Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

An Evaluation of Self-Attentive Sequential Recommendation (SASRec) Algorithm Using Hyperparameter Tuning

Agung Toto Wibowo*[1], Hasmawati[2], Hani Nurrahmi[3], Imtitsal Ulya Salsabila[4]

[1,2,3,4]School of Computing, Universitas Telkom, Indonesia

Email: [1]agungtoto@telkomuniversity.ac.id

Received : Jul 22, 2025; Revised : Oct 14, 2025; Accepted : March 19, 2026; Published : Apr 18, 2026

Abstract

Sequential recommendation is a branch of Recommender Systems that aims to predict the next item a user will interact with based on their historical sequence of interactions. The main challenge in SR is to capture both short-term and long-term dependencies among items within a sequence. Self-Attentive Sequential Recommendation (SASRec) is a self-attention-based deep learning model designed to recognize sequential interaction patterns. Despite its effectiveness, the performance of SASRec is highly dependent on hyperparameter configurations, yet comprehensive evaluations remain limited. This research aims to evaluate the influence of SASRec's configuration through hyperparameter tuning on sequential recommendation performance. The hyperparameters used are *hidden_size*, *inner_size*, number of attention heads (*num_heads*), and number of layers (*num_layers*). The evaluation was conducted on two public datasets with different sparsity characteristics: MovieLens-1M (Sparsity ≈ 95.80%) and Amazon Musical Instruments (Sparsity ≈ 99.99%). In this study, Recall@k and MRR@k were used as performance metrics. The test results showed that *hidden_size* and *inner_size* had a significant positive impact on performance, especially on the dense dataset. The optimal *hidden_size* was obtained at *hidden_size* = 64 on the Amazon Musical Instrument dataset, and at *hidden_size* = 256 on the Movielens 1M dataset. The optimal *inner_size* was obtained at *inner_size* = 256 on both datasets. Meanwhile, the *num_heads* and *num_layers* hyperparameters did not provide a significant performance improvement. Furthermore, in the comparison between SASRec, GRU4Rec, and BERT4Rec, SASRec outperforms GRU4Rec and BERT4Rec in handling highly sparse datasets such as Amazon Musical Instruments obtained average recall@20 = 0.0678, and average MRR@20 = 0.0223.

Keywords: *Hyperparameter Tuning, Recommender System, SASRec, Self-Attention, Sequential Recommendation.*

1. INTRODUCTION

Recommender Systems (RecSys) have been widely developed to provide product recommendations to users [1], [2]. Initially, RecSys were developed by utilizing the Collaborative Filtering (CF) approach to find patterns from user interactions using memory-based algorithms (such as KNN), Matrix Factorization (such as SVD, SVD++, NMF), and model-based CF (such as GNN, DeepFM, and LightGCN) [2], [3], [4]. The use of model-based CF with neural networks generally outperforms memory-based approaches due to its ability to capture non-linear relationships [2]. Although popular, CF faces several problems, including its inability to capture context and its limitations in handling temporary changes in user preferences (temporal drift) [5]. The problem of temporal drift is addressed through Sequential Recommendation (SR) [6], [7].

GRU4Rec is an SR algorithm based on Recurrent Neural Networks (RNN) [1], [8]. GRU4Rec utilizes the Gated Recurrent Unit (GRU) to learn sequential interactions from users, since the training process is performed on sequential data [8], [9], [10]. In its process, GRU4Rec is only able to capture short-term dependencies from sequential interactions [8], [11]. GRU4Rec encounters difficulties in modeling long-term dependencies [8]. In its development, self-attention-based models such as SASRec

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

emerged [8], [10]. SASRec is the first model to leverage self-attention from the Transformer for SR [10]. SASRec transforms user interactions in a sequential context into item embedding representations enriched with positional encoding [10]. SASRec also employs multi-head self-attention to capture both short-term and long-term dependencies, thereby addressing the temporal drift problem [8], [10], [12]. Subsequent SR developments led to the use of bidirectional self-attention implemented in BERT4Rec [3], [8]. BERT4Rec also combines the masked item prediction technique, where several items are masked, and the model is trained to predict the masked items [6], [8], [13].

