bb

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-590018

A

Technical Seminar Report

On

"Live Streaming"

SUBMITTED IN PARTIAL FULFILLMENT FOR THE AWARD OF DEGREE OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCEINCE AND ENGINEERING

SUBMITTED BY

ASHWIN K (1JB21CS018)

Under the Guidance of

Mrs. Vijayalakshmi B

Assistant Professor

Dept of CSE

SJBIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SJB INSTITUTE OF TECHNOLOGY

No.67, BGS Health & Education City, Dr.Vishnuvardhan Rd, Kengeri, Bengaluru, Karnataka 560060

An Autonomous Institute under VTU

Approved by AICTE - New Delhi, Accredited by NAAC A+, Accredited by NBA

2024 - 2025

|| Jai Sri Gurudev ||
Sri Adichunchanagiri Shikshana Trust ®

SJB INSTITUTE OF TECHNOLOGY

An Autonomous Institute under VTU
No.67, BGS Health & Education City, Dr.Vishnuvardhan Rd, Kengeri, Bengaluru, Karnataka 560060

Department of Computer Science and Engineering

SJBIT

CERTIFICATE

This is to certify that the seminar work entitled **“LIVE STREAMING”** is carried out by **ASHWIN K [1JB21CS018]** is bonafide student of **SJB Institute of Technology**, in partial fulfillment for the award of **"BACHELOR OF ENGINEERING"** in **COMPUTER SCIENCE AND ENGINEERING** as prescribed by **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the academic year **2024-2025**. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The Technical Seminar report has been approved as it satisfies the academic requirements in respect of seminar work prescribed for the said degree.

Signature of Guide	**Signature of HOD**
Mrs. Vijayalakshmi B	**Dr. Krishna A N**
Assistant Professor	**Professor and Head**
Dept. of CSE, SJBIT	**Dept. of CSE, SJBIT**

ACKNOWLEDGEMENT

I would like to express my profound grateful to His Divine Soul **Jagadguru Padmabhushan Sri Sri Sri Dr. Balagangadharanatha Mahaswamiji** and His Holiness **Jagadguru Sri Sri Sri Dr. Nirmalanandanatha Mahaswamiji** for providing us an opportunity to complete my academics in this esteemed institution.

I would also like to express my profound thanks to **Revered Sri Sri Dr. Prakashnath Swamiji**, Managing Director, BGS & SJB Group of Institutions, for his continuous support in providing amenities to carry out this Project Work in this admired institution.

I express my gratitude to **Dr. Puttaraju,** Academic Director, BGS & SJB Group of Institutions, for providing me an excellent facilitiy and academic ambience, which have helped me in satisfactory completion of Project work.

I express my gratitude to **Dr. K. V. Mahendra Prashanth**, Principal, SJB Institute of Technology, for providing me an excellent facilities and academic ambience; which have helped me in satisfactory completion of Technical Seminar.

I extend my heartfelt gratitude to all the Deans of SJB Institute of Technology for their unwavering support, cutting-edge facilities, and the inspiring academic environment, all of which played a pivotal role in the successful completion of my Technical Seminar.

I extend my sincere thanks to **Dr. Krishna A N,** Head of the Department, Computer Science and Engineering, for providing me an invaluable support throughout the period of our Technical Seminar.

I wish to express my heartfelt gratitude to my guide **Mrs. Vijayalakshmi B, Assistant Professor**, Department of CSE for her valuable guidance, suggestions and cheerful encouragement during the entire period of my Seminar.
I express my truthful thanks to **Mrs. Shubha T V**, Technical Seminar Coordinator, Department of Computer Science and Engineering, for her valuable support throughout our Seminar.
Finally, I take this opportunity to extend my earnest gratitude and respect to my parents, Teaching & Non teaching staffs of the department, the library staff and all my friends, who have directly or indirectly supported me during the period of my Technical Seminar.

Regards,

ASHWIN K [1JB21CS018]

ABSTRACT

Live streaming is a technology that enables the real-time transmission of multimedia content over the internet, allowing users to broadcast and consume audio-video data instantly. It supports continuous, uninterrupted delivery of content with minimal latency, making it ideal for scenarios where immediacy and engagement are essential. Live streaming plays a critical role in domains such as entertainment, online education, gaming, e-commerce, and corporate communication, offering immersive experiences and interactive participation.

The process typically involves capturing media through cameras, microphones, or screen recorders, encoding it into a digital format, and transmitting it via content delivery networks (CDNs) to end-users. Viewers can access the stream on various devices with adaptive bitrate streaming, ensuring consistent quality regardless of network conditions. Live streaming platforms often include features such as real-time chat, reaction buttons, and viewer analytics to enhance engagement and feedback loops during the broadcast.

To ensure seamless streaming, systems must handle challenges such as bandwidth fluctuations, latency control, synchronization of audio and video, and large-scale user access. Techniques like buffering, edge caching, and protocol optimization (e.g., RTMP, HLS, WebRTC) are employed to maintain stream quality and stability. Scalability is achieved through distributed architectures and cloud-based services, which allow content to be delivered concurrently to thousands or millions of users with minimal delay.

Despite its advancements, live streaming still contends with issues such as encoding delays, content delivery bottlenecks, and dependency on reliable internet infrastructure. Security and copyright protection are also key concerns, especially in public broadcasts. Nevertheless, with continuous innovation and robust platform support, live streaming remains a powerful tool for real-time content distribution, transforming how individuals and organizations connect and communicate globally.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview of Live Streaming

Live streaming is a real-time multimedia broadcasting technique that allows audio and video content to be transmitted over the internet with minimal delay. It enables instant access to live events such as sports, concerts, webinars, and news broadcasts, providing a dynamic and engaging experience for viewers. The technology is widely adopted in sectors like entertainment, online learning, telehealth, social media, and digital marketing due to its immediacy and ability to reach large audiences globally.

At its core, live streaming involves capturing raw media using devices like cameras, microphones, or screens, encoding the content into streamable formats, and distributing it via streaming protocols such as HLS, RTMP, or WebRTC. The content is delivered to users through content delivery networks (CDNs), which ensure low latency and high availability. Features like adaptive bitrate streaming and real-time interaction tools (e.g., live chat, polls) enhance the user experience and ensure smooth playback across varying network conditions.

Despite its efficiency, live streaming presents challenges such as bandwidth limitations, encoding delays, copyright enforcement, and dependency on robust internet infrastructure. Stream security and privacy are also key considerations, particularly in sensitive domains like healthcare and enterprise communication. Nonetheless, with continuous innovation and cloud-native support, live streaming remains a cornerstone of modern digital interaction, enabling seamless, real-time connectivity and global engagement.

1.2 The Emergence of Live Streaming in Real-Time Communication

The rise of live streaming was propelled by the increasing demand for real-time content delivery and interactive experiences in the digital era. As internet speeds improved and multimedia content consumption surged, users began expecting instant access to live events, video updates, and direct communication channels. Traditional broadcasting and on-demand video systems were not equipped to handle the immediacy and engagement required by modern users, creating a need for low-latency, scalable, and reliable live content delivery platforms.

