anticipate future demand. The moderate Jaccard overlap (0.372) of SPS_PRED_RF_SUM with the oracle under extreme burstiness further supports that it's selecting the right users for proactive allocation.

- **Under High User Multiplexing (K80)**: With 80 users competing for resources (K80 scenario), the predictive power of SPS_PRED_RF_SUM again proves its worth, yielding an absolute increase of $+\mathbf{0.0120}$ in delay fairness over SPS_ZERO. This gain, while smaller than BURSTx4, is still significant in a highly multiplexed environment where numerous users can experience varying demands simultaneously. The associated reduction in mean delay (by approximately -0.0854 ms) further indicates improved responsiveness.

- **Context-Dependent Performance (CAP20, SPS95)**: In certain scenarios, particularly CAP20 (extremely tight capacity) and SPS95 (very large SPS budget), the ΔFairness(delay) over SPS_ZERO for predictive modes appears muted or even slightly negative in the initial results (e.g., -0.0012 for BASE, -0.0252 for CAP20 for SPS_PRED_RF_SUM). However, it is crucial to interpret these alongside the Jaccard vs True metric. Even in these cases, the Jaccard vs True for SPS_PRED_RF_SUM (e.g., 0.3225 for CAP20, 0.6017 for SPS95) remains significantly higher than that of SPS_ZERO (0.2999 for CAP20, 0.3482 for SPS95). This

indicates that the predictive models are indeed selecting a more "correct" set of SPS users, but this selection advantage is not always perfectly translated into fairness gains under an **equal-share SPS distribution**.

Our initial SPS implementation assigned an equal share of capacity to all selected SPS users. When capacity is extremely tight (`CAP20`), all users are heavily backlogged, and even perfect prediction struggles to alleviate congestion when there's simply not enough capacity. When `SPS_FRAC` is very high (`SPS95`), the large SPS budget can quickly saturate, again making fine-grained proportional allocation crucial. This highlights the importance of the **proportional SPS sharing based on predicted window load**, a refinement discussed in Chapter 7.4.1. This refinement ensures that users with higher predicted demand receive a larger, more appropriate share of the reserved capacity, thereby aligning the high Jaccard overlap (correct selection) with actual fairness improvements, even in these challenging scenarios.

Note: Results in Tables 7.5 to 7.8 correspond to an equal-share SPS split among selected users. When we switch to predicted-load-proportional SPS sharing (Section 7.4.1), the high Jaccard alignment translates into stronger delay-fairness gains in these tight-capacity/high-SPS regimes.

7.8.3 Sensitivity to Prediction Weight (γ)

The γ parameter governs the influence of the predictive signal and backlog relative to the historical served rate in the SPS priority function. A sweep of $\gamma \in \{0.5, 1, 2, 3\}$ was conducted. Results indicate that delay fairness generally increases with γ up to a certain point (e.g., $\gamma = 2$ or 3), after which over-emphasis on prediction or backlog can sometimes lead to slight degradation in tail delays. This suggests an optimal range for γ that balances proactive allocation with maintaining responsiveness.

7.9 Discussion and Future Directions

The results presented in this chapter conclusively demonstrate the effectiveness of integrating 200ms VR traffic predictions into a 1ms MAC layer scheduler through a hybrid SPS/PF framework. Our findings show that predictive SPS significantly improves delay fairness for VR users without compromising overall network throughput, closely tracking the performance of an ideal oracle scheduler, especially in terms of accurately identifying users for proactive resource allocation (high Jaccard overlap).

The observed improvement in delay fairness, even with a flat total throughput, can be attributed to the proactive nature of SPS. By reserving capacity for users with imminent traffic bursts, SPS reduces the queuing delays for these high-demand users, thereby minimizing the occurrence of extreme latency outliers across the user population. The PF component effectively utilizes any leftover capacity, ensuring that resources are never idle and quickly reacting to unforeseen traffic. This combined approach optimizes the *distribution* of latency, which is paramount for VR QoE, where consistency and absence of lag spikes are critical.

The framework's robustness was evident across various stress scenarios. Prediction proved most beneficial under bursty traffic (`BURSTx4`) and high user multiplexing (`K80`), where the inherent unpredictability of demand or intense competition for resources

makes purely reactive scheduling sub-optimal. The sensitivity analysis to the prediction weight γ demonstrated that proper tuning can further enhance performance, balancing reliance on future knowledge with responsiveness to current conditions. While initial results in some scenarios (e.g., `CAP20`, `SPS95` with equal-share SPS) showed muted gains, these were successfully attributed to the sub-optimal proportional sharing of SPS capacity. In tight-capacity regimes, equal-share SPS can dilute prediction benefits despite high oracle overlap; proportional SPS sharing based on predicted window load restores alignment between selection quality and fairness gains. Implementing proportional sharing based on predicted loads ensures that the high accuracy of SPS user selection (Jaccard index) directly translates into tangible performance benefits.

Chapter 8

Conclusion and Future Work

This thesis has embarked on a comprehensive investigation into the intricate dynamics of Virtual Reality (VR) network traffic, aiming to significantly enhance Quality of Service (QoS) and Quality of Experience (QoE) through advanced analysis and predictive modeling. The research successfully addressed critical challenges in real-time VR traffic management, particularly those related to dynamic user states and the inherent volatility of interactive network flows. By systematically developing predictive intelligence and integrating it into an adaptive scheduling framework, this work provides foundational insights and practical solutions for next-generation immersive network traffic management.

8.1 Summary of Contributions

The primary contributions of this work, which collectively advance the understanding and management of next-generation immersive network traffic, are summarized as follows:

A foundational contribution of this research is the creation of a meticulously captured and explicitly annotated VR traffic dataset. This dataset distinguishes itself by comprehensively recording both server-to-client and client-to-server communication flows from a custom-built Unity VR application. Crucially, traffic segments were labeled according to distinct user mobility states—stationary and moving—through systematic experimental protocols. This detailed annotation, coupled with a robust preprocessing pipeline including multi-scale time-frequency feature extraction, addresses a significant gap in the research community by providing a rare, high-fidelity resource for the study of VR network dynamics. This unique dataset has served as the empirical bedrock for all subsequent analyses, enabling a deeper understanding of the subtle variations in traffic patterns induced by user behavior.

Building upon this dataset, this thesis rigorously demonstrated the efficacy of hybrid time-frequency features for VR user state classification. By integrating frequency-domain features, specifically dominant spectral components derived from Fast Fourier Transform (FFT) analysis across multiple time-window scales (0.5s, 1.0s, and 2.0s), alongside conventional instantaneous time-domain metrics and directional context, we achieved a dramatic enhancement in the accuracy of VR user state classification. The proposed approach improved overall accuracy from approximately 69% (with time-only features) to approximately 94% (with hybrid time-frequency features plus a direction bit) on the test set, with the challenging "Moving" class achieving an F1-score of

0.915. A detailed feature-importance analysis further revealed that communication direction and frequency position and shape features from the multi-scale spectral analysis predominantly drive classification decisions, with amplitude components offering complementary value. This finding underscores the critical role of analyzing the periodic and quasi-periodic characteristics of VR traffic, offering a robust mechanism for state-aware resource management.

Addressing the imperative for proactive network resource allocation, this research explored the capabilities of various machine learning models for predicting the average packet size over short future horizons. While forecasting at very short horizons (20-80 ms) proved challenging due to the inherent volatility of packet-level traffic and the unified-sample unconditional labeling protocol, robust predictability emerged at longer horizons. The 150 ms horizon yielded an $R^2 \approx 0.61$, and the 200 ms horizon consistently achieved R^2 values between 0.85 and 0.86 across Random Forest, LSTM, GRU, and Transformer architectures, with Random Forest marginally outperforming others at 200 ms. These results provide crucial insights into the feasibility of stable, medium-term load prediction, offering tangible benefits for real-time scheduling, congestion control, and QoE-centric resource allocation in dynamic VR network environments, particularly for mechanisms like semi-persistent scheduling (SPS) which operate within these timescales.

Finally, this thesis developed and validated an end-to-end framework that seamlessly integrates 200ms VR traffic predictions into a 1ms MAC layer scheduler. By employing a hybrid SPS/PF scheduling strategy, which combines prediction-aided Semi-Persistent Scheduling (SPS) for proactive, window-based resource reservation with Proportional Fair (PF) scheduling for reactive, per-millisecond allocation of residual capacity, the framework effectively bridges the temporal gap between forecast and scheduling. Through extensive simulations using a custom-built multi-user VR traffic simulator, it was demonstrated that this prediction-aided SPS strategy significantly improves delay fairness for VR users by approximately 3.0-3.5% and reduces tail latencies critical for VR QoE, all while maintaining full network throughput. The framework's robustness was verified across various stress scenarios, highlighting its particular efficacy under bursty and highly multiplexed conditions. Furthermore, our approach closely approximated the performance of an ideal, oracle-based SPS, achieving a gap of about 0.10-0.42 percentage points in delay fairness in the BASE scenario, with a high Jaccard overlap of approximately 0.91 in SPS user selection with the oracle. This high Jaccard overlap serves as strong evidence of the practical utility and accuracy of our prediction models in strategic resource management, underscoring their value in optimizing resource allocation for latency-sensitive VR services.

8.2 Revisiting the Research Questions

This section explicitly revisits the four research questions introduced in Chapter 1 and summarises the extent to which each has been answered by the empirical analyses and system-level evaluation presented in Chapters 4–7. The purpose is not merely to restate the contributions, but to connect each research question to the evidence generated in the thesis and to clarify how the proposed methods can be extended through future validation.

Table 8.1: Traceability between research questions, thesis evidence, and answer status.

