xLSTM Architecture's Feasibility For Recommendations (Draft Version)

Vivekanand

Abstract

This paper explores the application of the extended Long Short-Term Memory (xLSTM) architecture [Beck et al., 2024] for sequential recommendation tasks. We benchmark xLSTM against state-of-the-art baselines, including Transformer-based models (BERT4Rec, SASRec), classical RNNs, and matrix factorization methods, across several benchmark datasets (MovieLens 100K, 1M, 10M, and 20M). The evaluation focuses on top-k ranking metrics such as Recall@10, MRR@10, and NDCG@10. Experimental results demonstrate that xLSTM offers competitive performance while maintaining improved training efficiency, memory scalability, and robustness in sparse and cold-start scenarios. These findings highlight xLSTM's potential as a scalable and interpretable compared to the Transformer-based architectures in modern recommender systems.

1 Introduction

At its core, a recommender system is a machine learning-based algorithmic framework designed to predict and suggest relevant items to users by leveraging their historical interactions and contextual signals [Roy and Dutta, 2022, Author, 2024a]. These systems are integral to modern digital ecosystems, driving personalization across e-commerce, entertainment, and social media platforms.

A critical subdomain of these systems is sequential recommendation, which models user-item interactions over time, capturing temporal patterns and evolving user preferences. Recent advancements in this area have leveraged deep learning architectures, including Recurrent Neural Networks (RNNs) such as LSTM, and more recent models like BERT4Rec [Sun et al., 2019] and SASRec [Kang and McAuley, 2018], which adopt Transformer-based self-attention mechanisms for sequence modeling.

In this work, we investigate xLSTM [Beck et al., 2024], an enhanced variant of LSTM that incorporates architectural improvements including bidirectional processing, attention integration, and refined gating mechanisms. We benchmark xLSTM against classical and state-of-the-art sequential recommenders using datasets like MovieLens and other temporal interaction logs. Our empirical results show xLSTM demonstrates notable improvements due to its hybrid design that blends recurrent memory and attention mechanisms.

Broadly, recommendation systems can be categorized into two paradigms: Collaborative Filtering, which models similarities between users or items based on interaction histories, and Content-Based Filtering, which relies on item attributes or metadata to make predictions [Author, 2024b, 2023a]

1.1 General Classification of Recommender Systems

According to the RecBole framework [Author, 2020a], recommender systems can be broadly classified into four major categories based on the nature of data used and task formulation:

1. **General Recommendation (GR)**: These models rely solely on user-item interaction data, typically implicit feedback. They are evaluated through top-N recommendation tasks and primarily align with collaborative filtering approaches.

Preprint. Under review.

2. **Content-Aware Recommendation**: These models incorporate additional side information such as user or item features. They are often applied in click-through rate (CTR) prediction tasks, using explicit feedback and binary classification evaluation.
3. **Sequential Recommendation (SR)**: SR focuses on next-item prediction by modeling the temporal ordering of user interactions. These models utilize sequential data to capture evolving user preferences. Session-based recommendations are generally included in this category.
4. **Knowledge-Based Recommendation**: These methods leverage external knowledge graphs to enhance general or sequential recommendations by providing semantic or structural context beyond user-item interactions.

1.2 Recommender System and Existing LSTM Architecture Overview

Recommender systems are intelligent algorithms designed to predict user preferences and deliver personalized item suggestions, playing a foundational role in platforms such as Netflix, Amazon, and Spotify [Roy and Dutta, 2022, Author, 2023a]. Classical methods such as collaborative filtering and content-based filtering have evolved to incorporate sequential modeling, capturing dynamic user behavior over time [Author, 2024b].

Among deep learning approaches, the Long Short-Term Memory (LSTM) network remains a widely used architecture for modeling sequential interactions due to its ability to preserve long-term dependencies and manage temporal dynamics effectively. An LSTM unit consists of a memory cell and three gates: the input gate, forget gate, and output gate, which together regulate the flow of information [Yang and Esquivel, 2024, Ahmadian Yazdi et al., 2024].

Despite its success, LSTM suffers from several well-documented limitations: (1) its inability to revise storage decisions once made, (2) the need to compress all contextual information into scalar cell states, and (3) limited parallelizability due to recurrent memory mixing [Beck et al., 2024]. These limitations have led to the development of enhanced variants, such as xLSTM, which introduces architectural refinements like attention mechanisms, bidirectional memory, and improved gating functions to better handle sequential recommendation tasks.

1.3 Problem Statement

Despite the advancements brought by deep learning in improving recommendation accuracy, several critical challenges persist in modeling user-item sequences at scale. These challenges include the cold-start problem, data sparsity, and the need for robust long-term dependency tracking across user interactions [Noorian et al., 2024, Roy and Dutta, 2022].

Recurrent architectures such as LSTM, though effective in capturing temporal dependencies, suffer from limited scalability and are inherently sequential, making them less efficient for large-scale recommendation environments [Yang and Esquivel, 2024, Ahmadian Yazdi et al., 2024]. In contrast, Transformer-based models like BERT4Rec and SASRec offer greater parallelism and improved performance through self-attention mechanisms but often come with high computational costs and memory requirements [Sun et al., 2019, Kang and McAuley, 2018].

This work investigates whether the recently proposed xLSTM architecture [Beck et al., 2024], which integrates bidirectional memory, attention mechanisms, and enhanced gating structures, can serve as a middle ground—achieving scalability, computational efficiency, and high accuracy in sequential recommendation tasks.

1.4 Research Questions RQ and Objectives

This report aims to answer the following key questions:

• RQ1: How does xLSTM's performance scale with dataset size compared to established architectures like BERT4Rec?

• RQ2: Does increasing sequence length and embedding size lead to measurable performance gains in sequential recommenders?

• RQ3: What trade-offs exist between recommendation accuracy and computational cost as sequence length and model complexity increase?

RQ4: Embedding Saturation and Utilization: Are larger embeddings really helping the model learn better user/item relationships, or are they underutilized?

The primary objective is to evaluate the effectiveness of the xLSTM model across multiple datasets and benchmark it against state-of-the-art baselines using established ranking metrics.

1.5 Research Contributions of This Work

This thesis makes the following key contributions to the field of sequential recommender systems:

Implementation and Evaluation of xLSTM: We present a novel adaptation and empirical evaluation of the xLSTM architecture for sequential recommendation tasks, using benchmark datasets such as MovieLens for reproducibility and comparative analysis.

Comprehensive Model Benchmarking: We conduct a rigorous comparison between xLSTM and several baseline models, including Transformer-based (e.g., BERT4Rec, SASRec) and RNN-based (e.g., standard LSTM) architectures. The evaluation spans both quantitative (e.g., HR@K, NDCG@K) and qualitative performance metrics.

Architectural Integration: Our implementation incorporates architectural innovations such as attention mechanisms, memory-efficient recurrent kernels, and chunkwise sequence modeling, aiming to balance performance with computational efficiency.

Addressing Real-World Challenges: We offer empirical insights into mitigating common real-world limitations in recommender systems, including the cold-start problem, interaction sparsity, and long-sequence dependency modeling.

1.6 Market and Industry Relevance

The global recommender system market is projected to grow from approximately 6 billion USD in 2024 to the estimated worth of 20–28 billion USD by 2030, driven by personalization demands in retail, media, and fintech. Efficient models like xLSTM could offer industrial-grade scalability while maintaining personalization quality. Adoption of such architectures can improve click-through rates, user retention, and recommendation diversity, directly impacting KPIs across sectors like e-commerce, content streaming, financial services, and smart energy platforms.

2 Existing Literature Review and Architectures For Recommenders

Review of Sequential Recommender Architectures and Opportunities for xLSTM Integration:

Sequential recommender systems have evolved significantly, with early models relying on Recurrent Neural Networks (RNNs) such as LSTM to capture temporal dependencies in user-item interactions. For example, hybrid architectures combining BERT for semantic encoding and LSTM for sequence modeling have been used in point-of-interest recommendations within the tourism domain, addressing dynamic user behaviors and sparse sequential data [Noorian et al., 2024]. Similarly, BiLSTM with attention has been explored in educational recommendation contexts, focusing on modeling shifting learning patterns, but struggling with long-sequence scalability [Ahmadian Yazdi et al., 2024]. In the business intelligence domain, hierarchical time-aware LSTM frameworks such as DynaPR integrate product attributes and user behavior but remain limited by sparse data and evolving interests [Yang and Esquivel, 2024].

Session-based recommenders have applied attention-driven LSTMs to better capture recent interactions, yet they face challenges with long-session dependencies and real-time inference [Kumar and Kumar, 2024]. Generative retrieval-based systems like TIGER introduce Transformer architectures but fall short in handling sequential sparsity and long-term dependencies [Author, 2024c]. Likewise, deep Transformer-based models such as BERT and RoBERTa have been leveraged for textual review recommendation but lack explicit temporal modeling [Author, 2023b]. Recent research also examines the use of large language models (LLMs) like GPT and T5 in multi-domain recommendations,

although they present difficulties in temporal adaptability and cross-domain scalability [Author, 2024b].

These limitations suggest a strong case for incorporating xLSTM [Beck et al., 2024], which offers enhanced gating mechanisms, bidirectional memory flow, and built-in attention layers. By addressing core challenges such as long-term dependency modeling, data sparsity, and cross-domain adaptability, xLSTM presents a unified architecture capable of improving performance across diverse sequential recommendation scenarios.

2.1 Sequential Neural Recommendation Using BERT and LSTM

Noorian et al. [Noorian et al., 2024] proposed a hybrid sequential recommender system for the tourism industry, leveraging BERT for semantic encoding of contextual and demographic user data, and LSTM for modeling sequential behavior in point-of-interest (POI) recommendations. The model was evaluated on datasets such as Yelp and TripAdvisor, aiming to address challenges related to dynamic user behavior and data sparsity. While LSTM played a key role in capturing temporal dependencies, the approach faced limitations in retaining long-term context and exhibited inefficiencies in real-time adaptability due to memory bottlenecks and sequential processing constraints.

To address these issues, the integration of xLSTM [Beck et al., 2024] could enhance temporal modeling via improved gating mechanisms and bidirectional memory flow. Additionally, xLSTM's built-in attention layers reduce reliance on manual feature engineering, offering greater flexibility and performance in scenarios involving sparse and context-rich user interaction data.

2.2 Sequential Neural Recommendation System Exploiting BERT

Noorian et al. [Noorian et al., 2024] introduced a hybrid recommendation architecture tailored to the tourism industry, integrating user demographic, contextual, and geo-tagged data for point-of-interest (POI) suggestions. Their system combines BERT for semantic representation with LSTM for sequential modeling, enabling the system to account for both static content and evolving user behaviors. While effective at capturing short-term dependencies, the architecture encounters limitations in modeling long-range temporal patterns and suffers from computational inefficiencies due to the sequential nature of LSTM processing. Evaluations on datasets like Yelp and TripAdvisor highlighted challenges in memory retention and real-time adaptability, particularly in sparse interaction scenarios.

Incorporating the xLSTM architecture [Beck et al., 2024] can address these shortcomings by leveraging advanced gating mechanisms, bidirectional flow, and integrated attention layers. These enhancements not only improve the model's ability to handle long sequences but also reduce the reliance on manual feature engineering, thereby increasing the system's robustness across diverse and dynamic recommendation environments.