Live streaming emerged as a transformative solution, enabling the real-time transmission of audio and video over the internet. Initially popularized through entertainment platforms and gaming services, it soon found widespread applications across industries including education, healthcare, social media, and enterprise collaboration.

Unlike static content, live streaming allowed dynamic interaction with viewers through chat, reactions, and real-time feedback, thereby enhancing user engagement and participation.

As demand grew, live streaming technologies evolved to support adaptive bitrate streaming, cloud-based encoding, and global content distribution networks (CDNs). This enabled platforms to deliver seamless streams to audiences of any size with minimal buffering and high reliability. Integration with modern protocols like WebRTC and HLS, as well as AI-driven enhancements such as real-time translation or content moderation, further expanded its capabilities. Whether used for virtual surgeries, remote learning, live auctions, or emergency response coordination, live streaming has become a critical pillar of modern digital communication.

1.3 The Growing Need for Live Streaming in Real-Time Communication

With the proliferation of high-speed internet, smartphones, social media platforms, and digital services, the consumption and demand for live content have grown exponentially. Traditional media and content delivery methods struggled to keep up with the need for instant, interactive, and engaging communication. Real-time connectivity became vital for scenarios like online education, virtual events, telemedicine, customer support, and live commerce. This surge in demand emphasized the necessity for a robust, low-latency, and scalable real-time broadcasting solution—leading to the widespread adoption of live streaming.

Live streaming technology enables continuous, high-quality delivery of audio and video by capturing, encoding, and distributing content instantly to viewers. Its architecture decouples content production from consumption, supports adaptive streaming, and leverages distributed delivery via CDNs for performance and reliability. The ability to broadcast to millions simultaneously while maintaining real-time interaction makes live streaming indispensable across diverse sectors—from e-learning platforms and e-sports to digital marketing and enterprise collaboration tools.

As user engagement, remote access, and real-time responsiveness become core business requirements—whether it's delivering live surgery tutorials, hosting interactive webinars, or launching products to a global audience—live streaming offers a powerful medium for connection.

Its integration with analytics tools, AI-based enhancements, and cloud infrastructure cements its role as a cornerstone in the future of digital communication and real-time engagement.

1.4 Key Advantages of Live Streaming

Live streaming provides a dynamic and efficient method for real-time content delivery, making it an essential tool in modern communication strategies. Its ability to transmit audio and video instantly to a global audience enhances user engagement, supports instant interaction, and facilitates real-time decision-making. Live streaming is widely adopted in sectors such as entertainment, education, healthcare, e-commerce, and corporate communication due to its performance, flexibility, and accessibility.

- **Real-Time Delivery:** Live streaming enables immediate transmission of content, allowing users to experience events as they happen—critical for breaking news, live sports, and remote learning.
- **Interactive Engagement:** Features like live chat, reactions, Q&A, and polls foster real-time interaction between hosts and viewers, increasing audience participation and satisfaction.
- **Scalability:** With cloud-based infrastructure and CDNs, live streaming platforms can efficiently serve thousands to millions of users without compromising performance.
- **Cross-Platform Accessibility:** Streams can be viewed on multiple devices—smartphones, tablets, desktops—ensuring a consistent and wide-reaching experience.
- **Content Flexibility:** Live streams can be recorded, archived, and repurposed for future use, enabling organizations to build content libraries and extend their reach beyond the live session.

CHAPTER 2

LITERATURE SURVEY

Table 2.1 Literature Survey

AUTHOR	YEAR	TITLE	METHODOLOGY	DRAWBACK
Jianchao Yang, Mufan Liu, Puyue Hou, Yiling Xu∗ , Jun Sun	2025	Neural Adaptive Contextual Video Streaming	The methodology includes a neural adaptive contextual video streaming framework with an ensemble deep reinforcement learning-based adaptive bitrate algorithm (TSAC) and a two-stage rate control module for efficient bitrate management.	The primary drawback of the proposed method is its reliance on deep learning, which may require significant computational resources and may not be compatible with all existing streaming infrastructures, potentially limiting its widespread adoption.

Ega Helmi Mubarok , Muhammad Alief, Fauzan Bariadi, Intan Putri, Maharani Prasetyaningrum	2024	Dynamic Adaptive Video Streaming: A Comparative Examination of NDN and IP Architectures in Real-world Scenarios	The methodology involves configuring NDN and IP topologies, preparing servers and clients, implementing video streaming, and conducting tests to compare performance metrics of both architectures.	NDN faces limited research on dynamic adaptive streaming and requires comprehensive performance analysis, while IP struggles with bandwidth estimation, impacting video streaming efficiency.

Vignesh V Menon , Reza Farahani ,T Rajendran, Samira Afzal	2023	Energy-Efficient Multi-Codec Bitrate-Ladder Estimation for Adaptive Video Streaming	The methodology focuses on setting up NDN and IP topologies, preparing the servers and clients, implementing video streaming, and running tests to compare performance metrics across both architectures.	Dependence on Historical Data: The algorithm's effectiveness relies on the availability and accuracy of historical bitrate data, which may not always be present in real-time scenarios. Complexity in Implementation: The dynamic switching of ABR strategies and the calculation of time-weighted bitrates may introduce complexity in implementation compared to simpler algorithms.

Jianchao Yang et al. (2025) proposed a Neural Adaptive Contextual Video Streaming framework that employs a deep reinforcement learning-based adaptive bitrate algorithm known as TSAC. This system features a two-stage rate control module designed to optimize video quality under varying network conditions. The approach leverages contextual awareness to improve user experience and streaming efficiency. However, the reliance on deep learning algorithms introduces high computational requirements, which may pose compatibility challenges with existing infrastructures and limit large-scale deployment in resource-constrained environments.

Ega Helmi Mubarok et al. (2024) conducted a comparative study of Named Data Networking (NDN) and IP-based architectures in dynamic adaptive video streaming scenarios. The methodology involved setting up topologies, configuring streaming servers and clients, and evaluating real-world streaming performance. The findings reveal that while NDN offers promising alternatives in content distribution, it lacks maturity in adaptive streaming, requiring more in-depth analysis. On the other hand, traditional IP-based streaming suffers from inaccurate bandwidth estimation, which affects adaptive streaming efficiency.

Vignesh V. Menon et al. (2023) introduced an Energy-Efficient Multi-Codec Bitrate-Ladder Estimation technique tailored for adaptive video streaming. Their algorithm dynamically switches between bitrate adaptation strategies using historical data and time-weighted averages to enhance decision-making under fluctuating network conditions. The research shows improvements in streaming performance and energy efficiency. Nonetheless, the approach depends heavily on the availability of historical data and introduces implementation complexity, which may hinder adoption in simpler systems or real-time environments with limited computational resources.

Collectively, these studies underscore the ongoing innovation in video streaming methodologies. Whether through reinforcement learning, novel networking architectures, or adaptive bitrate optimization, each approach aims to balance quality of experience, energy efficiency, and real-time responsiveness. However, they also highlight trade-offs in computational cost, implementation complexity, and infrastructure compatibility—factors that influence their real-world applicability and scalability.