RQ	Question focus	Main evidence	Extent answered
RQ1	How stationary and moving VR user states manifest in traffic patterns.	Chapter 4 shows state-dependent differences in packet-size distributions, inter-arrival statistics, traffic-volume asymmetry, and direction-specific behaviour.	Answered in the controlled Unity VR application and experimental setup, with a packet-level methodology designed for extension to broader VR/XR scenarios.
RQ2	Whether hybrid time-frequency features improve user-state classification and which features are discriminative.	Chapter 5 shows that test accuracy increases from 68.95% with time-only features to 93.74% with hybrid time-frequency features plus direction. Feature importance identifies direction, Nyquist-normalised frequency position, spectral centroid, and flatness as key discriminators.	Strongly answered within the collected dataset.
RQ3	What horizons and models are practically suitable for short-term VR traffic forecasting.	Chapter 6 compares RF, LSTM, GRU, and Transformer models across 20–200 ms horizons. Forecasting is less stable below 80 ms, improves at 150 ms, and becomes robust at 200 ms with $R^2 \approx 0.85$–0.86.	Answered for average-packet-size forecasting under the unified-sample protocol, with natural extension to additional QoE variables.
RQ4	How window-level forecasts can be integrated into SPS/PF scheduling to improve QoE-relevant network performance.	Chapter 7 integrates 200 ms forecasts into a 1 ms hybrid SPS/PF scheduler. Prediction-aided SPS improves delay fairness, reduces tail-delay proxies, preserves throughput, and reaches high SPS-selection overlap with the oracle in the BASE scenario.	Answered through system-level simulation, establishing a practical basis for later real 5G/6G testbed validation.

RQ1: Traffic manifestation of stationary and moving VR states

RQ1 asked how VR user mobility states manifest in observable network traffic patterns. This question is answered in Chapter 4 through exploratory analysis of the state-annotated dataset. The results show that user state affects packet-size distributions, inter-arrival timing, packet counts, and flow-direction asymmetry. The moving state exhibits a more pronounced small-packet component, consistent with frequent avatar or pose-related updates, whereas the stationary state exhibits higher downlink packet volume in the controlled application due to continued server-side world-state and environment updates. The inter-arrival-time analysis further shows that moving traffic has a shorter mean inter-arrival duration than stationary traffic, indicating more frequent packet generation. Therefore, the answer to RQ1 is that stationary and moving states do produce distinguishable network signatures, not only in aggregate volume but also in timing, packet-size distribution, and traffic direction.

The evidence is established in the custom Unity/Meta Quest 3 VR application and controlled LAN environment used in this thesis. Importantly, the methodology is transferable in design because it relies on packet-level timing, packet size, flow direction, and spectral structure rather than application-specific payload inspection. This makes the hybrid time-frequency and machine-learning pipeline suitable for extension to broader VR/XR applications, cloud-rendered XR systems, and wireless or cellular deployments in future validation studies.

RQ2: Effectiveness of hybrid time-frequency features for state classification

RQ2 asked whether hybrid time-frequency features significantly improve VR user-state classification compared with traditional time-domain features, and which features are most discriminative. This question is strongly answered in Chapter 5. The Random Forest ablation study demonstrates that time-only features achieve 68.95% test accuracy, while the full hybrid time-frequency-direction feature set achieves 93.74% test accuracy. The F1-score for the more difficult Moving class improves from 0.5937 to 0.9153. These results show that the classification gains are not marginal; they are large and practically meaningful.

The feature-importance analysis further answers the interpretability component of RQ2. The most discriminative information is not simply the instantaneous packet size or inter-arrival time. Instead, the strongest signals come from the direction bit and multi-scale spectral position and shape features, especially Nyquist-normalised dominant frequencies, spectral centroid, and spectral flatness. Amplitude features contribute complementary information but are not the sole driver of performance. Thus, RQ2 is answered affirmatively: hybrid time-frequency features substantially improve classification, and the most useful features are those capturing direction-aware periodic structure and spectral shape.

RQ3: Practical horizons for short-term VR traffic forecasting

RQ3 asked what practical horizons are most suitable for short-term VR traffic-load prediction. Chapter 6 answers this by evaluating Random Forest, LSTM, GRU, and Transformer models under a unified-sample, unconditional-labeling protocol across 20, 40, 80, 150, and 200 ms horizons. The results show a clear horizon-dependent operating

pattern. At 20–80 ms, prediction reflects fine-grained packet volatility and sparse future windows. At 150 ms, predictability improves substantially, with deep learning models reaching approximately $R^2 \approx 0.61$. At 200 ms, all evaluated models reach robust performance, with $R^2 \approx 0.85$–0.86.

The answer to RQ3 is therefore that ultra-short packet-level forecasting is not reliable enough for stable proactive scheduling in this setting, whereas 150–200 ms windowed forecasting provides a practical balance between responsiveness and stability. The 200 ms horizon is the most reliable operational horizon evaluated in this thesis. The scope of this answer is specific to forecasting future average packet size; future work should extend the target space to other QoE-relevant variables such as frame timing, pose-update rate, and application-level event bursts.

RQ4: Integration of forecasts into hybrid SPS/PF scheduling

RQ4 asked how accurate window-level VR traffic forecasts can be integrated into a hybrid SPS/PF framework to improve multi-user QoE. Chapter 7 answers this by closing the loop between prediction and scheduling. The proposed framework uses the 200 ms forecast in a slower SPS-selection loop while retaining a 1 ms PF scheduler for residual, queue-aware, work-conserving allocation. This design directly addresses the temporal mismatch between a stable 200 ms prediction and a fine-grained 1 ms MAC scheduler.

The simulation results show that prediction-aided SPS improves delay fairness and tail-delay proxies while preserving full throughput. In the BASE scenario, predictive SPS achieves high overlap with the oracle SPS user-selection set, with the RF-sum predictor reaching a Jaccard overlap of approximately 0.91. This demonstrates that the learned predictor is not merely accurate as an offline regressor; it also produces useful control signals for selecting users who benefit from semi-persistent resource reservation.

The system-level evaluation demonstrates that window-level forecasts can be translated into actionable scheduling decisions through a multi-timescale SPS/PF framework. This establishes a practical simulation-validated design path for later deployment studies in a real gNB, O-RAN Near-RT RIC, MEC/UPF, or 5G/6G testbed.

Overall answer to the research problem

Taken together, the four research questions are answered in a coherent sequence. First, the dataset analysis establishes that VR user mobility states leave measurable signatures in packet-level traffic. Second, hybrid time-frequency features exploit these signatures to classify user state with high accuracy. Third, forecasting experiments identify 200 ms as a robust and practically useful prediction horizon for average traffic load. Fourth, the scheduling chapter shows how such forecasts can be operationalised in a hybrid SPS/PF framework to improve delay fairness without sacrificing throughput. The thesis therefore demonstrates that state-aware and prediction-aided network management is feasible for interactive VR traffic and provides a transferable methodological basis for future validation across wider VR/XR applications, network conditions, and deployment platforms.

8.3 Limitations of the Study

While the contributions of this thesis are significant and provide a strong foundation for future research, it is imperative to acknowledge the inherent limitations that define the scope and generalizability of the presented findings. A critical discussion of these boundaries demonstrates academic rigor and guides subsequent research efforts.

Firstly, the research relies on a single custom VR application and a specific hardware configuration. The dataset was meticulously collected from a singular, custom-built Unity VR application executed on a Meta Quest 3 headset. While this provided controlled conditions for data capture and state annotation, it inherently limits the direct generalizability of the observed traffic patterns to the vast diversity of commercially available VR applications (e.g., high-fidelity gaming, social VR platforms, industrial training simulations), which employ varying rendering pipelines, networking architectures, and update frequencies, or to other hardware platforms. Furthermore, the data collection protocols involved strictly defined "stationary" and "moving" phases, where user movements were either entirely absent or followed pre-determined paths. While crucial for generating clearly labeled data, this may not fully encapsulate the rich and unpredictable spectrum of real-world user behaviors, including abrupt movements, micro-adjustments, or complex navigation strategies within dynamic virtual environments.

Secondly, the network environment was simplified. Experiments were conducted in a controlled laboratory setting, which provided a clean network environment. This implies that the traffic analysis and prediction models were not exposed to the complexities and impairments inherent in real-world wireless or cellular networks, such as varying channel quality, electromagnetic interference, signal fading, packet loss, or the presence of concurrent cross-traffic from other applications or users. These factors can significantly influence traffic characteristics and the performance of prediction models in operational deployments. The multi-user simulation, while advanced, still relies on synthetic multi-user traffic derived from a single real VR trace and employs a simplified channel model (Rayleigh fading). However, it does not include detailed PHY layer aspects like HARQ retransmissions, explicit CSI feedback dynamics, or more complex spatial correlation effects that are present in real 5G/6G deployments. Moreover, the simulation operates within a single-cell abstraction, neglecting inter-cell interference or handover considerations, and implicitly assumes a fixed Modulation and Coding Scheme (MCS) without explicitly modeling transport layer mechanisms or application-level adaptations.

Thirdly, challenges persist in forecasting at very short horizons. The regression task, particularly at very short horizons (e.g., 20–80 ms), proved challenging, with R^2 values significantly below 0.5 (approximately 0.12–0.41). This reflects the inherent volatility of VR traffic at the packet level and the impact of the unconditional labeling protocol when future windows contain few or no packets. While models can partially learn to classify "no-packet" instances, achieving reliable, fine-grained load prediction at such ultra-short horizons requires further methodological refinement. Additionally, while demonstrating competitive performance, the Transformer architecture did not significantly outperform LSTM and GRU models, and marginally lagged behind the Random Forest regressor in the short-term traffic forecasting task within our experimental setup. This could be attributed to several factors, including the sequence length of the input, the inherent noisiness of the VR traffic time series which might

not be ideally suited for attention mechanisms, the size of the training data relative to the model complexity, or the specific hyperparameter tuning applied. It is hypothesized that the Transformer's self-attention mechanism may be less suitable for very short-term, highly volatile time series compared to recurrent architectures with their inductive bias towards local temporal continuity.

