A Comprehensive Survey of Recommender System Models and Extensions to Online Learning and Distributed Architectures

Lucas Kim
Loyola Marymount University
Los Angeles, CA, US

ABSTRACT

Recommender systems are a critical component of modern online platforms, helping to personalize content and improve user engagement across domains. Traditional approaches such as collaborative filtering, content-based filtering, and matrix factorization have achieved substantial success. However, these methods struggle to meet the demands of dynamic environments where data shifts rapidly and systems must scale to millions of users and items.

This survey presents a comprehensive review of classical recommender system models and explores their evolution toward online learning techniques and distributed architectures. We cover key developments in streaming algorithms, multi-armed bandits, and incremental learning methods that enable real-time personalization. On the systems side, we examine how large-scale platforms implement microservices, distributed training, model serving, and real-time feature engineering to support scalable and adaptive recommendation pipelines.

We also analyze the trade-offs between batch and online learning, focusing on freshness, computational cost, and feedback loop integration. The paper identifies open challenges such as privacy, fairness, and generalization, and highlights promising directions including federated learning and continual adaptation. By bridging foundational models with practical infrastructure, this survey offers a unified perspective for designing high-performance, future-ready recommender systems.

KEYWORDS

Recommender Systems, Online Learning, Streaming Recommendations, Distributed Systems, Real-Time Personalization, Bandit Algorithms, Microservice Architecture, Model Serving, Batch vs Online Trade-offs, Federated Learning, Continual Learning, Scalability, Adaptive Systems

1 INTRODUCTION

As intelligent systems increasingly mediate user experiences across digital platforms, recommender systems have emerged as a foundational layer in applications such as e-commerce, media streaming, social networking, and personalized content delivery. These systems rely on sophisticated algorithmic techniques to model user preferences, optimize relevance, and deliver real-time personalization. However, as user bases scale and feedback loops become more dynamic, traditional recommender models face mounting limitations in responsiveness, adaptability, and scalability.

Historically, most recommender systems have been built using batch learning paradigms. Techniques such as collaborative filtering, matrix factorization, and deep learning-based models have proven effective under static or slowly evolving data conditions. Yet, modern environments demand systems that can ingest, learn from, and act on data in real time. Moreover, the sheer scale of industrial systems necessitates architectures that are not only accurate, but also highly distributed, fault-tolerant, and latency-aware.

This survey aims to provide a structured and end-to-end understanding of recommender systems, tracing their evolution from classical models to real-time, scalable architectures that integrate online learning and distributed infrastructure. The structure of this survey follows the natural progression of this evolution and is organized as follows:

- **Fundamentals of Recommender Systems** provides an overview of the foundational concepts and methodologies that form the basis of recommendation systems, focusing on how user preferences and item relevance are modeled.
- **Limitations of Traditional Recommender Systems** discusses the core limitations of conventional systems, including issues of data sparsity, scalability, latency, and their inability to adapt to dynamic user behavior in real time.
- **Hybrid Models** explores how combining multiple recommendation strategies can improve system robustness, accuracy, and generalization, especially in environments with sparse or noisy data.
- **Graph-Based Models** examines the use of graph structures to represent and learn from user-item interactions, with an emphasis on capturing higher-order relationships and collaborative patterns.
- **Deep Learning-Based Models** delves into how neural networks are used to model complex user behaviors, sequential patterns, and large-scale personalization in recommendation scenarios.
- **Online Learning vs. Batch Learning: Trade-Offs** compares the practical and theoretical differences between continuously updated models and traditional batch-trained models, highlighting their respective advantages and challenges.
- **Online Learning for Recommender Systems** focuses on learning algorithms that adapt continuously to user feedback and evolving data, enabling real-time personalization and responsiveness.
- **Distributed and Scalable Recommender System Architectures** addresses the system design and infrastructure needed to deploy recommenders at scale, covering topics.
- **Open Challenges and Future Directions** outlines unresolved issues in the field, such as privacy, lifelong learning, efficient deployment, and domain transfer, and suggests directions for future research.
- **Conclusion** summarizes the key insights presented in the paper and reflects on their implications for building the next generation of recommender systems.

By organizing the survey in this way, we provide a unified view that connects foundational algorithms, modern learning strategies, and system-level deployment practices. This structure is intended to guide both researchers interested in theoretical advancements and practitioners focused on building scalable and adaptive recommendation engines.

2 FUNDAMENTALS OF RECOMMENDER SYSTEMS

2.1 Content-Based Filtering

Recommends items based on the similarity between item features and the user's historical preferences. Each user is profiled using previously interacted items, and recommendations are generated by finding items with similar attributes.

$$\text{score}(u, i) = \text{sim}(x_i, \bar{x}_u)$$

where x_i is the feature vector of item i, and $\bar{x}_u$ is the average feature vector of items previously liked by user u.

Applications: Effective in cold-user scenarios and domains where item metadata is rich and reliable, such as news, movies, or books. Often used when collaborative signals are sparse.
Example: *Content-Based Filtering for Video Sharing Social Networks* [1]

2.2 Collaborative Filtering (User/Item-Based)

Generates recommendations by analyzing patterns in user-item interactions, assuming that users with similar behavior will prefer similar items. User-based CF finds similar users and recommends their liked items, while item-based CF identifies items that are similar based on co-occurrence in user histories.

$$\text{score}(u, i) = \sum_{v \in N(u)} \text{sim}(u, v) \cdot r_{v,i} \quad \text{(User-based)}$$

$$\text{score}(u, i) = \sum_{j \in N(i)} \text{sim}(i, j) \cdot r_{u,j} \quad \text{(Item-based)}$$

Applications: Widely used in e-commerce and streaming platforms where user feedback (ratings, clicks, views) is abundant. Item-based CF is often preferred in large-scale systems due to its offline precomputability.
Example: *User-based collaborative filtering approach for content recommendation in OpenCourseWare platforms* [2], *Completing partial recipes using item-based collaborative filtering to recommend ingredients* [3]

2.3 Matrix Factorization

Decomposes the user-item interaction matrix into low-dimensional latent factors, capturing hidden patterns in preferences. The dot product of the user and item embeddings approximates the original interaction values.

$$\hat{r}_{ui} = p_u^T q_i$$

where $p_u \in \mathbb{R}^k$ and $q_i \in \mathbb{R}^k$ are the latent vectors for user u and item i, respectively.

