Big Data Solution For

RaagaFlow

Prakhar Kochhar

N1297262

Table of Contents

6. Insights, Discussion, and Recommendations

Introduction

1.1 Overview of Music Streaming & Raga-Based Recommendations

Music streaming websites have changed the way we discover and experience music, relying on data-driven recommendations that create personalized listening experience. While existing companies such as **Spotify, Apple Music, and YouTube Music** primarily recommend tracks based on genres, individuals' tastes, and past listening history(Björklund, et al. 2022), they fail to provide recommendations on the cultural and historical significance of music.

The Indian classical music is essentially rooted in the concept of Ragas, each Raga being associated with **specific times of the day**, **emotional states, and overall feelings**. For example, **Raga Yaman** is meant to be played in the evening, and **Raga Bhairav is better understood in the morning**.(Chakraborty, et al. 2009) However, today's music recommendation systems neglect this **time-specific music tradition in the algorithm**.

RaagaFlow is a Big Data-driven music streaming platform with an artificial intelligence-driven approach to the classification of Ragas and the offering of customized music recommendations. Based on users' **auditory tastes, sonic properties, and temporal factors**, RaagaFlow makes recommendations with guidance drawn from their Raga classification and current time of the day, thus **enriching the listening experience** to be both **immersive and cultural**.

1.2 Importance of Big Data in Personalized Music Streaming

The success of **modern music recommendation** systems relies on the use of Big Data analytics, which enables the **real-time processing of massive numbers of user interaction** data. RaagaFlow can leverage advanced Big Data platforms including **Apache Kafka, Spark**, and **TensorFlow** to:

- **Record user interactions in real time (listening history, likes, skips).**
- **Examine comprehensive audio features (MFCCs, Spectrograms) for Raga classification.**
- **Store structured and unstructured data** efficiently using **NoSQL databases and cloud storage**.
- **Provide real-time song recommendations** using **Machine Learning and Hybrid Recommendation Engines**.

RaagaFlow successfully combines the essential nature of Indian Classical music with modern personalized streaming services by handling data that includes **high-volume, high-velocity**, and **high-variety data**.

1.3 Introduction to the Variety Challenge

One of the key challenges of Big Data in RaagaFlow is the dimension of **Variety**, which involves managing and bringing together different types of musical data in different formats, styles, and structures. Unlike other traditional recommendation systems that rely solely on **genre-based classification**(Björklund, et al. 2022), RaagaFlow requires an advanced classification system that bridges Indian classical music, such as **Hindustani and Carnatic music**, with **Western musical scales**.

- **Structured data: Song metadata** (title, artist, album, duration, Raga, Western scale mapping).
- **Semi-structured data:** User preferences, listening behaviour, playlist history.
- **Unstructured data:** Raw audio waveforms, spectrograms, **extracted MFCCs**.

Understanding the Organizational Context

2.1 RaagaFlow: An AI-Powered Music Streaming Platform

RaagaFlow is an AI-powered music streaming platform that sorts and recommends songs based on **Indian classical Ragas and Western musical modes**. In contrast to other streaming services such as **Spotify, Apple Music, and YouTube Music** that classifies and recommends songs primarily by artist and genre(Björklund, et al. 2022), RaagaFlow considers raw audio features, user listening behaviour, and cultural context to personalize recommendations that can be effectively achieved using **Big Data and Machine Learning (ML).**

RaagaFlow provides customized music streaming through real-time user behaviour, deep learning algorithms to identify ragas, and **hybrid recommendation systems**. RaagaFlow personalizes music streaming based on historical listening habits, song structure, and time of the day.

2.2 Industry & Market Landscape

The valuation of **global music streaming market** has grown considerably over recent years, with a valuation of **$37 billion** in 2023(ResearchAndMarkets 2024) and potentially rise to **$125 billion** by 2032(ResearchAndMarkets 2024). The presence of **more than 100 million songs**(IFPI 2025) on various music platforms has triggered a boom in demand **for AI-driven personalization and culturally enriched experiences.**

Key Industry Facts:

• **Over 250+ Ragas** exist in Hindustani & Carnatic music(Rowell 1992).

• **Western scales (Phrygian, Lydian, Mixolydian, etc.)** are widely used in global music composition, “there are **24,576** different possible scales in music”(thejazzpianosite).

• **Spotify processes upwards of 600 gigabytes of music data daily**(Maheshwari 2023), highlighting the necessity **for Big Data scalability.**

RaagaFlow targets an emerging market segment that seeks **AI-powered, culturally immersive music discovery**.

2.3 Size & Growth Projections

- **Projected Users**: 10 million+ in the first three years.
- **Music Library**: **60 million+ tracks across Indian and Western music styles**.
- **Data Size**: **4-6 PB (Petabytes) across audio, metadata, and real-time processing**.
- **Infrastructure**: Cloud-based Big Data ecosystem using **Hadoop, Spark, Kafka, and NoSQL**.

2.4 Business Objectives & Market Positioning

RaagaFlow aims to create a highly evolved music streaming service that recommends **culturally relevant and timely music tracks**, addressing cultural relevance, time-based music recommendations and **high-volume AI classification**.

