International Journal of Scientific Research in Engineering & Technology
Volume5, Issue2 (March-April 2025), PP: 01-07.
https://www.doi.org/10.59256/ijsreat.20250502001
www.ijsreat.com ISSN No: 2583-1240

A Scalable System Design for Real-Time Personalized Recommendation Engines in E-Commerce

Karthik Kamarapu[1], Kali Rama Krishna Vucha[2]
[1]*Independent Software Researcher, Osmania University, Hyderabad, Telangana, India.*
[2]*Independent Software Researcher, Acharya Nagarjuna University, Guntur, Andhra Pradesh, India.*

To Cite this Article: *Karthik Kamarapu[1], Kali Rama Krishna Vucha[2], "A Scalable System Design for Real-Time Personalized Recommendation Engines in E-Commerce", International Journal of Scientific Research in Engineering & Technology, Volume 05, Issue 02, March-April 2025, PP: 01-07.*

Abstract: *The increasing demand for real-time personalization in e-commerce has highlighted the need for recommendation engines that are both scalable and efficient. Traditional systems often rely on centralized architectures that challenge such as scalability, latency and safeguarding user data privacy. This study introduces a novel design for a real-time personalized recommendation engine that operates within a distributed microservices framework by integrating multiple approaches such as collaborative filtering, content-based filtering, and reinforcement learning while also addressing privacy concerns using federated learning and differential privacy techniques. At the same time, by leveraging real-time data streaming and advanced caching mechanisms, the proposed system delivers low-latency recommendations and makes it highly suitable for large-scale e-commerce applications. Experimental results indicate that this approach significantly outperforms conventional centralized systems in scalability, accuracy of recommendations and response times.*

Key Word: *Scalable recommendation system, Federated learning, differential privacy, Real-time personalization, E-commerce architecture*

1. INTRODUCTION

Personalized recommendation systems became essential for enhancing user engagement and driving revenue in e-commerce platforms. This system improves user experience by providing tailored product suggestions. Traditional recommendation engines rely on centralized architectures that sometimes do not meet the demands of modern platforms with limitations in scalability, latency and data privacy. But with the exponential growth of e-commerce, there is an urgent need for a system that satisfies the criteria of scalability, latency and data privacy.

Several existing works have contributed significantly to the field of recommendation systems. Collaborative filtering techniques such as those introduced by Koren et al. [1] is widely adopted for their ability to leverage user-item interactions. However, these methods are limited by their reliance on centralized data aggregation that poses privacy risks and scalability challenges. He et al. [2] extended collaborative filtering with neural network-based models that demonstrated improved accuracy but at the cost of increased computational complexity and latency.

Liu et al. [3] explored the integration of differential privacy into recommendation systems, providing strong privacy guarantees but without addressing the need for real-time recommendations. Federated learning, as investigated by Yang et al. [4], enables decentralized model training across distributed nodes, reducing privacy risks. However, these systems often lack the adaptability required for dynamic user behavior.

Reinforcement learning-based recommendation systems such as those proposed by Zhao et al. [5] have shown promise in adapting to user preferences in real-time. Despite their potential, these systems are rarely integrated into a scalable and privacy-preserving architecture.

This paper proposes a scalable system design for a real-time personalized recommendation engine that integrates collaborative filtering, content-based filtering and reinforcement learning within a distributed microservices architecture. Unlike some of the existing works, our approach combines federated learning and differential privacy to ensure user data remains secure. Through extensive experimentation, we demonstrate that the proposed system achieves superior scalability, accuracy, and latency compared to traditional centralized approaches.

II.PROPOSED FRAMEWORK

The proposed framework for a real-time personalized recommendation engine is built on a scalable micro services architecture.

2.1 System Architecture

The framework uses a micro services-based architecture where each micro service has a specific functionality such as data ingestion, feature storage, model training and recommendation serving.