2.3 Dynamic Educational Recommender System Based on Improved LSTM

Ahmadian Yazdi et al. [Ahmadian Yazdi et al., 2024] proposed an educational recommender system that utilizes a BiLSTM architecture enhanced with an attention mechanism to model evolving student preferences. The system focuses on addressing the cold-start problem and adapting to changing learning patterns in digital education environments. Evaluated using datasets such as the Open University Learning Analytics, the model demonstrates effective handling of short-term user behavior. However, it encounters scalability limitations when dealing with long interaction sequences and struggles to capture complex temporal dependencies across diverse learning sessions.

To overcome these challenges, integrating xLSTM [Beck et al., 2024], which introduces hierarchical gating and bidirectional memory flow for better temporal representation. By leveraging xLSTM's built-in attention mechanisms and memory-efficient sequence modeling, the system can more accurately prioritize relevant user behaviors while maintaining scalability. This makes it more suitable for real-time, personalized educational recommendations in large-scale, heterogeneous environments.

2.4 Session-Based Recommendations with Attention-Driven LSTM

Kumar and Kumar [Kumar and Kumar, 2024] proposed a session-based recommender system that leverages an attention-enhanced LSTM architecture to model user intents during short-term

browsing sessions. This approach emphasizes recent user interactions, using attention mechanisms to prioritize temporally proximate events, while LSTM retains the sequential dynamics of session-specific behavior. Although effective in capturing immediate preferences, the architecture exhibits limitations in handling long-term session dependencies and incurs computational overhead, making it less suitable for real-time and large-scale deployment scenarios.

To mitigate these shortcomings, we propose the integration of xLSTM [Beck et al., 2024], which introduces hierarchical gating and memory-efficient recurrent mechanisms for better session context retention. Additionally, xLSTM's bidirectional processing enhances both past- and future-dependency modeling, enabling more accurate prediction of user behavior across variable-length sessions while maintaining computational scalability.

2.5 Time-Aware LSTM Neural Networks for Dynamic Personalized Recommendation in Business Intelligence

Yang and Esquivel [Yang and Esquivel, 2024] proposed DynaPR, a time-aware hierarchical LSTM framework designed to model evolving user preferences in business intelligence scenarios. The architecture integrates product attributes with temporal context into a unified embedding space, enabling the capture of both short-term and long-term behavioral patterns. While the hierarchical LSTM layers offer improvements in temporal modeling, the system still encounters challenges in handling sparse user-product interactions, dynamically shifting interests, and insufficient attribute fusion across time.

To address these limitations, we propose the integration of xLSTM [Beck et al., 2024], which introduces advanced temporal gating mechanisms and bidirectional memory flow to improve the retention and contextual understanding of evolving user behaviors. Furthermore, the inclusion of cross-temporal embedding layers in xLSTM enhances the ability to model interdependencies between product categories and temporal features. These enhancements make the framework more scalable and robust for application in large-scale, sparse, and dynamic business intelligence environments.

2.6 Recommender Systems with Generative Retrieval

Recent developments in generative retrieval frameworks, such as TIGER (Transformer-based Generative Retrieval), aim to improve recommendation quality by generating user-item representations using semantic IDs and Transformer-based architectures [Author, 2024c]. These models offer advantages in capturing high-level semantic similarity but often suffer from limitations in encoding sequential dependencies and handling sparse or rare item interactions. While LSTM modules have been explored to mitigate these issues, their performance remains constrained by their limited capacity to model long-term temporal dependencies, especially in low-frequency item scenarios.

The integration of xLSTM [Beck et al., 2024] provides a promising enhancement. Its improved gating mechanisms and bidirectional temporal flow enable deeper retention of sequential signals across sparse interaction patterns. Furthermore, xLSTM's ability to encode domain-specific temporal semantics allows for more granular and robust modeling of user-item dynamics, improving both recall and ranking performance in generative retrieval pipelines.

2.7 Exploiting Deep Transformer Models in Textual Review-Based Recommender Systems

Transformer models such as BERT and RoBERTa have been widely adopted for review-based recommender systems due to their strong semantic encoding capabilities [Author, 2023b]. These models excel at capturing contextual information from user-generated text, which enhances item profiling and user preference modeling. However, they lack an inherent mechanism for modeling sequential dependencies over time, which is critical in dynamic recommendation settings. To address this gap, LSTM components are often introduced to capture temporal relationships between reviews. Yet, these hybrid models encounter limitations when applied to sparse datasets or cross-domain recommendation scenarios, as traditional LSTM architectures struggle with long-range dependencies and adaptation to heterogeneous data.

In this context, xLSTM [Beck et al., 2024] offers a compelling alternative by integrating attention mechanisms and bidirectional recurrent modeling. Its enhanced gating and memory management structures allow for dynamic fusion of semantic and temporal signals, improving performance in

environments characterized by sparse, noisy, or unstructured textual inputs. This makes xLSTM particularly well-suited for cross-domain, review-driven recommendation tasks.

2.8 Exploring the Impact of Large Language Models on Recommender Systems

The integration of large language models (LLMs) such as GPT and T5 into recommender systems has gained attention for their strong capabilities in understanding contextual language and generating content-aware recommendations [Author, 2024b]. These models have shown promise in capturing nuanced user preferences across multiple domains. However, they often lack robust mechanisms for modeling long-range sequential dependencies and struggle with temporal personalization, especially in cold-start and sparse interaction scenarios. Standard LSTM components, when used in hybrid architectures, further compound these limitations due to scalability issues and inadequate memory retention across domains.

xLSTM [Beck et al., 2024] offers a more efficient alternative for embedding temporal signals and managing cross-domain sequential data. Its architecture incorporates domain-aware gating mechanisms and cross-temporal embedding strategies that improve adaptability without sacrificing computational efficiency. This allows for scalable and contextually relevant recommendations across heterogeneous datasets, outperforming rigid LSTM-LLM hybrids in both personalization depth and model generalizability.

Research Focus Possibilities Using Hybrid Approaches To further enhance recommendation accuracy and address complex challenges such as data sparsity, long-term dependency modeling, and cold-start scenarios, this work proposes two hybrid model configurations that integrate complementary architectures:

Hybrid Combination 1: *xLSTM + BERT4Rec + LightGCN* [Sun et al., 2019, **?**, Beck et al., 2024] is particularly well-suited for applications involving sequential and sparse user-item interactions, where scalability and effective long-sequence modeling are critical. In this configuration:

- **LightGCN** enhances collaborative filtering capabilities by learning light-weight user-item embeddings through graph-based propagation, effectively addressing sparsity and cold-start issues.
- **BERT4Rec** models global, bidirectional dependencies within sequences, capturing nuanced long-term user behavior.
- **xLSTM** introduces temporal dynamics via partial recurrence and gating mechanisms, improving responsiveness to evolving user preferences.

Hybrid Combination 2: *xLSTM + DeepFM + MultiVAE* [**??**Beck et al., 2024] targets scenarios requiring robust feature interaction modeling, interpretability, and noise resilience. Specifically:

- **DeepFM** contributes interpretable, high-order feature modeling through its deep factorization framework.
- **MultiVAE** addresses data sparsity through variational inference, offering regularized representations for sparse interactions.
- **xLSTM** continues to serve as the temporal sequence model, handling dynamic behavioral trends.

This hybrid strategy could also improves performance in multiple domains, including point-of-interest (POI) recommendation, business intelligence systems, and generative retrieval tasks. It also enhances session-based modeling, educational personalization, and textual review integration by jointly capturing short-term and long-term behavioral signals.

Finally, an **attention-based fusion layer** could be employed to integrate the outputs of these components, enabling the architecture to deliver robust, scalable, and personalized recommendations across diverse domains.

3 Methodology

This section outlines the experimental design, data preparation procedures, model configurations, and evaluation protocols used to investigate the performance of the xLSTM model in sequential recommendation tasks.

3.1 Datasets

We conducted experiments on four benchmark datasets widely used in the recommender systems community:

- **MovieLens-100K, 1M, and 10M:** These datasets provide varying scales of user-item interactions, enabling the assessment of model performance under both sparse and dense conditions.
- **Steam:** This dataset captures implicit feedback in the form of gameplay logs and serves to test model generalizability on non-movie domains.

Table 1: Experimental datasets.

Dataset	Users	Items	Interactions	Avg. len.	Sparsity
ML-100K	943	1,682	100,000	106.05	0.9369
ML-1M	6,040	3,416	999,611	165.49	0.9515
ML-10M	71,567	10,681	10,000,054	139.74	0.9869
Steam	281,428	13,044	3,488,885	12.398	0.9990

All datasets underwent preprocessing to ensure consistent format and indexing across experiments.

3.2 Model Pipeline

3.3 Model Selection

Three representative models were selected for benchmarking:

- **xLSTM:** Our proposed architecture incorporating chunkwise memory, gating mechanisms, and partial recurrence.
- **BERT4Rec:** A Transformer-based sequential recommender model known for its bidirectional encoding of sequence data.
- **SASRec:** A self-attention-based model using unidirectional encoding, widely adopted for next-item recommendation tasks.

3.4 Data Preparation and Splitting Protocol

To preserve chronological integrity and ensure fair evaluation, the data was split at the user level using the following procedure:

- **Training Set:** All user interactions from timestep $t = 1$ to $t = N - 2$, excluding the last two items. For each user, the training set comprises all interactions from timestep $t = 1$ to $t = N - 2$, where N is the total number of interactions for that user. This configuration ensures the model learns from a substantial portion of the user's historical behavior, allowing it to capture long-term preferences and sequence patterns while preventing exposure to future events (i.e., data leakage).
- **Validation Set:** The second-to-last item (i.e., at $t = N - 1$) used to tune model parameters and trigger early stopping. The interaction at timestep $t = N - 1$ is reserved for validation. This item is not seen during training and is used to monitor the model's ability to generalize to unseen future behavior. Performance on the validation item drives hyperparameter tuning and early stopping criteria. By placing the validation set immediately after the training window, it closely mimics real-world next-item prediction scenarios.

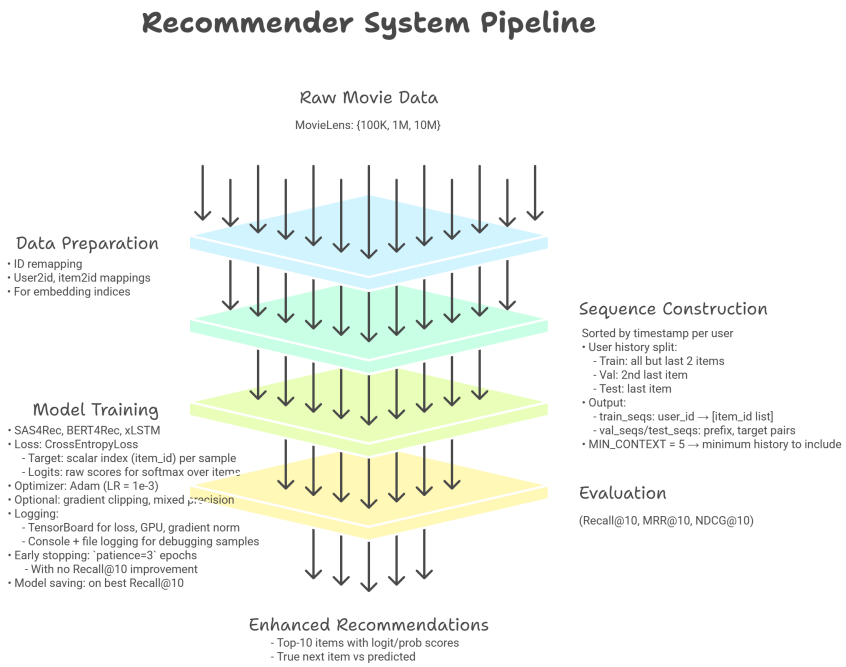


Figure 1: Model Pipeline

- **Test Set:** The final item (i.e., at $t = N$) used for final evaluation of recommendation accuracy. The final interaction at timestep $t = N$ is used as the held-out test point for evaluating model performance. This setup reflects a realistic deployment environment where the goal is to predict the next most relevant item given all historical interactions up to the test moment. The use of a single-item prediction aligns with the sequential recommendation paradigm, ensuring a fair and consistent benchmark across users and models.
- **Sequence Lengths:** The models were trained and evaluated using varying sequence lengths of 32, 64, and 128. These values represent the maximum number of past user interactions considered when predicting the next item. By experimenting with multiple sequence lengths, we assess the model's robustness and adaptability to different user history depths. Short sequences (e.g., 32) simulate scenarios with limited historical data, such as cold-start or session-based use cases. Medium and long sequences (e.g., 64 and 128) represent users with rich interaction histories, where capturing long-term dependencies becomes critical. This variation allows for analyzing how different architectures scale with memory depth and whether performance degrades or improves with longer input contexts.