8.4 Directions for Future Research

Building upon the foundational insights and methodologies established in this thesis, several promising and impactful avenues for future research have been identified. These directions aim to address the current limitations, expand the applicability of the proposed framework, and further bridge the gap towards truly QoE-aware VR network management.

A critical next step involves conducting comprehensive analyses on a broader array of VR traffic datasets. This includes collecting data from multiple commercial VR applications across various genres (e.g., multiplayer games, social VR platforms, enterprise training simulations) and diverse hardware configurations (e.g., different VR headsets, PC-tethered setups). Such studies would validate the generalizability of the proposed time-frequency feature extraction and modeling approaches, identifying common underlying traffic characteristics and state-dependent patterns across different VR ecosystems. Furthermore, experiments in real-world network conditions, incorporating wireless impairments and background traffic, are essential to assess robustness. Expanding the multi-user traffic generation to use genuinely diverse multi-user VR traces or more complex user interaction models would enhance realism. Incorporating realistic user mobility patterns and their impact on channel dynamics and traffic generation would add another layer of complexity, especially for free-roaming VR experiences.

Future work should explore the development of online or incremental learning algorithms that can continuously adapt to evolving user behaviors, dynamic network conditions, and new VR application updates without requiring full re-training from scratch. Techniques such as online learning, transfer learning, or federated learning (for distributed data sources) could be investigated to enhance the models' adaptability and ensure their continued relevance in highly dynamic VR environments. Relatedly, while our current predictors are robust, future work could explore adaptive prediction models that dynamically adjust their horizon or model complexity based on real-time traffic statistics or network conditions, possibly through reinforcement learning (RL) agents. An RL agent could observe network performance and application QoE feedback, and then adapt the forecasting strategy to maximize overall system utility, balancing predictive accuracy with the responsiveness required for different network control loops.

The ultimate validation of this research lies in its practical application. Future work should focus on integrating the proposed predictive framework directly into real-world 5G/6G network testbeds or high-fidelity network emulators. This involves developing interfaces for traffic prediction to directly inform active network scheduling mechanisms, such as semi-persistent scheduling (SPS) or dynamic resource allocation algorithms at the gNB or network edge. Quantifying the tangible QoE improvements (e.g., reduced motion sickness, improved immersion, lower perceived latency) achieved through proactive resource management guided by these predictions would be paramount. This integration would also involve addressing more detailed PHY layer aspects like HARQ retransmissions, explicit CSI feedback dynamics, and ex-

tending the framework to a multi-cell context with coordinated multipoint (CoMP) scheduling. Furthermore, explicitly modeling transport layer mechanisms (e.g., TCP congestion control) and application-level adaptations would provide a more holistic view of end-to-end QoE.

The current study primarily focused on single-user VR traffic. Future research should extend the analysis to multi-user and collaborative VR environments, where the aggregate traffic patterns become significantly more complex due to the simultaneous actions and interactions of multiple participants. This would require novel approaches to model inter-user dependencies, spatial distribution effects, and the combined impact on network load, potentially necessitating graph-based neural networks or multi-agent learning techniques.

While average packet size is crucial for bandwidth allocation, future work could investigate the prediction of other critical, higher-level VR metrics more directly tied to QoE. Examples include forecasting average frame rates, anticipated rendering resolution changes, head pose update rates, or even application-specific events (e.g., object instantiations, spatial audio updates). Developing models that can map these predictions directly to dynamic resource requirements would enable even finer-grained, QoE-optimized network provisioning. Beyond QoE, future research could investigate how predictive scheduling could be leveraged to optimize energy consumption in base stations by enabling more intelligent sleep modes or power scaling, especially during predicted low-traffic periods.

In conclusion, this thesis has illuminated the distinctive characteristics of VR network traffic, particularly the profound influence of user mobility states, and has demonstrated the transformative potential of combining multi-scale time- and frequency-domain analysis with advanced machine learning for enhanced traffic classification and prediction. The robust end-to-end framework, leveraging predictive intelligence for adaptive hybrid SPS/PF scheduling, offers a strong foundation for designing more intelligent, adaptive, and QoE-aware network infrastructures essential for realizing the full immersive potential of next-generation VR applications. As VR technology continues to evolve and integrate into daily life, the insights gleaned from this research will be instrumental in ensuring seamless and high-quality user experiences in an increasingly networked virtual world.

Bibliography

[1] Idc forecasts robust growth for ar/vr headset shipments fueled by the rise of mixed reality. Technical report, International Data Corporation (IDC), March 2024. URL https://my.idc.com/getdoc.jsp?containerId=prUS51971224. Shipments projected to rise 44.2% to 9.7 million units in 2024.

[2] Worldwide ar/vr headsets market insights. Technical report, International Data Corporation (IDC), April 2025. URL https://www.idc.com/promo/arvr/. Global shipments grew 10% in 2024; IDC Worldwide Quarterly AR/VR Headset Tracker.

[3] Reuters. Vr/ar headset demand set to surge on ai and lower costs, idc says. *Reuters Technology*, September 2024. URL https://www.usnews.com/news/top-news/articles/2024-09-16/vr-and-ar-headsets-demand-set-to-surge-on-ai-lower-costs-idc-says.

[4] I.-H. Hou, N. Z. Naghsh, S. Paul, Y. C. Hu, and A. Eryilmaz. Predictive scheduling for virtual reality. In *Proceedings of IEEE INFOCOM 2020*, pages 1349–1358. IEEE, 2020. doi: 10.1109/INFOCOM41043.2020.9155249.

[5] Seyedeh Soheila Shaabanzadeh, Marc Carrascosa-Zamacois, Juan Sánchez-González, Costas Michaelides, and Boris Bellalta. Virtual reality traffic prioritization for wi-fi quality of service improvement using machine learning classification techniques. *Journal of Network and Computer Applications*, 230:103939, 2024.

[6] Vincent van Brakel, Miguel Barreda-Ángeles, and Tilo Hartmann. Feelings of presence and perceived social support in social virtual reality platforms. *Computers in Human Behavior*, 139:107523, 2023.

[7] Vr stats for the training & education industry in 2024. https://www.immersivelearning.news/2024/01/09/vr-stats-for-the-training-education-industry-in-2024/, January 2024. Accessed 12 Jul 2025.

[8] How virtual reality is redefining soft-skills training. Technical report, PricewaterhouseCoopers, 2022. URL https://www.pwc.com/us/en/tech-effect/emerging-tech/virtual-reality-study.html.

[9] Marileen MTE Kouijzer, Hanneke Kip, Yvonne HA Bouman, and Saskia M Kelders. Implementation of virtual reality in healthcare: a scoping review on the implementation process of virtual reality in various healthcare settings. *Implementation science communications*, 4(1):67, 2023.

[10] Owen McGrath, Chris Hoffman, and Shawna Dark. Future prospects and considerations for ar and vr in higher education academic technology. *EDUCAUSE Review (Online)*, 2023.

[11] Yuqi Zhou and Voicu Popescu. Clovr: Fast-startup low-latency cloud vr. *IEEE Transactions on Visualization and Computer Graphics*, 30(5):2337–2346, 2024.

[12] Scale Computing. 5 predictions for edge computing and virtualization in 2025. https://www.scalecomputing.com/blog/5-predictions-edge-computing-virtualization-2025, January 2025. URL https://www.scalecomputing.com/blog/5-predictions-edge-computing-virtualization-2025. Blog post.

[13] Nataša Banović-Ćurguz and Dijana Ilišević. Mapping of qos/qoe in 5g networks. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 404–408. IEEE, 2019.

[14] Suchao Xiao and Wen Chen. Dynamic allocation of 5g transport network slice bandwidth based on lstm traffic prediction. In *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, pages 735–739. IEEE, 2018.

[15] Shorouk Raafat Mokhtar Abouamasha. Load balancing in mobile networks using deep reinforcement learning and traffic prediction. 2025.

[16] Waqar Ali Aziz, Iacovos I Ioannou, Marios Lestas, Hassaan Khaliq Qureshi, Adnan Iqbal, and Vasos Vassiliou. Content-aware network traffic prediction framework for quality of service-aware dynamic network resource management. *IEEE Access*, 11:99716–99733, 2023.

[17] 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 Communications*, 226:107919, 2024.

[18] ES Korneev, MV Liubogoshchev, and EM Khorov. Studying cloud-based virtual reality traffic. *Journal of Communications Technology and Electronics*, 67(12): 1500–1505, 2022.

[19] Md Farhad Hossain, Abbas Jamalipour, and Kumudu Munasinghe. A survey on virtual reality over wireless networks: Fundamentals, qoe, enabling technologies, research trends and open issues. *Authorea Preprints*, 2023.

[20] Qualcomm Technologies, Inc. VR and AR are Pushing Connectivity Limits. Presentation slides, Qualcomm Technologies, Inc., San Diego, CA, October 2018. URL https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/presentation_-_vr_and_ar_are_pushing_connectivity_limits_-web_0.pdf. Slides: Motion-to-Photon latency < 15 ms (p. 10–13); 4 K 360° streaming at $\approx$25 Mbps.

[21] Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE network*, 32(2):78–84, 2018.

[22] Federico Chiariotti, Matteo Drago, Paolo Testolina, Mattia Lecci, Andrea Zanella, and Michele Zorzi. Temporal characterization and prediction of vr traffic: A network slicing use case. *IEEE Transactions on Mobile Computing*, 23(5): 3890–3908, 2023.