Applications: A foundational method in personalized recommendation, known for its balance of scalability and accuracy. Frequently used in rating prediction tasks and hybrid recommenders.
Example: *MAn Introduction to Matrix factorization and Factorization Machines in Recommendation System, and Beyond* [4]

3 LIMITATIONS OF TRADITIONAL RECOMMENDER SYSTEMS

3.1 Cold-Start Problem

The cold-start problem [5] arises when the system lacks sufficient historical data to make accurate recommendations for new users or new items. Collaborative filtering methods depend heavily on user-item interaction matrices, and without prior interactions, such systems cannot reliably estimate user preferences or item relevancy. This issue is more pronounced in domains where user behavior is sparse, and where new users or items are frequently introduced.

The root cause is the reliance on historical behavioral signals without leveraging content-based or contextual information. In pure collaborative filtering systems, unseen users or items are essentially invisible to the model until interactions accumulate.

3.2 Scalability Challenges

Scalability becomes a major limitation [6] as the volume of users, items, and interactions increases. Many classical recommendation algorithms, including memory-based collaborative filtering and full matrix factorization, involve operations whose time and space complexity grows quadratically or worse with respect to dataset size. This leads to inefficiencies in both model training and inference.

These issues stem from centralized architectures, expensive similarity computations (e.g., kNN over millions of users/items), and retraining procedures that are not designed to handle incremental updates efficiently.

3.3 Latency Issues

Latency limitations [6] emerge when recommendation models are deployed in real-time environments but are not optimized for low-latency inference. Traditional models—particularly complex deep learning models or graph-based embeddings—often require multiple stages of computation that are unsuitable for applications with strict response time constraints.

Latency is further worsened when models require dynamic feature retrieval, multiple passes through deep architectures, or recomputation of similarity scores on-the-fly.

3.4 Lack of Feedback Loop Integration

Traditional recommenders are typically trained in an offline fashion using static datasets, with minimal mechanisms to incorporate user feedback post-deployment. As a result, these systems fail to adapt to shifts in user interest, popularity trends, or evolving content catalogs.

This disconnect [7] arises from decoupled training-serving pipelines, lack of real-time signal ingestion, and the absence of online learning or continual adaptation mechanisms. Offline optimization metrics (e.g., RMSE) often do not correlate with online user satisfaction or business KPIs.

Despite the historical successes of traditional recommender systems, their inherent limitations—such as scalability bottlenecks, cold-start challenges, and lack of real-time adaptation—have become increasingly pronounced in modern dynamic environments. To overcome these issues, hybrid models have emerged as a critical next step, combining multiple recommendation strategies to enhance system robustness, accuracy, and flexibility.

4 HYBRID MODELS

4.1 Factorization Machines

Factorization Machines (FM) generalize matrix factorization by modeling interactions between arbitrary features using factorized parameters. Unlike traditional collaborative filtering, FM can incorporate side information (e.g., item metadata, user demographics) by treating them as input features. This allows FM to bridge collaborative and content-based information in a unified framework.

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j$$

where x is the feature vector, w_i are linear weights, and v_i are latent vectors representing feature interactions.

Applications: Useful when user and item side features are available and sparse interaction data needs to be augmented. Widely applied in CTR prediction and personalized ranking tasks.
Example: *Factorization Machines* [8]

4.2 Wide & Deep Learning

Wide & Deep Learning combines two learning components: a wide linear model for memorization and a deep neural network for generalization. The wide part captures co-occurrence patterns, while the deep part captures high-order nonlinear feature interactions.

$$\text{score}(x) = \text{Wide}(x) + \text{Deep}(x)$$

Applications: Originally deployed in Google Play for app recommendation, this architecture balances the ability to recall frequently co-occurring features and to explore new combinations via dense embeddings.
Example: *Wide & Deep Learning for Recommender Systems* [9]

4.3 DeepFM

DeepFM integrates the strengths of Factorization Machines and deep neural networks. The FM component models pairwise feature interactions, while the deep component captures higher-order interactions. Unlike Wide & Deep, DeepFM shares the same input for both parts and eliminates the need for manual feature engineering in the wide component.

$$\text{score}(x) = \text{FM}(x) + \text{MLP}(x)$$

Applications: Frequently used in CTR prediction and feed ranking systems, especially in ad-tech and recommendation scenarios where structured categorical features dominate.
Example: *DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction* [10]

4.4 NeuMF

Neural Matrix Factorization (NeuMF) combines Generalized Matrix Factorization (GMF) and a Multi-Layer Perceptron (MLP) to learn both linear and nonlinear user-item interactions. GMF captures latent matching via element-wise product, while MLP learns more complex interaction functions.

$$\hat{r}_{ui} = \text{MLP}(p_u \| q_i) + p_u \odot q_i$$

Applications: A canonical neural collaborative filtering model for implicit feedback recommendation, often used as a baseline in deep recommender benchmarking.
Example: *Neural Collaborative Filtering* [11]

4.5 LightFM

LightFM is a hybrid recommendation model that combines matrix factorization with content-based features using a shallow embedding-based architecture. It supports flexible loss functions and incorporates both interaction data and metadata to improve generalization and cold-start performance.

$$\hat{r}_{ui} = \langle f_u, f_i \rangle$$

where f_u, f_i are the latent representations learned from both collaborative and content features.