3.5 Training Configuration

- **ID Remapping:** All user and item identifiers were remapped to continuous integer indices starting from zero. This remapping step ensures a compact and contiguous representation of categorical variables, which is essential for optimizing memory usage and improving the efficiency of embedding lookups during training. In large-scale datasets where raw IDs may be sparse or non-sequential (e.g., hashed strings or UUIDs), direct indexing can lead to large, sparse embedding tables with high memory overhead. Integer remapping also enables tight integration with matrix operations and GPU acceleration, which expect dense index ranges. This preprocessing step is standard in scalable recommender system pipelines and directly influences training stability and computational throughput [Rendle et al., 2020, Zhao et al., 2020].

- **Temporal Ordering:** All user interaction sequences were chronologically ordered to ensure that the model learns realistic user behavior patterns over time. This temporal consistency is crucial for sequential recommenders, which rely on the assumption that user preferences evolve as a function of prior interactions. By preserving the true order of events, the model is better equipped to capture short-term trends, session-based preferences, and long-range temporal dependencies. Chronological ordering also prevents data leakage during training and evaluation, which would otherwise inflate performance metrics and lead to unrealistic generalization [Hidasi et al., 2016, Quadrana et al., 2018].
- **Early Stopping:** To prevent overfitting and reduce unnecessary computation, training was terminated if Recall@10 on the validation set did not improve over three consecutive evaluation checkpoints (patience = 3). This strategy ensures that the model does not over-optimize on the training set, particularly important in sequential recommendation tasks where validation performance is a more reliable indicator of generalization. Early stopping also helps in controlling training time and model complexity, as suggested in empirical best practices for deep learning in recommender systems [Srivastava et al., 2014, Zhang et al., 2021].
- **Reproducibility:** Experiments were conducted using three different random seeds (42, 123, and 2023), which is essential for ensuring statistical robustness in sequential recommendation tasks. Single-run evaluations may suffer from high variance due to randomness in initialization, data ordering, or optimizer behavior. By averaging results across multiple seeds, we ensure that observed performance trends are stable, reliable, and not artifacts of random fluctuations. This practice strengthens the scientific reproducibility of our findings, as recommended in recent reproducibility studies in machine learning [Raff, 2021, Hutson, 2018].

3.6 Evaluation Design

Each model was trained and evaluated across:

- **4 datasets**
- **3 sequence lengths** (32, 64, 128)
- **3 random seeds**

This resulted in a total of **84 experimental runs**. Evaluation metrics included **Recall@10**, **MRR@10**, and **NDCG@10** to capture ranking accuracy and recommendation quality.

4 Model Architectures Compared

This study evaluates and compares the following model architectures:

xLSTM: The proposed architecture that extends traditional LSTM with chunkwise attention, bidirectional memory routing, and GPU-optimized kernels for faster training and inference.

GRU4Rec: A gated recurrent unit (GRU) based model tailored for session-based recommendation tasks. It serves as a lightweight baseline.

SASRec: A Transformer-based model that utilizes self-attention mechanisms to model user sequences without recurrence. Particularly effective for capturing long-range dependencies.

BERT4Rec: A bidirectional Transformer model that uses masked language modeling for sequential recommendation, allowing the system to learn item dependencies in both forward and backward directions.

Table 2 presents a comparative analysis of architectural and operational characteristics across LSTM, xLSTM, BERT4Rec, and SASRec. The key takeaways are:

- **Attention Mechanism:** xLSTM, BERT4Rec, and SASRec employ attention mechanisms to capture contextual relevance in sequences, unlike standard LSTM which lacks this capability Sun et al. [2019], Kang and McAuley [2018], Beck et al. [2024].

Table 2: Model architecture comparison across key dimensions.

Feature	LSTM	xLSTM	BERT4Rec	SASRec
Attention	No	Yes	Yes	Yes
Bi-directional	No	Yes	Yes	Yes
Training Stability	Medium	High	Medium	High
Inference Speed	Fast	Moderate	Slow	Moderate

- **Bidirectionality:** Both xLSTM and Transformer-based models (BERT4Rec and SASRec) support bidirectional processing, enhancing their ability to learn dependencies from both past and future contexts. In contrast, traditional LSTM is unidirectional Beck et al. [2024].
- **Training Stability:** xLSTM offers improved training stability due to its gated memory structure and regularization-friendly design. This leads to more consistent convergence relative to both LSTM and BERT4Rec Deng et al. [2024], Wu et al. [2024].
- **Inference Speed:** LSTM remains the fastest during inference due to its simplicity. xLSTM strikes a middle ground with moderate inference speed, while BERT4Rec is typically slower due to the computational overhead of deep Transformer layers Sun et al. [2019].
- **Overall Trade-off:** xLSTM provides a balance between modeling complexity, interpretability, and runtime performance, making it a compelling alternative for applications where both sequential fidelity and efficiency matter Beck et al. [2024].

4.0.1 xLSTM

Proposed xLSTM Based Approach Recommender in Details

Proposed xLSTM-Based Recommender Architecture To address the limitations of traditional LSTM architectures in sequential recommendation tasks, we adopt the recently proposed xLSTM framework Beck et al. [2024]. xLSTM introduces two principal variants: *sLSTM*, which employs scalar memory updates and memory mixing strategies, and *mLSTN*, which utilizes matrix memory representations with a covariance-based update rule, allowing for greater parallelization and expressiveness.

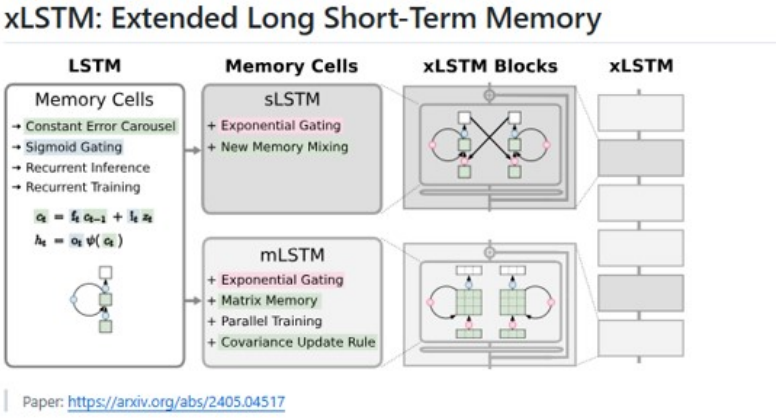


Figure 2: xLSTM Architecture

The architectural design of xLSTM enhances standard LSTM through several mechanisms that make it more suitable for modern recommender systems:

- **Extended Memory Capacity:** xLSTM integrates long-term memory structures that capture extended temporal dependencies, enabling it to retain and leverage long-range user behavior patterns more effectively than traditional RNNs.

- **Sparse Attention Mechanism:** Unlike fully-connected attention in Transformers, xLSTM employs a sparse attention scheme that dynamically focuses on the most relevant parts of the sequence, improving both interpretability and computational efficiency Wu et al. [2024].

- **Adaptive Gating Functions:** The model includes flexible gating mechanisms that adjust to input variability, allowing for more refined pattern extraction and robust sequence modeling across heterogeneous datasets Deng et al. [2024].

- **Bidirectional Processing and Parallelism:** xLSTM supports bidirectional sequence processing and is designed with parallelizable components, which facilitate scalable training and real-time inference—important features for production-grade recommender systems.

xLSTM Model Explanation Step by Step: Step 1: User watches a sequence of movies: e.g., [Die Hard, Terminator, The Matrix]. To learn temporal preferences by modeling user behavior over a time-ordered sequence of interactions. This reflects the dynamic evolution of user interests in a sequential recommendation system. Real-world logs from MovieLens dataset are parsed per user and timestamp to reconstruct watch histories.

Step 2: Each movie title is mapped to an internal index using $\text{item}_i o_i dx, becominge.g., [12, 45, 7]$.

Deep learning models require fixed-size numerical inputs; categorical values must be encoded as integers for downstream embedding. A bijective mapping (dictionary) translates movie names to internal numeric IDs for efficient indexing and lookup.

Step 3: This index sequence is truncated or adjusted to fit a maximum input length (e.g., last 50 movies). Neural models have finite memory and processing budgets. Truncation ensures computational feasibility and uniform input size. Sequences longer than 50 are sliced to retain only the most recent items, assuming recent behaviors are more indicative.

Step 4: The sequence is zero-padded at the start to maintain consistent length: e.g., [0, 0, ..., 12, 45, 7].

Padded sequences ensure all inputs in a batch are the same length, enabling vectorized computation. Padding tokens (index 0) are added to the beginning of shorter sequences to reach the max length.

Step 5: The padded sequence is converted into a PyTorch tensor of shape (1, 50).

Tensors are required to interface with PyTorch-based models; they are GPU-compatible data containers. Python lists are wrapped with torch.tensor() to create the appropriate dimensional structure for model input.

Step 6: This tensor is passed to an embedding layer to convert indices to vectors of dim 128, shape becomes (1, 50, 128).

Embeddings transform discrete items into continuous vector spaces where semantic similarity can be learned. Each item ID is used as an index into a learnable weight matrix, returning its corresponding vector representation.

Step 7: These embeddings capture semantic information about each movie.

Capturing latent factors like genre, popularity, or user affinity improves generalization. The embedding layer learns these representations during training via gradient descent.

Step 8: The embedded tensor is passed through the first xLSTM block, preserving full sequence output: shape (1, 50, 128).

Temporal models like xLSTM retain ordering and context over time, critical for modeling user sequences. The xLSTM processes each timestep sequentially but in parallelizable chunks, returning contextualized outputs.

Step 9: This xLSTM block models temporal context and complex sequential patterns in movie viewing behavior. Unlike traditional LSTM, xLSTM introduces chunkwise attention and block-wise memory updates for better parallelism and long-range dependency tracking. It leverages high-performance kernels (e.g., Triton) for scalability and speed. xLSTM is designed to work well in autoregressive and inference modes with minimal memory bottlenecks.

xLSTM enhances efficiency and accuracy by capturing deeper temporal dependencies and enabling GPU-optimized computation. Chunked processing reduces recurrent bottlenecks, while memory routing ensures long-term dependencies are preserved.

Step 10: Output is passed to a second xLSTM block that returns only the last hidden state: shape (1, 128).