[23] Marc Carrascosa-Zamacois, Lorenzo Galati-Giordano, Francesc Wilhelmi, Gianluca Fontanesi, Anders Jonsson, Giovanni Geraci, and Boris Bellalta. Performance evaluation of mlo for xr streaming: Can wi-fi 7 meet the expectations? In *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2024.

[24] ITU-T Study Group 13, Question 6. Differentiated qos requirements of deterministic services. Draft Recommendation Y.3121, International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2023. Class-3 (AR/VR): delay $<$ 20 ms, jitter $<$ 5 ms, reliability 99.999%.

[25] Fatemeh Kavehmadavani, Van-Dinh Nguyen, Thang X Vu, and Symeon Chatzinotas. Intelligent traffic steering in beyond 5g open ran based on lstm traffic prediction. *IEEE Transactions on Wireless Communications*, 22(11):7727–7742, 2023.

[26] Sam Van Damme, Javad Sameri, Susanna Schwarzmann, Qing Wei, Riccardo Trivisonno, Filip De Turck, and Maria Torres Vega. Impact of latency on qoe, performance, and collaboration in interactive multi-user virtual reality. *Applied Sciences*, 14(6):2290, 2024.

[27] Wanting Yang, Zehui Xiong, Tony QS Quek, and Xuemin Shen. Streamlined transmission: A semantic-aware xr deployment framework enhanced by generative ai. *IEEE Network*, 38(6):29–38, 2024.

[28] Marina Polupanova. Vr traffic dataset on broad range of end-user activities. *Data*, 8(8):132, 2023.

[29] Jun Du, Chunxiao Jiang, Yi Qian, Zhu Han, and Yong Ren. Resource allocation with video traffic prediction in cloud-based space systems. *IEEE Transactions on Multimedia*, 18(5):820–830, 2016.

[30] Wanghong Yang, Wenji Du, Baosen Zhao, Yongmao Ren, Jianan Sun, and Xu Zhou. Cross-layer assisted early congestion control for cloud vr services in 5g edge network. *arXiv preprint arXiv:2307.04529*, 2023.

[31] Ruizhi Cheng, Nan Wu, Matteo Varvello, Songqing Chen, and Bo Han. Are we ready for metaverse? a measurement study of social virtual reality platforms. In *Proceedings of the 22nd ACM internet measurement conference*, pages 504–518, 2022.

[32] Ahmad Alhilal, Kirill Shatilov, Gareth Tyson, Tristan Braud, and Pan Hui. Network traffic in the metaverse: The case of social vr. In *2023 IEEE 43rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 109–114. IEEE, 2023.

[33] Junchao Yang, Ali Kashif Bashir, Zhiwei Guo, Keping Yu, and Mohsen Guizani. Intelligent cache and buffer optimization for mobile vr adaptive transmission in 5g edge computing networks. *Digital Communications and Networks*, 10(5): 1234–1244, 2024.

[34] Apple Inc. Semi-persistent scheduling enhancements for 5g xr services. World Intellectual Property Organization (WIPO) Patent WO2024011365A1, January 2024. URL https://patents.google.com/patent/WO2024011365A1/en. Published January 18, 2024.

[35] Qing He, György Dán, and Georgios P Koudouridis. Semi-persistent scheduling for 5g downlink based on short-term traffic prediction. In *GLOBECOM 2020-2020 IEEE global communications conference*, pages 1–6. IEEE, 2020.

[36] Yoga Suhas Kuruba Manjunath, Mathew Szymanowski, Austin Wissborn, Mushu Li, Lian Zhao, and Xiao-Ping Zhang. Reslearn: Transformer-based residual learning for metaverse network traffic prediction. *arXiv e-prints*, pages arXiv–2411, 2024.

[37] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Frequency domain feature based robust malicious traffic detection. *IEEE/ACM Transactions on Networking*, 31 (1):452–467, 2022.

[38] Jinjia Ruan and Dongliang Xie. Networked vr: State of the art, solutions, and challenges. *Electronics*, 10(2):166, 2021.

[39] Fenghe Hu, Yansha Deng, Walid Saad, Mehdi Bennis, and A Hamid Aghvami. Cellular-connected wireless virtual reality: Requirements, challenges, and solutions. *arXiv preprint arXiv:2001.06287*, 2020.

[40] Matthew Warburton, Mark Mon-Williams, Faisal Mushtaq, and J Ryan Morehead. Measuring motion-to-photon latency for sensorimotor experiments with virtual reality systems. *Behavior research methods*, 55(7):3658–3678, 2023.

[41] International Telecommunication Union – Telecommunication Standardization Sector (ITU-T). Functional requirements of end-to-end network platforms to enhance the delivery of cloud-vr services over integrated broadband cable networks. Recommendation J.1631, International Telecommunication Union – Telecommunication Standardization Sector, Geneva, November 2021. URL https://handle.itu.int/11.1002/1000/14648. Approved 24 November 2021.

[42] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–80, 2018.

[43] 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 Communications*, 226:107919, 2024.

[44] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. Latency and cybersickness: Impact, causes, and measures. a review. *Frontiers in Virtual Reality*, 1:582204, 2020.

[45] Seyedmohammad Salehi, Abdullah Alnajim, Xiaoqing Zhu, Malcolm Smith, Chien-Chung Shen, and Leonard Cimini. Traffic characteristics of virtual reality over edge-enabled wi-fi networks. *arXiv preprint arXiv:2011.09035*, 2020.

[46] Mattia Lecci, Federico Chiariotti, Matteo Drago, Andrea Zanella, and Michele Zorzi. Temporal characterization of xr traffic with application to predictive network slicing. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 406–415. IEEE, 2022.

[47] Aditya Dhakal, Xukan Ran, Yunshu Wang, Jiasi Chen, and K. K. Ramakrishnan. Slam-share: Visual simultaneous localization and mapping for real-time multi-user augmented reality. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22)*, pages 293–306, Rome, Italy, December 2022. Association for Computing Machinery. ISBN 978-1-4503-9508-3. doi: 10.1145/3555050.3569142. URL https://doi.org/10.1145/3555050.3569142.

[48] Ali J. Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. Edge-slam: Edge-assisted visual simultaneous localization and mapping. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys '20)*, pages 325–337, Toronto, Ontario, Canada, 2020. Association for Computing Machinery. ISBN 978-1-4503-7954-0. doi: 10.1145/3386901.3389033. URL https://doi.org/10.1145/3386901.3389033.

[49] *ARCore Client (Android) — NVIDIA CloudXR SDK Documentation*. NVIDIA, 2025. URL https://docs.nvidia.com/cloudxr-sdk/usr_guide/cxr_arcore_client.html. Describes AR streaming: client sends pose (motion) data to the server; server renders frames and streams them to the client.

[50] Yihang Zhang, Zhidong Jia, Li Jiang, Qingyang Li, Xinggong Zhang, and Zongming Guo. Modeling virtual reality traffic with head movement in remote rendering. *Meta*, 3(2), 2023.

[51] 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 Communications*, 226:107919, 2024.

[52] Thuy T. T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008. doi: 10.1109/SURV.2008.080406.

[53] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 50–60, New York, NY, USA, 2005. ACM. doi: 10.1145/1064212.1064220.

[54] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994. doi: 10.1109/90.282603.

[55] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[56] Federico Chiariotti. A survey on 360-degree video: Coding, quality of experience and streaming. *Computer Communications*, 177:133–155, 2021.

[57] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic. In *Conference proceedings on Communications architectures, protocols and applications*, pages 183–193, 1993.

[58] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on networking*, 3(3):226–244, 1995.

[59] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[60] Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

[61] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[62] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[63] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Pearson Education India, 1999.

[64] Peter D. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 2003.

[65] Xiang Chen, Joo Yeo, Gyu Lee, and Anthony Lau. Power spectrum entropy based detection and mitigation of low-rate dos attacks. *Computer Networks*, 132: 24–36, 2018.

[66] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 2005.

[67] L. Cohen. *Time-Frequency Analysis*. Prentice Hall, Englewood Cliffs, 1995.

[68] J. B. Allen and L. R. Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 2005.

[69] Boualem Boashash. *Time-Frequency Signal Analysis and Processing: A Comprehensive Reference*. Academic Press, 2015.

[70] Stephane G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 2002.

[71] David Anick, Debasis Mitra, and Man Mohan Sondhi. Stochastic theory of a data-handling system with multiple sources. *Bell System Technical Journal*, 61 (8):1871–1894, 1982.

[72] Wolfgang Fischer and Klaus Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Performance Evaluation*, 18(2):149–171, 1993.

[73] Ilkka Norros. A storage model with self-similar input. *Queueing Systems*, 16(3): 387–396, 1994.

[74] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[75] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[76] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[77] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[79] Bharat Agarwal, Mohammed Amine Togou, Marco Marco, and Gabriel-Miro Muntean. A comprehensive survey on radio resource management in 5g hetnets: Current solutions, future trends and open issues. *IEEE Communications Surveys & Tutorials*, 24(4):2495–2534, 2022.

[80] Sumanta Saha and Rumana Quazi. Priority-coupling-a semi-persistent mac scheduling scheme for voip traffic on 3g lte. In *2009 10th International Conference on Telecommunications*, pages 325–329. IEEE, 2009.

[81] Dennis Overbeck, Niklas A Wagner, Fabian Kurtz, and Christian Wietfeld. Proactive resource management for predictive 5g uplink slicing. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 1000–1005. IEEE, 2022.

[82] Habtegebreil Haile, Karl-Johan Grinnemo, Simone Ferlin, Per Hurtig, and Anna Brunstrom. End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks. *Computer Networks*, 186:107692, 2021.