In this study, we focus on evaluating the SASRec algorithm, considering that SASRec is efficient in modeling SR [10]. SASRec has a simpler architecture compared to other models, with the capability to model long-term dependencies effectively [5], [8]. SASRec only uses user–item interaction logs in modeling SR [8], [14]. In many studies, SASRec is often used as a benchmark algorithm [15]. However, no research has yet conducted a comprehensive evaluation of the influence of SASRec's hyperparameters (such as number of layers, hidden size, and number of attention heads) in modeling SR, especially on datasets with different levels of sparsity.

The contributions of this research can be described as follows: a. conducting systematic experiments on four hyperparameters of the SASRec algorithm (namely hidden size, inner size, *num_heads*, and *num_layers*), b. comparing SASRec's performance on two datasets with different sparsity characteristics (MovieLens 1M sparsity ~95.80% and Amazon Musical Instruments sparsity 99.99%), c. providing hyperparameter analysis on SR performance, which can serve as a reference for future research. In addition to these contributions, we also compare SASRec's performance with two other algorithms, namely GRU4Rec and BERT4Rec. This comparison is necessary to identify the advantages of SASRec in modeling SR.

Based on this background and motivation, this study aims to evaluate the performance of SASRec in SR scenarios. The focus is on examining the impact of hyperparameter configurations—such as hidden size, inner size, number of attention heads (*num_heads*), and number of layers (*num_layers*)—on SR performance. The evaluation is conducted using the Recall@K and Mean Reciprocal Rank (MRR) metrics. These performance metrics are chosen because they account for the order of user-item interactions, making them highly representative of SR effectiveness [1], [10]. Accuracy-based metrics (such as MAE and RMSE) are not suitable, as they do not consider the sequential nature of interactions.

2. RELATED WORKS

2.1. Recommender Systems

Recommender Systems (RecSys) are software applications designed to select products or items that match user preferences [2], [16]. The recommended products are chosen from thousands or even millions of candidates in a catalog [16], [17]. For example, when a user visits Netflix, the system selects several videos that are highly likely to be consumed by that user [1], [5]. The use of RecSys is highly beneficial for improving user retention and loyalty [5].

Two of the most widely used approaches in RecSys are Collaborative Filtering (CF) [2], [4], [18] and Content-Based Filtering (CBF) [2], [4], [16], [19]. CF leverages the history of user interactions with products/items to identify patterns of similarity among users or items [2], [4], [16]. Recent studies have also applied Graph Neural Networks (GNNs) to learn more expressive latent representations of user and items. Other works have proposed Transformer-based architectures to enhance CF modeling [5], [20], as well as the integration of graph attention networks [1] or group attention mechanisms [3] for exploiting contextual attributes.

In User-Based CF, recommendations are generated by predicting user preferences based on their similarity with other users, typically measured using Cosine Similarity or Pearson Correlation [2]. Item-Based CF, on the other hand, finds similarity between items based on co-consumption patterns [2].

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

Matrix Factorization approaches embed users and items into latent dimensions, and user preferences are reconstructed by the interaction of these latent features [2], [9]. Well-known MF variants include Singular Value Decomposition (SVD), Alternating Least Squares (ALS), Stochastic Gradient Descent (SGD), Non-negative Matrix Factorization (NMF), and SVD++ [2], [4]. More recent works have shown that neural CF models (e.g., Neural CF) outperform traditional memory-based CF because of their ability to capture complex non-linear relationships [2].

The CBF method, in contrast, relies on item features and recommends new items similar to those previously consumed by the user [2], [16]. While both CF and CBF have been effective, they share a limitation: they do not consider the temporal sequence of interactions [5], [17]. Both approaches assume static user preferences, which is often unrealistic [3], [5], [7]. This limitation has led to the development of Sequential Recommendation (SR), which explicitly models user behavior by taking into account the order of interactions [7], [17].