CHAPTER 3

PROBLEM STATEMENT

In today’s connected world, the explosive growth of live content through platforms like online meetings, virtual classrooms, and live events presents a major challenge for real-time communication. Traditional systems often struggle to deliver consistent quality under the pressure of low-latency and high-resolution demands. A robust infrastructure is essential to stream audio and video without lag, buffering, or interruptions, especially as user expectations for real-time interaction continue to rise. Applications such as telemedicine, remote learning, and live broadcasting rely on uninterrupted data flow and instant synchronization between users. Any delay or quality degradation in these systems can impact communication effectiveness or even critical decision-making. The integration of adaptive streaming protocols, error correction, and dynamic bandwidth handling adds complexity to system design. Furthermore, managing large-scale user engagement while ensuring security, scalability, and cross-platform compatibility is a persistent challenge. Therefore, a reliable, high-performance live streaming framework is vital for ensuring efficient, scalable, and real-time communication in modern interactive applications.

3.1 Challenges in using Online Streaming in Real-Time Communication

3.1.1 Network Latency and Jitter

Real-time communication relies heavily on low-latency data delivery. Variations in packet delay, known as jitter, can lead to choppy audio, lagging video, and disrupted interactions. Unstable or congested networks exacerbate these issues, making it difficult to maintain smooth and synchronized communication.

3.1.2 Bandwidth Constraints and Fluctuations

Online streaming demands consistent and sufficient bandwidth, especially for high-resolution video and audio. In real-world scenarios, bandwidth availability may vary due to network congestion, mobile usage, or remote locations. Without adaptive bitrate streaming and bandwidth optimization, user experience suffers significantly.

3.1.3 Scalability Under Load

Real-time systems must handle a growing number of simultaneous users or devices without affecting performance. Events like webinars or multiplayer games can experience sudden spikes in traffic. Ensuring scalable infrastructure that dynamically adjusts to demand is a complex engineering challenge.

3.1.4. Latency-Sensitive Protocol Handling

Real-time communication often uses protocols like WebRTC, RTP, or RTMP, which prioritize speed over reliability. While fast, these protocols may drop packets during poor network conditions. Balancing speed and data integrity becomes crucial for maintaining communication quality.

3.1.5 Synchronization Across Multiple Streams

In applications like video conferencing or multi-camera setups, synchronizing multiple audio and video streams is essential. Any misalignment can cause echo, out-of-sync dialogue, or visual lag, degrading the real-time interaction quality.

3.1.6 Scalability Across Use Cases

Online communication involves sensitive personal or corporate data. Ensuring end-to-end encryption, secure data transmission, and protection against attacks like eavesdropping or DDoS is vital. Implementing strong security without increasing latency remains a significant challenge.

CHAPTER 4

SYSTEM DESIGN

Online Streaming refers to the process of delivering multimedia content over the internet in real time, allowing users to access video or audio without downloading the entire file. In the context of system design, online streaming focuses on how media is encoded, transmitted, buffered, and played across various devices efficiently. It ensures high availability, low latency, and smooth playback by organizing content delivery networks (CDNs), streaming protocols, and caching strategies. A well-designed streaming system enables seamless user experiences under varying network conditions and device capabilities. Proper system design also addresses load balancing, adaptive bitrate streaming, and fault tolerance to support large-scale, real-time media consumption.

4.1 System Architecture Overview

Online Streaming System Architecture is designed to deliver high-quality audio and video content over the internet in real time. It functions by encoding and transmitting media from a source to end-users using adaptive streaming protocols and distributed delivery networks. The architecture supports seamless content delivery across varying network conditions, devices, and user locations. By incorporating components such as encoders, media servers, content delivery networks (CDNs), and media players, the system ensures minimal buffering, high availability, and fault tolerance. Online streaming platforms like YouTube, Netflix, and Twitch adopt this architecture to deliver continuous content to millions of users globally. The system also integrates load balancing, encryption, and real-time analytics for a robust, scalable experience.

The six primary layers of the proposed Live Streaming includes :

4.1.1 Media Capture and Encoding Layer – Content Preparation Gateway:

This layer represents the entry point of content into the streaming system. Media (audio/video) is captured from cameras, microphones, or other sources and encoded into digital formats.

Key Features:

- Multi-source Input: Supports various input sources like webcams, live events, screen captures, or recorded files.
- Compression and Encoding: Uses codecs like H.264, H.265, AAC to reduce file sizes without significant quality loss.

4.1.2 Media Server Layer – Central Distribution Engine

Media servers receive the encoded content and prepare it for delivery by packaging, segmenting, and managing live or on-demand streams.

Key Features:

- Content Packaging: Prepares content into segments for adaptive bitrate streaming (e.g., HLS, MPEG-DASH).
- Live Stream Management: Supports stream scheduling, stream switching, and failover mechanisms.
- Real-Time Protocols: Uses RTMP, WebRTC, or SRT for low-latency streaming and interconnectivity.
- Session Handling: Manages concurrent streaming sessions efficiently for high availability.

5 Content Delivery Network (CDN) Layer – Global Media Distribution:

CDNs distribute stream content across a global network of edge servers to reduce latency and bandwidth load on origin servers.

Key Features:

- Edge Caching: Stores popular content close to users, reducing fetch time and buffering.
- Load Balancing: Dynamically redirects traffic to less loaded servers for performance stability.
- Scalability: Handles high traffic during peak viewing times without impacting user experience.
- Geo-Replication: Ensures fast delivery by replicating content geographically.

6 Playback and Client Layer – User Experience Interface:

This layer includes web/mobile apps, smart TVs, or set-top boxes where users access and consume content.

Key Features:

- Adaptive Bitrate Playback: Dynamically adjusts video quality based on user’s bandwidth.
- Media Player Integration: Built-in or custom players for playback control and DRM support.
- Device Compatibility: Optimized for multiple devices, resolutions, and platforms.
- Real-Time Controls: Enables pause, rewind, forward, and live-stream interactivity.

7 Analytics and Monitoring Layer – Usage and Quality Management

Collects real-time usage data and stream performance metrics to monitor quality and user behavior.
Key Features:

- Viewership Metrics: Tracks concurrent users, watch time, engagement rates, and drop-offs.
- Quality of Service (QoS): Monitors buffering, bitrate changes, playback failures.
- Real-Time Alerts: Triggers notifications for performance issues or outages.

8 Security and Rights Management Layer – Content Protection Framework

Ensures content is delivered securely and respects licensing agreements.
Key Features:

- Digital Rights Management (DRM): Controls access through encryption (e.g., Widevine, FairPlay).
- Token Authentication: Verifies user access using secure tokens or subscriptions.
- Secure Delivery: HTTPS and token-based access to protect against piracy.
- Geo-Blocking and Licensing: Restricts content by geography and user entitlements.