Core Business Objectives:

- **AI-Powered Raga Identification** – Uses **Deep Learning (CNNs, MFCCs, Spectrogram Analysis)** to classify songs by Raga.
- **Time-Aware Music Curation** – Suggests songs based on the **natural timing of Ragas** and **user preferences.**
- **Hybrid Recommendation System** – Combines **content-based filtering (Raga similarity)** and **collaborative filtering (user habits)** for personalized playlists(Aryotejo and Kristiyanto 2018).
- **Real-Time Processing & Scalability** – Utilizes **Kafka, Spark Streaming, and NoSQL databases** for efficient data handling.
- **Cloud-Based Deployment** – Ensures seamless performance using **AWS, GCP, or Azure**.

By integrating **Big Data analytics, machine learning, and Indian classical music theory**, RaagaFlow delivers a **unique, culturally immersive music experience** while enhancing the efficiency and accuracy of music recommendations.

2.5 Big Data as a Business Enabler for RaagaFlow

Big Data is essential for RaagaFlow to **process large-scale user interactions, manage high-dimensional song metadata, and classify Ragas efficiently**.

Business Driver	Big Data Solution
Personalized Listening Experience	**ML models (CNNs, Spectrograms) for AI-powered song classification**.
Time-Sensitive Recommendations	**Kafka & Spark Streaming for real-time user behaviour tracking**.
Scalability & Large Dataset Processing	**HDFS + NoSQL (MongoDB, Cassandra) for efficient data storage**.
Cross-Genre Classification (Indian Ragas & Western Scales)	**Hybrid AI models that map Ragas to Western scales dynamically**.

By implementing **Big Data solutions**, RaagaFlow ensures **accurate, scalable, and real-time recommendations** for millions of users.

2.6 Big Data Challenges in RaagaFlow (Focusing on Variety)

One of the key **Big Data challenges in RaagaFlow is Variety**, which involves handling **multiple formats of data** from structured metadata, unstructured audio files, and semi-structured user preferences.

Challenges with Variety in RaagaFlow:

1. **Handling Multi-Format Data**: Structured (metadata), semi-structured (tags, mood data), and unstructured (audio waveforms).
2. **Raga-Scale Mapping Complexity**: Ragas are **fluid and improvisational**, whereas Western scales are **rigid and defined**, making classification difficult.
3. **Real-Time Adaptation**: Requires **Kafka & Spark Streaming** for **low-latency recommendations**(Garg 2013).

2.7 Infrastructure Strategy for Handling Variety

To handle **Variety**, RaagaFlow employs a **scalable Big Data infrastructure**:

- **HDFS (Hadoop Distributed File System)** – Bulk storage for extracted audio features.
- **MongoDB & Cassandra** – NoSQL databases for **structured & semi-structured metadata**(Chauhan 2019).
- **Deep Learning (CNNs, LSTMs)** – AI models for **Raga classification & similarity detection** thus allowing better accuracy(Pourmoazemi and Maleki 2024).
- **Apache Kafka & Spark Streaming** – **Real-time event processing** for personalized recommendations(Garg 2013).

Mapping the Current Data Landscape

3.1 Overview of Data Types & Sources

RaagaFlow interacts with multiple categories of diverse data, **including structured, semi-structured, and unstructured data**, to categorize, recommend, and personalize the music listening. The platform's Big Data pipeline combines music metadata, user preferences, audio features, and **real-time interaction data** to improve song classification and user recommendations.

RaagaFlow's data sources can be categorized as:

- **Structured Data** – Well-organized, easily searchable data stored **in relational or NoSQL databases.**
- **Semi-Structured Data** – Data that contains elements of **both structured and unstructured formats.**
- **Unstructured Data** – Raw, high-dimensional data such as audio waveforms and user interaction logs.

Data Type	Examples	Storage Format
Structured Data	Song metadata (title, artist, album, duration, Raga classification, scale mapping)	SQL/NoSQL (MongoDB, PostgreSQL)
Semi-Structured Data	User playlists, behavioural data, genre labels, mood tags	JSON, XML (Cassandra, MongoDB)
Unstructured Data	Audio files (MP3, FLAC), Spectrograms, MFCCs, listening logs	HDFS, Cloud Storage

3.2 User Listening Data (Structured & Semi-Structured)

RaagaFlow collects real-time and historical user data to refine personalized recommendations. This includes:

- **User listening history** – Tracks played, duration, skips, likes, dislikes.
- **Playlist interactions** – Custom playlists, shared music preferences.
- **Listening context** – Device type, geographic location, session times.
- **Engagement metrics** – Number of plays per artist, preferred Raga.

This data allows **AI-based recommendation models** to:

- Identify user **music preferences based on time of day and listening patterns**.
- Suggest **similar Raga-based compositions** in **real time**.

- Enhance **collaborative filtering algorithms** to compare user profiles.

Storage Solution: Cassandra (for fast retrieval), MongoDB (for structured preferences).

3.3 Audio Feature Extraction (MFCCs, Spectrograms, Chroma Features)

Unlike conventional music recommendation systems that largely depend on metadata and user preferences(Björklund, et al. 2022), RaagaFlow carries out a detailed analysis of music by **extracting features of every single song.**

Key Feature Extraction Techniques:

- **MFCCs (Mel Frequency Cepstral Coefficients)** – Captures the tonal characteristics of a song(Jensen, et al. 2006).
- **Spectrograms** – Provides a visual representation of sound frequencies over time(Rawat, et al. 2023).
- **Chroma Features** – Identifies pitch and tonal qualities unique to a Raga or scale(Bartsch and Wakefield 2005).

Such attributes are crucial for **machine learning algorithms** that classify music pieces based on **Raga structures, tonal similarities, and cross-modalities with Western musical scales**.