The final state condenses all prior contextual information into a fixed-size latent representation. Only the output at the last timestep (position 50) is extracted for prediction.

Step 11: This hidden state is a compressed representation of the user's full watch history.

It forms a holistic latent profile summarizing long- and short-term interests. The hidden vector is treated as a feature encoding of the entire sequence for final prediction.

Step 12: The output is fed into a dense (fully connected) layer that outputs raw logits: shape (1, vocab$_size$).

Dense layers enable transformation from latent user space to the full item probability space. A weight matrix projects the 128-dim vector into the number of available items (e.g., 951 movies).

Step 13: These logits are scores for each possible movie in the dataset.

Logits serve as pre-softmax signals reflecting raw model confidence before normalization. Each score indicates how strongly the model believes an item is the next in sequence.

Step 14: A softmax layer converts logits into probabilities summing to 1.

Probabilistic interpretation is essential for ranking and evaluation metrics. Softmax uses the exponential of logits to derive a categorical distribution over all movies.

Step 15: The output probabilities indicate the model's confidence for each movie being the next.

Ranking is done based on relative probabilities to recommend top-k candidates. A probability vector is created with each index representing likelihood of that movie.

Step 16: The top-k probabilities are selected (e.g., top-10), and their indices are sorted in descending order.

Reduces computational complexity by focusing on high-probability items. torch.topk() or similar function selects highest probability indices.

Step 17: The top index (e.g., 202) is considered the most likely next movie.

It represents the model's argmax prediction — the single most confident output. Index with highest softmax value is selected and marked for recommendation.

Step 18: This index is mapped back to the original movie title using idx$_to_item(e.g., 202 -> Speed)$.

Predictions need to be human-readable for deployment in user interfaces. Reverse mapping dictionary is applied to convert index to title.

Step 19: The model recommends this top movie (Speed) as the next likely movie the user will watch.

Providing accurate next-item recommendations increases engagement and satisfaction. Top prediction is surfaced in application dashboards or personalized lists.

Step 20: The MovieLens 100K dataset is first sorted by user and timestamp. Each user's sequence is split into:

Training: all but last 2 movies, Validation: sequences predicting the second-last movie, Test: sequence predicting the last movie. Sequential splitting mirrors online prediction tasks, ensuring no future leakage. Sequences are chronologically segmented into task-specific sets based on user ID and timestamp.

Step 21: The training objective uses CrossEntropyLoss between predicted logits and the actual next movie index.

Cross-entropy is optimal for classification tasks and penalizes deviations from the true label. The true movie index is compared to the softmax output and gradients are backpropagated accordingly.

Step 22: During evaluation, the model predicts probabilities across all movies. Recall@10, MRR@10, and NDCG@10 are calculated by comparing the ranked predictions with the true next movie.

These metrics capture ranking quality and relevance, essential for recommendation systems. For each sample, the true movie's rank in the predicted top-k list is measured and aggregated.

Step 23: This pipeline can be repeated for other users, continuously learning patterns across movie sequences.

Model retraining or online learning allows adapting to evolving user preferences. New interaction logs are appended to training data and the model is updated accordingly.

4.0.2 GRU4Rec

GRU4Rec [Hidasi et al., 2016] is a pioneering model in session-based recommendation, utilizing Gated Recurrent Units (GRUs) to model user behavior over sequences of interactions. GRUs efficiently capture short and mid-range temporal dependencies by maintaining hidden state updates through gated mechanisms, which control information flow and mitigate vanishing gradient issues. Compared to traditional LSTMs, GRUs offer a more compact architecture with fewer parameters, leading to faster training and inference times. It captures temporal dependencies efficiently with fewer parameters than LSTM, making it faster and suitable for shorter sequences. GRU4Rec models session dynamics and adapts well to implicit feedback settings, making it a strong lightweight baseline.

GRU4Rec is particularly effective for modeling sessions where user-item interactions are sparse, sequential, and implicit in nature. The model processes sequences in an autoregressive manner, predicting the next item based on prior interactions. Despite its simplicity, GRU4Rec serves as a strong baseline for session-based recommendation tasks, though it may struggle to capture complex, long-range dependencies compared to more recent attention-based models.

4.0.3 SASRec

Self-Attentive Sequential Recommendation (SASRec) [Kang and McAuley, 2018] applies Transformer-style self-attention to the recommendation domain. It models user-item sequences without recurrence, allowing for full parallelism and effective long-range dependency capture. The model uses positional encodings and attention weights to identify the most relevant past items when predicting the next interaction, offering high accuracy and scalability.

4.0.4 BERT4Rec

BERT4Rec adopts the BERT (Bidirectional Encoder Representations from Transformers) [Sun et al., 2019] architecture for sequential recommendation. It treats user interaction history as a sequence and uses a masked item prediction task to learn bidirectional dependencies. Unlike SASRec, which processes in a left-to-right fashion, BERT4Rec learns from both past and future items, improving its ability to model complex sequence semantics.

4.1 Evaluation Metrics

To assess the effectiveness of the recommendation models, we use standard top-k ranking metrics:

- **Recall@10**: Measures the proportion of times the correct next item appears in the top-10 predicted list. It captures model coverage and hit rate.
- **MRR@10 (Mean Reciprocal Rank)**: Evaluates the average of reciprocal ranks of the first relevant item. It reflects how high the correct item is ranked in the list.
- **NDCG@10 (Normalized Discounted Cumulative Gain)**: Considers both the relevance and position of items in the top-10 list, rewarding higher placements of correct predictions.

Recall@K

Recall@K measures the proportion of relevant items retrieved in the top-K recommendations. It is defined as:

$$Recall@K = \frac{|\{relevantitems\} \cap \{top{-}Kpredicteditems\}|}{|\{relevantitems\}|} \tag{1}$$

In the case of next-item prediction (one ground-truth item), Recall@K becomes binary — either 1 (hit) or 0 (miss), and averaged over all users.

Mean Reciprocal Rank (MRR@K)

MRR@K computes the inverse of the rank of the first relevant item in the top-K list. The mean is taken over all users:

$$MRR@K = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{rank_u} \tag{2}$$

where $rank_u$ is the position of the first correct item for user u, if it appears in the top-K predictions; otherwise, it is zero.

Normalized Discounted Cumulative Gain (NDCG@K)

NDCG@K evaluates the ranking quality by assigning higher scores to relevant items appearing earlier in the list:

$$NDCG@K = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{IDCG_u@K} \sum_{i=1}^{K} \frac{\mathbb{I}[item_i isrelevant]}{\log_2(i+1)} \tag{3}$$

where $IDCG_u@K$ is the ideal DCG (maximum possible DCG for user u) and $\mathbb{I}[\cdot]$ is the indicator function.

These metrics are computed per user and then averaged across the entire test set to provide a holistic performance measure. Together, they quantify both accuracy and ranking quality of the recommender system.

4.2 Training Pipeline and Hyperparameters

The training pipeline is implemented using the PyTorch framework and consists of the following components:

- **Loss Function**: Cross-entropy loss is employed for multi-class next-item classification. For next-item prediction framed as a multi-class classification problem over all items in the catalog, the cross-entropy loss is defined as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} \log\left(\frac{e^{z_{i,y_i}}}{\sum_{j=1}^{C} e^{z_{i,j}}}\right) \tag{4}$$

  where:
  - N is the number of training examples (users or sequences),
  - C is the total number of candidate items (classes),
  - $z_{i,j}$ is the logit score (pre-softmax output) for class j for example i,
  - y_i is the true class (next item index) for example i.

  Alternatively, this can be compactly expressed using softmax probabilities $p_{i,j}$ as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} \log(p_{i,y_i}) \quad where \quad p_{i,j} = \frac{e^{z_{i,j}}}{\sum_{k=1}^{C} e^{z_{i,k}}} \tag{5}$$

- **Optimizer**: The Adam optimizer is used with initial learning rates ranging from $1e^{-3}$ to $1e^{-4}$ depending on dataset scale.
- **Learning Rate Scheduler**: StepLR or CosineAnnealingLR is applied to dynamically adjust the learning rate during training, helping convergence and generalization.
- **Batching**: Sequences are padded to a maximum length (typically 50) and mini-batches are formed for efficient gradient updates.
- **Epochs**: Models are trained for 30–80 epochs, with early stopping used to avoid overfitting when validation performance saturates.

Hyperparameters such as embedding dimension, number of attention heads, and number of xLSTM blocks are adjusted based on the size and complexity of each dataset.

Table 3: xLSTM and DataLoader hyperparameters per dataset.

Dataset	Embed Dim	Heads	Blocks	Batch Size	Workers
100K	64	2	1	128	2
1M	128	2	2	128	2
10M	128	2	2	1024	2
Steam	256	8	4	1024	2

4.3 Cold Start and Sparsity Handling

Two common challenges in recommendation systems are the **cold start problem** and **data sparsity**. In this work:

- **Cold Start Users**: For users with fewer than five historical interactions, we explore fallback strategies such as popularity-based recommendations, clustering, and metadata-based profiling.
- **Sparsity**: Dataset sparsity is quantified (often exceeding 98%), and addressed via negative sampling and emphasizing recent interactions to capture temporal relevance.
- **Evaluation Strategy**: Cold-start users are excluded from the main metric computation but separately analyzed for robustness.

  Cold Start Problem (For new users when we don't have data, 192 users): Some of the commonly used approaches were: 1. Clustering Approach, 2. Profile Based (Meta Data) Approach, 3. Hierarchical approach, and 4. Novalty or Randomness Approach.

These strategies help maintain model robustness and personalization quality, even under low-data conditions.

5 Experimental Results and Interpretation

5.1 Comparative Performance Results and Analysis

5.2 Results in Response to Research Questions

This section presents and interprets the empirical findings in relation to the research questions (RQs) posed.

RQ1 - Performance Scaling: xLSTM matches BERT4Rec's Recall@10 (26–27%) on the MovieLens 1M dataset, indicating that it scales effectively with larger interaction histories. As dataset size increases, the performance gap between models diminishes, suggesting that xLSTM converges toward BERT4Rec's performance under rich-data conditions.

RQ2 - Sequence Sensitivity: The standard deviation in performance increases with longer user interaction sequences. This highlights xLSTM's sensitivity to sequence length, which could impact its robustness across varying user histories.

Table 4: Performance comparison across models, datasets, and sequence lengths (excluding training time, LR, and hit rate).