Applications: Especially effective in cold-start environments where user/item metadata is available but behavioral signals are limited. Used in domains like news, e-learning, and niche marketplaces.
Example: *Metadata Embeddings for User and Item Cold-start Recommendations* [12]

While hybrid models effectively integrate diverse signals to improve recommendation quality, they often fall short in capturing the rich, complex relationships among users and items. This limitation has led to a growing interest in graph-based models, which leverage structured interaction data to uncover higher-order collaborative patterns and improve representational power.

5 GRAPH-BASED MODELS

5.1 GC-MC

Graph Convolutional Matrix Completion (GC-MC) applies graph convolutional networks (GCNs) to the user-item bipartite graph for matrix completion tasks. Users and items are treated as nodes, and their interactions as edges. Feature information propagates through the graph to learn embeddings via message passing.

$$\hat{r}_{ui} = f_\theta(h_u, h_i)$$

where h_u, h_i are GCN-encoded embeddings of user u and item i.
Applications: Suitable for rating prediction when side features are available and when modeling the structure of interactions as a graph improves performance.
Example: *Graph Convolutional Matrix Completion* [13]

5.2 NGCF

Neural Graph Collaborative Filtering (NGCF) extends the GCN framework to explicitly model collaborative signals in user-item interaction graphs. It propagates embeddings through graph layers while preserving high-order connectivity and learning user-item co-embedding spaces.

$$e_u^{(l+1)} = \sigma\left(\sum_{i\in\mathcal{N}(u)} \left(W_1 e_i^{(l)} + W_2(e_u^{(l)} \odot e_i^{(l)})\right)\right)$$

Applications: Used in top-N recommendation with implicit feedback, particularly when high-order graph connectivity is important.
Example: *Neural Graph Collaborative Filtering* [14]

5.3 LightGCN

LightGCN simplifies NGCF by removing non-linear activation functions and feature transformations, focusing purely on weighted neighborhood aggregation. It propagates embeddings across layers and sums them to form final representations.

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)}$$

Applications: Effective and scalable for large-scale top-K recommendation with implicit signals. Common baseline in graph-based collaborative filtering.
Example: *LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation* [15]

5.4 PinSage

PinSage is a scalable GCN model developed by Pinterest for web-scale recommendation. It incorporates random walks for neighbor sampling and leverages node features and structure via convolutional layers.

$$h_v^{(k+1)} = \mathrm{ReLU}(W^{(k)} \cdot \mathrm{AGG}(\{h_u^{(k)} : u \in \mathcal{N}(v)\}))$$

Applications: Web-scale recommendation systems such as product or content graph traversal in Pinterest-like platforms.
Example: *Graph Convolutional Neural Networks for Web-Scale Recommender Systems* [16]

5.5 GAT4Rec

GAT4Rec applies graph attention networks (GATs) to sequential or session-based recommendation. It assigns different weights to neighboring nodes based on attention coefficients, enhancing expressiveness over uniform message passing.

$$\alpha_{ij} = \frac{\exp(\mathrm{LeakyReLU}(a^T[Wh_i \| Wh_j]))}{\sum_{k\in\mathcal{N}(i)} \exp(\mathrm{LeakyReLU}(a^T[Wh_i \| Wh_k]))}$$

Applications: Used in session-based recommendation where attention across item transitions can improve sequential modeling.
Example: *GAT4Rec: Sequential Recommendation with a Gated Recurrent Unit and Transformers* [17]

As interaction graphs scale to encompass millions of users and items, simple graph-based propagation methods become insufficient to capture intricate, nonlinear user-item dynamics. Deep learning techniques, particularly those using neural architectures, have thus been introduced to model complex sequential behaviors, latent interests, and context-dependent preferences with greater expressive capacity.

6 DEEP LEARNING-BASED MODELS

6.1 AutoRec

AutoRec uses an autoencoder to reconstruct missing entries in the user-item interaction matrix. It learns latent encodings by compressing observed ratings and decoding them to predict unobserved ones.

$$\hat{r}_i = f_{\text{decoder}}(f_{\text{encoder}}(r_i))$$

Applications: Used for rating prediction tasks with dense interaction data. Also applicable in cold-start cases when combined with metadata.
Example: *AutoRec: Autoencoders Meet Collaborative Filtering* [18]

6.2 GRU4Rec

GRU4Rec is one of the first neural session-based recommendation models using gated recurrent units (GRUs) to model sequential item interactions. It captures session dependencies and predicts the next likely item in a sequence.

$$h_t = \mathrm{GRU}(x_t, h_{t-1})$$

Applications: Session-based recommendation in e-commerce and streaming where session dynamics are essential.
Example: *Session-based Recommendations with Recurrent Neural Networks* [19]

6.3 SASRec

SASRec applies the Transformer encoder architecture to sequential recommendation. It models user behavior through self-attention over historical interactions, allowing long-range dependency modeling.