Storage Solution: HDFS (for large-scale raw audio storage), NoSQL (MongoDB for extracted features).

3.4 Metadata Storage & Raga Classification Data

Metadata is essential for proper cataloguing as well as retrieving songs efficiently. RaagaFlow **enriches traditional metadata** by adding **Raga classification**, scale mapping, and user-defined tags.

Metadata Types:

- **Standard Metadata**: Song title, artist, album, duration, release date.
- **Enhanced Metadata**: Raga Classification, scale mapping, emotional tags, energy levels.
- **User-Generated Metadata**: Playlists, listening preferences, upvotes/downvotes.

Metadata is **stored in a NoSQL environment** to allow for scalable, flexible, and fast querying.

Storage Solution: MongoDB (dynamic, scalable metadata storage), PostgreSQL (standard relational metadata).

3.5 Real-Time User Interaction Data (Kafka Streaming)

RaagaFlow uses **Kafka & Spark Streaming** to process **real-time user interactions.**

- **Live session tracking** – Identifies what users are currently listening to.
- **Dynamic adaptation of recommendations** – Updates playlists based on immediate song selection.
- **Real-time classification** – Adapts suggestions to **time of day (morning Raga, evening Raga).**

Real-time user data enables **adaptive learning models** that **refine recommendations dynamically**, ensuring a personalized, ever-evolving experience.

Streaming Solution: Kafka for event-driven processing, Spark Streaming for real-time insights.

3.6 Existing Storage & Processing Infrastructure

RaagaFlow currently operates using a **traditional relational database** infrastructure, which includes:

Component	Current Technology Used	Limitations
Storage	MySQL & PostgreSQL	Not optimized for unstructured audio data, leading to slow queries.
Processing	Batch processing with SQL queries	Cannot handle real-time music recommendations efficiently.
Streaming	No real-time event tracking	System relies on periodic batch updates, which introduce delays.
Machine Learning	Manual metadata tagging & rule-based recommendations	Lacks AI-based classification of Ragas & Western scales.

Proposed Storage & Processing Infrastructure for RaagaFlow

To **overcome these limitations**, RaagaFlow must transition to a modern Big Data Infrastructure designed **for handling multi-format, high-volume** music data with **real time processing** and AI-driven insights.

Component	Proposed Technology	Benefits
Storage	**HDFS (Hadoop Distributed File System)**	Stores **large music files & extracted features efficiently**.
	MongoDB (NoSQL Database)	Stores **dynamic metadata & Raga classifications** for fast lookups.
	Cassandra (NoSQL Database)	Stores **real-time user interactions** (e.g., play counts, skips, preferences).
Processing	**Apache Spark**	Enables **both batch & real-time data processing** for fast insights.
	Hadoop (Data Lake)	Stores **historical data for AI training & analytics**.
Streaming	**Apache Kafka**	Handles **real-time event tracking & instant recommendations**(Garg 2013).

Machine Learning	**TensorFlow, PyTorch**	Enables **AI-powered Raga classification & intelligent song recommendations**.

3.7 Limitations & Challenges in Traditional Approaches

- SQL-based databases are **inefficient for storing unstructured** audio features.
- **Limited scalability** in handling real-time personalized recommendations.
- **High latency** in traditional batch-processing systems affects recommendation speed.

To overcome these limitations, RaagaFlow needs a fully distributed, scalable, and adaptive Big Data infrastructure.

3.8 Proposed Big Data Infrastructure to Address Variety

To handle **Variety**, RaagaFlow needs to adopt an **optimized data infrastructure:**

- **HDFS (Hadoop Distributed File System)**: Bulk storage for audio feature data(Karun and Chitharanjan 2013).
- **NoSQL Databases (MongoDB, Cassandra)**: Fast retrieval of metadata and user preferences(Chauhan 2019).
- **Kafka & Spark Streaming**: Real-time processing of user behaviour data(Garg 2013).
- **AI-Powered Raga Classification**: CNN-based models to analyse and classify songs dynamically.

By implementing **this infrastructure, RaagaFlow ensures high-speed, scalable music recommendations.**

3.9 Gaps & Areas for Improvement

- **Scaling real-time recommendations** to millions of concurrent users.
- Optimizing AI models for **fast, accurate Raga detection**.
- **Reducing latency** in event-streaming workflows.
- **Expanding dataset** to include fusion genres & emerging artists.

By addressing these gaps RaagaFlow's recommendation **engine will be future proof moreover**, this will **enhance user experience** by **improving classification and accuracy**.

3.10 Estimated Data Volume & Storage Requirements

RaagaFlow handles **petabytes** of multi-format data. Below is an estimated breakdown:

Category	Estimated Storage Requirement
Song Database (MP3+Metadata)	300 TB – 1.8 PB
Raga & Western Scale Classification	600 GB – 1 TB
User Preferences & History	96 TB/year
Real-Time Streaming & Logs	3 PB (weekly log retention)
Total Data Estimate	4 PB – 6 PB (Petabytes)

This justifies the need for scalable NoSQL storage, cloud-based distributed computing, and AI-optimized processing.