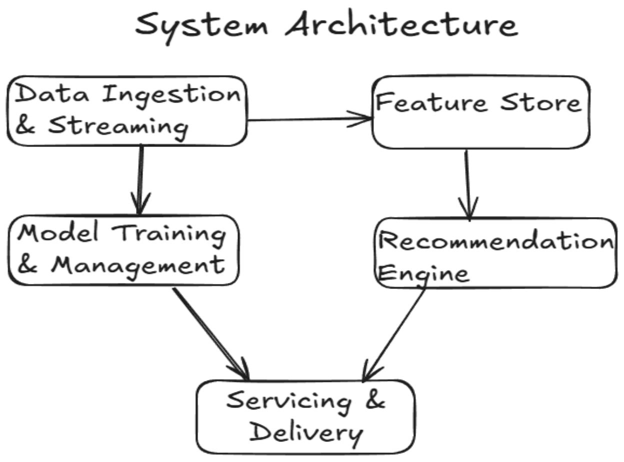

Fig 1. System Architecture

The data ingestion and streaming service collects real-time user interaction data such as clicks, purchases and search queries by utilizing Apache Kafka for high-throughput and fault-tolerant data ingestion. Kafka ensures all the incoming data is logged and streamed efficiently to subsequent services for processing. The feature store manages user and item features required for recommendation generation. This component is implemented using distributed NoSQL database Dynamo DB to handle high-speed operations.

The model training and management service is responsible for training both global and local recommendation models. For federated learning, local models are trained on user devices or edge nodes and encrypted updates are aggregated on the central server using tools like TensorFlow Federated. The recommendation engine combines collaborative filtering, content-based filtering and reinforcement learning algorithms to generate personalized suggestion. This engine interacts with the feature store to retrieve relevant user and item data before performing inference. Finally, the serving and delivery service provides low-latency recommendations through APIs and uses caching mechanisms such as Redis to precompute and store frequently accessed recommendations.

2.2 Data Ingestion and Streaming

This is an important part of the architecture that is responsible for capturing, processing and delivering real-time user interaction data to downstream services. This component ensures that system can handle high volumes of user activity while keeping latency low. The primary purpose of this layer is to capture real-time user interactions such as clicks, views, searches and purchases and transform them into structured data stream for further processing.

Implementation and Technologies

This service uses Apache Kafka for its ability to handle millions of events per second with low latency and high fault tolerance. This architecture consists of producers, brokers and consumers

Producers: Event Producers such as user-facing front-end applications and mobile apps send interaction data to Kafka topics. For example, when a user clicks on a product, the event is immediately captured by a JavaScript embedded in the client application and forwarded to Kafka producer.

Kafka Brokers: These are central nodes in Kafka cluster responsible for storing and managing data streams. Kafka partitions the incoming data based on topics e.g., user clicks, search queries, purchases and replicates them across brokers for fault tolerance.

Consumers: Downstream services such as the feature store or model training components consume the data form Kafka topics in real time. A Python based Kafka client, Confluent-Kafka, processes the data and prepares it for feature extraction.

Workflow

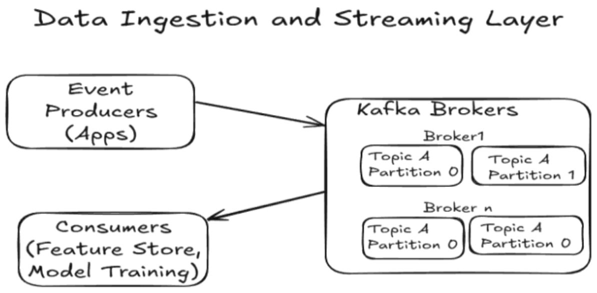

Fig 2. Data Ingestion and Streaming Layer

User interactions are captured in real time via SDKs or APIs embedded in the front-end application. The structured event is transmitted to Kafka producers which publish relevant topics. For example, a product click event would be sent to the user clicks topic. Kafka partitions data stream based on a unique key such as user id which ensures that all the vents related to a specific user are processed in order. Downstream services such as the feature store or model training service consume the data from Kafka topics. The services deserialize the data into usable formats such as Pandas Data Frames in Python for further processing.

Integration with the System

The data ingestion service seamlessly integrates with other components. Real-time feature updates are driven by the data streamed through Kafka. The Recommendation engine uses data to adapt to user behavior dynamically. The data is also utilized to train or fine-tune models that ensures the recommendations remain accurate and relevant. To ensure the ingestion services operate efficiently, the metrics such as throughput, latency time and error rates are monitored using tool Prometheus.