2.2. Sequential Recommendation

Sequential Recommendation (SR) is a technique in RecSys that predicts the product or item a user is most likely to consume next [10], [17], [21]. Unlike other approaches, SR explicitly considers the temporal order of user-item interactions [17], [21]. To address SR tasks, researchers have proposed several models, including Markov Chains, Factorizing Personalized Markov Chains (FPMC), and Personalized Ranking Metric Embedding (PRME) [22], [23]. A Markov Chain is a probabilistic model that learns transition patterns between items under the assumption that the next interaction depends only on one or a few previous items. FPMC combines Matrix Factorization with Markov Chains to capture personalized item transitions and has been shown to be effective in next-basket recommendation tasks. PRME, on the other hand, maps users and items into latent vectors through an embedding process and optimizes them using a ranking metric that accounts for the order of user interactions [17].

Beyond these models, Recurrent Neural Network (RNN)-based approaches have also been widely adopted. Notable examples include GRU4Rec [9], [10], [11] and the Short-Term Attention/Memory Priority Model (STAMP) [17], [24]. GRU4Rec leverages Gated Recurrent Units (GRU) to represent historical sequences and was one of the first models designed for session-based recommendations, where predictions are based on the order of items consumed within a session [9], [10], [11]. STAMP combines short-term attention with explicit memory to better capture user intent [17], [24]. However, RNN-based models face limitations in modeling long-range dependencies [17], [22], which has motivated the adoption of self-attention mechanisms.

With the advent of Transformer architecture, recommendation systems began to leverage self-attention for SR. Transformers have proven highly effective in capturing long-range dependencies in sequential prediction. SASRec (Self-Attentive Sequential Recommendation) was the first Transformer-based model to apply self-attention in modeling user interaction sequences. Its strength lies in efficiently capturing both short-term and long-term patterns [6]. Building on this, BERT4Rec introduced bidirectional self-attention and masked item training to enhance sequential modeling[6]. TiSASRec incorporated both positional information and temporal intervals [10], while SSE-PT extended SASRec by integrating user embeddings into item embeddings for personalization, along with Stochastic Shared Embeddings (SSE) to avoid overfitting. More recently, Coase employed a SudokuFormer architecture to capture global session context by combining item encoding, positional encoding, and time interval encoding.

SR approaches can be categorized into [17]: a. Pure ID-Based SR, which represents items only by unique IDs without side information. This approach is computationally efficient and lightweight. Algorithms in this category include Markov Chains, FPMC [22], [23], PRME, GRU4Rec [9], [10], [11], STAMP [17], [24], SASRec [10], [11], BERT4Rec [3], [11], and SSE-PT [25]. b. SR with Side

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

Information [17], which leverages additional data such as category, price, ratings, reviews, temporal context, knowledge graphs, or social networks. Algorithms in this category include TiSASRec [25] and Coase.

2.3. Self-Attentive Sequential Recommendation

SASRec (Self-Attentive Sequential Recommendation) is a sequential prediction model that was first introduced by Kang & McAuley in 2018 [26]. SASRec is known to be the first deep-learning model to utilize the Self-attention mechanism to solve sequential recommendation [7], [10]. Self-attention is a mechanism that allows each element in a sequence to interact with other elements [1], [5], [11]. Self-attention also learns a representation based on the relative contribution of all elements involved in the sequence [1], [17]. In the context of sequential recommendation, the items that appear in a transaction sequence represent the influence of a customer's/user's preferences [10], [27]. Thus, by learning self-attention, it is hoped that the sequential consumption pattern of a user can be learned [10], [11]. Broadly, the SASRec architecture generally consists of four main parts: a. Input Embedding [5], [18], b. Self-Attention Encoder [5], [10], c. Feedforward and Normalization Layer [1], [5], and d. Output/Prediction Layer [5].