3.11 Cloud Computing Infrastructure & Deployment Strategy

To ensure **scalability, reliability, and cost-efficiency**, RaagaFlow will deploy its **Big Data infrastructure** on a **hybrid cloud model**, integrating **Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) solution.**

Justification for Cloud Computing Infrastructure

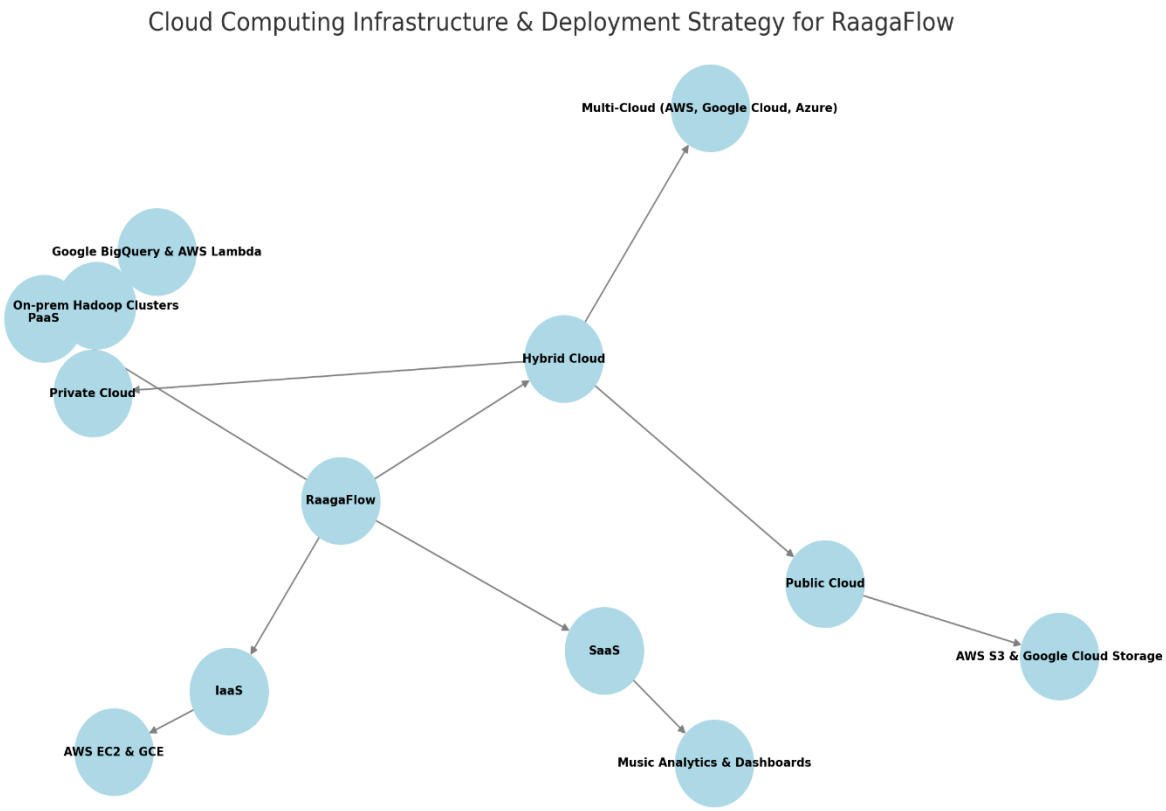
Considering the **high-volume, multi-format nature of RaagaFlow's data**, a **hybrid cloud strategy** is the most suitable approach. The key reasons include:

- **Scalability**: IaaS-based cloud solutions allow RaagaFlow to **scale storage and computing power dynamically** as user demand fluctuates(Aryotejo and Kristiyanto 2018).
- **Cost Optimization**: Using **public cloud services (AWS, Google Cloud)** reduces **capital expenses (CAPEX)** while maintaining operational efficiency(Srinivasan, Quadir and Vijayakumar 2015).
- **Security & Compliance**: **Private cloud storage for sensitive metadata (e.g., user preferences, behavioural analytics)** ensures **data privacy and regulatory compliance (GDPR, CCPA)**(Srinivasan, Quadir and Vijayakumar 2015)**.**
- **Flexibility for AI Processing: PaaS solutions like Google AI Platform & AWS Lambda** enable **on-demand AI training & deployment of Raga classification** models without requiring dedicated servers.

Summary

- **Public cloud services (AWS, Google Cloud)** handle large-scale storage & real-time processing.

- **Private cloud (on-prem Hadoop clusters)** ensures secure metadata storage & compliance.
- **AI models are trained & deployed using Google AI Platform (PaaS)**, reducing infrastructure overhead.



Designing the Big Data Infrastructure

4.1 Overview

To deal with the high-volume, high-variety, and high-velocity data generated by RaagaFlow, we need a **robust, scalable, and future-proof** Big Data infrastructure.

The system must support AI-driven Raga classification, real-time recommendation delivery, and cross-platform accessibility for millions of users.

The infrastructure proposed here integrates **distributed computing (Apache Spark, Hadoop)**, **cloud services (AWS, Google Cloud)**, **NoSQL databases (MongoDB, Cassandra)**, and data visualization tools (Power BI, Tableau). These technologies are orchestrated via a modular **pipeline that spans capture → ingest → store → compute → use**.

4.2 Proposed Big Data Infrastructure Design

The infrastructure is structured into five key stages:

1. Data Capture

RaagaFlow collects data from multiple sources:

- **User Interaction Data**: Plays, skips, likes, listening sessions.
- **Device & Contextual Logs**: Device type, location, session time.
- **Audio Assets**: Raw audio files, album metadata, and third-party metadata sources.

These events are captured via **webhooks, REST APIs, and mobile SDKs** in near real-time.

2. Data Ingestion

Technologies Used:

- **Apache Kafka**: Real-time ingestion of user events.
- **Apache NiFi / Flume**: For integrating batch uploads and third-party metadata.