2.3 Feature Store

This is a critical component that ensures efficient management, storage and retrieval of user and item features. The feature store serves as the central repository for all users and item-related features where is readily accessible to all other components such as recommendation engine and model training service. This store maintains up-to-date profiles for users and items by ingesting data from the data ingestion layer and supports both real-time and batch access patterns.

Feature Storage and Management

The feature store is implemented using a distributed No SQL database – Dynamo DB which is efficient for high-speed operations and easy scalability. Features are categorized into two primary types:

User Features: Include demographic data e.g., age, gender and behavioral patterns e.g., browsing history, click-through rates and session-based attributes e.g., device type, location.

Item Features: Includes static attributes e.g., category, price, manufacturer and dynamic attributes e.g., inventory status, average rating.

For efficient storage and retrieval, the data is partitioned by User ID. This ensures all the features related to a specific user are co-located, minimizing access latency.

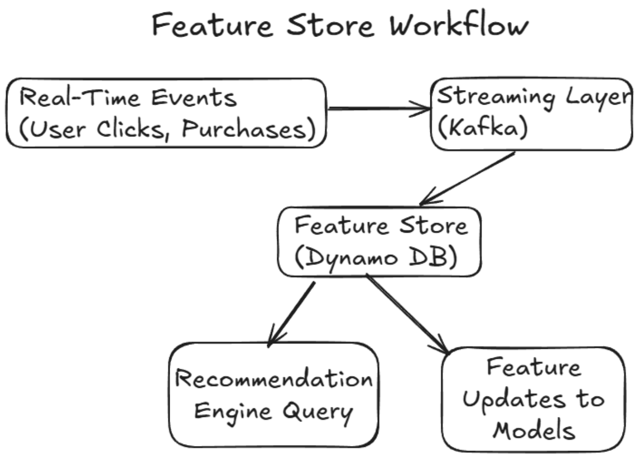
Feature Store Workflow
Real-Time Events (User Clicks, Purchases)
Streaming Layer (Kafka)
Feature Store (Dynamo DB)
Recommendation Engine Query
Feature Updates to Models

Fig 3. Feature Store Workflow

Real-Time Updates

To ensure that the features remain accurate and up to date, the features store integrate with the data ingestion layer. As user interactions are captured in real time the corresponding features are updated in the database. For example, A click event updates the user’s browsing history and click-through rate. Another example, where a purchase event modifies the inventory status and purchase frequency of the item. A combination of event-driven processing and streaming pipelines ensures these updates are propagated with minimal delays.

Feature Engineering and Transformation

The feature store is responsible for basic features transformations to ensure data consistency and usability. This includes transformations such as normalizing numerical features, encoding categorical features and aggregating user behavior. These transformations are performed as part of pre-query processing.

Integration with the Recommendation Engine

When the recommendation engine requires features to generate personalized suggestions, it queries the feature store for both user and item attributes. For example, when a user opens a product page, the recommendation engine retrieves the user’s browsing history, purchase patterns and device type along with the attributes of similar items. These features are then passed into

pre-trained models to generate recommendations. To optimize response times, frequently accessed features are cached in-memory using Redis.
Scalability and Fault Tolerance: The features storage is designed to handle many concurrent read and write operations. This is achieved through:
Horizontal Scaling: New nodes are added to the NoSQL cluster as data volume grows.
Replication: Data is replicated across multiple nodes to ensure fault tolerance.
Consistency Levels: The database configuration allows for eventual consistency for write-heavy workloads while reading queries prioritize fast responses.

2.4 Model Training and Management

This layer is responsible for training, updating and managing the recommendation models used by the system. This section focuses on how the component supports federated learning and ensures privacy-preserving mechanisms.

Federated Learning Framework

This framework is implemented to train models across distributed nodes without sharing raw user data. Each edge node such as a mobile device trains a local version of the model using its data. Only the model updates are shared with the central server for aggregation where it combines these updates to improve the global model.