The Input Embedding part represents the user's interaction history as a numerical vector representation (embedding). This representation is then augmented with positional information to represent the sequential interaction pattern, ensuring the pattern is recognized by the model. Suppose a user has a sequential item interaction as follows: $s = [i_1, i_2, \ldots, i_t]$ where i_j denotes the -j-th item in the sequential interaction, and t denotes the maximum historical sequence length of the model. Each item i_j is represented by a d-dimensional embedding vector so that: $e_{ij} \in \mathbb{R}^d$. SASRec then stores an embedding matrix for all items in the training dataset as: $E \in \mathbb{R}^{Nxd}$, where N is the total number of items. The embedding vector for item i_j can be retrieved as: $e_{ij} = E[i_j]$. This value is then added with positional information representing a sequence. The positional information for the -j-th position in a sequence is represented by: $p_j \in \mathbb{R}^d$. The model stores positional encoding information in: $P \in \mathbb{R}^{txd}$. The final process of the Input Embedding of an item is the sum of the item embedding and the positional encoding, denoted as: $x_j = e_{ij} + p_j$. Thus, the entire input sequence (items in a sequence) can be represented as the matrix: $X = [x_1; x_2; \ldots; x_t] \in \mathbb{R}^{txd}$.

The Self-Attention Encoder part implemented a multi-head self-attention mechanism to learn the relationships between items in a sequential interaction. The use of Self-Attention was expected to capture user preferences based on the order of item consumption, both for short-term and long-term interactions. Mathematically, self-attention worked by multiplying the items in the sequential transaction through the transformations: Query (Q), Key (K), and Value (V). The attention value itself was calculated using the Equation (1):

$$Attention(Q, K, V) = soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right).V \quad (1)$$

where: Q, K, and V are the results of transforming the input items into vector form, each representing the query, key, and value. Meanwhile d_k indicates the size or length of the key vector and is used to normalize the result to prevent it from becoming too large. The multiplication between Q and K^T produces a score that indicates how much attention one item pays to another. The softmax function is used to convert these scores into probability values that serve as attention weight.

Feedforward dan Normalization: After self-attention, the result was passed to a feedforward layer and stabilized through a normalization process and residual connections to make training more stable. The Feedforward and Normalization part received the encoding from the Self-attention, processed the existing patterns, and then passed them to the Output Prediction part. The feedforward neural network

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

(FFN) itself consisted of two linear layers that utilized a non-linear activation function (usually ReLU). To make the training process more convergent, each sub-layer in the SASRec architecture (both self-attention and feedforward layer) was followed by a normalization layer and a residual connection. The normalization layer kept the activation distribution under control during training. Meanwhile, the residual connection helped avoid gradient degradation in the deep neural network.

The Output Prediction part is the final section that produced the prediction of the next item in the sequential recommendation. The prediction process was done by calculating the similarity (via dot product) between the final output vector and all item embedding vectors. The item with the highest score would be recommended as the candidate for the next interaction.

3. METHOD

This study aimed to evaluate the performance of SASRec in performing the Sequential Recommendation task. Since SASRec has several components with different parameter values, we tested several important hyperparameter values in SASRec (sub-chapter 0). The comparison was performed using appropriate performance metrics for Sequential recommendation, namely: a. Recall@k and b. MRR@k (sub-chapter 3.2). To measure robustness in this study, we utilized 2 (two) datasets: a. ML-1M, and b. Amazon Musical Instrument (sub-chapter 3.3). The experimental results were then analyzed and described in Chapter 4. The analysis was conducted by measuring the performance of Recall@k and MRR@k with k values of [5, 10, 15, 20, ..., 100]. The SASRec implementation used in this study was by utilizing the Recbole library[1]. Recbole is a python library that has been equipped with several Recommender System algorithms for Collaborative Filtering, Content-Based Filtering, and Sequential Recommendation methods, including the SASRec algorithm. The research methodology employed in this study is illustrated in Figure 1, which presents the Sequential Recommendation process using the SASRec algorithm.