[83] Jian Su, Huimin Cai, Zhengguo Sheng, AX Liu, and Abdullah Baz. Traffic prediction for 5g: A deep learning approach based on lightweight hybrid attention networks. *Digital Signal Processing*, 146:104359, 2024.

[84] Filip Lemic, Jakob Struye, and Jeroen Famaey. Short-term trajectory prediction for full-immersive multiuser virtual reality with redirected walking. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 6139–6145. IEEE, 2022.

[85] Marcin Bosk, Marija Gajić, Susanna Schwarzmann, Stanislav Lange, Riccardo Trivisonno, Clarissa Marquezan, and Thomas Zinner. Using 5g qos mechanisms to achieve qoe-aware resource allocation. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 283–291. IEEE, 2021.

[86] Zhilbert Tafa and Veljko Milutinovic. Machine learning in congestion control: A survey on selected algorithms and a new roadmap to their implementation. *arXiv preprint arXiv:2112.15522*, 2021.

[87] Federico Chiariotti, Matteo Drago, Paolo Testolina, Mattia Lecci, Andrea Zanella, and Michele Zorzi. Temporal characterization of vr traffic for network slicing requirement definition. *arXiv preprint arXiv:2206.00317*, 2022.

[88] Seyedmohammad Salehi, Abdullah Alnajim, Xiaoqing Zhu, Malcolm Smith, Chien-Chung Shen, and Leonard Cimini. Traffic characteristics of virtual reality over edge-enabled wi-fi networks. *arXiv preprint arXiv:2011.09035*, 2020.

[89] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114, 2018.

[90] Quentin Guimard, Lucile Sassatelli, Francesco Marchetti, Federico Becattini, Lorenzo Seidenari, and Alberto Del Bimbo. Deep variational learning for 360° adaptive streaming. *ACM Transactions on Multimedia Computing, Communications and Applications*, 20(9):1–25, 2024.

[91] Majed Al Zayer, Paul MacNeilage, and Eelke Folmer. Virtual locomotion: a survey. *IEEE transactions on visualization and computer graphics*, 26(6):2315–2334, 2018.

[92] Aniruddha Prithul, Isayas Berhe Adhanom, and Eelke Folmer. Teleportation in virtual reality; a mini-review. *Frontiers in Virtual Reality*, 2:730792, 2021.

[93] Changhyun Lee, Yunho Jung, Seongjoo Lee, Yunje Oh, and Jaeseok Kim. Cost-effective scene change detection algorithm for real-time h. 264 rate control. *Optical Engineering*, 47(3):030501–030501, 2008.

[94] Sung Min Kim, Ju Wan Byun, and Chee Sun Won. A scene change detection in H.264/AVC compression domain. In Yo-Sung Ho and Hyoung-Joong Kim, editors, *Advances in Multimedia Information Processing – PCM 2005*, volume 3768 of *Lecture Notes in Computer Science*, pages 1072–1082, Berlin, Heidelberg, 2005. Springer. doi: 10.1007/11582267_93.

[95] Mark Claypool. Motion and scene complexity for streaming video games. In *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09)*, pages 34–41, Orlando, Florida, USA, April 2009. ACM. ISBN 978-1-60558-437-9. doi: 10.1145/1536513.1536529.

[96] Juzheng Duan, Min Zhang, Jing Wang, Shuai Han, Xun Chen, and Xiaolong Yang. VCC-DASH: A video content complexity-aware DASH bitrate adaptation strategy. *Electronics*, 9(2):230, 2020. ISSN 2079-9292. doi: 10.3390/electronics9020230.

[97] Alberto Dainotti, Antonio Pescapé, Pierluigi Salvo Rossi, Francesco Palmieri, and Giorgio Ventre. Internet traffic modeling by means of hidden markov models. *Computer Networks*, 52(14):2645–2662, 2008.

[98] Luca Muscariello, Marco Mellia, Michela Meo, M Ajmone Marsan, and R Lo Cigno. Markov models of internet traffic and a new hierarchical mmpp model. *Computer communications*, 28(16):1835–1851, 2005.

[99] Alberto Dainotti, Walter De Donato, Antonio Pescape, and Pierluigi Salvo Rossi. Classification of network traffic via packet-level hidden markov models. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*, pages 1–5. IEEE, 2008.

[100] S Duraimurugan, R Avudaiammal, and PM Durai Raj Vincent. Enhanced qos through optimized architecture for video streaming applications in heterogeneous networks. *Wireless Personal Communications*, 118(2):1655–1673, 2021.

[101] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2009.

[102] Ahmad Azab, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*, 10(3):676–692, 2024.

[103] AI Getman and MK Ikonnikova. A survey of network traffic classification methods using machine learning. *Programming and Computer Software*, 48(7):413–423, 2022.

[104] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of deep packet inspection (dpi) tools for traffic classification. 2013.

[105] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[106] Ruchira Purohit, Satish Kumar, Sameer Sayyad, and Ketan Kotecha. Time-frequency analysis and autoencoder approach for network traffic anomaly detection. *MethodsX*, 14:103228, 2025.

[107] Xuan Kong, Congcong Wang, Yanmiao Li, Jiangang Hou, Tongqing Jiang, and Zhi Liu. Traffic classification based on cnn-lstm hybrid network. In *International Forum on Digital TV and Wireless Multimedia Communications*, pages 401–411. Springer, 2021.

[108] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[109] Niloofar Bayat, Weston Jackson, and Derrick Liu. Deep learning for network traffic classification. *arXiv preprint arXiv:2106.12693*, 2021.

[110] Wei Quan. *Emerging Networking Architecture and Technologies: First International Conference, ICENAT 2022, Shenzhen, China, November 15–17, 2022, Proceedings*. Springer Nature, 2023.

[111] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[112] Irina Kochetkova, Anna Kushchazli, Sofia Burtseva, and Andrey Gorshenin. Short-term mobile network traffic forecasting using seasonal arima and holt-winters models. *Future Internet*, 15(9):290, 2023.

[113] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking*, 2(1):1–15, 2002.

[114] Gabriel O Ferreira, Chiara Ravazzi, Fabrizio Dabbene, Giuseppe C Calafiore, and Marco Fiore. Forecasting network traffic: A survey and tutorial with open-source comparative evaluation. *IEEE Access*, 11:6018–6044, 2023.

[115] Mayukh Biswas, Ayan Chakraborty, and Basabdatta Palit. A kalman filter based low complexity throughput prediction algorithm for 5g cellular networks. *IEEE Transactions on Vehicular Technology*, 73(5):7089–7101, 2023.

[116] Yongtao Wei, Jinkuan Wang, and Cuirong Wang. Network traffic prediction based on wavelet transform and season arima model. In *International Symposium on Neural Networks*, pages 152–159. Springer, 2011.

[117] Matthew Roughan and Darryl Veitch. Measuring long-range dependence under changing traffic conditions. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 3, pages 1513–1521. IEEE, 1999.

[118] Dalal Aloraifan, Imtiaz Ahmad, and Ebrahim Alrashed. Deep learning based network traffic matrix prediction. *International Journal of Intelligent Networks*, 2:46–56, 2021.

[119] Weiwei Jiang. Internet traffic matrix prediction with convolutional lstm neural network. *Internet Technology Letters*, 5(2):e322, 2022.

[120] Phi Le Nguyen, Yusheng Ji, et al. Deep convolutional lstm network-based traffic matrix prediction with partial information. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 261–269. IEEE, 2019.

[121] Min Tan, Ruixuan Ba, and Guohui Li. Data center traffic prediction algorithms and resource scheduling. *Sensors*, 22(20):7893, 2022.

[122] Yi Wang and Peiyuan Chen. Network traffic prediction based on transformer and temporal convolutional network. *PloS one*, 20(4):e0320368, 2025.

[123] André Felipe Zanella, Antonio Bazco-Nogueras, Cezary Ziemlicki, and Marco Fiore. Characterizing and modeling session-level mobile traffic demands from large-scale measurements. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 696–709, 2023.

[124] Xinwei Chen, Ali Taleb Zadeh Kasgari, and Walid Saad. Deep learning for content-based personalized viewport prediction of 360-degree vr videos. *IEEE Networking Letters*, 2(2):81–84, 2020.

[125] Patrice Abry and Darryl Veitch. Wavelet analysis of long-range-dependent traffic. *IEEE transactions on information theory*, 44(1):2–15, 2002.

[126] Wei Lu and Ali A Ghorbani. Network anomaly detection based on wavelet analysis. *EURASIP Journal on Advances in Signal processing*, 2009(1):837601, 2008.

[127] Chin-Tser Huang, Sachin Thareja, and Yong-June Shin. Wavelet-based real time detection of network traffic anomalies. In *2006 securecomm and workshops*, pages 1–7. IEEE, 2006.

[128] Mackenzie Haffey, Martin Arlitt, and Carey Williamson. Modeling, analysis, and characterization of periodic traffic on a campus edge network. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 170–182. IEEE, 2018.

[129] Matthew Price-Williams, Nick Heard, and Melissa Turcotte. Detecting periodic subsequences in cyber security data. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 84–90. IEEE, 2017.

[130] Fanchao Meng, Jiaping Gui, Futai Zou, Yunbo Li, and Yue Wu. Distr: Detecting multi-stage iot botnets through contextual traffic and causal analytics. *Computers & Security*, page 104531, 2025.

[131] Molham Alsakati, Charlie Pettersson, Sebastian Max, Vishnu Narayanan Moothedath, and James Gross. Performance of 802.11 be wi-fi 7 with multi-link operation on ar applications. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2023.

[132] Muhammad Shahid Anwar, Ahyoung Choi, Sadique Ahmad, Khursheed Aurangzeb, Asif Ali Laghari, Thippa Reddy Gadekallu, and Andrew Hines. A moving metaverse: Qoe challenges and standards requirements for immersive media consumption in autonomous vehicles. *Applied Soft Computing*, 159:111577, 2024.