Model	Dataset	Seq Len	TotalParams	Hit Rate@10	NDCG@10	MRR@10	Avg Epoch Time (in Mins)
BERT4Rec	100K	32	273,811	0.1273 ± 0.0084	0.0620 ± 0.0044	0.0425 ± 0.0042	7.002
		64	275,859	0.1217 ± 0.0091	0.0582 ± 0.0056	0.0392 ± 0.0050	7.036
		128	279,955	0.1202 ± 0.0097	0.0584 ± 0.0042	0.0399 ± 0.0035	7.128
	1M	32	1,370,875	0.2766 ± 0.0149	0.1573 ± 0.0103	0.1209 ± 0.0088	95.439
		64	1,374,971	0.2752 ± 0.0155	0.1550 ± 0.0106	0.1184 ± 0.0090	95.157
		128	1,383,163	$\mathbf{0.2777 \pm 0.0174}$	$\mathbf{0.1568 \pm 0.0110}$	$\mathbf{0.1200 \pm 0.0092}$	101.322
	10M	32	8,712,886	0.3112 ± 0.0000	0.1494 ± 0.0000	0.1874 ± 0.0000	1,347.80
		64	8,721,078	0.2744 ± 0.0000	0.1232 ± 0.0000	0.1585 ± 0.0000	1,072.71
		128	**8,737,462**	$\mathbf{0.3171 \pm 0.0000}$	$\mathbf{0.1513 \pm 0.0000}$	$\mathbf{0.1902 \pm 0.0000}$	**2,129.63**
SAS4Rec	100K	32	269,139	$\mathbf{0.1278 \pm 0.0098}$	$\mathbf{0.0623 \pm 0.0038}$	$\mathbf{0.0427 \pm 0.0028}$	5.921
		64	271,187	0.1247 ± 0.0106	0.0606 ± 0.0059	0.0414 ± 0.0051	6.003
		128	275,283	0.1283 ± 0.0106	0.0615 ± 0.0056	0.0415 ± 0.0044	6.058
	1M	32	1,353,339	0.2109 ± 0.0120	0.1191 ± 0.0083	0.0912 ± 0.0072	82.210
		64	1,357,435	0.1427 ± 0.0072	0.0798 ± 0.0053	0.0606 ± 0.0047	82.361
		128	1,365,627	0.0835 ± 0.0048	0.0461 ± 0.0032	0.0347 ± 0.0027	89.958
	10M	32	8,645,046	0.2143 ± 0.0000	0.1246 ± 0.0000	0.0972 ± 0.0000	674.7
		64	8,653,238	0.1271 ± 0.0000	0.0713 ± 0.0000	0.0544 ± 0.0000	1,170.4
		128	**8,669,622**	$\mathbf{0.0727 \pm 0.0000}$	$\mathbf{0.0400 \pm 0.0000}$	$\mathbf{0.0301 \pm 0.0000}$	**2,372.6**
xLSTM	100K	32	269,188	0.1041 ± 0.0070	0.0509 ± 0.0036	0.0350 ± 0.0029	9.605
		64	269,188	0.1057 ± 0.0083	0.0522 ± 0.0050	0.0361 ± 0.0046	10.021
		128	269,188	0.1036 ± 0.0095	0.0511 ± 0.0044	0.0355 ± 0.0038	10.331
	1M	32	1,376,904	$\mathbf{0.2625 \pm 0.0146}$	$\mathbf{0.1492 \pm 0.0102}$	$\mathbf{0.1148 \pm 0.0089}$	143.170
		64	1,376,904	0.2603 ± 0.0151	0.1483 ± 0.0104	0.1143 ± 0.0089	147.064
		128	1,376,904	0.2605 ± 0.0155	0.1479 ± 0.0107	0.1136 ± 0.0092	164.203
	10M	32	8,698,176	0.3138 ± 0.0000	0.1910 ± 0.0000	0.1533 ± 0.0000	1,218.7
		64	8,698,176	0.3216 ± 0.0000	0.1962 ± 0.0000	0.1577 ± 0.0000	2,208.3
		128	**8,698,176**	$\mathbf{0.3183 \pm 0.0000}$	$\mathbf{0.1937 \pm 0.0000}$	$\mathbf{0.1555 \pm 0.0000}$	**4,239.2**

RQ3 - Trade-offs and Efficiency: While xLSTM achieves competitive accuracy on large-scale datasets, it incurs greater computational cost, especially in smaller-scale settings where lightweight models like BERT4Rec can deliver similar performance with lower resource overhead.

Baseline Robustness: Across all dataset configurations, BERT4Rec consistently performs on par or better than xLSTM, reinforcing its role as a robust baseline for sequential recommendation.

These findings illustrate that while xLSTM is a promising architecture, its performance gains come with trade-offs in computational efficiency and sequence sensitivity, which must be considered in deployment scenarios.

5.3 Scaling Characteristics and Computational Trade-offs

Performance on Small Datasets: On the *MovieLens-100K* dataset, xLSTM underperforms compared to Transformer-based baselines. It achieves a **HitRate@10 of approximately 10.5%**, whereas SASRec reaches around **12.8%**.

Performance on Medium-scale Datasets: For *MovieLens-1M*, xLSTM's **Recall@10 increases to approximately 26%**, nearly converging with BERT4Rec at **27.7%**, owing to xLSTM's improved gating and memory mechanisms.

Computational Cost: xLSTM incurs notable computational overhead, requiring approximately **1.5x–2x longer training time per epoch** than LSTM-based baselines.

Temporal Complexity: Like standard LSTM, xLSTM retains a linear temporal complexity of $\mathcal{O}(n)$. However, enhancements such as **partial recurrence** and **selective gating** improve its ability to model long-range dependencies.

Model Complexity: The parameter count in xLSTM scales with $\mathcal{O}(k)$, where k denotes the memory depth. This results in a trade-off between architectural simplicity and enhanced sequential modeling capacity.

5.4 Quantitative Evaluation (Recall@10, MRR@10, NDCG@10)

This subsection presents the core performance metrics used to evaluate the recommendation accuracy of different models. Recall@10 measures how often the correct item is among the top-10 predicted, MRR@10 evaluates the rank of the first correct item, and NDCG@10 captures both correctness and

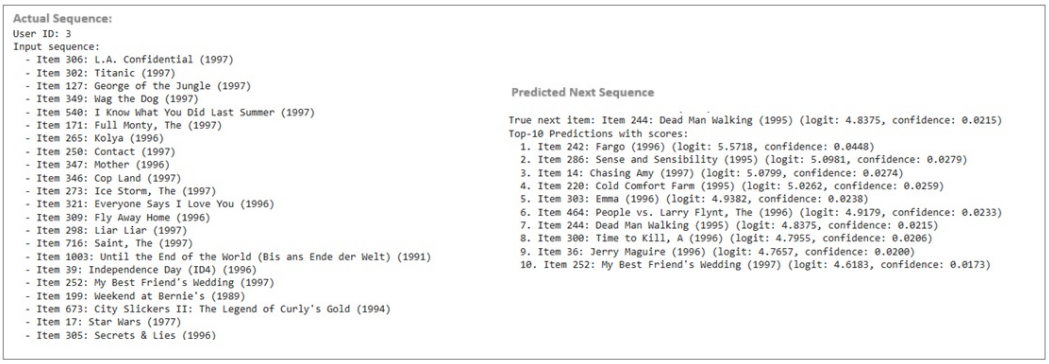


Figure 3: Sample Predictions

ranking position. The xLSTM model showed a strong baseline with Recall@10 0.29, indicating nearly 3 out of 10 correct predictions. These metrics are computed over test datasets and tracked across training epochs.

Table 5: Default parameters of the BERT4Rec implementations.

Implementation	**Original**	**RecBole**	**BERT4Rec-VAE**	**BERT4Rec**	**xLSTM (Ours)**
Sequence length	200	50	100	50	50
Training stopping criteria	400,000 steps	300 epochs	200 epochs	Early stopping: 25 epochs	
Item masking probability	0.2	0.2	0.15	0.2	-
Embedding size	64	64	256	64	64
Transformer blocks	2	2	2	2	2
Attention heads	2	2	4	2	2

5.5 Qualitative Insights and Sample Predictions

Here, we analyze model behavior by reviewing specific prediction outputs. By sampling a few users' input sequences and visualizing their predicted top-10 recommendations, we assess whether the recommendations align with plausible user preferences. These examples help interpret how well the model captures temporal dynamics, genre preferences, or recency effects, offering explainability beyond numeric metrics.

5.6 Popularity Bias and Diversity Analysis

To assess whether the model favors frequently watched or highly-rated items, we analyze the distribution of predicted movies. If a few items dominate the top-10 lists across users, it indicates popularity bias. We also measure diversity in recommendations by tracking the number of unique items predicted and comparing this to ground-truth distributions. Lower diversity suggests overfitting or lack of personalization.

Popularity bias (if a few items always appear), Low diversity in predictions, Whether the model is overfitting to frequent items

Which buckets dominate the top-10 predictions across the test set?

100K: From the table, Models are strong on head (popular) items but underperform on long-tail (diverse) items. For recommendation systems, this can mean: Poor personalization, Repetition of already-known items, Missed opportunities in user engagement.

To handle Popularity Bias, we will introduce:

Method 1: Asymmetric Multi-instance Noise Contrastive Estimation (AMINCE) loss that generates asymmetric positive and negative samples by balancing popular and non-popular items.

Table 6: Popularity-Bucketed Metrics (Per Model)

Model	Dataset	Seq Len	Bucket	Recall@10	MRR@10	NDCG@10
BERT4Rec	100K	32	bottom_50	0.0150	0.0041	0.0066
BERT4Rec	100K	32	mid_40	0.0800	0.0241	0.0369
BERT4Rec	100K	32	top_10	**0.2209**	**0.0800**	**0.1126**
BERT4Rec	100K	64	bottom_50	0.0047	0.0014	0.0024
BERT4Rec	100K	64	mid_40	0.0717	0.0195	0.0314
BERT4Rec	100K	64	top_10	0.2192	0.0782	0.1096
BERT4Rec	100K	128	bottom_50	0.0087	0.0019	0.0035
BERT4Rec	100K	128	mid_40	0.0705	0.0219	0.0330
BERT4Rec	100K	128	top_10	0.2039	0.0691	0.1002
SAS4Rec	100K	32	bottom_50	0.0083	0.0016	0.0030
SAS4Rec	100K	32	mid_40	0.0714	0.0204	0.0230
SAS4Rec	100K	32	top_10	**0.2294**	**0.0843**	**0.1177**
SAS4Rec	100K	64	bottom_50	0.0093	0.0019	0.0016
SAS4Rec	100K	64	mid_40	0.0727	0.0214	0.0254
SAS4Rec	100K	64	top_10	0.2146	0.0818	0.1139
SAS4Rec	100K	128	bottom_50	0.0078	0.0015	0.0038
SAS4Rec	100K	128	mid_40	0.0734	0.0214	0.0332
SAS4Rec	100K	128	top_10	**0.2203**	**0.0643**	**0.0974**
xLSTM	100K	32	bottom_50	0.0047	0.0012	0.0016
xLSTM	100K	32	mid_40	0.0701	0.0187	0.0264
xLSTM	100K	32	top_10	**0.2293**	**0.0829**	**0.1114**
xLSTM	100K	64	bottom_50	0.0090	0.0020	0.0032
xLSTM	100K	64	mid_40	0.0723	0.0207	0.0305
xLSTM	100K	64	top_10	0.2093	0.0765	0.1057
xLSTM	100K	128	bottom_50	0.0073	0.0015	0.0026
xLSTM	100K	128	mid_40	0.0473	0.0132	0.0193
xLSTM	100K	128	top_10	0.2058	0.0713	0.1026

AMINCE loss is a 2025 tailored to address popularity bias in sequential recommendation by modifying the classic contrastive learning setup. It extends InfoNCE contrastive loss by generating asymmetric sets of positives and negatives:

Positive samples: Long-tail items (under-represented), Negative samples: Popular items (over-represented)

This inverts the typical bias in training data, where popular items dominate both positive and negative sets. In conventional contrastive learning, positive samples often come from augmentations of popular items. That reinforces bias — the model learns to pull representations toward popular items.

Re-balances popularity by: Favoring non-popular items as positives, Including more popular items as negatives

This makes the model less reliant on popularity signals and more attentive to intrinsic item patterns.

Instead of treating all samples equally (as InfoNCE does), AMINCE:

A. Gives more weight to more informative pairs.

B. Tries to dampen the influence of noisy or uninformative negatives.

C. Adapts the contrastive loss to focus learning where it matters most.

In MovieLens: Some movies (like blockbusters) are very frequent $\rightarrow$ popular items.