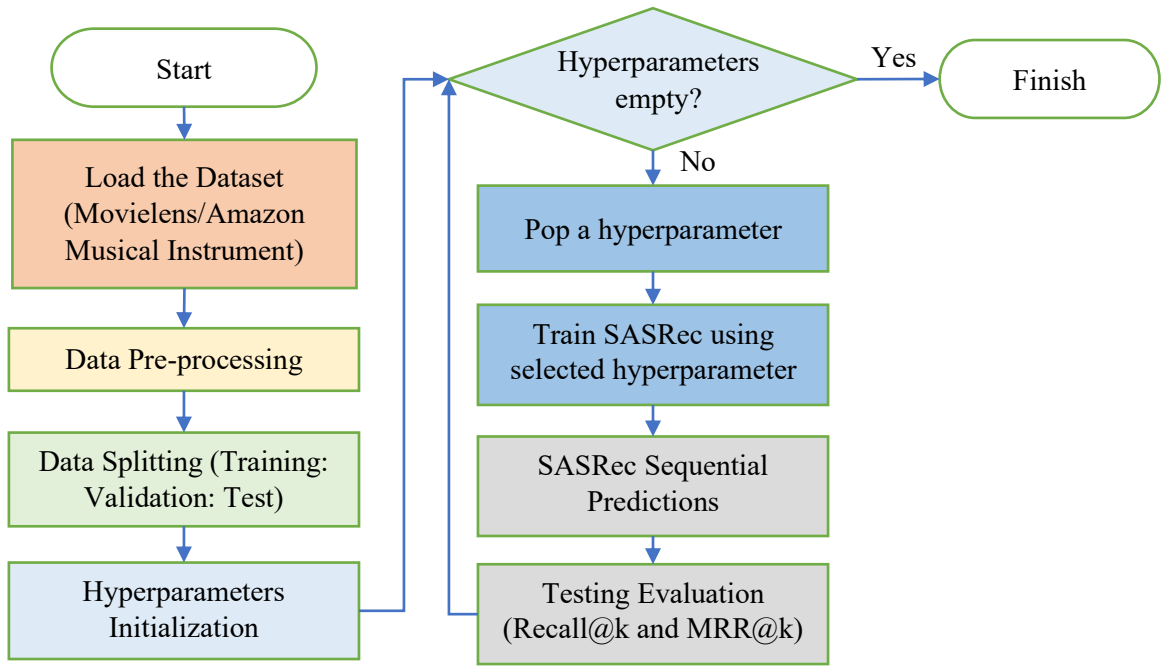


Figure 1. Research Methodology for Sequential Recommendation using SASRec Algorithm

[1] https://www.recbole.io/

3.1. Load Dataset

In implementing SASRec, the first step we carried out was load the dataset (see Figure 1). In this research, 2 datasets were used to measure the effectiveness of the SASRec algorithm in performing sequential recommendation. Datasets used in this study were: a. Amazon Musical Instrument [28] and b. Movielens 1 M [29] different levels of sparsity. Sparsity itself is calculated by comparing the number of unknown interactions with the total number of interactions that should exist. In RecSys, a higher sparsity value represents a greater proportion of unknown interactions in the dataset. The sparsity level can be calculated using the Equation (2):

$$Sparsity = 1 - \frac{|interaction|}{|user|x|item|} \quad (2)$$

where |interaction| denotes the number of observed interactions in the dataset, meanwhile |user| and |item| denotes the number of users and items in the dataset.

The Amazon Musical Instrument dataset itself has explicit interaction data from 1,800,000 users who provided 3,000,000 ratings for 213,600 items (musical instruments). With this amount of data, this dataset is considered moderately sized for modeling in a recommender system. This dataset was collected in 2023 by the McAuley Lab from the University of California San Diego https://amazon-reviews-2023.github.io/ [28]. With this number of interactions, the Amazon Musical Instrument dataset has a Sparsity ≈ 99.99%. A sparsity value approaching 100% makes the Amazon Musical Instrument dataset very challenging to model.