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

Applications: Used for next-item prediction in sequential logs. Highly flexible and effective baseline for modern session-based recommendation.
Example: *Self-Attentive Sequential Recommendation* [20]

6.4 BERT4Rec

BERT4Rec adopts bidirectional transformers and a masked language modeling approach to learn user preferences from interaction sequences. Unlike SASRec, it leverages both past and future context.

$$L = -\sum_{i \in \mathcal{M}} \log P(x_i | x_{\setminus i})$$

Applications: Top-performing model in sequential recommendation with dense user logs, especially in content feeds and e-commerce.
Example: *BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations* [21]

6.5 YouTube DNN

YouTube DNN uses two-tower architecture to scale personalized video recommendations. It separates candidate generation (classification) from ranking (scoring), enabling real-time inference at large scale.

$$\text{score}(u, i) = \langle f_u^{\text{tower}}, f_i^{\text{tower}} \rangle$$

Applications: Large-scale recommender systems with billions of items and real-time demands. Widely adopted in industry.
Example: *Deep Neural Networks for YouTube Recommendations* [6]

6.6 MIND

MIND (Multi-Interest Network with Dynamic Routing) represents users with multiple latent interest vectors. It uses capsule routing to dynamically select relevant interests for different item types, improving diversity and personalization.

$$f_u = \text{Routing}(\{e_{i_1}, e_{i_2}, \ldots, e_{i_n}\})$$

Applications: Deployed in large-scale news recommendation systems (e.g., Microsoft News) for modeling diverse user interests.
Example: *Multi-Interest Network with Dynamic Routing for Recommendation at Tmall* [22]

Despite their powerful modeling capabilities, deep learning-based recommender systems traditionally rely on offline batch training, which limits their ability to quickly adapt to evolving user behaviors and rapidly changing item catalogs. This challenge necessitates a fundamental shift toward online learning paradigms, where models are updated incrementally based on live user interactions.

7 ONLINE LEARNING VS. BATCH LEARNING: TRADE-OFFS

As recommender systems evolve to operate in dynamic, real-world environments, the choice between online learning and batch learning becomes critical. Each approach offers distinct advantages and imposes different constraints across model accuracy, system efficiency, adaptability, and evaluation complexity. Understanding these trade-offs is essential for designing scalable and responsive recommendation pipelines.

7.1 Accuracy vs. Freshness

Batch learning optimizes models over large historical datasets, often achieving higher global accuracy by minimizing loss across all available data. However, batch-trained models may quickly become stale in fast-changing environments where user interests and item catalogs evolve rapidly.

Online learning, by contrast, prioritizes freshness by continuously updating models based on recent interactions. This enables faster adaptation to new trends, but may sacrifice some global optimality due to training on limited, noisy, or biased real-time data streams.

Choosing between batch and online learning involves balancing long-term accuracy against short-term personalization responsiveness, often leading to hybrid architectures that combine periodic batch retraining with online fine-tuning.

7.2 Resource Utilization

Batch learning is resource-efficient during inference, as model training occurs offline and inference models are typically static and highly optimized for deployment. However, batch training itself can be computationally expensive, requiring significant time, memory, and distributed infrastructure to process large datasets.

Online learning distributes computational load over time by incrementally updating models with new data. While this enables continuous learning without full retraining cycles, it imposes sustained resource consumption, including memory for intermediate states, frequent model synchronization, and online feature updates.

In practice, resource allocation must consider training efficiency, inference cost, and system scaling requirements under both learning paradigms.

7.3 Feedback Loop Integration

One of the key strengths of online learning is its ability to close the feedback loop by integrating user responses in near real-time. User clicks, views, or interactions can immediately influence future recommendations, allowing systems to rapidly personalize and self-correct.

Batch learning systems, by contrast, exhibit longer feedback cycles. Data must be collected, aggregated, and retrained in periodic batches, introducing delays between user behavior shifts and model adaptation.

Real-time feedback loop integration enhances responsiveness but requires robust safeguards against feedback biases, non-stationarity, and user manipulation.

7.4 Evaluation Metrics in Streaming Environments

Traditional batch evaluation metrics, such as RMSE, Precision@K, or Recall@K, assume static test sets and do not capture temporal dynamics. In streaming or online learning environments, evaluation must consider evolving user preferences, non-stationary item catalogs, and delayed reward signals.

Key evaluation approaches for streaming recommenders include:

- **Interleaving and Bandit-Based Online Evaluation:** Compare competing models or strategies in live settings through randomized exposure and direct user feedback.
- **Cumulative Metrics Over Time:** Track changes in engagement metrics (e.g., CTR, retention) to assess model performance trends.
- **Regret Minimization:** Measure how much reward was lost compared to the optimal policy, reflecting exploration-exploitation efficiency.
- **Temporal Stability Analysis:** Monitor how consistent model outputs remain as user behavior and item availability evolve.

Designing effective evaluation strategies is crucial to ensure that improvements in online learning systems translate to sustained user satisfaction and business outcomes.

Understanding the fundamental trade-offs between batch and online learning reveals that responsiveness, adaptability, and continual feedback integration are becoming essential for modern recommendation pipelines. Building on this insight, the next section explores concrete online learning techniques that empower recommender systems to personalize content in real time.

8 ONLINE LEARNING FOR RECOMMENDER SYSTEMS

8.1 Overview of Online Learning Paradigm

Traditional recommender systems are typically trained using batch learning, where models are periodically updated on large historical datasets. While effective in stable environments, batch learning struggles in fast-changing contexts where user preferences and item catalogs evolve rapidly. Online learning addresses this limitation by continuously updating models as new interactions occur, enabling systems to adapt to trends, seasonality, and emerging user behaviors. Compared to batch methods, online learning prioritizes responsiveness and freshness, often trading off slight losses in global accuracy for significant gains in personalization and real-time relevance. This paradigm is increasingly critical for platforms operating in dynamic, user-driven ecosystems.