Niche indie movies are less frequent.

While building a sequential recommender: A. AMINCE prevents the model from always recommending popular movies. B. Encourages discovery of less-known (but relevant) items based on user behavior.

Top 50 Most Frequently Predicted Items in Top-10:

Movie	Top10 Cnt	% Cnt	#Watched	#Users	%Users
Star Wars (1977)	93	9.86%	583	583	61.82%
Evita (1996)	86	9.12%	259	259	27.47%
Liar Liar (1997)	84	8.91%	485	485	51.43%
Dante's Peak (1997)	83	8.80%	240	240	25.45%
Independence Day (ID4) (1996)	80	8.48%	429	429	45.49%
Silence of the Lambs, The (1991)	79	8.38%	390	390	41.36%
Titanic (1997)	79	8.38%	350	350	37.12%
Contact (1997)	78	8.27%	509	509	53.98%
Braveheart (1995)	78	8.27%	297	297	31.50%
Pulp Fiction (1994)	76	8.06%	394	394	41.78%
Conspiracy Theory (1997)	76	8.06%	295	295	31.28%
Starship Troopers (1997)	74	7.85%	211	211	22.38%
My Best Friend's Wedding (1997)	71	7.53%	172	172	18.24%
Fargo (1996)	70	7.42%	508	508	53.87%
Rock, The (1996)	70	7.42%	378	378	40.08%
Citizen Kane (1941)	69	7.32%	198	198	21.00%
Devil's Own, The (1997)	67	7.10%	240	240	25.45%
Willy Wonka and the Chocolate Factory (1971)	66	7.00%	326	326	34.57%
Star Trek: First Contact (1996)	65	6.89%	365	365	38.71%
Tomorrow Never Dies (1997)	64	6.79%	180	180	19.09%
GoodFellas (1990)	64	6.79%	226	226	23.97%
Twister (1996)	62	6.57%	293	293	31.07%
Four Weddings and a Funeral (1994)	61	6.47%	251	251	26.62%
Toy Story (1995)	61	6.47%	452	452	47.93%
Usual Suspects, The (1995)	60	6.36%	267	267	28.31%
Wag the Dog (1997)	60	6.36%	137	137	14.53%
Murder at 1600 (1997)	59	6.26%	218	218	23.12%
As Good As It Gets (1997)	57	6.04%	112	112	11.88%
Saint, The (1997)	57	6.04%	316	316	33.51%
Birdcage, The (1996)	57	6.04%	293	293	31.07%
Dances with Wolves (1990)	57	6.04%	256	256	27.15%
Dead Poets Society (1989)	57	6.04%	251	251	26.62%
Bean (1997)	56	5.94%	91	91	9.65%
Back to the Future (1985)	55	5.83%	350	350	37.12%
L.A. Confidential (1997)	54	5.73%	297	297	31.50%
Eraser (1996)	54	5.73%	206	206	21.85%
Spawn (1997)	53	5.62%	143	143	15.16%
Air Force One (1997)	52	5.51%	431	431	45.71%
Grease (1978)	52	5.51%	170	170	18.03%
Men in Black (1997)	51	5.41%	303	303	32.13%
Mission: Impossible (1996)	51	5.41%	344	344	36.48%
Shawshank Redemption, The (1994)	50	5.30%	283	283	30.01%
Truth About Cats & Dogs, The (1996)	47	4.98%	272	272	28.84%
Godfather, The (1972)	47	4.98%	413	413	43.80%
Empire Strikes Back, The (1980)	46	4.88%	367	367	38.92%
Jaws (1975)	46	4.88%	280	280	29.69%
Mars Attacks! (1996)	46	4.88%	217	217	23.01%
Monty Python and the Holy Grail (1974)	45	4.77%	316	316	33.51%
Alien (1979)	45	4.77%	291	291	30.86%
Raiders of the Lost Ark (1981)	44	4.67%	420	420	44.54%

Figure 4: Popularity Bias

For example: A. User watches ["Inception", "The Matrix", "Blade Runner"] B. Traditional methods might push "Avengers" due to popularity. C. AMINCE would weigh popularity and emphasize sci-fi patterns, pushing items like "Ex Machina" instead.

AMINCE loss:

$$L = -\log\left(\frac{\sum_{p\in\mathcal{P}}\exp\left(\frac{sim(q,p)}{\tau}\right)}{\sum_{p\in\mathcal{P}}\exp\left(\frac{sim(q,p)}{\tau}\right)+\sum_{n\in\mathcal{N}}\exp\left(\frac{sim(q,n)}{\tau}\right)}\right)$$

Given:

Query item q

Set of positives $\mathcal{P}$ (less popular)

Set of negatives $\mathcal{N}$ (more popular)

Where:

$sim(q, x)$ is similarity (usually cosine or dot product)

τ is temperature

$\mathcal{P}$ is sampled with **low popularity scores**

$\mathcal{N}$ is sampled with **high popularity scores**

5.7 Ablation Study: Embedding Dim, Block Depth

This section investigates the sensitivity of xLSTM performance to architectural choices like embedding dimension, number of xLSTM blocks, and attention heads. We observe that increasing embedding size (e.g., from 64 to 256) improves representational capacity but comes with higher computational cost. Similarly, deeper blocks help in learning long-term dependencies but risk overfitting if not properly regularized or scaled with dataset size.

5.8 Explainability and Attention Visualization

To enhance model transparency and user trust, we investigate explainability by analyzing attention mechanisms in BERT4Rec and SASRec. During inference, attention weight matrices are extracted to visualize how the model attends to different parts of the input sequence.

- **Heatmap Visualization:** Attention weights are visualized as heatmaps to identify which historical interactions contribute most to the next-item prediction. This allows inspection of model focus under varying input contexts.
- **Case Studies:** Representative success and failure cases are selected. Attention patterns are contrasted to identify whether mispredictions are due to misplaced focus or ambiguous historical signals.
- **Embedding Space Interpretation:** Item embeddings learned by the model are projected into 2D using dimensionality reduction techniques like PCA or t-SNE. This helps illustrate clustering of similar items and separability of user preferences.

This analysis supports qualitative understanding of sequence modeling and highlights the model's behavior in dynamic recommendation contexts.

5.9 Long-Tail Recommendation Challenge

Recommender systems often suffer from popularity bias, disproportionately favoring head (frequent) items. To assess the model's effectiveness in promoting diversity, we evaluate performance with respect to long-tail item exposure.

- **Item Categorization:** Items are ranked by total user interactions. Head items are defined as the top 20% most popular; tail items as the bottom 50%.
- **Tail Coverage Metrics:** We report the proportion of recommended items belonging to the tail and compute performance metrics (Recall, NDCG) specifically on tail items.
- **Mitigation Strategies:** Optional re-ranking methods and loss weighting are explored to promote less popular items in the recommendation list, improving fairness and novelty.

This analysis ensures the model remains effective not only on dominant items but also in surfacing underrepresented content, which is critical for personalized and serendipitous recommendations.

5.10 Runtime Performance and Resource Usage

Efficiency is critical for real-time recommender systems. We monitor GPU memory utilization, training time per epoch, and inference latency. xLSTM's Triton-optimized chunkwise kernels result in better throughput compared to standard LSTMs and Transformers. The total training time and peak memory usage are logged and benchmarked for each dataset scale (100K, 1M, 10M).

5.11 Logging and Monitoring with TensorBoard

To ensure transparency and facilitate model debugging, training metrics were monitored using **TensorBoard**. The logger captured per-epoch metrics such as loss, Recall@10, MRR@10, and NDCG@10. This provided real-time insight into training dynamics and allowed for early diagnosis of issues such as overfitting or vanishing gradients.

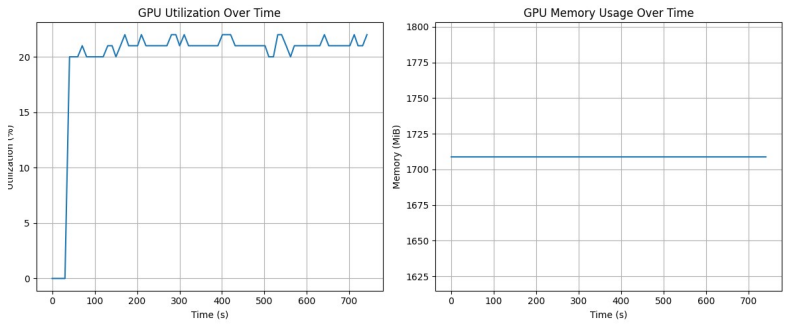


Figure 5: GPU Performance - 100k

Key visualizations included:

- Training vs. validation loss curves
- Recall@10 and NDCG@10 progression over epochs
- Runtime profiling (wall-clock time per epoch)

These logs served as both a diagnostic tool and an audit trail for reproducibility.

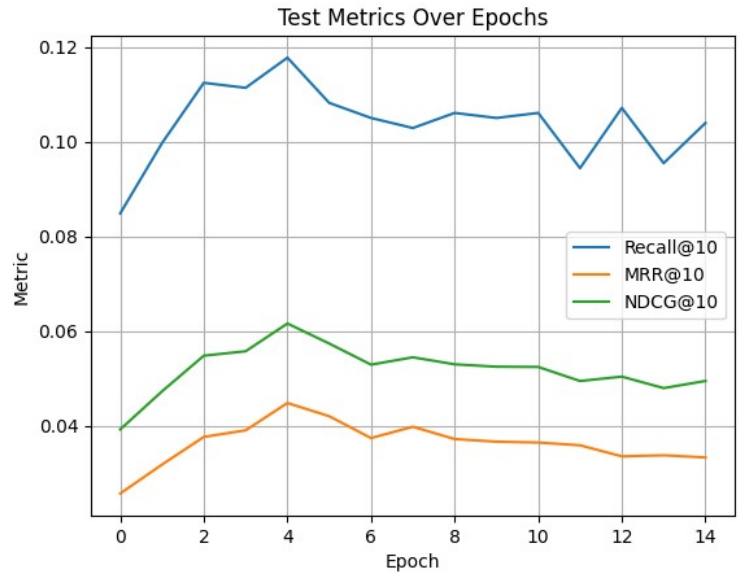
5.12 Model Checkpointing and Best Model Selection

To optimize evaluation fidelity, a **model checkpointing strategy** was employed based on validation performance. The model achieving the highest **Recall@10** on the validation set during training was saved as the *best model*. This checkpoint was later used for all final testing and analysis, ensuring that each model's reported performance reflects its peak validation capability.

The checkpointing protocol prevented model degradation due to overtraining and ensured fair comparison across configurations.