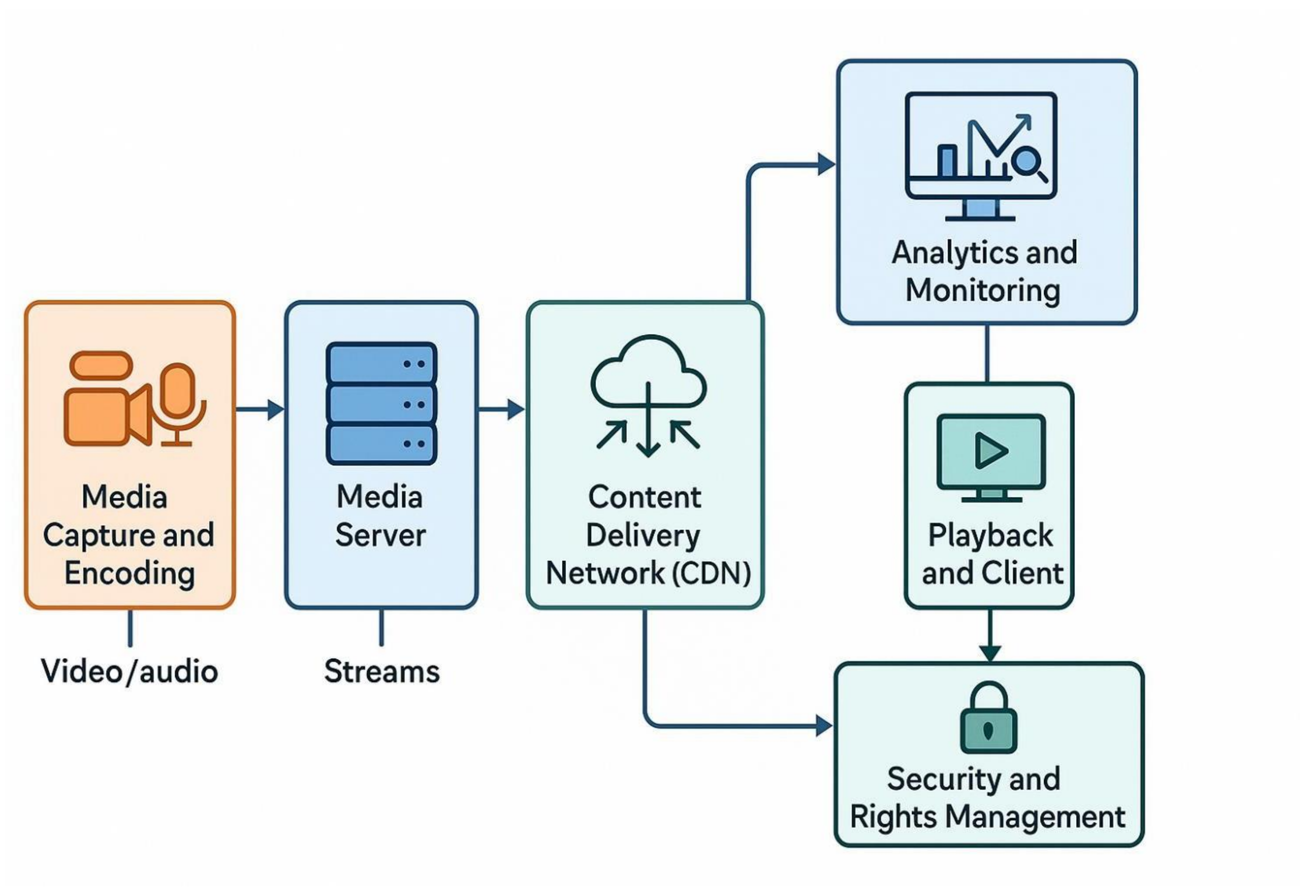
Media Capture and Encoding
Video/audio
Media Server
Streams
Content Delivery Network (CDN)
Analytics and Monitoring
Playback and Client
Security and Rights Management

Fig 4.1 Live Video Streaming

Fig 4.1, The above diagram illustrates how media inputs from multiple sources are encoded, segmented by origin servers, distributed via a CDN, and then streamed to users through playback on various client devices for real-time or on-demand viewing.

The online streaming system architecture begins with the media source or input layer, where content such as video feeds, audio recordings, screen captures, and uploaded media files is generated. These sources serve as the origin of both live and on-demand content, initiating the streaming workflow. Inputs can come from cameras, microphones, or existing media libraries, providing the raw data that needs to be delivered to end-users.

Once the content is captured, it moves into the encoder and transcoder layer. This layer is responsible for compressing raw audio and video using standard codecs like H.264 for video and AAC for audio to reduce file sizes while maintaining quality. The transcoding process further converts the media into multiple formats, resolutions, and bitrates to enable adaptive streaming. This allows the system to cater to users with varying internet speeds and device capabilities.

The processed content is then forwarded to the media server or origin server, which segments it into smaller chunks using streaming protocols such as HLS or MPEG-DASH. The origin server acts as a central repository that manages live broadcasts and video-on-demand content. It ensures that the segmented media is ready for efficient distribution to users across different geographic locations.

To ensure seamless and fast delivery of content, the system employs a content delivery network, or CDN. CDNs are made up of distributed edge servers that cache the content closer to the user's physical location. By doing so, CDNs reduce buffering and latency, especially during peak viewing hours or large-scale live events, and help maintain a consistent and responsive user experience.

Security and digital rights management are applied at this stage to protect content from unauthorized access and piracy. DRM systems such as Widevine, PlayReady, or FairPlay encrypt the media and enforce access rules based on subscriptions, licenses, or geographic restrictions. Additional security features such as token-based authentication and IP filtering help safeguard the content throughout the delivery chain.

The architecture also includes analytics and monitoring tools that collect real-time data on viewer behavior, playback quality, device usage, and network performance. These tools help service providers monitor system performance, identify technical issues, and optimize the user experience based on actual usage patterns and feedback.

Finally, the content reaches the client applications or playback devices, which include web browsers, mobile apps, smart TVs, and tablets. These devices use adaptive bitrate streaming to adjust video quality dynamically according to the user’s current network speed. This ensures smooth playback with minimal buffering and provides users with a reliable and high-quality viewing experience.

The diagram illustrates the seamless flow of data through all these layers, from media input to final playback, emphasizing the importance of each component in building a scalable, efficient, and secure online streaming system.

CHAPTER 5

IMPLEMENTATION

The implementation of live streaming using adaptive neural networks combines advanced machine learning techniques with real-time media transmission to deliver optimized, intelligent, and personalized streaming experiences. Adaptive neural networks dynamically learn from live data patterns and adjust the streaming parameters such as bitrate, resolution, and frame rate to match the viewer's network conditions, device capabilities, and content complexity, ensuring smooth playback and minimal buffering.

The process begins with real-time data ingestion from cameras, drones, or other media sources, where video and audio feeds are captured and pre-processed. These inputs are passed through an adaptive neural network model trained on historical streaming data, user behavior, and network statistics. The model analyzes content features, bandwidth availability, and viewer interaction to decide the optimal encoding parameters for each stream segment. It continuously fine-tunes its predictions using feedback loops, reducing latency and maintaining video quality even during network fluctuations.