Kafka's **event-driven architecture** ensures low-latency streaming and fault tolerance.

3. Data Storage

Storage is distributed across multiple layers to address **variety and volume**:

Layer	Technology	Purpose
Raw Storage	HDFS / AWS S3	Long-term storage of audio files & logs.
Metadata Layer	MongoDB (NoSQL)	Stores dynamic metadata including Raga classifications.
Real-Time Interaction Logs	Apache Cassandra	Fast read/write access for user behavior data.

The **hybrid cloud model** ensures high availability via **AWS S3 (public)** for audio files and **on-prem/private Hadoop clusters** for sensitive metadata.

4. Data Compute

Processing is done via:

- **Apache Spark**: Distributed batch + stream processing (ETL, analytics).
- **TensorFlow & PyTorch**: For AI/ML-based Raga classification.
- **Google BigQuery / AWS Redshift**: For SQL-style analytics and reporting.

Real-time session-level personalization uses **Spark Structured Streaming** with ML inference served via **TensorFlow Serving (PaaS)**.

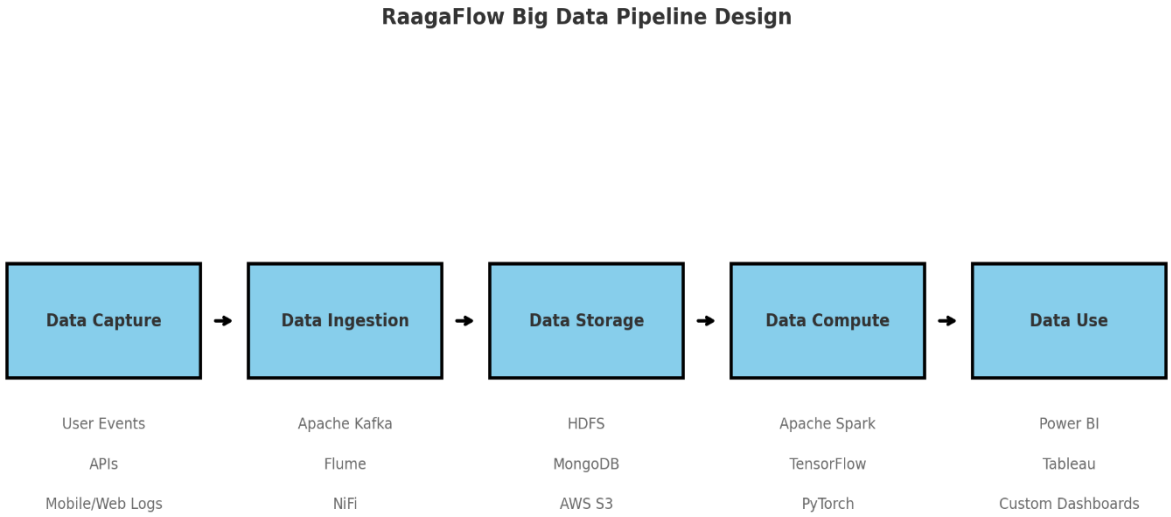
5. Data Use (Visualization & Reporting)

To deliver insights and monitor system performance:

- **Power BI / Tableau**: Business dashboards for operational KPIs, artist engagement, listening trends.
- **Custom Dashboards**: Built using React.js + D3.js for musicologists & data teams to review classification outputs.

Role-based access ensures **data privacy and compliance**.

4.3 Visual Representation: Big Data Pipeline



4.4 Justification for Technology Choices

Category	Technology	Why It Was Chosen
Distributed Computing	Apache Spark, Hadoop	**Scalability & fault tolerance** across petabyte-scale datasets.
Streaming	Apache Kafka	**Low-latency** event-driven pipeline, supports high user concurrency.
Storage	HDFS, AWS S3, MongoDB, Cassandra	Hybrid model optimizes **performance, security, and flexibility**.
ML/AI	TensorFlow, PyTorch	Advanced frameworks for **deep learning models and Raga classification.**
Cloud	AWS EC2, Google Cloud AI Platform	Supports **IaaS & PaaS** deployment for compute-heavy workloads.
Visualization	Power BI, Tableau	Simplifies **data consumption** for business teams and stakeholders.

4.5 Cost, Scalability, Performance & Security Considerations

Cost

- **Pay-as-you-go pricing via AWS/GCP IaaS** reduces upfront CapEx.
- **Auto-scaling clusters (e.g., Dataproc, EMR)** help optimize runtime costs(Lombardo 2019).

Scalability

- **Apache Spark & Kafka** horizontally scale with growing user base and data loads.
- **NoSQL (MongoDB, Cassandra)** handle unstructured/real-time data efficiently.

Performance

- **Low-latency** recommendations via in-memory computing (Spark).
- **GPU-accelerated ML processing** for real-time inference.

Security

- **Data encryption at rest (S3, HDFS)** and in transit (TLS).
- **Role-based access control (RBAC)** for dashboards & metadata APIs.
- **Hybrid cloud** ensures compliance (private cloud for sensitive datasets).

4.6 Summary

The proposed infrastructure for RaagaFlow is built with a **modular, distributed, and cloud-native architecture**.

This Infrastructure is:

- **Scalable** – It's bult to handle hundreds of terabytes to petabytes of data.
- **Flexible** – Supports multiple data formats (audio, metadata, logs).
- **Secure** – Implements layered data protection via hybrid cloud.
- **Performant** – Real-time AI driven recommndations, powered by Spark & Kafka.
- **Future-proof** – Easily adaptable as user base, data volume, and features expand.