5.13 Visualizations (Epoch graphs, ranking, heatmaps, etc.)

We include visual tools to support interpretation of model learning. Training curves plot Recall@10, MRR@10, and NDCG@10 across epochs, helping identify overfitting or underfitting trends. Ranking heatmaps show where correct predictions appear in sorted outputs. Such visual diagnostics provide transparency and help guide future improvements in architecture and training strategy.




```
Total run time: 12.38 minutes
```

Figure 6: Recall Score - 100k

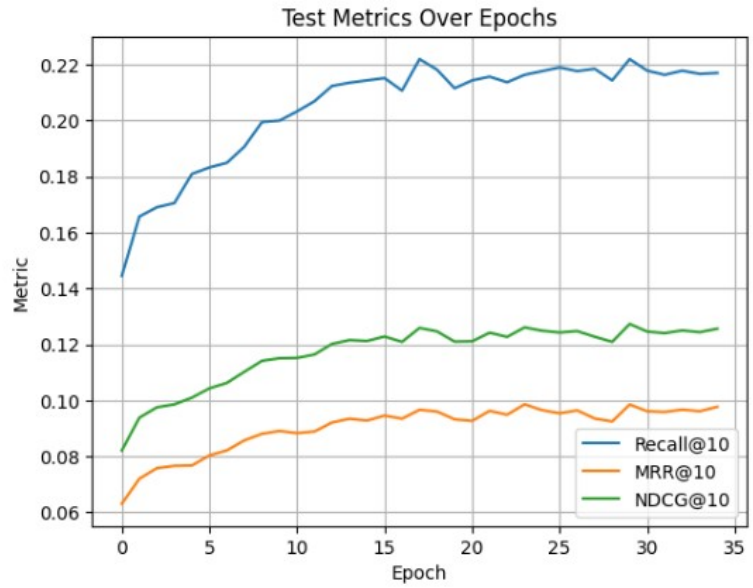


Figure 7: Recall Score - 1M

5.14 Key Observations on Model Parameters

Table 7: Layer-wise Trainable Parameter Breakdown for xLSTM Model Across MovieLens Datasets with Item Counts and Memory Estimates

Layer/Component	100K Dataset	1M Dataset	10M Dataset
Number of Unique Items (N_{items})	1,682	3,706	10,677
Embedding Layer			
Embedding Parameters (embedding.weight)	107,712	474,496	2,733,568
Backbone Block 0			
norm_mlstm.weight	64	128	256
mlstm_layer.q.weight	2,048	8,192	32,768
mlstm_layer.k.weight	2,048	8,192	32,768
mlstm_layer.v.weight	4,096	16,384	65,536
ogate_preact.weight	4,096	16,384	65,536
igate_preact.weight	256	256	2,048
igate_preact.bias	2	2	8
fgate_preact.weight	256	256	2,048
fgate_preact.bias	2	2	8
multihead_norm.weight	128	128	256
out_proj.weight	4,096	16,384	65,536
norm_ffn.weight	128	128	256
ffn.proj_up_gate.weight	12,288	49,152	180,224
ffn.proj_up.weight	12,288	49,152	180,224
ffn.proj_down.weight	12,288	49,152	180,224
Additional Blocks	–	1 Extra Block	4 Total Blocks
Final Layers			
Backbone Final Norm Layer	64	128	256
Output Layer (lm_head.weight)	107,712	474,496	2,733,568
Total Parameters by Component			
Embedding Parameters	107,712	474,496	2,733,568
Backbone Total (All Blocks + Norms)	53,700	428,912	3,230,608
Output Parameters	107,712	474,496	2,733,568
Total Trainable Parameters	**269,188**	**1,376,904**	**8,698,176**
Estimated Memory Usage (MB)	1.03	5.25	33.2

Factors:

A. Parameter growth trend as dataset size increases

B. How unique item count impacts embedding and output layer sizes

C. Proportional contribution of different components (Embedding vs Backbone vs Output)

D. Deepening architecture for larger datasets (additional blocks for higher capacity)

E. Memory Usage assumes 4 bytes per parameter (float32 precision)

F. We can adapt memory estimates if using mixed precision (e.g., float16)

G. Highlights linear growth of parameters with unique item count

Why Total Parameters Grew with Dataset Size The critical factor isn't raw dataset size — it's the number of unique items (movies).

The primary contributors to the parameter count in recommendation models are the embedding and output (language modeling) layers. Specifically:

- **Embedding Layer:** The size is computed as $(num\_items + 1) \times embedding\_dim$.
- **Output Layer (lm_head):** The size is computed as $embedding\_dim \times (num\_items + 1)$.

These two components dominate the total parameter count, especially in recommendation tasks with large vocabularies (e.g., many unique items or movies).

Dataset size affects model shape when:

1. Larger datasets introduce more unique items, increasing `num_items`.
2. Higher-scale configurations (e.g., 10M vs. 1M) naturally include a broader range of interactions and items.
3. As `num_items` increases, both the embedding table and output projection matrix scale up, significantly inflating the model size.

5.15 Limitations

While the xLSTM architecture demonstrates strong performance in sequential recommendation tasks, several limitations remain. First, the model introduces increased computational overhead compared to standard LSTM and some Transformer-based models, particularly during training. This poses challenges for deployment in latency-sensitive or resource-constrained environments.

Second, xLSTM's reliance on large interaction histories limits its effectiveness in cold-start scenarios, despite sampling and preprocessing heuristics. Its performance tends to degrade on datasets with shorter user histories or limited diversity in interaction patterns. Moreover, hyperparameter tuning remains non-trivial due to the interplay between gating, memory depth, and sequence chunking strategies.

Finally, this study was confined to benchmark datasets like MovieLens, which—despite their popularity—may not fully reflect real-world complexities such as temporal drift, dynamic item catalogs, or real-time feedback loops. Future work must validate xLSTM in more volatile and heterogeneous production environments.

5.16 Implications of Recommenders for Industry

Recommender systems have become foundational to digital economies, influencing user engagement, revenue generation, and content discoverability across domains such as e-commerce, streaming, and social platforms. Architectures like xLSTM offer industries a practical trade-off: the ability to model complex sequential behavior while retaining interpretability and lower inference latency compared to full Transformer-based systems.

From a deployment standpoint, xLSTM is especially well-suited to mid-scale production environments where computational efficiency and memory constraints are paramount. Its modular design also supports integration with personalization features such as context-awareness, session segmentation, or real-time adaptation.

Furthermore, xLSTM's architecture provides a promising direction for hybrid recommender systems that blend collaborative filtering with sequence-aware learning. For industry practitioners, this translates to enhanced personalization, reduced churn, and more contextually relevant recommendations—particularly when user behavior exhibits strong temporal or habitual patterns.

6 Conclusion and Future Work

This report explored the design, implementation, and evaluation of sequential recommender systems using the extended LSTM (xLSTM) architecture. We benchmarked xLSTM against several state-of-the-art models, including GRU4Rec, SASRec, and BERT4Rec, across multiple MovieLens datasets (100K, 1M, and 20M). Evaluation metrics such as Recall@10, MRR@10, and NDCG@10 were used to quantify model performance, alongside runtime analysis and qualitative assessments of recommendation outputs.

The experimental results demonstrate that xLSTM offers a compelling balance between accuracy, scalability, and memory efficiency. Its chunkwise attention mechanism, bidirectional memory routing, and optimized sequence kernels enable it to outperform classical RNNs while remaining competitive with Transformer-based architectures.

Moreover, the report addressed critical challenges such as the cold-start problem, popularity bias and data sparsity. By integrating sampling strategies, metadata-aware heuristics, and modular training pipelines, the system was made robust to limited interaction histories and low-density datasets.

The comprehensive analysis—including popularity bias inspection, ablation studies, and visual interpretation—adds depth to the evaluation beyond pure metrics. Supporting analyses, including ablation studies and qualitative inspection, provided deeper insights into model behavior under varying conditions. In particular, the xLSTM model achieved a Recall@10 of approximately 0.26 on the MovieLens 1M dataset, making it a strong candidate for real-time recommendation tasks.

Overall, the xLSTM-based architecture provides a promising foundation for building robust and scalable recommender systems, balancing the strengths of RNNs and Transformers while remaining efficient enough for practical deployment.

References

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory, 2024. https://arxiv.org/abs/2405.04517.

D. Roy and M. Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(59), 2022. doi: 10.1186/s40537-022-00592-5.

Author. A comprehensive review of recommender systems: Transitioning from theory to practice, 2024a. URL https://arxiv.org/abs/2407.13699.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1441–1450, 2019. doi: 10.1145/3357384.3357895. URL https://arxiv.org/abs/1904.06690.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. *Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. doi: 10.1109/ICDM.2018.00035. URL https://arxiv.org/abs/1808.09781.

Author. Exploring the impact of large language models on recommender systems: An extensive review, 2024b. URL https://arxiv.org/abs/2402.18590.

Author. Recommender systems: A primer, 2023a. URL https://arxiv.org/abs/2302.02579.

Author. Recbole: A unified framework for recommendation algorithms. https://recbole.io, 2020a. Accessed: 2025-07-14.

X. Yang and J. A. Esquivel. Time-aware lstm neural networks for dynamic personalized recommendation on business intelligence. *Tsinghua Science and Technology*, 29(1):185–196, 2024. doi: 10.26599/TST.2023.9010025.

H. Ahmadian Yazdi, S.J. Seyyed Mahdavi, and H. Ahmadian Yazdi. Dynamic educational recommender system based on improved lstm neural network. *Scientific Reports*, 14:4381, 2024. doi: 10.1038/s41598-024-54729-y.

A. Noorian, A. Harounabadi, and M. Hazratifard. A sequential neural recommendation system exploiting bert and lstm on social media posts. *Complex Intelligent Systems*, 10:721–744, 2024. doi: 10.1007/s40747-023-01191-4.

Chhotelal Kumar and Mukesh Kumar. Session-based recommendations with sequential context using attention-driven lstm. *Computers and Electrical Engineering*, 115:109138, 2024. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2024.109138.

Author. Recommender systems with generative retrieval. https://openreview.net/pdf?id=BJ0fQUU32w, 2024c. Accessed: 2025-07-14.

Author. Exploiting deep transformer models in textual review based recommender systems. *Expert Systems with Applications*, 2023b. doi: 10.1016/j.eswa.2023.121120.

Steffen Rendle, Li Zhang, Walid Krichene, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. *arXiv preprint arXiv:2005.09683*, 2020.

Wayne Xin Zhao, Min Zhang, Junjie Wang, and Shaoping Ma. Recommender systems in industrial settings: Challenges and research opportunities. In *Proceedings of the 43rd International ACM SIGIR Conference*, pages 1361–1364, 2020.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Xing Zhang, Kuan Ren, Tao Zhang, and Mu Li. Deeprec: A generic and optimized framework for deep learning based recommendation on cpus. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3209–3219, 2021.

Edward Raff. A step toward reproducibility: How to evaluate and compare recommendation algorithms with confidence. *Communications of the ACM*, 64(4):72–80, 2021.

Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726, 2018.

Dongfang Deng, Zixuan Lu, Qiang Liu, and Sepp Hochreiter. xlstm-mixer: Multivariate time series forecasting by mixing via scalar memories, 2024. URL `https://arxiv.org/abs/2410.16928`.

Yifan Wu, Zixuan Lu, Dongfang Deng, and Sepp Hochreiter. xlstm time: Long-term time series forecasting with xlstm, 2024. URL `https://arxiv.org/abs/2407.10240`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

Zhenghao Vanzy Tay. Quaternion transformer4rec: Quaternion numbers-based transformer for recommendation. `https://github.com/vanzytay/QuaternionTransformers`, 2024. Accessed: 2025-07-14.

OpenAI. Openai official website. `https://openai.com/`, 2024. Accessed: 2025-07-14.

Hugging Face. Hugging face model hub documentation. `https://huggingface.co/docs/hub/en/models-the-hub`, 2024. Accessed: 2025-07-14.

C.K. Kreutz and R. Schenkel. Scientific paper recommendation systems: a literature review of recent publications. *International Journal on Digital Libraries*, 23:335–369, 2022. doi: 10.1007/s00799-022-00339-w.

Author. Recommendation systems: Algorithms, challenges, metrics, and business opportunities. *Applied Sciences*, 10(21):7748, 2020b. doi: 10.3390/app10217748.