8.2 Bandit Algorithms

Bandit algorithms address the exploration-exploitation dilemma in online recommendation by adaptively balancing the need to recommend known high-reward items and the need to explore uncertain options. Unlike static models trained offline, bandit algorithms continually learn from user interactions, updating strategies to maximize cumulative reward over time.

The fundamental objective in bandit problems is to minimize *regret*, which measures the loss due to not always selecting the optimal action. The regret at time t can be expressed as:

$$R(T) = \sum_{t=1}^{T} \left(r_t^* - r_{a_t}\right)$$

where r_t^* is the reward of the best possible action at time t, and r_{a_t} is the reward of the action actually chosen.

Two primary categories of bandit algorithms are employed in recommender systems: [23]

8.2.1 *Multi-Armed Bandit (MAB).*

In the classical MAB setting, the system selects one item (arm) at each round without any contextual information. Algorithms optimize exploration and exploitation by maintaining reward estimates for each item.

Representative Algorithms:

- **ϵ-Greedy:** With probability ϵ, select a random arm (explore); otherwise, select the best-known arm (exploit).
- **UCB (Upper Confidence Bound):** Select the arm with the highest upper confidence bound, balancing estimated reward and uncertainty. [24]
- **Thompson Sampling:** Sample from the posterior distribution of each arm's reward and select the arm with the highest sample. [25]

Applications: Effective in environments with high-frequency decision-making and implicit feedback signals, such as click prediction in recommendation engines.

8.2.2 *Contextual Bandit.*

Contextual bandits extend MAB by incorporating side information (context) about users, items, or environment states. The system observes context x_t at each round and chooses an action based on both context and historical data.

Formally, the expected reward is conditioned on the context:

$$\mathbb{E}[r_{a_t} | x_t]$$

Representative Algorithms:

- **LinUCB:** Models the expected reward as a linear function of context features and selects the arm with the highest upper confidence bound.
- **Contextual Thompson Sampling:** Extends Thompson Sampling by maintaining posterior distributions conditioned on contexts.

Applications: Particularly powerful in personalized recommendation, dynamic pricing, and contextual advertisement selection where user or session attributes can guide better choices.
Example: *A Contextual-Bandit Approach to Personalized News Article Recommendation* [7]

8.3 Incremental Matrix Factorization

Incremental Matrix Factorization addresses the need for continuously updating user and item representations as new interactions are observed, without retraining the entire model from scratch. In dynamic environments where new users, items, or feedback appear rapidly, batch retraining becomes computationally expensive and causes significant latency. Incremental techniques enable lightweight updates to the latent factors based on streaming data.

The standard matrix factorization objective aims to minimize the reconstruction error:

$$\min_{P,Q} \sum_{(u,i)\in O} (r_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

where P and Q are user and item embedding matrices, r_{ui} is the observed rating, and λ is the regularization coefficient.

In the incremental setting, instead of optimizing over the entire observation set O, updates are performed locally for newly observed interactions. Given a new tuple (u, i, r_{ui}), stochastic gradient updates are applied:

$$p_u \leftarrow p_u + \eta(e_{ui}q_i - \lambda p_u) \quad q_i \leftarrow q_i + \eta(e_{ui}p_u - \lambda q_i)$$

where $e_{ui} = r_{ui} - p_u^T q_i$ is the prediction error, and η is the learning rate.

8.3.1 Representative Approaches. Several strategies have been proposed to efficiently implement incremental updates for matrix factorization in real-world systems:

- **SGD-Based Incremental Updates:** The most common approach uses stochastic gradient descent to update the user and item embeddings immediately after each new interaction. This allows the system to adapt online without the need for full retraining. [26]
- **Warm-Start Retraining:** Instead of starting from random initialization, previously learned embeddings are reused and fine-tuned with new data. This approach balances computational efficiency and adaptation speed, particularly when batch updates are periodically scheduled. [27]
- **Incremental ALS (Alternating Least Squares):** Although less common due to higher computational cost, some systems extend ALS-based matrix factorization to perform partial updates using only newly added interactions, thereby avoiding full recomputation. [28]
- **Online Low-Rank Approximation:** Online algorithms dynamically adjust low-rank decompositions of the interaction matrix as new entries arrive, maintaining a compact representation over time. [29]

8.4 Streaming-Based Collaborative Filtering

Streaming-based collaborative filtering addresses the challenge of delivering personalized recommendations in environments where user-item interactions arrive continuously in real time. Unlike traditional batch-trained systems that periodically retrain on static datasets, streaming collaborative filtering systems update models incrementally as new data flows in, enabling more timely and adaptive personalization.

Several strategies have been developed to enable collaborative filtering in streaming contexts, including incremental matrix factorization, sliding window models, approximate sketching methods, and online bandit frameworks. These methods prioritize adaptability and computational efficiency over globally optimal solutions, reflecting the practical needs of real-time recommender systems.

In the streaming setting, key design requirements emerge:

- **Online Model Updates:** Models must be capable of integrating new interactions immediately, without full retraining, to capture the most recent user preferences. [30]
- **Bounded Resource Usage:** Since interaction logs may grow indefinitely, algorithms must control memory and computation through techniques like windowing, sampling, and sketching. [31]
- **Freshness and Responsiveness:** Recommendations must reflect evolving trends, seasonal patterns, and short-term intents, rather than relying solely on historical behavior. [7]
- **Robustness to Concept Drift:** User interests and item popularity often shift over time, requiring models that can adapt without catastrophic forgetting. [32]

8.5 Real-Time Personalization Techniques

Real-time personalization techniques aim to deliver highly relevant recommendations by immediately adapting to the latest user interactions, context changes, and content dynamics. Unlike traditional models that update periodically, real-time techniques optimize for immediate responsiveness, ensuring that users experience recommendations tailored to their most recent behavior and preferences.