[133] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[134] Remote operation of vehicles with 5g. URL https://www.ericsson.com/en/reports-and-papers/mobility-report/articles/remote-monitoring-and-control-of-vehicles. Reports system response time and network RTT mostly under 50 ms.

[135] Playout delay: Rtp header extension to control playout delay. URL https://webrtc.github.io/webrtc-org/experiments/rtp-hdrext/playout-delay/.

[136] Iason P. Achieving <100 ms latency for remote control with webrtc. URL https://gethopp.app/blog/latency-exploration. Industry blog; demonstrates sub-100 ms end-to-end latency with WebRTC.

[137] Alexander Novotny, Rowan Gudmundsson, and Frederick C Harris. A unity framework for multi-player vr applications. *International Journal of Computers and Their Applications*, 27(3):115–121, 2020.

[138] 3rd Generation Partnership Project (3GPP). System architecture for the 5G System (5GS). Technical Specification 3GPP TS 23.501, 3GPP, July 2025. URL https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/18.10.00_60/ts_123501v181000p.pdf. Release 18.

[139] O-RAN Alliance. O-RAN Architecture Description. Publicly Available Specification (PAS) O-RAN.WG1.OAD-R003-v08.00, O-RAN Alliance / ETSI, January 2024. URL https://www.etsi.org/deliver/etsi_ts/103900_103999/103982/08.00.00_60/ts_103982v080000p.pdf.

[140] 3rd Generation Partnership Project (3GPP). NR; Physical layer procedures for data. Technical Specification 3GPP TS 38.214, 3GPP, July 2025. URL https://www.etsi.org/deliver/etsi_ts/138200_138299/138214/18.07.00_60/ts_138214v180700p.pdf. Release 18.

[141] 3rd Generation Partnership Project (3GPP). 5G System Enhancements for Edge Computing; Stage 2. Technical Specification 3GPP TS 23.548, 3GPP, January 2025. URL https://www.etsi.org/deliver/etsi_ts/123500_123599/123548/18.08.00_60/ts_123548v180800p.pdf. Release 18.

[142] 3rd Generation Partnership Project (3GPP). Interface between the Control Plane and the User Plane nodes (PFCP); Stage 3. Technical Specification 3GPP TS 29.244, 3GPP, January 2021. URL https://cdn.standards.iteh.ai/samples/62064/edd5cef388f64c00a0e274c4802d89b2/ETSI-TS-129-244-V16-6-0-2021-01-.pdf. Release 16.

[143] 3rd Generation Partnership Project (3GPP). Procedures for the 5G System (5GS). Technical Specification 3GPP TS 23.502, 3GPP, January 2024. URL https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/17.11.00_60/ts_123502v171100p.pdf. Release 17.

[144] 3rd Generation Partnership Project (3GPP). Policy and charging control framework for the 5G System (5GS); Stage 2. Technical Specification 3GPP TS 23.503, 3GPP, July 2025. URL https://www.etsi.org/deliver/etsi_ts/123500_123599/123503/18.10.00_60/ts_123503v181000p.pdf. Release 18.

[145] A. Jalali, R. Padovani, and R. Pankaj. Data throughput of CDMA-HDR: A high efficiency–high data rate personal communication wireless system. In *Proc. IEEE VTC 2000-Spring*, volume 3, pages 1854–1858, Tokyo, Japan, 2000.

[146] Matthew Andrews, Krishnan Kumaran, Kavita Ramanan, Alexander Stolyar, Rajiv Vijayakumar, and Phil Whiting. Providing quality of service over a shared wireless link. *IEEE Communications Magazine*, 39(2):150–154, 2001. doi: 10.1109/35.900644.

[147] Jong-Hun Rhee and Kihong Kim. Scheduling of real/non-real time services in an AMC/TDM system: EXP/PF algorithm. In *Advances in Computer Science –*

ASIAN 2003, volume 2896 of *Lecture Notes in Computer Science*, pages 506–513. Springer, 2003. doi: 10.1007/3-540-36555-9_52.

[148] 3rd Generation Partnership Project (3GPP). NR; Medium Access Control (MAC) protocol specification. Technical Specification 3GPP TS 38.321, 3GPP, July 2025. URL https://www.etsi.org/deliver/etsi_ts/138300_138399/138321/18.06.00_60/ts_138321v180600p.pdf. Release 18.

[149] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley, 1975. ISBN 978-0471491101.

[150] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013. ISBN 978-1107027503. doi: 10.1017/CBO9781139017329.

[151] Jinjia Ruan and Dongliang Xie. A survey on qoe-oriented vr video streaming: Some research issues and challenges. *Electronics*, 10(17):2155, 2021. doi: 10.3390/electronics10172155. URL https://doi.org/10.3390/electronics10172155.

Appendix A

Details of the Custom-built Virtual Reality Application

This appendix delineates the technical specifications and architectural design of the custom-built Virtual Reality (VR) application, developed using Unity (version 2022.3.5.1f1), which served as the primary instrument for generating the network traffic dataset. A comprehensive understanding of the application's underlying mechanisms, particularly its networking logic and user interaction model, is crucial for accurately interpreting the observed traffic patterns and validating the findings presented in this dissertation.

A.1 Unity Project Structure and Core Components

The application was meticulously developed within the Unity game engine environment, leveraging several key packages and SDKs that are indispensable for VR and robust multiplayer functionalities. The selection and integration of these components directly influenced the traffic generation characteristics observed during data collection.

1. **Meta XR All-in-One SDK** provided foundational prefabs and utilities for rapid VR environment setup, encompassing critical elements such as the `Camera Rig`, essential for defining the user's viewpoint in the virtual space; `Hand Tracking` and `Controller Tracking`, which enabled precise capture of user hand and controller movements; and `Network Avatar`, facilitating the visual representation of users in the shared virtual environment. These components were central to capturing immersive user interactions and their corresponding kinematic data, which are subsequently translated into network traffic.

2. **Unity Netcode for GameObjects** constituted the high-level networking library that formed the architectural backbone of the multiplayer environment. This robust framework enabled seamless server-to-client and client-to-server communication, facilitated the synchronization of game objects across connected clients, and supported the execution of remote procedure calls (RPCs). The chosen authoritative server model inherently dictated the flow of information, where the server maintained the singular, authoritative global game state, while clients submitted input and received validated state updates. This server-centric design is fundamental to understanding the traffic asymmetry and message patterns observed.

3. **Lobby and Relay Packages**, while initially included for simplified matchmaking and indirect client-host connections via Unity's relay services, were subsequently configured or bypassed for direct IP communication in the controlled experimental setup. This modification was implemented to eliminate any potential variability or latency introduced by third-party relay services, ensuring that the captured network traffic reflected direct client-server interactions within the local network environment, as further elaborated in Section A.3.

A.2 Application-Level Network Logic and Message Types

The custom VR game environment, featuring distinct "island" zones and connecting "paths" (conceptually illustrated in Figure 1 of the main text), was specifically designed to elicit and differentiate between stationary and moving user behaviors. The application's inherent network logic directly governs the generation and characteristics of the traffic patterns observed in these distinct states.

1. **Fixed World Tick Rate.** The server component of the application maintained a consistent world update rate, specifically operating at a fixed 30 Hz tick. This implies that server-driven state synchronizations and non-player character (NPC) updates were consistently attempted 30 times per second, irrespective of the dynamic activity of individual clients. This constant server-side heartbeat contributes a baseline traffic component.

2. **Dynamic Client-to-Server Updates.** Client-originated traffic, primarily comprising `NetworkTransformMessage` for pose updates and `SpawnBall` requests for interactive object instantiation, was directly driven by user actions.

    - **`NetworkTransformMessage`** synchronizes the player's real-time pose. Its structure includes `NetworkObjectId`, `NetworkBehaviourId`, and a `State` variable containing kinematic fields `PositionX, PositionY, PositionZ`, `RotAngleX, RotAngleY, RotZ`, and `ScaleX, ScaleY, ScaleZ`. The C# `Serialize` method in `NetworkTransformMessage.cs` packs these data into the payload, affecting packet size.
    - **`SpawnBall` mechanism** exemplifies interactive, bursty traffic. The `SpawnBall.cs` script (see Listing A.1) shows how a client input triggers a `ServerRpc` to the authoritative server to instantiate a network object, ensuring consistent state across clients.

3. **User Mobility Impact.** The observed distinct traffic volume dynamics between stationary and moving states, which are thoroughly characterized in Section 3.1 of the main text, are a direct consequence of this underlying network logic. For example, the Meta Quest SDK's "pose-on-change" strategy dictates that client-to-server `NetworkTransformMessage`s are transmitted only when angular or translational velocity exceeds a predefined threshold. This mechanism naturally leads to a significantly lower update rate and thus lower client-to-server traffic when the user is stationary. Conversely, while stationary, the application's engine may widen the area-of-interest, triggering more frequent server-to-client updates for

high-fidelity assets (e.g., NPC idle animations, particle effects, UI prompts). This results in a higher number of smaller `ObjectUpdate` or `AssetChunk` packets being sent from the server, explaining the increased downlink volume during stationary periods, as detailed in Section 3.1.

A.3 Network Configuration and Deployment

The experimental setup was carefully configured to ensure precise control over network conditions and accurate traffic capture. This involved specific configurations for network connectivity and application deployment.