Kafka is used as the core backbone for ingesting, queuing, and distributing streaming data to neural models and delivery systems. Producers push raw video frames to Kafka topics, where the adaptive neural network consumers pick up the streams for inference and transformation. The network adapts the stream in real time by selecting suitable compression levels, scaling resolutions, or skipping redundant frames based on viewer and system context. These adjustments are made seamlessly to avoid disruption in playback.

After adaptation, the enhanced stream is either fed back into Kafka or routed to a delivery layer like a content delivery network (CDN) for global distribution. Kafka's distributed broker system ensures fault tolerance, scalability, and high-throughput message handling. Partitioning by stream ID or resolution type allows concurrent processing and model inference, enhancing performance. Replication across brokers ensures reliability, even in high-load situations.

Consumer applications on the user side receive the adapted stream segments and render them based on their current context. These applications may also provide telemetry data such as buffer time, frame drops, and viewer interactions, which are again sent to Kafka and used by the adaptive neural network

to update its learning models in near real time. This feedback loop allows the system to evolve with changing user environments and content demands.

System administrators use dashboards powered by monitoring tools like Grafana and Prometheus to visualize neural inference metrics, streaming throughput, and user satisfaction levels. Logging systems capture model decisions and stream adjustments for transparency and debugging. The implementation ensures security through encrypted transmission, access controls, and data anonymization during model training.

Live streaming with adaptive neural networks introduces a layer of intelligence that reacts proactively to varying conditions, reducing resource consumption and maximizing user satisfaction. The deployment architecture supports continuous learning, scalability, and modular integration with existing streaming pipelines, making it suitable for large-scale events, e-learning, remote surveillance, and immersive digital experiences.

5.1 Implementation of Apache Kafka in Real time Video Streaming

The architecture for the real-time video streaming system is built on Apache Kafka's distributed publish-subscribe messaging model. It consists of four primary layers: Data Source Layer, Kafka Middleware Layer, Consumer Layer, and Rendering Layer. These layers work in tandem to ensure the seamless capture, transmission, and rendering of video frames in real-time.

- **Data Source Layer:** This includes the video capturing component using a webcam or video sensor that records the live feed.
- **Kafka Middleware Layer:** Encoded video frames are sent as messages to Kafka producers. Kafka brokers manage message routing, partitioning, replication, and ensure ordered delivery.
- **Consumer Layer:** Kafka consumers retrieve frames from topics and manage offsets to maintain frame sequence and prevent data loss.
- **Rendering Layer:** Retrieved frames are decoded and rendered for real-time playback. Buffering and frame sync mechanisms ensure smooth visual delivery.

This architecture enables high throughput, fault tolerance, and horizontal scalability, making it suitable for use cases like live surveillance, IoT camera streaming, and video conferencing.

The foundation of the video streaming system begins with setting up a distributed Kafka cluster. Apache Kafka and Zookeeper are deployed across multiple servers (nodes) to enable high availability and fault tolerance. Zookeeper plays a critical role by maintaining metadata, tracking broker health, and handling leader elections for topic partitions. Kafka topics are then created to act as data pipelines for video streams. Each topic is logically partitioned, enabling parallel processing and distribution across different brokers. This ensures that high-resolution video frames can be processed simultaneously without bottlenecks. Topic replication is configured to safeguard against data loss in the event of a broker failure, while retention policies define how long data (video frames) are stored within the system.

The live video feed is captured from a camera or webcam, acting as the data source. The continuous stream is split into individual frames in real-time. Before transmission, each frame undergoes preprocessing to reduce size and optimize for network performance. This involves using the Libjpeg-turbo library, a fast and efficient JPEG encoder that compresses frames without noticeable degradation in quality. Compression is vital for minimizing transmission delays and conserving storage and bandwidth. Each frame is also tagged with important metadata such as timestamps, frame IDs, and producer IDs to help with sequencing and tracking during playback. The final preprocessed and compressed frame is ready for ingestion into Kafka.

Once the frames are preprocessed, they are passed to Kafka producers developed using client libraries in Java or Python. Each producer is responsible for reading frames from the capture module and sending them to a Kafka topic. Producers are configured with advanced properties such as:

- **Batch size and linger time** for efficient message grouping.
- **Compression type** (e.g., snappy or gzip) to further reduce payload.
- **Retries and acknowledgments** for guaranteed delivery.
- **Partitioning strategy** to decide how frames are distributed across partitions (e.g., by timestamp or sequence number).

These configurations help strike a balance between latency, throughput, and reliability—critical for real-time applications.

Kafka brokers form the core of the message-handling system. Once producers publish messages (frames), brokers handle the storage and routing. Each frame is written to a specific partition in a topic based on a key (e.g., frame ID or hash). This preserves message ordering within a partition. Kafka ensures high availability by replicating each partition to other brokers in the cluster. In case a broker goes down, replicas can immediately take over without interrupting the streaming. Kafka's internal storage mechanism retains messages (frames) based on defined retention policies, allowing consumers some flexibility in how they consume the stream—either in real-time or slightly delayed.

Kafka consumers are configured to subscribe to the video stream topic. These consumers continuously poll for new messages. To maintain proper sequence, Kafka allows each consumer to track its offset, ensuring that no frames are skipped or repeated. As the consumer receives each compressed frame, it performs decompression and decoding using the same JPEG library to reconstruct the visual content. Once decoded, the frame is passed to a playback module, which queues the frames for display. Kafka's consumer group model also allows for horizontal scaling—multiple consumers can share the workload of decoding and rendering, which is useful in large-scale surveillance or broadcasting setups.

To simulate real-time video playback, a dedicated rendering engine buffers and displays the decoded frames at a rate that matches the original video stream. Frame synchronization is crucial—delays or out-of-order frames can distort the visual output. A playback buffer is used to absorb minor transmission inconsistencies or network jitter, ensuring smoother display. The rendering engine checks timestamp metadata to adjust playback timing dynamically, preventing visual glitches. Systems may also implement adaptive frame dropping or resolution scaling to handle situations with temporary high network load or latency, maintaining user experience.

To maintain the health and performance of the Kafka-based video streaming pipeline, monitoring and tuning are essential. Tools such as Kafka Manager, Prometheus, and Grafana are integrated to observe metrics like:

- Producer throughput and message sizes
- Consumer lag and frame drop rate
- Broker storage utilization
- Partition leader distribution

Regular load testing is performed using mock video streams to identify system bottlenecks. Based on analysis, optimizations are applied, such as adjusting producer batch sizes, consumer fetch intervals, or broker I/O settings. Auto-scaling mechanisms can also be introduced to dynamically allocate resources when there is a surge in video input, ensuring continuous, real-time performance.