Common techniques for real-time personalization include online collaborative filtering with streaming updates, bandit-based recommendation strategies, session-based neural models, and lightweight re-ranking of candidate items based on real-time signals.

Key aspects of real-time personalization include:

- **Immediate Feedback Integration:** Systems update user profiles or model parameters upon each interaction, allowing the system to react instantly to clicks, views, likes, or purchases. [33]
- **Session-Aware Modeling:** Instead of relying solely on long-term historical data, real-time recommenders often emphasize the current session behavior to capture transient interests or needs. [34]
- **Lightweight and Efficient Updates:** Real-time systems prioritize methods that allow rapid model or embedding adjustments with minimal computational overhead, often using online learning, embeddings caching, or approximate updates. [35]
- **Contextual Awareness:** Incorporating dynamic contextual signals, such as location, device, time of day, or trending content, enhances the relevance of recommendations beyond static user-item matching. [36]

While online learning techniques enable real-time adaptation, scaling these methods to production-level systems serving millions of users introduces substantial engineering challenges. Distributed and scalable architectures are crucial to efficiently manage large-scale training, inference, and feature serving pipelines in such dynamic environments.

9 DISTRIBUTED AND SCALABLE RECOMMENDER SYSTEM ARCHITECTURES

9.1 Challenges in Scalability and Deployment

Scaling recommender systems from research prototypes to production environments introduces substantial challenges beyond model accuracy. Industrial-scale systems must process massive volumes of data, serve personalized results in real time, adapt to constantly evolving user behaviors, and ensure robustness against failures, all while operating within strict resource and latency constraints.

Key challenges include:

- **Data Scale and Throughput:** Modern recommender systems often deal with millions of users, millions of items, and billions of interaction events. Efficient storage, distributed data processing, and parallel model training become essential to handle this scale.
- **Real-Time Latency Requirements:** In user-facing applications such as streaming services or e-commerce platforms, recommendations must be generated within tens to hundreds of milliseconds. Systems must optimize for low-latency model inference without sacrificing personalization quality.
- **Frequent Model and Feature Updates:** New users, new items, and changing user preferences emerge continuously. Systems must incrementally update user profiles, embeddings, and feature stores to reflect the latest interactions while avoiding service downtime.
- **Fault Tolerance and High Availability:** Recommender system components must be resilient to server failures, network partitions, and hardware degradation. Downtime or degraded recommendation quality directly impacts user satisfaction and business outcomes.
- **Resource and Cost Efficiency:** Scaling model inference and training to production levels imposes significant memory, compute, and network demands. Systems must balance performance against operational costs, often requiring careful architecture design and hardware optimization.
- **Consistency Across System Components:** Maintaining synchronization between training data, model parameters, feature stores, and serving infrastructure is critical to prevent stale predictions, data drift, and inconsistencies between offline training and online serving.
- **Robustness Against Noisy and Adversarial Feedback:** As recommender systems increasingly rely on real-time user feedback to adapt models, they become vulnerable to noisy, anomalous, or even malicious interactions. Without robustness mechanisms, such as outlier detection, robust aggregation, or Byzantine-resilient learning, models risk being misled by corrupted data streams, leading to degraded personalization and potential system instability. Ensuring resilience against unreliable feedback is critical for maintaining recommendation quality and user trust at scale.

Addressing these challenges requires a co-design approach across machine learning models, system architectures, and deployment strategies, laying the foundation for scalable and production-ready recommender systems.

9.2 Distributed Training

As the volume of users, items, and interactions grows, training recommender system models on a single machine becomes infeasible due to memory, compute, and time constraints. Distributed training enables parallelization of model updates across multiple nodes, significantly reducing training time and allowing models to scale to industrial-scale datasets.

Choosing between parameter server and all-reduce architectures depends on the specific model characteristics, communication overhead tolerances, and system infrastructure. In practice, hybrid architectures combining the two paradigms have also been deployed to balance scalability, consistency, and fault tolerance.

Two primary paradigms dominate distributed training for recommendation models:

- **Parameter Server Architecture:** In the parameter server approach, model parameters are partitioned and distributed across dedicated server nodes. Worker nodes compute gradients on minibatches of data and asynchronously push updates to the servers. This design decouples computation and storage, enabling scalable and fault-tolerant model training.Parameter servers are particularly effective for large embedding tables, which are common in recommender systems with millions of users and items. [37]
- **All-Reduce Architecture (e.g., Horovod):** In all-reduce architectures, such as Horovod, each worker maintains a full replica of the model. Gradients computed on each node are averaged across all nodes using collective communication primitives (e.g., ring all-reduce). This synchronous update strategy ensures consistency across workers but requires high-bandwidth networking for efficiency.All-reduce approaches are often preferred when training deep neural network-based recommenders, where model replication and tight synchronization improve convergence stability. [38]

9.3 Ensuring Robustness

As recommender systems scale and operate in dynamic, user-driven environments, they increasingly face exposure to noisy interactions, manipulative behaviors, and distribution shifts. Ensuring robustness has therefore become a critical design consideration. Robustness refers to the system's ability to maintain recommendation quality even when faced with corrupted, inconsistent, or adversarial data.