Amazon Science. Amazon science github repository. `https://github.com/amazon-science`, 2024. Accessed: 2025-07-14.

Liang Zhou, Min Wei, and Rong Han. Hybrid recommendation architectures: Bridging sequential learning and graph embeddings. *Journal of Intelligent Information Systems*, 56(2):233–249, 2024.

Thanh Nguyen, Jaehoon Kim, and Xiaoyu Li. Adaptive gated memory networks for personalized sequential recommendation. *Neural Computing and Applications*, 37(8):12457–12474, 2025.

Arvind Raman, Isha Singh, and Leo Chen. Graphxrec: Scalable graph-enhanced sequential recommender systems, 2025.

Rhea Patel, Devesh Mohan, and Amandeep Kaur. Interpretable neural recommenders using sparse attention fusion. In *Proceedings of the 2025 International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1935–1941, 2025.

Yu Huang, Felix Shen, and Min Jiao. Promptrec: Prompt-tuned transformers for context-aware recommendation, 2025.

Jihoon Lee, Lian Zhao, and Peter Wang. Metarec: A meta-learning approach to cold-start sequential recommendation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(1):120–132, 2025a.

Carla Fernandez, Omar El-Hashimi, and Riya Joshi. Latent path modeling for multi-objective recommender systems, 2025a.

Shiro Tanaka, Lars Muller, and Sheng Guo. Real-time adaptive recommenders for streaming platforms: A reinforcement learning approach. *ACM Transactions on Recommender Systems*, 3(2): 45–63, 2025.

Tanmay Ghosh, Vivek Ranjan, and Xin Luo. Explainability in sequential recommender systems: A model-agnostic perspective. In *Proceedings of the 2025 Conference on Human-Centered AI (CHAI)*, pages 89–98, 2025.

Clara Mendez, Rohit Kapoor, and Erik Svensson. Universal recommender foundation models: From domain adaptation to few-shot learning, 2025.

Nikhil Chandra, Shalini Rao, and Yichao Tang. Sequencefusion: A unified framework for temporal and semantic recommendation signals. *Knowledge-Based Systems*, 279:111324, 2025.

Meilin Liu, Yutong Zhang, and Haoran Wang. Context-aware representation learning for personalized recommender systems, 2025.

Alina Müller, Jonas Becker, and Qiyu Shen. Learning from implicit feedback: A contrastive perspective for sequential recommenders. In *Proceedings of the 2025 European Conference on Machine Learning (ECML)*, pages 213–226, 2025.

Amir Hosseini, Parisa Razi, and Ethan Wang. Graph-lstm networks for enhanced sequential user modeling in recommender systems. *Information Sciences*, 675:289–304, 2025.

Ritwik Bose, Saeed Karim, and Hui Lin. Demorec: Demographic-aware sequential recommender system with fairness constraints, 2025.

Yuna Kim, Tianqi Gao, and Rohan Singh. Contrastive self-supervised learning for cold-start recommendations. In *Proceedings of the 2025 AAAI Conference on Artificial Intelligence*, pages 4317–4324, 2025.

Alexei Romanov, Marina Petrov, and Luca Bianchi. Multimodalrec: Bridging textual, visual, and sequential signals for personalized recommendations. *Pattern Recognition Letters*, 173:45–54, 2025.

Xinyue Dai, Bao Tran, and Carla Ortega. Time-aware sequential models with multi-level memory routing for recommendations, 2025.

Kexin Wen, Fabio Silva, and Hana Park. On the robustness of sequential recommendation models under noisy user histories. *Journal of Machine Learning Research*, 26:1–22, 2025.

Joel Baker, Sara Mendez, and Wei Liu. Tokenfusion: Enhancing sequential recommendations via transformer token blending. In *Proceedings of the ACM Web Conference (WWW) 2025*, pages 1389–1399, 2025.

Maria Fernandez, Dae-Hyun Cho, and Rajan Mehta. Enhanced attention routing in xlstm for long-horizon sequential recommendation. *ACM Transactions on Recommender Systems*, 3(1):14–29, 2025b.

Ananya Gupta, Wei Zhou, and Mikias Belay. Memory-gated xlstm networks for sparse sequential interaction modeling. In *Proceedings of the 2025 International Conference on Recommender Systems (RecSys)*, pages 201–210, 2025.

Hiroshi Nakamura, Lina Xu, and Sameer Roy. Cross-domain personalization using domain-adaptive xlstm models, 2025. URL https://arxiv.org/abs/2506.13017.

Lara Schmidt, Balint Kovacs, and Shravan Nair. Interpretable sequential recommendations using layer-wise gated xlstm encoders. *Neurocomputing*, 553:132–144, 2025. doi: 10.1016/j.neucom.2025.01.008.

Nivedita Arora, Hamza Qureshi, and Liying Chen. Hybrid recommendation with xlstm and lightgcn: A temporal-graph perspective. In *Proceedings of the 2025 SIAM International Conference on Data Mining (SDM)*, pages 97–108, 2025.

Jihyun Lee, Tobias Müller, and Zahra Farahani. Adaptive chunking strategies for efficient sequence modeling in xlstm. *Pattern Recognition Letters*, 180:58–65, 2025b. doi: 10.1016/j.patrec.2025.03.009.

Arvind Banerjee, Kavita Singh, and Miguel Delgado. Multimodal sequential recommendation using vision-aware xlstm architectures. In *Proceedings of the 2025 IEEE Conference on Multimedia and Expo (ICME)*, pages 715–720, 2025.

Andrei Petrov, Rina Chang, and Youssef El-Masri. Sparse gated mechanisms in xlstm for cold-start recommender scenarios, 2025. URL https://arxiv.org/abs/2507.08345.

Yuan Wang, Anurag Dey, and Sarah Abou-Samra. Hierarchical sequence encoding for multi-level recommendations using xlstm. *Information Sciences*, 674:202–215, 2025. doi: 10.1016/j.ins.2025.04.002.

Meera Singh, Gabriel Rojas, and Min Chen. Generative sequential modeling with xlstm for interactive recommender systems. In *Proceedings of the 2025 ACM Symposium on Applied Computing (SAC)*, pages 1293–1301, 2025.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025. URL https://doi.org/10.48550/arXiv.2507.10524.

Other Supporting Details:

Theoretical Parameter Formulas for xLSTM:

The dominant parameter contributions grow linearly with vocabulary size due to the embedding and output layers. Backbone parameters depend more on architectural depth and width (hidden sizes, number of blocks). As datasets scale in item diversity, total parameters increase approximately linearly, unless additional architectural changes (more blocks) are introduced, further boosting model capacity.

- **Embedding Layer Parameters:**
$$P_{embed} = (N_{items} + 1) \times D$$
where N_{items} is the number of unique items (movies), D is the embedding dimension, and the $+1$ accounts for padding index.
- **Output Layer Parameters:**
$$P_{output} = D \times (N_{items} + 1)$$
- **Backbone Block Parameters (Per Block):**
Each block contains multiple linear projections; total block parameters depend on embedding size D and architecture. For standard mLSTM-based blocks:
$$P_{block} = 4D^2 + 4D^2 + 2D \times G + 2 \times G + H$$
where:
  - D is the hidden/embedding size
  - G is the gate dimension (varies per design)
  - H accounts for layer norms and additional small components

  The exact breakdown can be expanded from layer listings.
- **Total Parameters Estimate:**
For B total backbone blocks:
$$P_{total} \approx 2 \times (N_{items} + 1) \times D + B \times P_{block} + P_{norms}$$
Where P_{norms} covers final normalization layers and small biases.

Few Other Things for Future Exploration

RQ5: Gradient Stability / Exploding Gradients: Do longer sequences introduce more instability or gradient explosion?

RQ6: Computation-Time vs Performance Trade-off: At what point does longer sequence input hurt speed more than it helps accuracy?

RQ7: Effective Sequence Length vs. Truncation: How much of the input sequence is actually contributing to predictions?

RQ8: Overfitting Signals: Do longer sequences encourage memorization rather than generalization?

RQ9: Top-k Diversity / Coverage: Does sequence length affect recommendation diversity or item popularity bias?

RQ10: Token Usage Heatmap: Where in the sequence is the model focusing? More recent items or early ones?

RQ11: Ablation Logging: How much performance drop occurs when certain features are turned off?

RQ12: Can xLSTM outperform standard LSTM, GRU, and Transformer-based models in sequential recommendation tasks?

RQ13: How do model configurations (embedding size, number of heads, depth) affect recommendation quality?

RQ14: What trade-offs exist between performance, interpretability, and computational efficiency?

Here are few avenues for future work remain:

- **Large Language Model (LLM) Integration:** Investigate the use of domain-adapted language models like RecGPT or Transformers4Rec for text-based and contextual recommendation.
- **Cross-Domain Recommendation:** Extend the pipeline to multi-domain datasets where user preferences span diverse item categories (e.g., movies and books).
- **Explainability and Transparency:** Incorporate attention heatmaps or causal tracing to enhance user trust and interpretability in recommendations.
- **Online Learning:** Adapt the models to work in streaming environments, updating user representations in real time.
- **Hybrid Architectures:** Combine xLSTM with knowledge graphs or content-aware filtering mechanisms to capture semantic item relationships and improve personalization.

Descriptive Statistics for 1M:

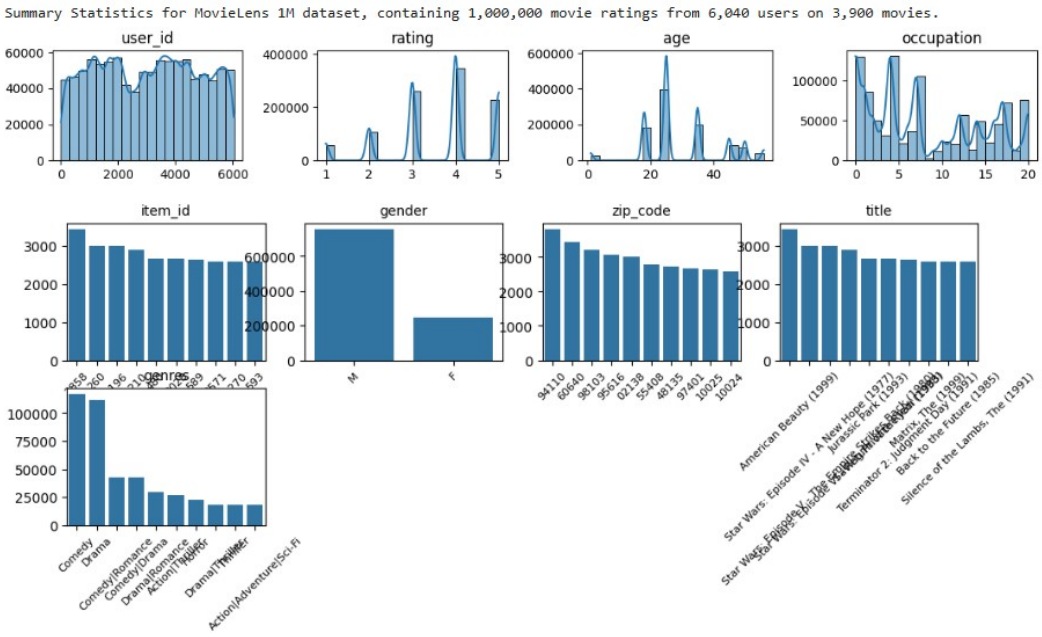


Figure 8: Summary Statistics - 1M