1. **Direct IP Connection.** To eliminate the potential for variability or additional latency introduced by Unity's default relay services, the `Auto Match Making` building block was explicitly removed from the Unity project. Instead, a custom C# script, `NetworkConnect.cs` (provided in Listing A.2), was implemented. This script dynamically determines whether the current Unity instance should launch as a network server or a client based on the active Unity Build Settings (e.g., targeting "Server" vs. "PC" platform). It then initiates a direct network connection using pre-configured IP addresses (e.g., localhost `127.0.0.1` for local testing or a private IP for local network setups). This direct connection approach ensures that the captured network traffic accurately reflects the raw application-layer exchanges without intermediate relaying overhead.

2. **Headset and Server Deployment.** For generating the real-world traffic traces, the client-side VR application was deployed onto a Meta Quest 3 headset. This required configuring Unity's Build Settings to target the Android platform and activating developer mode on the headset. The server application, which manages the authoritative game state and network connections, ran on a dedicated Linux machine. The Unity-built Linux executable (`.x86_64` file) was launched via command line, ensuring a stable and controlled server environment throughout the data collection sessions.

This detailed overview of the VR application and its deployment provides essential context for understanding the genesis of the network traffic patterns that are analyzed and predicted throughout this dissertation.

A.4 Custom C# Scripts for VR Application Functionality

This section includes the C# source code for the custom scripts developed within the Unity VR application, which are directly relevant to its networking behavior and interaction mechanisms.

```
1
2 using Unity.Netcode;
3 using UnityEngine;
4 using UnityEngine.UI;
5 using System.Collections.Generic;
```

```
6
7  public class SpawnBall: NetworkBehaviour
8  {
9      public GameObject prefab;
10     public float spawnSpeed = 10f;
11     public float ballLifetime = 10f; // Lifetime of the ball in
           seconds
12     public float cooldownDuration = 2f; // Cooldown duration in
           seconds
13
14     private Dictionary<ulong, float> playerCooldowns = new
           Dictionary<ulong, float>();
15     public Text cooldownText; // Reference to the UI Text component
16
17     void Start()
18     {
19         // Ensure the cooldown Text is not null
20         if (cooldownText == null)
21         {
22             Debug.LogError("Cooldown␣Text␣is␣not␣assigned.");
23         }
24     }
25
26     void Update()
27     {
28         if (OVRInput.GetDown(OVRInput.Button.SecondaryIndexTrigger)
               )
29         {
30             Debug.Log("Trigger␣pressed,␣attempting␣to␣spawn␣ball.")
                   ;
31             AttemptToSpawnBall();
32         }
33
34         // Update the cooldown display
35         UpdateCooldownDisplay();
36     }
37
38     void AttemptToSpawnBall()
39     {
40         ulong playerId = NetworkManager.Singleton.LocalClientId;
41
42         // Check if the player is on cooldown
43         if (playerCooldowns.ContainsKey(playerId) && Time.time <
               playerCooldowns[playerId])
44         {
45             Debug.Log("Player␣is␣on␣cooldown.");
46             return;
47         }
48
49         // Update the player's cooldown
50         playerCooldowns[playerId] = Time.time + cooldownDuration;
```

```
51
52          // Call the server RPC to spawn the ball
53          SpawnBallServerRpc(transform.position, transform.forward *
                spawnSpeed);
54      }
55
56      void UpdateCooldownDisplay()
57      {
58          ulong playerId = NetworkManager.Singleton.LocalClientId;
59
60          if (playerCooldowns.ContainsKey(playerId))
61          {
62              float timeLeft = playerCooldowns[playerId] - Time.time;
63              if (timeLeft > 0)
64              {
65                  cooldownText.text = $"Cooldown: {timeLeft:F1}s";
66              }
67              else
68              {
69                  cooldownText.text = "Ready";
70              }
71          }
72          else
73          {
74              cooldownText.text = "Ready";
75          }
76      }
77
78      [ServerRpc(RequireOwnership = false)]
79      void SpawnBallServerRpc(Vector3 position, Vector3 velocity,
            ServerRpcParams rpcParams = default)
80      {
81          Debug.Log($"SpawnBallServerRpc called with position: {
                position}, velocity: {velocity}");
82
83          if (prefab == null)
84          {
85              Debug.LogError("Prefab is not set.");
86              return;
87          }
88
89          GameObject spawnedBall = Instantiate(prefab, position,
                Quaternion.identity);
90          NetworkObject networkObject = spawnedBall.GetComponent<
                NetworkObject>();
91
92          if (networkObject == null)
93          {
94              Debug.LogError("The prefab does not have a
                    NetworkObject component.");
95              Destroy(spawnedBall);
```

```
96              return;
97          }
98
99          networkObject.Spawn(true); // Spawn the network object on
                all clients
100
101         Rigidbody rb = spawnedBall.GetComponent<Rigidbody>();
102         if (rb == null)
103         {
104             Debug.LogError("The prefab does not have a Rigidbody
                    component.");
105             Destroy(spawnedBall);
106             return;
107         }
108
109         rb.velocity = velocity; // Set the initial velocity
110         rb.collisionDetectionMode = CollisionDetectionMode.
                Continuous; // Use continuous collision detection mode
111
112         Destroy(spawnedBall, ballLifetime); // Destroy the ball
                after its lifetime
113     }
114 }
```

Listing A.1: C# Script for Spawning Network Objects (SpawnBall.cs)

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Unity.Netcode;
5
6  public class NetworkConnect: MonoBehaviour
7  {
8      private void Start()
9      {
10         var singleton = NetworkManager.Singleton;
11
12 #if UNITY_SERVER
13         singleton.StartServer();
14         Debug.Log("Server started.");
15 #else
16         singleton.StartClient();
17         Debug.Log("Client started.");
18 #endif
19     }
20 }
```

Listing A.2: C# Script for Network Connection Management (NetworkConnect.cs)

Appendix B

Sample Code for Data Processing and Feature Extraction

This appendix provides illustrative Python code snippets from the data preprocessing and feature engineering pipeline. These scripts are crucial for ensuring the reproducibility of the dataset generation and for the robust extraction of both time-domain and frequency-domain features utilized in the subsequent classification and regression tasks. All scripts were developed in Python 3.9 and primarily leverage standard scientific libraries such as NumPy, Pandas, and SciPy.

B.1 Raw Data Filtering and Application Log Parsing

The initial stage of data preparation involved cleaning and structuring raw network captures and application logs into a unified, machine-readable format.

1. **Wireshark JSON Filtering** (`1-filtering.py`): This script processes raw network packet captures, typically in `.pcap` format, which were first converted to JSON using TShark. Its primary function is to extract only the most pertinent packet-level information, thereby reducing data volume and focusing on relevant features. The key fields retained from each packet's JSON representation include the high-resolution arrival timestamp (`frame.time`), the UDP payload size (`data.len`), and the raw payload content (`data.data`). The filtering logic, exemplified by the `filter_dict` function, ensures that timestamps are uniformly parsed and other specified keys are extracted.

```
import json
import re

# Dictionary defining the keys we want to keep from each layer
keys_to_keep = {
    "frame": [
        "frame.time"
    ],
    "data": [
        "data.len",
        "data.data"
    ]
}
```

```
14
15 # Function to filter the dictionary ‘d‘ and keep only the
      specified ‘keys‘
16 def filter_dict(d, keys):
17     filtered = {}
18     for k in keys:
19         if k in d:
20             if k == "frame.time":
21                 # Extracting the time in the format HH:MM:SS.
                      ssssss using regex
22                 match = re.search(r'\d{2}:\d{2}:\d{2}\.\d+', d
                      [k])
23                 if match:
24                     filtered[k] = match.group(0)
25             else:
26                 filtered[k] = d[k]
27     return filtered
28
29 # Open the input JSON file containing packet data
30 with open('./data/client_to_server.json', 'r') as f:
31     packets = json.load(f)
32
33 # Open the output file to write the filtered data
34 with open('filtered_client_to_server.json', 'w') as f:
35     f.write('[\n')
36
37     # Process each packet in the input file
38     for idx, packet in enumerate(packets):
39         filtered_packet = {}
40         for layer, keys in keys_to_keep.items():
41             if layer in packet['_source']['layers']:
42                 filtered_packet[layer] = filter_dict(packet['
                      _source']['layers'][layer], keys)
43
44         # Add a comma and newline before writing the next
              packet, except for the first one
45         if idx > 0:
46             f.write(',\n')
47         json.dump(filtered_packet, f, indent=2)
48
49     f.write('\n]')
50
51 print("File has been created : 'filtered_move_packet.json'.")
```

Listing B.1: Python script for filtering Wireshark JSON data (`1-filtering.py`).

2. **Unity Log to JSON Conversion** (`2-TxtToJson-client.py`, `2-TxtToJson-server.py`): These two specialized Python scripts are responsible for parsing the application-level console logs generated directly by the Unity VR application. They extract structured metadata such as the Unity-generated timestamp, the message type (`TypeUnity`), and the relevant game object identifier (`IDobject`), along with the

serialized message content (`SerializeData`).

- For client-side logs (`2-TxtToJson-client.py`), a crucial aspect is the inclusion of a `compare_packets` function. This function performs a heuristic analysis by comparing the kinematic data (position and Euler angles) of the current `NetworkTransformMessage` with the previous one. Based on observed changes in these values, a `SpecialLabel` (e.g., "positive X motion", "negative Y rotation", or "stationary") is assigned. This initial labeling provides a preliminary indication of user activity, which was then refined through manual annotation for the final dataset.
- The server-side script (`2-TxtToJson-server.py`) includes an additional filtering step by a specific `IDobject`. This is necessary because the Unity server's logs often contain multiple entries for synchronized objects within what corresponds to a single network packet transmission. This filtering ensures that only unique and representative log entries for specific entities (e.g., the player's avatar) are processed, avoiding data redundancy.