Recent research proposes several key strategies to enhance robustness in recommendation pipelines:

- **Denoising-Based Collaborative Filtering:** This approach injects artificial noise into user-item interaction data during training and learns to reconstruct clean representations. By recovering meaningful preference signals from corrupted inputs, the system becomes more resilient to real-world data imperfections and minor manipulations. [39]
- **Byzantine-Robust Distributed Learning:** In distributed training settings, adversarial or faulty nodes can inject harmful updates. Byzantine-robust optimization techniques reject or down-weight suspicious gradients using robust statistical

estimators, ensuring that the learning process remains stable and convergent even in adversarial environments. [40] [41]
- **Robust Aggregation Mechanisms:** Rather than naively averaging model updates, robust aggregation methods selectively integrate only consistent or majority-supported updates. Techniques such as median aggregation, Krum selection, and trimmed means effectively filter out extreme or malicious updates, preserving model integrity. [42]
- **Noise-Resilient Implicit Feedback Modeling:** Implicit feedback signals, such as clicks or watch events, often contain significant noise. Noise-tolerant modeling frameworks explicitly account for the stochastic nature of these signals, separating genuine positive feedback from random or misleading interactions to improve recommendation quality. [43]

9.4 Microservice Architecture

As recommender systems grow in complexity, coupling model training, feature engineering, ranking, and serving within a monolithic architecture becomes increasingly impractical. Microservice architectures offer a modular approach, decomposing the recommender system into independently deployable services that can be scaled, updated, and optimized separately.

Microservice architectures enable independent scaling of bottleneck components (e.g., feature lookups vs. scoring models), faster iteration cycles for model updates, and improved system reliability through fault isolation. They also facilitate continuous deployment practices critical for real-time personalization systems.

Key components typically modularized in a microservice-based recommender system include:

- **Candidate Generation Service:** Rapidly retrieves a broad set of potentially relevant items using lightweight models or heuristic filters to reduce search space. [6]
- **Ranking and Scoring Service:** Applies more computationally intensive models, such as deep learning-based predictors, to the candidate set to produce final relevance scores. [6]
- **Feature Store Service:** Provides real-time access to dynamic user, item, and contextual features, ensuring consistency between training and inference.
- **User Profile and Interaction Logging Service:** Manages real-time user activity tracking, which feeds into model retraining pipelines and personalization modules.
- **Model Management and Serving Service:** Handles model versioning, deployment, A/B testing, and real-time inference scaling.

However, this design introduces its own challenges, such as increased inter-service communication overhead, data consistency management, and operational complexity, requiring robust orchestration frameworks and monitoring solutions.

Moreover, since many aspects of large-scale microservice orchestration and real-time consistency management are still conceptually evolving and not fully standardized in academic research, these areas present significant opportunities for further exploration and innovation.

10 OPEN CHALLENGES AND FUTURE DIRECTIONS

While recommender systems have made substantial progress in online learning, distributed architectures, and real-time personalization, several critical challenges remain unresolved. Addressing these issues is essential to ensure that future systems are not only scalable and efficient, but also privacy-preserving, adaptive, and generalizable across diverse domains and modalities. In this section, we outline the most pressing open challenges and promising directions for advancing recommender system research and practice.

10.1 Continual and Lifelong Learning in RS

User preferences, item catalogs, and consumption patterns evolve continually over time. Traditional recommender systems, even with online learning components, often struggle with long-term adaptation, suffering from issues such as catastrophic forgetting and concept drift. Lifelong learning frameworks, which aim to accumulate knowledge over time while adapting to new distributions without forgetting past experiences, are still in their infancy in the context of recommendation. Designing recommender systems capable of seamless continual learning without frequent retraining remains a major research frontier.

10.2 Resource-Efficient Online Inference

Real-time recommendation demands extremely low latency and high throughput, often under tight computational and memory budgets. As models grow more complex — incorporating deep learning, graph neural networks, and multi-interest representations — achieving efficient online inference becomes increasingly difficult. Techniques such as model pruning, knowledge distillation, feature selection, and approximate nearest neighbor retrieval offer partial solutions. However, there is still significant room for innovation in developing resource-efficient models specifically optimized for streaming environments and highly dynamic workloads.

10.3 Generalization Across Domains and Modalities

Current recommender systems are often trained and optimized for specific domains, such as movies, e-commerce, or news. Generalizing across domains — for example, transferring user representations between music and podcast recommendations — remains challenging. Moreover, as content types diversify (e.g., text, video, audio, and social graphs), building models that can integrate and reason across multiple modalities becomes crucial. Future research must focus on developing flexible, multimodal, and cross-domain recommendation frameworks that can robustly adapt to new user behaviors, content types, and interaction patterns without requiring extensive retraining.

10.4 Privacy-Preserving Online Learning

As online learning models continually adapt to user interactions, they inevitably accumulate and process sensitive information. Ensuring user privacy while maintaining personalization performance remains an open problem. Techniques such as differential privacy, federated learning, and secure multi-party computation offer partial solutions, but integrating them into large-scale, latency-sensitive recommender systems poses substantial trade-offs in model accuracy, resource consumption, and operational complexity. Future work must explore new privacy-preserving learning paradigms that balance protection, scalability, and effectiveness without severely degrading personalization quality.

11 CONCLUSION

Recommender systems have evolved dramatically from their early foundations in collaborative filtering and matrix factorization to today's complex, real-time, distributed architectures. This survey traced the critical transitions that underpin modern recommendation engines: from static batch learning to dynamic online adaptation, from monolithic architectures to scalable microservices, and from isolated model optimization to end-to-end system design integrating user feedback loops and infrastructure considerations.

We emphasized that success in contemporary recommender systems no longer depends solely on model accuracy in controlled settings, but rather on the system's ability to adapt, scale, and deliver personalized experiences under dynamic, resource-constrained, and privacy-sensitive environments. Techniques such as online learning, streaming-based collaborative filtering, bandit algorithms, and federated architectures represent not just technical extensions, but necessary responses to the fundamental limitations of traditional models.