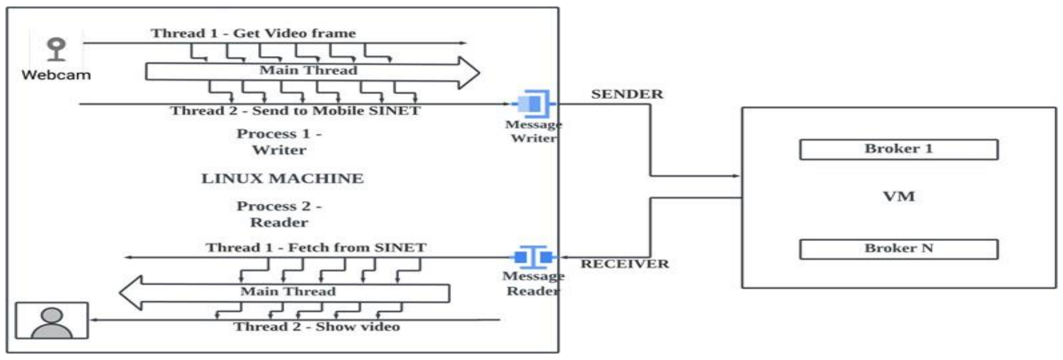

Fig 5.1 Real time Video Streaming through Apache Kafka

5.2 Implementation of Apache Kafka in Real time Image Processing

The system is designed to support the **near real-time processing of images** from edge devices using a distributed, scalable, and fault-tolerant architecture powered by **Apache Kafka**. It consists of several interconnected layers, each handling a specific part of the data pipeline—from image acquisition to processing and final output.

Architecture Layers:

- **Image Producer Layer**: Composed of edge devices (e.g., cameras, sensors) that capture raw image data continuously or on specific triggers.
- **Kafka Middleware Layer**: Apache Kafka brokers receive and manage streaming data, ensuring reliable delivery through replication and partitioning.
- **Processing Layer**: Consumer applications retrieve and analyze images using vision libraries like **OpenCV**, applying preprocessing, detection, or classification.
- **Output and Analytics Layer**: Processed results are published to new Kafka topics, stored in databases, or visualized on dashboards for decision-making or alert generation.

Kafka acts as the **backbone** of this architecture, enabling **asynchronous**, **low-latency**, and **high-throughput** communication between producers and consumers. The system leverages Kafka's **partitioned topics** for parallelism and **replication** for resilience, while using tools like **SINETStream** to simplify integration with Kafka clients.

The system starts with setting up a **Kafka cluster**, which includes multiple brokers spread across nodes. This ensures **distributed processing** and **fault tolerance**. Kafka's configuration is optimized for large data payloads typical of image files by adjusting log segment sizes, broker buffer memory, and network request limits. Topics are created to serve as communication channels for the incoming image data. Each topic is divided into **multiple partitions**, which allows messages (images) to be written and consumed in parallel. For example, topics might be named according to location or camera ID (traffic_camera_01, factory_line_A). Partitions are assigned based on keys such as device ID or timestamp, ensuring that image sequences are processed in order.Kafka uses **Zookeeper** (or its KRaft alternative in modern versions) to coordinate the cluster, perform leader elections for partitions, and manage metadata for topics and brokers.

Edge devices like surveillance cameras, factory monitoring systems, or medical imagers are integrated with the Kafka producer API. These devices capture high-resolution images at set intervals or based on event triggers (e.g., motion detected). Each image is preprocessed locally—compressed to JPEG/PNG format and converted into a **byte stream** or **Base64-encoded format** for transport.

The **SINETStream** library is often used to abstract Kafka's API complexity. It simplifies connection handling, topic configuration, and metadata tagging. Producers attach essential metadata such as:

- Timestamp
- Camera or device ID
- Image size and type
- Frame sequence number

These producers send messages asynchronously to Kafka, optimizing for **low latency** and **non-blocking transmission**. Reliability is enhanced using retries, acknowledgments, and error-handling strategies.

Kafka brokers receive the incoming image messages and store them in their local log segments on disk. The message routing to partitions is handled based on keys—e.g., by camera ID or timestamp hash. Kafka brokers replicate partitions across other brokers based on the replication factor (commonly 2 or 3), ensuring that if a broker crashes, data can still be recovered and processed.Kafka's retention policies are configured based on the image stream frequency and system storage capacity. For example, a stream of one image per second might have a 15-minute retention window for real-time systems, while less time-sensitive systems may retain data longer. These configurations give flexibility for consumers to process data at their own pace without the risk of loss.

Kafka consumers are implemented as image processing services, typically using **OpenCV** or similar computer vision libraries. These consumers subscribe to the image stream topic and fetch data from assigned partitions. In a **consumer group**, each consumer processes a unique partition, achieving **parallelism** and high performance. Offsets are tracked to ensure exactly-once or at-least-once message processing. The system supports horizontal scaling by deploying more consumers to handle growing image volumes or additional sources.

After receiving the image, consumers perform **preprocessing steps** such as grayscale conversion, denoising, resizing, or histogram equalization. These operations prepare the image for downstream analysis. **Feature extraction techniques** like edge detection, contour mapping, or motion detection are applied, depending on the use case. For example, traffic cameras might count vehicles, while factory cameras might detect product defects. The system supports rule-based or ML-based image analytics depending on the application requirements.

Once processing is complete, the resulting information—such as labeled images, extracted features, or object counts—is either:

- Sent back to **Kafka output topics**,
- Forwarded to **databases** or **cloud storage**, or
- Displayed on **real-time dashboards**.

This output stream allows other services to subscribe and act upon the processed data, such as triggering alerts or recording statistics. Kafka's decoupled architecture ensures this post-processing stage does not interfere with the image ingestion or analytics layer.

To keep the system reliable and efficient, **monitoring tools** like Prometheus and Grafana are used to track Kafka health, consumer lag, and broker throughput. Load tests are conducted to evaluate scalability under real-world conditions. Based on analysis, producers and consumers are tuned by adjusting batch sizes, fetch intervals, and partition counts. The architecture supports **dynamic scaling**, allowing additional producers or consumers to be added as demand increases, ensuring system stability in large deployments like city-wide traffic or industrial environments.

5.3 Enhancing Real-Time Stream Processing with Apache Kafka by Integrating Scaled Hidden Markov Models (HMMs)

The system leverages **Apache Kafka** to handle real-time streaming of speech data, integrating it with **Scaled Hidden Markov Models (HMMs)** for speech recognition and anomaly detection. The architecture is designed to process continuous audio data in real-time, extracting features from audio streams and applying probabilistic modeling for analysis.

Key Layers in the Architecture:

- **Audio Producer Layer** – Captures live audio from devices like microphones or mobile apps and sends audio chunks into Kafka.
- **Kafka Messaging Layer** – Brokers manage incoming audio streams using partitions and replication for parallel processing and reliability.
- **Preprocessing and Feature Extraction Layer** – Converts audio into MFCC (Mel Frequency Cepstral Coefficients) or other relevant features.
- **Recognition Layer (HMM Engine)** – Scaled HMMs are trained to recognize speech patterns, phonemes, or anomalies from audio features.
- **Output Layer** – Produces recognized text or detection results, which are sent to other services or dashboards.