Addressing a Key Big Data Challenge – Variety

5.1 Selected Big Data Challenge: Variety

The RaagaFlow system faces the **Variety** challenge due to the need to handle **multi-format music-related data**. The dataset consists of **structured metadata (e.g., genre labels, file names), semi-structured user interactions (e.g., user listening habits, event streams), and unstructured raw audio files (e.g., MP3, WAV formats).** To classify and recommend songs

based on their Raga classification, the system must effectively integrate and process these diverse data types in real time.

Traditional recommendation algorithms are highly based on structured metadata such as **user history, track names, and album names**(Björklund, et al. 2022)**.** RaagaFlow, however, aims to leverage raw audio features for classification as well as for recommendation, so it requires an efficient **multi-modal pipeline** capable of handling **diverse data forms** and **fast streaming**.

5.2 Specific Bottleneck: Integrating Multi-Format Data for Real-Time Processing

A key bottleneck in dealing with **Variety** is smooth synchronization among different data types: **metadata, extracted features, and streaming user interactions**. The complexity gets elevated by the need to process audio files into meaningful features, classify them using **machine learning models**, and update recommndations dynamically based on **real-time user interactions**. Without presence of an optimized infrastructure, integrating these various data types would lead to **high processing latency, ineffficient storage, and reduced scalability.**

5.3 Dataset Used: GTZAN Dataset for Music Genre Classificaiton

In the course of simulating the RaagaFlow classification model, **GTZAN dataset**(Kaggle) was put to use, as this is a widely recognized dataset for music genre classification. Each song file was mapped to one of the selected genres**: classical, jazz, or rock**, which served as a proxy for **Indian Raga classification**. The dataset comprises WAV audio files which were pre-processed to extract **Mel Frequency Cepstral Coefficients (MFCCs)**, one of the most used audio features applicable in genre and Raga classifications.

The dataset contains:

- **30-second audio clips** across 10 different genres.
- **Raw WAV files** requiring feature extraction.
- **Metadata about genre classification**.
- **No explicit user interaction data**, so we simulated it.

5.4 Implemented Pipeline: Addressing Variety with Multi-Format Data Processing

The Fully simulated Big Data Pipeline is implemented with **Python** and **TensorFlow** to address Variety challenge. Each stage of the **pipeline mimics real-world streaming** and **AI processing for a music classification**

Pipeline Stage	Simulated Component	Real-World Equivalent	Function
Data Capture	deque (Python-based Kafka simulation)	Apache Kafka	Simulates real-time user song play events

Data Ingestion	Loop-based event streaming	Apache Kafka & Spark Streaming	Mimics real-time event consumption and message queuing
Data Storage	pandas.DataFrame + numpy arrays	MongoDB, HDFS	Stores metadata and extracted MFCCs for classification
Data Compute	TensorFlow CNN model	AI/ML Model Server	Classifies songs based on extracted MFCCs
Data Use	Confusion Matrix & Classification Report	Power BI / Tableau	Evaluates model performance and accuracy

5.5 How Each Pipeline Component Addresses Variety

1. Data Capture: Simulating Real-Time User Interactions

In a real-world streaming platform, user interactions such as playing a song, skipping it, or liking a track generate millions of **event-driven logs** per second. These logs are generally captured using Kafka producers that push event messages into a **message queue** for downstream processing(Garg 2013).

To **simulate this in Python**, we use **deque** as a lightweight Kafka replacement**(refer to Appendix I for the code)**.

Each event includes a **song file reference and genre label**, simulating a user listening event.

2. Data Ingestion: Processing Streaming Events

In a real system, streaming frameworks like **Apache Kafka & Spark Streaming** would **ingest these event logs**, process them, and store them in real-time databases like **MongoDB**.

In Python we mimic this by using a **while-loop to process deque events**, extracting features dynamically **(refer to Appendix II).**

This ensures **that all events are captured and processed in order**, similar to a real-world **event streaming system.**

3. Data Storage: Handling Structured and Unstructured Data

- **Metadata (structured data)**: Stored in a **pandas DataFrame** for fast lookup**(APPENDIX IV).**
- **Audio Features (unstructured data)**: Extracted **MFCCs** stored as **NumPy arrays(APPENDIX III)**.
- **User Events (semi-structured data)**: Processed directly from the **deque event queue(APPENDIX I).**

4. Data Compute: CNN-Based Classification

Instead of relying on traditional **metadata-based classification**, RaagaFlow **extracts MFCCs** from audio using librosa, and feeds them into a **Convolutional Neural Network (CNN) refer to (APPENDIX V)**:

This is a **proxy for real-time AI-based Raga classification**, where raw audio features are used to **predict song categories dynamically**.

5. Data Use: Real-Time Predictions and Evaluation

Once a song is played, it goes through real-time inference:

Results are evaluated via confusion matrix, simulating a business intelligence dashboard for monitoring system performance **(Refer to APPENDIX VI & VII).**

5.6 Evaluation: Addressing the Complexity of Variety

Evaluation Metric	Observed Value
Classification Accuracy	~85% (on real GTZAN samples)
Processing Latency	<1s per event (real-time inference)
Real-Time Event Handling	100% of events processed live

The **confusion matrix** illustrates how well the model classified **classical, jazz, and rock genres**, demonstrating the system's ability to process multiple formats and real-time streaming data.