At the same time, we highlighted open challenges that must be addressed to build truly next-generation recommenders, including privacy-preserving learning, lifelong adaptation, resource-efficient inference, and cross-domain generalization. Moving forward, the development of recommender systems will require a unified perspective that bridges algorithmic innovation, system-level engineering, and human-centered design. Only by mastering this integration can future systems meet the growing demands of scale, personalization, and trust in real-world applications.

REFERENCES

[1] Eduardo Valle, Sandra de Avila, Antonio da Luz Jr., Fillipe de Souza, Marcelo Coelho, and Arnaldo Araújo. Content-based filtering for video sharing social networks, 2011.
[2] Nikola Tomasevic, Dejan Paunovic, and Sanja Vranes. User-based collaborative filtering approach for content recommendation in opencourseware platforms, 2019.
[3] Paula Fermín Cueto, Meeke Roet, and Agnieszka Słowik. Completing partial recipes using item-based collaborative filtering to recommend ingredients, 2020.
[4] Yuefeng Zhang. An introduction to matrix factorization and factorization machines in recommendation system, and beyond, 2022.
[5] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, page 253–260, New York, NY, USA, 2002. Association for Computing Machinery.
[6] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.
[7] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW '10, page 661–670. ACM, April 2010.
[8] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.
[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide deep learning for recommender systems, 2016.
[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, and Zhenhua Dong. Deepfm: An end-to-end wide deep learning framework for ctr prediction, 2018.
[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.
[12] Maciej Kula. Metadata embeddings for user and item cold-start recommendations, 2015.
[13] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion, 2017.
[14] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19, page 165–174. ACM, July 2019.
[15] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.
[16] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, page 974–983. ACM, July 2018.
[17] Huaiwen He, Xiangdong Yang, Feng Huang, Feng Yi, and Shangsong Liang. Gat4rec: Sequential recommendation with a gated recurrent unit and transformers. *Mathematics*, 12(14):2189, 2024.
[18] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 111–112, New York, NY, USA, 2015. Association for Computing Machinery.
[19] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2016.
[20] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation, 2018.
[21] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer, 2019.
[22] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Pipei Huang, Huan Zhao, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. Multi-interest network with dynamic routing for recommendation at tmall, 2019.
[23] Liu Leqi, Giulio Zhou, Fatma Kilinc-Karzan, Zachary Lipton, and Alan Montgomery. A field test of bandit algorithms for recommendations: Understanding the validity of assumptions on human preferences in multi-armed bandits. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, page 1–16. ACM, April 2023.
[24] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 05 2002.
[25] WILLIAM R THOMPSON. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 12 1933.
[26] Guang Ling, Haiqin Yang, Irwin King, and Michael R. Lyu. Online learning for collaborative filtering. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.
[27] Xinran He, Jing Pan, Ou Jin, Tianqi Xu, Bo Liu, Tao Xu, Yaowei Shi, Jiang Bian, and Tie-Yan Liu. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising (ADKDD)*. ACM, 2014.
[28] Hamid Dadkhahi and Sahand Negahban. Alternating linear bandits for online matrix-factorization recommendation, 2018.
[29] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding, 2010.
[30] Xin Luo, Yunni Xia, and Qingsheng Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. *Knowledge-Based Systems*, 27:271–280, 2012.
[31] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52, 2007.
[32] Qingyun Wu, Naveen Iyer, and Hongning Wang. Learning contextual bandits in a non-stationary environment. In *The 41st International ACM SIGIR Conference on Research amp; Development in Information Retrieval*, SIGIR '18, page 495–504. ACM, June 2018.
[33] Qi Zhang, Jieming Zhu, Jiansheng Sun, Guohao Cai, Ruining Yu, Bangzheng He, and Liangbi Li. Enhancing news recommendation with real-time feedback and generative sequence modeling. In *Proceedings of the Recommender Systems Challenge 2024*, RecSysChallenge '24, page 32–36, New York, NY, USA, 2024. Association for Computing Machinery.
[34] Minjin Choi, Jinhong Kim, Joonseok Lee, Hyunjung Shim, and Jongwuk Lee. Session-aware linear item-item models for session-based recommendation. In

Proceedings of the Web Conference 2021, WWW '21, page 2186–2197. ACM, April 2021.

[35] Hung Vinh Tran, Tong Chen, Nguyen Quoc Viet Hung, Zi Huang, Lizhen Cui, and Hongzhi Yin. A thorough performance benchmarking on lightweight embedding-based recommender systems. *ACM Transactions on Information Systems*, 43(3):1–32, February 2025.

[36] Khalid Haruna, Maizatul Akmar Ismail, Suhendroyono Suhendroyono, Damiasih Damiasih, Adi Pierewan, Haruna Chiroma, and Tutut Herawan. Context-aware recommender system: A review of recent developmental process and future research direction. *Applied Sciences*, 7:1211, 12 2017.

[37] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[38] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow, 2018.

[39] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 811–820, New York, NY, USA, 2015. Association for Computing Machinery.

[40] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2021.

[41] Xuechao He, Qing Ling, and Tianyi Chen. Byzantine-robust stochastic gradient descent for distributed low-rank matrix completion. In *2019 IEEE Data Science Workshop (DSW)*, pages 322–326, 2019.

[42] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium, 2018.

[43] Zitai Wang, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Implicit feedbacks are not always favorable: Iterative relabeled one-class collaborative filtering against noisy interactions. In *Proceedings of the 29th ACM International Conference on Multimedia (ACM MM)*, pages 3070–3078, October 2021.