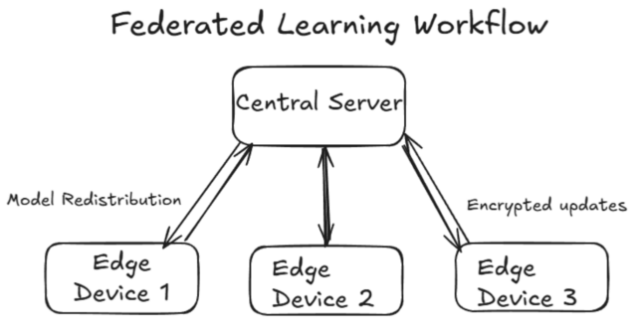

Fig 4. Federated Learning Workflow

The learning process follows the following steps:
Model Initialization: The central server initialized the global model and sends it to edge nodes for local training.
Local Training: Each edge node trains the model on its dataset producing updated model weights.
Update Aggregation: The central server collects encrypted updates from all nodes and aggregates them using algorithms such as Federated Averaging.
Global Model Update: The aggregated updates are applied to the global model which is redistributed to all the nodes.

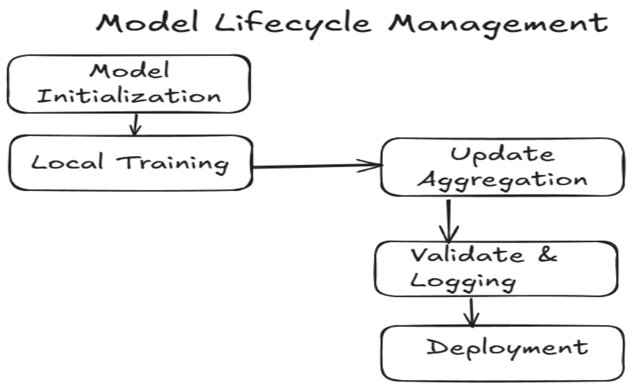

Fig. 5 Model Lifecycle Management

This framework is implemented using Tensor Flow Federated.

2.5 Privacy Mechanism and Real-Time Adaptation in Training

To ensure privacy during training, the following techniques are used to ensure compliance with data protections regulations such as GDPR and CCPA.

Differential Privacy: Noise is added to model updates at the edge nodes before they are sent to central server which ensures that individual contributions remain distinct while preserving the utility of the aggregated model

Homomorphic Encryption: Model updates are encrypted during transmission to prevent unauthorized access.
The training component supports continuous learning that allows the recommendation models to adapt to changing user behavior. For example, incremental training is performed on edge nodes as new user interactions are captured. The central server schedules periodic aggregation rounds to update the global model with recent data.

Model Management and Versioning: This component includes a robust model management system to handle multiple versions of the recommendation models which track the following:

Model Versions: each update to the global model is assigned a unique version that enables rollback if a specific version does not perform well.

Performance Metrics: Metrics such as accuracy, precision, recall and loss are logged for each version.

Deployment: Only models that meet predefined performance thresholds are deployed to the serving layer.
These functionalities are implemented using MLflow, a platform for managing machine learning lifecycles.

Recommendation Engine

The recommendation engine is the core system responsible for generating personalized suggestions based on user and item features that uses multiple recommendation techniques to ensure collaborative filtering, content-based filtering and reinforcement learning. This engine acts as the decision-making layer that combines insights from user interactions and item attributes to deliver tailored recommendations. By processing data from the feature store and leveraging trained models, this engine generates suggestions according to user preferences and browsing behavior.