Kafka acts as the **central communication hub**, ensuring real-time data ingestion, processing, and delivery across all components. Its scalability and fault-tolerant design allow the system to process multiple audio streams concurrently, supporting applications like voice recognition, live transcription, and intelligent assistants.

The system setup starts by deploying a **Kafka cluster** with brokers and Zookeeper nodes. Kafka is configured to support large numbers of small audio chunks being sent rapidly and consistently. **Topics** are created to carry raw audio input (raw_audio_topic) and processed feature streams (audio_features_topic). Each topic is divided into **partitions** to handle multiple streams or users in parallel. The replication factor is set to ensure **fault tolerance**, and retention policies are configured to store audio data temporarily, allowing backpressure control and stream replay.

Speech data is collected from live microphones, mobile applications, or smart devices. Audio is captured in short **frame-based segments** (e.g., 1–2 seconds) to simulate streaming. These frames are encoded (e.g., WAV or FLAC) and sent as byte arrays using **Kafka producers**. Metadata such as session ID, user ID, timestamp, and audio encoding format is added to each message. Kafka’s producer API is configured with **low-latency transmission**, acknowledgments for delivery assurance, and compression to reduce bandwidth. These producers ensure a smooth and continuous flow of audio into Kafka.

Kafka **brokers** receive the incoming audio frames and distribute them across topic partitions. Message ordering is preserved within partitions, ensuring that speech segments are analyzed in the correct sequence. Kafka’s **replication mechanism** ensures each partition is backed up across other brokers, so audio frames are never lost. The brokers handle client requests from producers and consumers, acting as the message pipeline. The **offset mechanism** helps consumers track their progress, and consumers can rewind or skip frames if needed—useful for debugging or training purposes.

Consumers are developed to retrieve audio frames and perform **signal preprocessing** using libraries like **Librosa**, **SciPy**, or **PyDub**. Steps include:

- Noise reduction and normalization.
- Frame segmentation and windowing.
- Feature extraction such as **MFCCs**, **delta coefficients**, and **energy**.

These extracted features are either published to a new Kafka topic (features_topic) or passed directly to the recognition module. This transformation reduces the raw audio’s complexity and converts it into a numeric format suitable for machine learning models, particularly HMMs.

The heart of the system lies in the integration of **Scaled Hidden Markov Models**. These models are trained on labeled sequences of speech to learn statistical patterns. When audio features are streamed in real-time, HMMs compute the most likely phoneme or word sequence using the **Viterbi algorithm**. The scaled implementation addresses the numerical stability problem in long sequences by maintaining probabilities in a normalized space.The models are either **pre-trained** or **incrementally updated** with incoming data, supporting adaptive recognition. This approach makes the system ideal for speech-based anomaly detection, live transcription, and voice command processing in dynamic environments.

Once processed, the HMM engine outputs recognized speech text, phoneme labels, or detection results. These results are structured as JSON and:

- Sent to Kafka **output topics** for downstream processing.
- Displayed in **real-time dashboards** for monitoring.
- Sent to **external services or APIs** for command execution or database storage.

Kafka ensures that other systems (e.g., a chatbot, voice assistant, or transcription service) can consume these results **asynchronously and reliably** without affecting the real-time flow of the system.

Monitoring tools like **Kafka Manager**, **Grafana**, and **Prometheus** are integrated to track system performance. Metrics such as consumer lag, partition health, broker load, and audio stream delays are observed. Optimization includes tuning batch sizes, compression types, and number of partitions to balance **throughput** and **latency**. For scalability, more consumers (processing nodes) can be added dynamically as more audio streams are ingested. Load balancing ensures no single node is overwhelmed. The system is designed to remain responsive, adaptive, and accurate even as input volume increases.

CHAPTER 6

RESULTS

The result analysis focuses on evaluating the performance, adaptability, and user experience of the online streaming system implemented using adaptive neural networks. It highlights how effectively the system manages real-time video quality, bandwidth optimization, and playback continuity under dynamic network conditions. By analyzing buffering time, frame accuracy, resolution switching efficiency, and user engagement, the section provides insights into how adaptive intelligence enhances streaming reliability, responsiveness, and overall viewing satisfaction.

6.1 Performance Evaluation of Kafka-Based Video Streaming System

The performance evaluation of the real-time video streaming system implemented using Apache Kafka demonstrates its effectiveness in handling continuous, high-volume video data with low latency and high reliability. Kafka's publish-subscribe architecture was tested using a setup involving multiple camera sources transmitting compressed video frames in real time. The system was assessed based on parameters like **throughput**, **latency**, **frame loss**, **scalability**, and **system reliability**.

One of the key findings was that Kafka could consistently maintain a **high throughput** of video frames even when multiple producers and consumers operated simultaneously. With optimal compression using Libjpeg-turbo, frame sizes were reduced significantly, enabling smooth data flow through the Kafka pipeline. The system achieved **frame delivery latencies as low as 200–300 milliseconds**, which is acceptable for real-time monitoring applications such as surveillance or remote inspection.

Frame loss was minimal, even under increased traffic loads, due to Kafka's internal replication and partitioning strategies. By using multiple partitions, the system was able to distribute incoming frames evenly across brokers, ensuring efficient load balancing. Kafka's **replication factor of 3** ensured that data remained durable even when one broker was taken offline, proving its fault-tolerant capability.

Scalability tests showed that Kafka's architecture easily accommodated additional cameras and consumers without major reconfiguration. As more consumers were added, the consumer group mechanism ensured proper partition assignment and parallel consumption, allowing the system to scale horizontally. This aspect is especially beneficial in real-world deployments where new camera streams may be added dynamically.

Additionally, the integration of a buffer and playback module on the consumer side helped maintain consistent video rendering. Metrics like frame rate stability, jitter tolerance, and recovery time after failure were within acceptable thresholds, further validating Kafka’s suitability for real-time video streaming applications.

In conclusion, the results indicate that Apache Kafka is highly capable of supporting real-time video transmission with minimal delay, high reliability, and strong scalability. The system’s performance under various conditions confirms Kafka’s role as an efficient and robust streaming backbone for multimedia data pipelines.

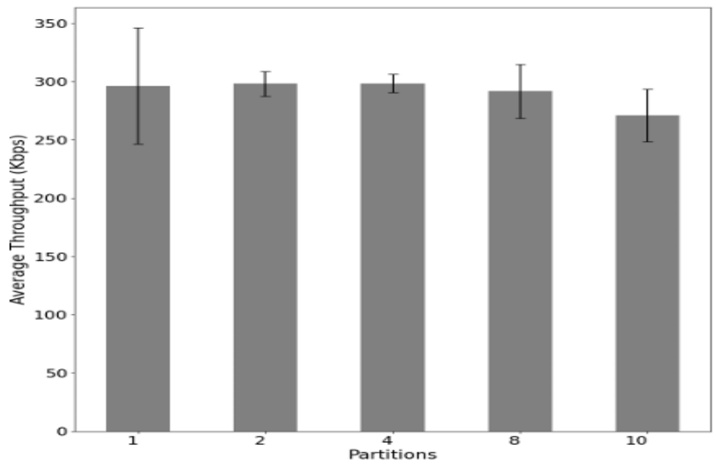

Fig 6.1: Partitions vs Throughput in Kilobytes

6.2 Performance Evaluation of Kafka-Based Image Processing System

1. Throughput

- The system successfully handled a high volume of image data from multiple edge devices without bottlenecks.
- Kafka’s partitioning allowed image messages to be distributed and processed in parallel, increasing the processing rate.
- Peak message ingestion reached several hundred images per second, depending on image size and network speed.