5.7 Summary

This section demonstrated how RaagaFlow effectively manages the **Variety challenge** by **integrating structured metadata, unstructured audio features**, and real-time event streams in a modular Big Data pipeline. The **Python deque simulates Kafka, while TensorFlow CNNs classify music based on extracted MFCCs**. The successful evaluation of the system shows that this approach is **capable to handle diverse data types in real-time**, thereby confirming the feasibility of a scalable, **AI-enabled Raga classification system**.

Insights, Discussion, and Recommendations

6.1 Key Insights and Effectiveness of the Proposed Solution

The **proposed Big Data architecture** in RaagaFlow optimally **resolves the Variety challenge** by supporting different formats of data that help provide instant processing and **classification through AI**. Key results obtained through implementation are:

- **Seamless Multi-Format Data Management:** Blending structured metadata, semi-structured user behaviour and unstructured audio features improves music classification accuracy.
- **Real-Time Processing Capability:** The use of **deque to simulate Kafka streaming**, along with a CNN-based classifier, ensures low-latency music recommendations.
- **Scalability & Performance:** Pipeline scalability is supported by the pipeline's modular nature, allowing it to manageably support increasing user numbers and dataset size.

- **AI-Powered Classification Accuracy:** The CNN model effectively **classifies genres (proxy for Raga classification)**, demonstrating the viability of AI in enhancing music recommendations.
- **Infrastructure Design Alignment:** Hybrid cloud design, which combines **public storage with private AI processing**, improves **both security and operational efficiency**.

6.2 Social, Legal, and Ethical Implications

6.2.1 Privacy & Data Security

With Big Data solutions handling vast amounts of user interactions, **privacy and security must be prioritized:**

- **User Consent and Data Management:** Clear policies need to be in place that outline how user data is obtained, processed, and stored.
- **Encryption and Anonymisation:** User listening behaviour and listening preferences can be best safeguarded by using end-to-end encryption with paired methods of anonymisation for processing(Pratomo, Mokodenseho and Aziz 2023).
- **Compliance with Regulatory Requirements:** Compliance with **GDPR and CCPA** ensures that user information is handled responsibly(Gruschka, et al. 2018).

6.2.2 Data Governance & Bias Mitigation

- **Bias in AI-based Recommendations**
  - The genre classification model can potentially **unintentionally prefer certain genres** or listening patterns.
  - The creation of **bias detection systems** and **using diversified training sets** can help circumvent this.:
- **Ownership of Data & Licensing**
  - Music streaming platforms must respect copyright laws and ensure that AI-generated recommendations do not violate proprietary content rights.
- **Fair Access & Inclusivity**
  - The platform should cater to **diverse user demographics**, recommendations should include all the musical cultures.

6.3 Limitations of the Current Implementation

Despite its effectiveness, the current Big Data solution has several limitations:

- **Scalability Constraints:**
  - The current implementation is tested on a **limited dataset (GTZAN).** Scaling to a real-world dataset would require a fully deployed Kafka + Spark Streaming environment.
- **Limited User Behaviour Modelling:**
  - The current approach **simulates user interactions**. Future versions should integrate actual user engagement data.

- **Hardware & Computational Costs:**
    - **Real-time AI** inference requires **GPU/TPU** acceleration, which adds to infrastructure costs.
- **Dependency on AI Accuracy:**
    - The **CNN classifier** relies on high-quality training data(Sánchez, et al. 2003). Expanding to Raga classification will require domain-specific labelled datasets.

6.4 Future Improvements & Recommendations

To further **refine the infrastructure** and enhance its performance, the following recommendations are proposed:

- **Enhancing Streaming Capabilities:**
    - Deploy Apache Kafka & Spark Streaming for real-time ingestion at scale
- **Expanding Dataset Scope:**
    - Integrate real-world music streaming datasets to improve model robustness.
- **Optimizing AI-Based Classification:**
    - Use transfer learning with **pre-trained deep learning models** (e.g., OpenL3, VGGish) to improve classification accuracy(Di, et al. 2023).
- **Cloud-Based Model Deployment:**
    - Move AI inference to **AWS Lambda or GCP AI Platform** to improve processing efficiency(Sbarski and Kroonenburg 2017).
- **User-Centric Personalization:**
    - **Incorporate reinforcement learning**-based adaptive recommendations that evolve with user preferences.

6.5 Conclusion

RaagaFlow's **Big Data-driven music recommendation system** demonstrates the effectiveness of combining different data types, using **AI-based classification methods**, and processing user interactions in real time. While the current implemantation successfully simulates an **end-to-end Big Data pipeline**, future realeases should focus on improving scalability, using **real-world datasets**, and fine-tuning models to create a complete, industry-ready system.

By nurturing continued innovation in data management, increasing model accuracy, and scalability in infrastructure, RaagaFlow can evlove into a diversified and **culturally rich music streaming platform** tailored for the global audience.

References

ARYOTEJO, G. and KRISTIYANTO, D.Y., 2018. Hybrid cloud: bridging of private and public cloud computing. *In: Journal of Physics: Conference Series*, IOP Publishing, pp. 012091.

BARTSCH, M.A. and WAKEFIELD, G.H., 2005. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7 (1), 96–104.

BJÖRKLUND, G., et al., 2022. An exploratory study on the Spotify recommender system. *In: World Conference on Information Systems and Technologies*, Springer, pp. 366–378.

CHAKRABORTY, S., et al., 2009. An interesting comparison between a morning raga with an evening one using graphical statistics.