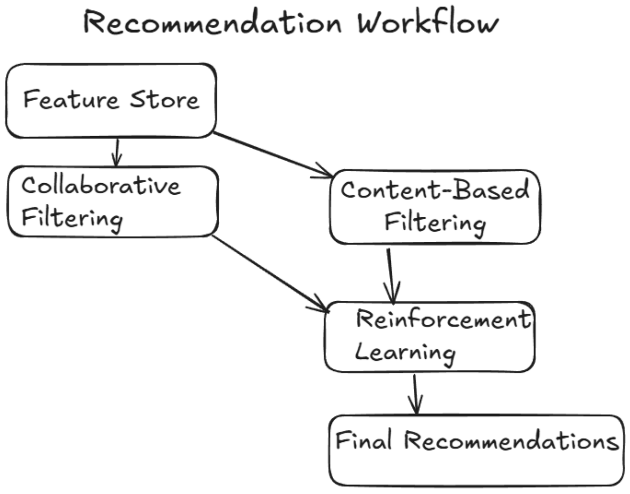

Fig. 6 Recommendation Workflow

2.6 Recommendation Techniques

This engine uses a hybrid approach by combining three key recommendation techniques:
Collaborative Filtering: This technique analyzes user-item interaction data to identify patterns and similarities. Matrix factorization is used to predict missing entries in the user-item interaction matrix that enables the system to recommend items that similar user have interacted with. For example, if User A and User B share similar preferences, then items preferred by User A but not explored by User B are recommended.

Content-Based Filtering: Content-based filtering leverages items such as category, price and textual descriptions to find items that match a user' profile. Text-based similarity measures such as cosine similarity are used to match user preferences with item descriptions. The approach ensures relevance for cold-start scenarios where new items lack interaction history.

Reinforcement Learning: Multi-armed bandit algorithms such Upper Confidence Bound (UCB) are used to dynamically adjust recommendations based on immediate user feedback. For example, if a user clicks on or purchases a recommended item, the system rewards the recommendation strategy.

2.7 Data Flow in the Recommendation Engine

The recommendation engine interacts with the features store and model management layer to retrieve relevant data and generate suggestions. In the workflow, as a first step when the engine receives a user request for recommendations, it queries the feature store to retrieve the user's profile including browsing history and preferences as well as item attributes. The retrieved data is passed into pre-trained collaborative filtering and content-based filtering models. The outputs from these models are combined and reinforcement learning dynamically adjusts the final recommendation list. The top N recommendations are delivered to the user through the serving layer.

To maintain relevance, the engine incorporates real-time adaptation mechanism. For instance, when a user interacts with a recommended item (e.g., clicks or purchases it), the engine immediately updates the user's profile and adjusts the future recommendations. This is achieved by incrementally updating user-item interaction matrices and re-weighting reinforcement learning strategies based on user feedback.

III.EXPERIMENTAL SETUPS AND RESULTS

The experimental setup outlines the data sources, system configuration, evaluation metrics and methodologies used to validate the scalability, accuracy and privacy-preserving capabilities of the recommendation engine.

Data Sources: The experiments are conducted using publicly available datasets that simulate real-world e-commerce interactions and user behavior.

Movie Lens 1M Dataset: This dataset contains one million ratings from users for various movies. It is used to simulate user preferences and validate collaborative filtering techniques.

Amazon Product Reviews Dataset: This dataset includes user reviews, product metadata and ratings. It is used to evaluate content-based filtering and reinforce the cold-start problem for new items.

These datasets are pre-processed to extract user profiles, item attributes and interaction matrices. For example, User profiles are constructed using demographic data and past interactions. Item attributes include product categories, textual descriptions and average ratings. Interaction metrics represent matrices user-time interactions such as ratings or purchases.

System Configuration: The setup is deployed on a distributed system with the following specifications:
Edge nodes: Raspberry Pi 4 with 4 GB RAM to simulate resource-constrained devices.

Central Server: 16-core CPU, 64 GM RAM and NVIDIA Telsa T4 GPU for global model aggregation and inference.

Software Stack:
Federated Learning: Tensor Flow Federated is used for decentralized training.
Data Ingestion: Apache Kafka handles real-time data streaming.
Feature Store: Cassandra servers as the distributed database for feature storage.
Model Management: ML flow is used to manage model versions and track performance metrics.
Privacy Mechanisms: Py Syft and Ten SEAL libraries are employed for implementing differential privacy and holomorphic encryption.
All components are containerized using Docker and orchestrated using Kubernetes. This step ensures scalability and fault tolerance across the distributed system.

Experimental Workflow
This is divided into three phases.
Data Preprocessing: The datasets are preprocessed to clean and standardize the data including handling of missing values and normalizing numerical features. Textual data from the Amazon Product Reviews dataset is tokenized and converted into embedding's using BERT.