The Movielens 1M dataset is a benchmark dataset often used in Recommender System experiments. The Movielens 1M dataset is medium-sized, storing 1,000,000 rating interactions given by 6,000 users for 4,000 movies. The dataset we used was collected by GroupLens in February 2023 https://grouplens.org/datasets/movielens/1m/ [29] With this number of interactions, the Movielens 1M dataset has a Sparsity ≈ 95.80%. The Sparsity value of the Movielens 1M dataset is smaller (more moderate) compared to the Amazon Musical Instrument dataset, which is much more difficult to model.

3.2. Data Pre-processing

The next process after loading the dataset is the pre-processing stage (see Figure 1 in the yellow-colored process), which was carried out in three subprocesses: a. data ordering, b. data cleansing, and c. sequential normalization. Data ordering was performed for all user interactions, which were sorted from the earliest timestamp to the most recent interaction. From both datasets, we used explicit interaction data that included a timestamp field. To model sequential recommendation, 3 fields were involved in the algorithm: a. "user_id", b. "item_id", and c. "timestamp". The interaction data was then sorted by 'timestamp' to represent a sequential recommendation case.

The data cleansing subprocess was carried out by ensuring that there were no duplicate interaction records. In addition, cleansing was also performed by removing incomplete interaction rows. After the data cleansing stage, the next subprocess was sequential normalization. In this stage, user interactions were constrained to ensure that all sequences had the same length, making them suitable for model training. The sequence length was limited by using a maximum interaction value (max_item_list_length = 50).

3.3. Data Splitting

The next process after pre-processing is Data Splitting (see Figure 1 in the green-colored process). In this study, data splitting was carried out following the Sequential Recommendation scheme. A key principle in data splitting for Sequential Recommendation is sorting interactions by timestamp. In the experiment, a data ratio of Training: Validation: Testing = 80% : 10% : 10% was used. With this

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

mechanism, 80% of each user's earliest data was taken as training data. The last 10% of each user's data, based on time order, was collected as testing data. It is on this testing data that we reported the performance of the Sequential Recommendation. As a note, since data splitting in Sequential Recommendation is done based on time order, it was not possible to perform cross-fold validation in this study. In the trials, only one experiment was conducted for each parameter combination.

3.4. Hyperparameters Initialization

In our research, we conducted tests on several hyperparameter values related to the SASRec algorithm (see Figure 1 in the light-blue-colored process). The hyperparameters we used were *hidden_sizes* = [64, 128, 256], *num_heads* = [2, 4, 8], n_layers = [1, 2, 3], and *inner_sizes* = [128, 256]. In SASRec, the *hidden_sizes* parameter represents the dimension size of the vector embedding representation of an item and also the item's position in the sequential recommendation. The larger the hidden size value, the better the model can represent information. However, a large *hidden_sizes* value requires more computation compared to a small *hidden_sizes* value.

The *num_heads* parameter represents the number of heads used in the multihead self-attention layer. The n_layers parameter represents the number of stacks of self-attention blocks used. A large n_layers value indicates model depth, which is expected to capture long-term dependencies from sequential interaction patterns. Meanwhile the *inner_sizes* value represents the size of the Feed-forward Network in each self-attention block in the SASRec algorithm [30]. Several studies have shown that hyperparameter configurations such as hidden size and depth greatly influence the model's generalization ability and capacity [31].

3.5. Train SASRec using Selected Hyperparameter

After the Hyperparameters Initialization process, we continued with the training of the SASRec algorithm (see Figure 1 in the dark-blue-colored process). In this study, to ensure fairness, we used several parameters that we set to be the same (constant). These parameters were: a. number of epochs = 100, b. max_item_list_length = 50, and c. learning_rate value= 0.001. We set these parameters as fixed values to ensure fairness in comparing the test results.

In the implementation of SASRec training, we used the Recbole library, which is a comprehensive library for Recommender Systems research. The training process was carried out by calling the SASRec model from the *recbole.model.sequential_recommender* module on the training data with the selected hyperparameter configuration. The training was repeated for multiple hyperparameter combinations. A 10% validation set was used to monitor the training process and prevent overfitting.