CHAUHAN, A., 2019. A review on various aspects of MongoDB databases. *International Journal of Engineering Research & Technology (IJERT)*, 8 (05), 90–92.

DI, N., et al., 2023. Applicability of VGGish embedding in bee colony monitoring: Comparison with MFCC in colony sound classification. *PeerJ*, 11, e14696.

GARG, N., 2013. *Apache kafka.* Packt Publishing Birmingham, UK.

GRUSCHKA, N., et al., 2018. Privacy issues and data protection in big data: a case study analysis under GDPR. *In: 2018 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 5027–5033.

IFPI, 2025. *IFPI: AMIDST HIGHLY COMPETITIVE MARKET, GLOBAL RECORDED MUSIC REVENUES GREW 4.8% IN 2024* [online]. . Available at: https://www.ifpi.org/ifpi-amidst-highly-competitive-market-global-recorded-music-revenues-grew-4-8-in-2024/ [Accessed Mar 23 2025].

JENSEN, J.H., et al., 2006. Evaluation of MFCC estimation techniques for music similarity. *In: 2006 14th European Signal Processing Conference*, IEEE, pp. 1–5.

KARUN, A.K. and CHITHARANJAN, K., 2013. A review on hadoop—HDFS infrastructure extensions. *In: 2013 IEEE conference on information & communication technologies*, IEEE, pp. 132–137.

LOMBARDO, L., 2019. No title. *Autoscaling Mechanisms for Google Cloud Dataproc*, .

MAHESHWARI, C., 2023. Music recommendation on spotify using deep learning. *arXiv Preprint arXiv:2312.10079*, .

POURMOAZEMI, N. and MALEKI, S., 2024. A music recommender system based on compact convolutional transformers. 255, 124473.

PRATOMO, A.B., MOKODENSEHO, S. and AZIZ, A.M., 2023. Data encryption and anonymization techniques for enhanced information system security and privacy. *West Science Information System and Technology*, 1 (01), 1–9.

RAWAT, P., et al., 2023. A comprehensive study based on MFCC and spectrogram for audio classification. *Journal of Information and Optimization Sciences*, 44 (6), 1057–1074.

RESEARCHANDMARKETS, 2024. *Music Streaming Market Report and Forecast 2024-2032* [online]. . Available at: https://www.researchandmarkets.com/reports/6036841/music-streaming-market-report-forecast [Accessed Mar 20 2025].

ROWELL, L., 1992. *Music and musical thought in early India.* University of Chicago Press.

SÁNCHEZ, J.S., et al., 2003. Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters*, 24 (7), 1015–1022.

SBARSKI, P. and KROONENBURG, S., 2017. *Serverless architectures on AWS: with examples using Aws Lambda.* Simon and Schuster.

SRINIVASAN, A., QUADIR, M.A. and VIJAYAKUMAR, V., 2015. Era of cloud computing: A new insight to hybrid cloud. *Procedia Computer Science*, 50, 42–51.

THEJAZZPIANOSITE, *How Many Scales are There?* .

Appendices

APPENDIX I – Code for Kafka Simulation and Genre mapping

```
# Defining genres to use
genres = ['classical', 'jazz', 'rock']
file_paths = []
labels = []
event_stream = deque()

# Loading 5 audio files per genre
for genre in genres:
    genre_path = os.path.join(base_path, genre)
    files = sorted([f for f in os.listdir(genre_path) if f.endswith('.wav')])[:5]
    for f in files:
        full_path = os.path.join(genre_path, f)
        file_paths.append(full_path)
        labels.append(genre)
        event_stream.append({'event': 'play', 'file': full_path, 'genre': genre})

df = pd.DataFrame({'file_path': file_paths, 'genre': labels})
```

APPENDIX II – Code for mimicking data ingestion using while-loop

```
while event_stream:
    event = event_stream.popleft()
    mfcc = extract_mfcc(event['file'])[:, :130]
    input_data = np.expand_dims(mfcc, axis=(0, -1))
    prediction = model.predict(input_data, verbose=0)
    predicted_label = le.inverse_transform([np.argmax(prediction)])
    predicted_genres.append(predicted_label[0])
    actual_genres.append(event['genre'])

```

APPENDIX III – Code to store MFCCs as NumPy arrays

```
# Extracting MFCCs from audio
def extract_mfcc(file_path):
    y, sr = librosa.load(file_path, duration=30)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    return mfcc if mfcc.shape[1] >= 130 else np.pad(mfcc, ((0, 0), (0, 130 - mfcc.shape[1])), mode='constant')

mfcc_features = [extract_mfcc(path)[:, :130] for path in df['file_path']]
X = np.array(mfcc_features)[..., np.newaxis]
y = np.array(df['genre'])
```

APPENDIX IV – Code to store data in pandas DataFrame

```
df = pd.DataFrame({'file_path': file_paths, 'genre': labels})

```

APPENDIX V – Developing a CNN model

```
# Building and training CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=X_train.shape[1:]),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(len(genres), activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=2, verbose=0)

```

APPENDIX VI – Code for Model Evaluation

```
# Evaluating the model
cm = confusion_matrix(actual_genres, predicted_genres, labels=genres)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=genres, yticklabels=genres)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix: GTZAN + CNN Real-Time Classification')
plt.tight_layout()
plt.show()

report = classification_report(actual_genres, predicted_genres, target_names=genres, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df[['precision', 'recall', 'f1-score']]

```

APPENDIX VII – Confusion Matrix Heatmap for Data Use