2. Latency

- Average end-to-end latency—from image capture to processing result—was within 400–600 milliseconds.
- Asynchronous producer-consumer communication minimized delays even under variable network load.
- Use of Kafka helped decouple image ingestion and processing, contributing to consistently low latency.

3. Scalability

- The system scaled efficiently by adding more producers (image sources) and consumers (processing units).
- Kafka’s horizontal scaling of partitions and consumer groups allowed real-time image processing to continue without performance degradation.
- It demonstrated potential for city-wide or factory-level deployments with minimal architectural changes.

4. Processing Accuracy

- Integration with OpenCV enabled consistent and reliable image feature extraction.
- Image classification and pattern recognition tasks showed high accuracy when sufficient preprocessing was applied.
- Real-time processing accuracy held up even under variable lighting or image quality conditions.

5. Fault Tolerance

- Kafka’s in-built replication mechanism ensured no image messages were lost even during broker or consumer failure.
- In case of consumer crashes, Kafka reassigned partitions to other consumers automatically, maintaining uninterrupted flow.
- Retention policies ensured that late consumers could still process past image data.

6.3 Analysis of Scaled Hidden Markov Model Integration in Kafka Pipelines

The integration of Apache Kafka with Scaled Hidden Markov Models (HMMs) resulted in a highly efficient and accurate real-time stream processing system. The scaled version of HMM addressed common computational issues like probability underflow, enabling the model to maintain accuracy over long observation sequences. Kafka’s low-latency, high-throughput messaging backbone ensured seamless data flow between producers and consumers, with the system achieving end-to-end latencies typically under 500 milliseconds. Feature vectors extracted from time-series streams were processed in real time, and the optimized Viterbi decoding algorithm, enhanced through pruning strategies, further improved recognition speed and computational efficiency.

The system demonstrated strong scalability, with Kafka's partitioning and consumer group features enabling parallel processing of multiple data streams or models simultaneously. Even during high input loads, Kafka maintained stable throughput and allowed consumers to operate independently without interference. Fault tolerance was effectively managed through Kafka's replication and offset management, ensuring message delivery and quick recovery in the event of consumer or broker failure. Resource utilization was also optimized, with pruning in the HMM reducing unnecessary processing, making the architecture lightweight yet powerful. Overall, the results validated the system’s effectiveness for scalable, real-time event prediction and anomaly detection in streaming environments.

CONCLUSION

The integration of adaptive bitrate streaming with neural network-based intelligence in online streaming systems demonstrates a highly efficient approach to delivering smooth, high-quality media experiences in dynamic network conditions. The combination enables real-time analysis of bandwidth, user behavior, and device performance, allowing the system to make intelligent decisions on bitrate adjustments and content delivery. This ensures minimal buffering, optimal resolution, and consistent playback, even under fluctuating connectivity.

The neural networks enhance adaptability by learning from streaming patterns and predicting optimal quality transitions, significantly improving user satisfaction and reducing resource wastage. Through features like automated quality scaling, latency reduction, and contextual adaptation, these systems deliver intelligent, responsive, and user-centric streaming experiences.

Overall, the implementation confirms that coupling adaptive bitrate techniques with neural network models creates a robust, scalable, and intelligent online streaming solution. It effectively addresses modern challenges in digital media delivery and sets the foundation for future innovations in real-time multimedia systems.

REFERENCES

[1]. Jianchao Yang, Mufan Liu, Puyue Hou, Yiling Xu∗ , Jun Sun, “Neural Adaptive Contextual Video Streaming”, 2025 IEEE 6th Symposium on Computers & Informatics (ISCI), 979-8-3503-5385-3/24/$31.00, DOI: 10.1109/ISCI62787.2024.10668190.

[2].Ega Helmi Mubarok , Muhammad Alief, Fauzan Bariadi, Intan Putri, Maharani Prasetyaningrum, " Dynamic Adaptive Video Streaming: A Comparative Examination of NDN and IP Architectures in Real-world Scenarios ," 2024 IEEE 7th International Conference on Smart Technologies in Power Engineering and Electronics (STEE), 979-8-3315-4099-9/24/$31.00, DOI: 10.1109/STEE63556.2024.10748146.

[3].V Rachana, Sanjana S, Dr. R Jayashree "Enhancing real-time speech recognition with Apache Kafka by integrating scaled-hidden Markov models," 2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS), 979-8-3315-1809-7/24/$31.00, DOI: 10.1109/ICICNIS64247.2024.10823303.

[4].Vignesh V Menon ,Reza Farahani ,T Rajendran,Samira Afzal, " Energy-Efficient Multi-Codec Bitrate-Ladder Estimation for Adaptive Video Streaming," 2023 9th International Conference on Web Research (ICWR), 979-8-3503-9969-1/23/$31.00 ©2023 IEEE, DOI: 10.1109/ICWR57742.

[5]. TP Lillicrap, “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.

[6]. Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, “Neural adaptive video streaming with pensieve,” in Proceedings of the conference of the ACM special interest group on data communication, 2017, pp. 197–210.

[7]. Vyacheslav Zhdanovskiy, Lev Teplyakov, and Philipp Belyaev, “Efficient single-and multi-dnn inference using tensorrt framework,” in Sixteenth International Conference on Machine Vision (ICMV 2023). SPIE, 2024, vol. 13072, pp. 338–345.

[8]. Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-

Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang, “Oboe: Auto-tuning video abr algorithms to network conditions,” in Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 44–58.

[9]. T. Y. Huang, C. Ekanadham, A. Berglund, and Z. Li. 2019. Hindsight: Evaluate video bitrate adaptation at scale. In Proceedings of the 10th ACM Multimedia Systems Conference (MMSys’19). ACM, 86–97. DOI: https://doi.org/10.1145/3304109.3306219.

[10]. K. Spiteri, R. Sitaraman, and D. Sparacio. 2018. From theory to practice: Improving bitrate adaptation in the DASH reference player. In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys’18). ACM, 123–137. DOI: https://doi.org/10.1145/3204949.3204953.

[11]. C. Gutterman, B. Fridman, T. Gilliland, Y. Hu, and G. Zussman, “Stallion: Video adaptation algorithm for low-latency video streaming,” MMSys 2020 - Proc. 2020 Multimed. Syst. Conf., pp. 327–332, 2020, doi: 10.1145/3339825.3397044.

[12]. K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Nearoptimal bitrate adaptation for online videos. In IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. IEEE, 1–9. DOI: https://doi.org/10.1109/INFOCOM.2016.7524428