3.6. Sequential Prediction and Evaluation

After the SASRec training stage, the next stage was Sequential Prediction and Evaluation (see Figure 1 in the gray-colored process). SASRec Sequential Prediction was performed on the 10% testing data obtained from the data splitting process. In the sequential recommendation prediction process, SASRec utilized the representation of the user's previous interaction sequence to estimate the most relevant next item. Each prediction generated a recommendation list (top-k list) for the user.

In Sequential Recommendation, performance assessment was done in the form of a ranked list. Several standard performance metrics based on error calculation such as Mean Absolute Error (MAE) or Root Mean Square Error (RMSE) could not be used as these measurements do not reflect attention to sequential prediction. Therefore, in this study, we used 2 (two) performance calculations for Sequential recommendation: a. $Recall@k$, and b. MRR. $Recall@k$ measured how many relevant items were in the top-k list divided by the total number of relevant items. Mean Reciprocal Rank (MRR) calculated the average of the reciprocal of the rank of the first relevant item in the recommendation list.

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

$Recall@k$ measures the proportion of relevant items that successfully appear in the top-k recommendation list. Formally, $Recall@k$ can be defined using Equation (3):

$$Recall@k = \frac{1}{|U|}\sum_{u\in U}\frac{|Rel(u)\cap Top-k(u)|}{|Rel(u)|} \quad (3)$$

Where U denotes the set of all users in the test set, Rel(u) denotes all relevant items for user u, $Top-k(u)$denotes the recommendation results of k items generated for user u.

$MRR@k$ measures the average reciprocal rank of the first relevant item in the recommendation list provided to a user u. formally, $MRR@k$ can be defined using Equation (4):

$$MRR@k = \frac{1}{|U|}\sum_{u\in U}\frac{1}{rank_u} \quad (4)$$

where $rank_u = min\{r \mid the\ relevant\ item\ is\ located\ at\ position\ r \leq k\}$.

4. RESULT & DISCUSSION

The Sequential Recommendation research we conducted using SASRec combined 54 hyperparameter combinations. These combinations originated from the parameters: a. *hidden_sizes* = {64, 128, 256}, b. *num_heads* = {2, 4, 8}, c. n_layers_list = {1, 2, 3}, and d. *inner_sizes* = {128, 256}. For each combination, we measured the performance of a. Recall@k, and b. MRR@k. To see the progress against different k values in top-K recommendations, we measured performance at 20 different k values, namely k = {5, 10, 15, ..., 100}. We then calculated the average performance based on hyperparameter groups. The average values were then displayed in the performance graphs in Figure 2, Figure 3, Figure 4, and Figure 5, and the performance detail was displayed in Table 1, Table 2, Table 3, and Table 4. Furthure analysis was performed for k = {10, 20, 30}.