Model Training: Federated Learning is initiated with local models trained on edge nodes using user-specific subsets of data. Updates from edge node are encrypted using holomorphic encryption and aggregated on the central server. Differential Privacy is applied during aggregation to ensure anonymity.

Evaluation: The trained global model is evaluated on holdout datasets measuring recommendation accuracy, latency and private metrics. The performance of the hybrid recommendation engine is compared against baseline methods including standalone collaborative filtering and content-based filtering models in table 1 below.

Table no 1 Accuracy metrics for Recommendation Methods

Metric	**Proposed Framework**	**Collaborative Filtering**	**Content-Based Filtering**
Precision (%)	92.1	85.4	87.2
Recall (%)	93.5	86.7	91.0
F1-Score (%)	92.8	86.0	90.2
MAP (Mean Average Precision	0.912	0.854	0.895
NDCG (Normalized Discounted Cumulative Gain)	0.925	0.868	0.902

The results indicate that the proposed framework achieves the highest precision, recall and ranking quality across all datasets.

Scalability

The system was evaluated for scalability under increasing workloads. The experiments tested throughput and latency for user workloads ranging from 1000 to 1 million requests and it was found that system maintained high throughput and acceptable latency even under peak loads.

Table no 2 Scalability Metrics of the Proposed Framework

Number of Requests	Throughput(request/sec)	Average Latency(ms)
1000	10500	15
10000	9800	22
100000	9200	38
1000000	8500	62

Privacy Preservation

The privacy-preserving mechanism was evaluated based on privacy loss (epsilon) and the impact of noise addition on model utility. The results show a trade-off between privacy and utility. The system achieves a balanced privacy-utility trade-off with epsilon values in the rate of 1.0 to 10.0. Table 3 compares the model accuracy at different levels of differential privacy.

Table no 3 Impact of Differential Privacy on Model Accuracy

Privacy Budget (Epsilon)	Accuracy (%)	Precision (%)	Recall (%)
0.1	85.3	83.2	82.9
1.0	90.5	89.4	89.1
10.0	92.1	91.8	91.2

IV.CONCLUSION

The proposed framework for a scalable, privacy-preserving recommendation engine successfully addresses key challenges in the modern e-commerce system by integrating federated learning, differential privacy and a hybrid recommendation strategy.

The results of extensive experiments validate the system's effectiveness. The recommendation engine demonstrated superior accuracy with a precision of 92.1 % and recall of 93.5% outperforming traditional centralized systems and standalone recommendation techniques. The scalability tests revealed the system's ability to handle workloads of up to 1 million user requests with low latency and high throughput. Privacy-preserving mechanisms including differential privacy and homomorphic encryption successfully protected user data without significant trade-offs in model utility.

This framework bridges critical gaps in the field by combining state-of-the-art privacy techniques with real-time adaptability and module design. Unlike existing approaches, the system is well suited distributed architectures supporting modern e-commerce platforms that require dynamic personalization and robust data privacy.

Future work will explore the integration of advanced deep learning techniques such as graph neural networks to enhance the contextual understanding of user behavior and item relationships. Additionally, the framework's applicability can be extended to the other domains such as health care where privacy and scalability are critical.

REFERENCES

[1] Koren, Y., Bell, R., and Volinsky, C., "Matrix Factorization Techniques for Recommender Systems," IEEE Computer, 2009. [Online]. Available: https://ieeexplore.ieee.org

[2] He, X., Liao, L., Zhang, H., et al., "Neural Collaborative Filtering," in Proceedings of WWW, 2017. [Online]. Available: https://dl.acm.org

[3] Liu, J., Wang, Z., and Li, X., "Differential Privacy for Personalized Recommendations," ACM SIGKDD Conference, 2019. [Online]. Available: https://dl.acm.org

[4] Yang, Q., Liu, Y., Chen, T., and Tong, Y., "Federated Learning for Recommender Systems," arXiv preprint, 2020. [Online]. Available: https://arxiv.org

[5] Zhao, X., Zhang, Z., and Zhou, H., "Reinforcement Learning for Personalized Recommendations," in Proceedings of RecSys, 2018. [Online]. Available: https://dl.acm.org