```
import re
import json

def parse_line(line):
    # Extract the time at the beginning of the line
    time_pattern = re.compile(r"(\d{2}:\d{2}:\d{2}\.\d{3})")
    time_match = time_pattern.match(line)

    if time_match:
        time = time_match.group(1)
    else:
        time = "00:00:00.000"  # Default value if no time is found

    # Extract key/value pairs after the time
    pattern = re.compile(r"/(\w+):([^/]+)")
    matches = pattern.findall(line)

    parsed_data = {key: value for key, value in matches}

    # If the key is 'SerializeData', remove the last character of the associated value
    if 'SerializeData' in parsed_data:
        parsed_data['SerializeData'] = parsed_data['SerializeData'][:-1]  # Remove the last character

    # Add the time to the dictionary with the key 'time'
    parsed_data['time'] = time

    return parsed_data

def compare_packets(prev_packet, curr_packet):
    special_labels = []

```

```
32      if 'position' in prev_packet and 'position' in curr_packet
           :
33          prev_position = tuple(map(float, prev_packet['position
               '].strip('()').split(',')))
34          curr_position = tuple(map(float, curr_packet['position
               '].strip('()').split(',')))
35
36          for i, (prev, curr) in enumerate(zip(prev_position,
               curr_position)):
37              if curr > prev:
38                  if i == 0:
39                      special_labels.append("positive X motion")
40                  elif i == 1:
41                      special_labels.append("positive Y motion")
42                  elif i == 2:
43                      special_labels.append("positive Z motion")
44              elif curr < prev:
45                  if i == 0:
46                      special_labels.append("negative X motion")
47                  elif i == 1:
48                      special_labels.append("negative Y motion")
49                  elif i == 2:
50                      special_labels.append("negative Z motion")
51
52      if 'euler' in prev_packet and 'euler' in curr_packet:
53          prev_euler = tuple(map(float, prev_packet['euler'].
               strip('()').split(',')))
54          curr_euler = tuple(map(float, curr_packet['euler'].
               strip('()').split(',')))
55
56          for i, (prev, curr) in enumerate(zip(prev_euler,
               curr_euler)):
57              if curr > prev:
58                  if i == 0:
59                      special_labels.append("positive X rotation
                           ")
60                  elif i == 1:
61                      special_labels.append("positive Y rotation
                           ")
62                  elif i == 2:
63                      special_labels.append("positive Z rotation
                           ")
64              elif curr < prev:
65                  if i == 0:
66                      special_labels.append("negative X rotation
                           ")
67                  elif i == 1:
68                      special_labels.append("negative Y rotation
                           ")
69                  elif i == 2:
70                      special_labels.append("negative Z rotation
```

```
                        ")
71
72      return special_labels
73
74  # List to store the data
75  data = []
76
77  # Dictionary to store the last packet for each IDobject with
        TypeUnity "Unity.Netcode.NetworkTransformMessage"
78  last_transform_packets = {}
79
80  # Read the text file
81  with open('./packet.txt', 'r') as file:
82      for line in file:
83          line = line.strip()
84          if line:
85              parsed_data = parse_line(line)
86              if parsed_data:
87                  if parsed_data.get("TypeUnity") == "Unity.
                        Netcode.NetworkTransformMessage":
88                      id_object = parsed_data.get("IDobject")
89                      if id_object:
90                          special_labels = []
91                          if id_object in last_transform_packets
                                :
92                              last_transform_packet =
                                    last_transform_packets[
                                    id_object]
93                              special_labels = compare_packets(
                                    last_transform_packet,
                                    parsed_data)
94                          if not special_labels:
95                              special_labels = ["stationary"]
96                          parsed_data['SpecialLabel'] =
                                special_labels
97                          # Update the last packet for this
                                IDobject
98                          last_transform_packets[id_object] =
                                parsed_data
99                  data.append(parsed_data)
100
101 # Save the data to a JSON file
102 with open('unity_client_to_server.json', 'w') as json_file:
103     json.dump(data, json_file, indent=4)
104
105 print("The data has been saved to 'data.json'.")
```

Listing B.2: Python script for processing Unity client logs into JSON, with motion labeling (`2-TxtToJson-client.py`).

```
1  import re
2  import json
```

```
3
4  # Function to analyse a line
5  def parse_line(line):
6      # Extract the hour
7      time_pattern = re.compile(r"(\d{2}:\d{2}:\d{2}\.\d{3})")
8      time_match = time_pattern.match(line)
9
10     if time_match:
11         time = time_match.group(1)
12     else:
13         time = "00:00:00.000"  # If no value add this by
               default
14
15     # Extract the other keys
16     pattern = re.compile(r"/(\w+):([^/]+)")
17     matches = pattern.findall(line)
18
19     parsed_data = {key: value for key, value in matches}
20
21     if 'SerializeData' in parsed_data:
22         parsed_data['SerializeData'] = parsed_data['
               SerializeData'][:-1]  # We do this because there is
                a : at the end that should not be there
23
24     # Add the time with its key
25     parsed_data['time'] = time
26
27     return parsed_data
28
29 # List to save the data
30 data = []
31
32 with open('./server_to_client_unity.txt', 'r') as file:
33     for line in file:
34         line = line.strip()
35         if line:
36             parsed_data = parse_line(line)
37             if parsed_data:
38                 # Filter by 'IDobject' the data
39                 if 'IDobject' in parsed_data:
40                     if parsed_data['IDobject'] == '4':
41                         data.append(parsed_data)
42                 else:
43                     data.append(parsed_data)
44
45
46 with open('./unity_server_to_client.json', 'w') as json_file:
47     json.dump(data, json_file, indent=4)
48
49 print("Data has been saved in 'data.json'.")
```

Listing B.3: Python script for processing Unity server logs into JSON, with object ID

filtering (`2-TxtToJson-server.py`).

B.2 Data Fusion and Time-Series Preparation

Following the initial parsing, the disparate datasets from Wireshark and Unity logs were meticulously merged and prepared for time-series analysis.

1. **Packet-Log Fusion** (`3-fusion-without-window.py`): This pivotal script facilitates the precise linkage between network-level packet characteristics (from Wireshark) and their corresponding application-level context (from Unity logs). The fusion process relies on identifying the serialized application payload (`SerializeData`) from Unity logs as a substring within the raw payload content (`data.data`) extracted from Wireshark packets. A hash table, mapping Wireshark packet payloads to their 'len' and 'time' attributes, is employed to optimize the lookup efficiency, ensuring that each application message is correctly associated with its network envelope. The script also reports unmerged data for quality control.

```
import json
import time

# Load JSON files
with open('./unity_server_to_client.json', 'r') as data_file:
    data = json.load(data_file)

with open('./filtered_server_to_client.json', 'r') as
   filtered_file:
    filtered = json.load(filtered_file)

# Build a hash table for the filtered data
hash_table = {
    packet['data']['data.data']: {
        'len': packet['data']['data.len'],
        'time': packet['frame']['frame.time']
    }
    for packet in filtered
}

# Initialize lists for merged and unmerged data
merged_data = []
unmerged_data = []

# Merging process with progress tracking
total_items = len(data)
start_time = time.time()

for index, item in enumerate(data):
    found = False
    for key in hash_table:
        if item['SerializeData'] in key:  # Check if the
           substring is present in the key
```

```
            item.update(hash_table[key])
            merged_data.append(item)
            found = True
            break
    if not found:
        unmerged_data.append(item)

    # Display speed and estimated remaining time
    if index % 1000 == 1:
        elapsed_time = time.time() - start_time
        items_processed = index + 1
        items_per_second = items_processed / elapsed_time
        remaining_items = total_items - items_processed
        estimated_time_remaining = remaining_items /
           items_per_second

        print(f"Progress: {items_processed}/{total_items}
           items processed.")
        print(f"Speed: {items_per_second:.2f} items/second")
        print(f"Estimated time remaining: {
           estimated_time_remaining:.2f} seconds")

# Save merged and unmerged data
with open('merged_data2.json', 'w') as merged_file:
    json.dump(merged_data, merged_file, indent=4)

with open('unmerged_data2.json', 'w') as unmerged_file:
    json.dump(unmerged_data, unmerged_file, indent=4)

print("Merging completed.")
```

Listing B.4: Python script for fusing Wireshark and Unity log data (`3-fusion-without-window.py`).

2. **Timestamp Standardization and Inter-arrival Time Calculation**: A series of sequential data preprocessing functions were applied to the merged datasets. The `ensure_datetime` function converts raw timestamp strings into a consistent 'datetime' object format and sorts the DataFrame chronologically, correcting for any minor logging or merging discrepancies. Subsequently, the `normalize_time_axis` function transforms these absolute timestamps into a zero-based relative time representation $T_i = t_i - \min(t)$, where t is the absolute timestamp of the i-th packet and $\min(t)$ is the earliest timestamp in the respective dataset. Finally, the `calculate_time_intervals` function computes the inter-arrival time $\Delta T_i = T_i - T_{i-1}$ for consecutive packets. This ΔT_i serves as a crucial time-domain feature, directly reflecting the observed packet frequency and burstiness, vital for both classification and prediction tasks.

Appendix C

AI Use Declaration

In accordance with Trinity College Dublin guidance on acknowledging and referencing the use of generative AI, I declare that no generative AI output has been used in this thesis as a source of original scholarly content. Generative AI tools were not used to generate the research questions, literature review, methodology, code, dataset, experimental results, analysis, citations, interpretation of findings, or thesis conclusions.

Standard software tools were used only for routine document preparation tasks such as spell checking, formatting, reference management, and LaTeX compilation. All research design choices, data collection, code implementation, experiments, analysis, interpretation of results, citations, corrections, and final thesis claims remain my own work and responsibility.

The public Trinity guidance consulted for this declaration is available at https://libguides.tcd.ie/gen-ai/acknowledging-referencing.