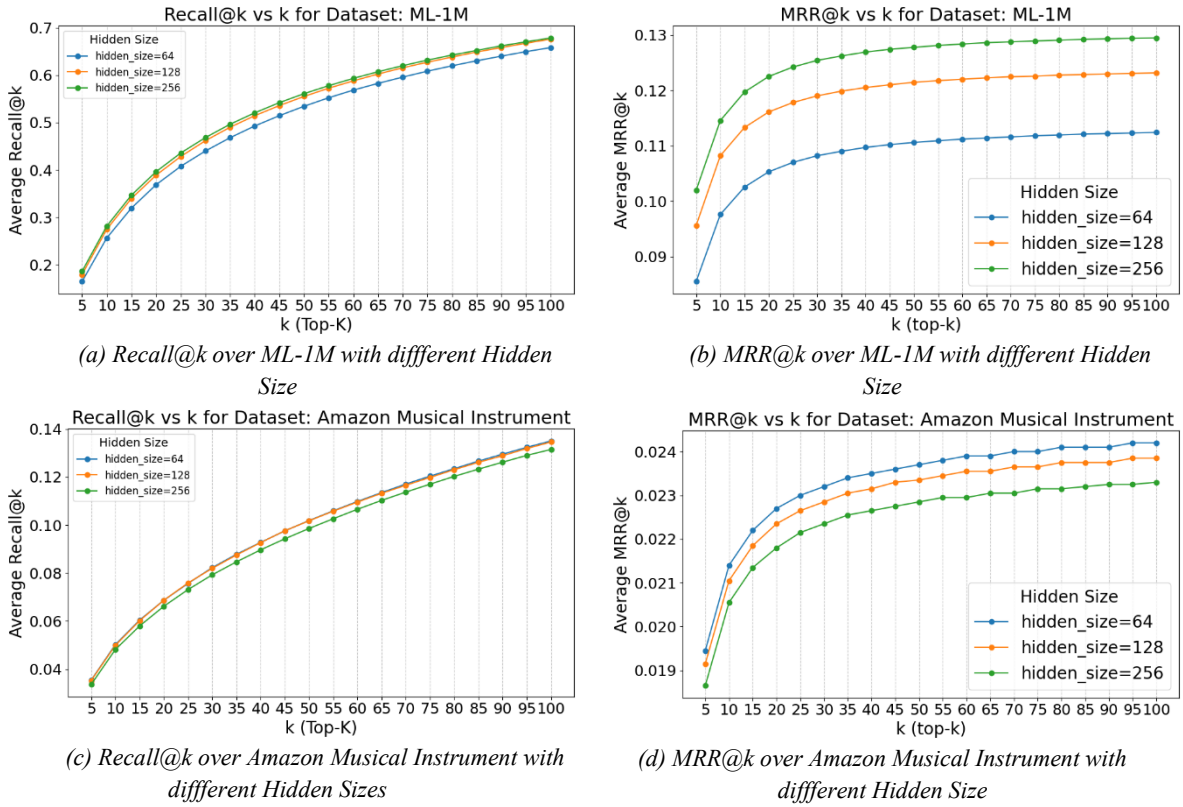
4.1. The Influence of the *hidden_sizes* Parameter

The influence of the *hidden_size* parameter can be seen in Figure 2, which consists of four sub-figure. The experimental results using the Movielens 1M (ML-1M) dataset are presented in Figure 2 (a) which measures Recall@k, and Figure 2 (b) which measures MRR@k. The experimental results using Amazon Musical Instrument dataset are shown in Figure 2 (c) which measures Recall@k, and Figure 2 (d) which measures MRR@k. The detailed impach of the *hidden_size* parameter on performance with k = {10, 20, 30} is presented in Table 1. In Table 1, the green-colored cells indicate the best performance obtained from each parameter combination.

Based on the experimental results, an increase in the *hidden_size* value generally had a positive impact on Recall@k performance (Figure 2), especially on a dense dataset like ML-1M (Figure 2 (a) ML-1M Recall@k). The Recall value consistently increased as the *hidden_size* value increased. This indicated that using a larger embedding dimension influenced capturing sequential interaction patterns. Analysis of the Recall values at k = 10, 20, and 30 with the Movielens dataset reinforced this finding.

Different *hidden_size* values of 64, 128, and 256 increased the recall value. Respectively, the *hidden_size* values of (64, 128, 256) produced Recall@10 performances of (0.2563, 0.2747, 0.2814), Recall@20 of (0.3687, 0.3889, 0.3963), and Recall@30 of (0.4399, 0.4617, 0.4682). Based on the overall average Recall, the *hidden_size*=256 configuration produced the best performance. Although on the Amazon Musical Instrument dataset (Figure 2 (c) Amazon Musical Instrument Recall@k), the increase in *hidden_size* showed a slight decrease.

Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158



(a) Recall@k over ML-1M with diffferent Hidden Size

(b) MRR@k over ML-1M with diffferent Hidden Size

(c) Recall@k over Amazon Musical Instrument with diffferent Hidden Sizes

(d) MRR@k over Amazon Musical Instrument with diffferent Hidden Size

Figure 2. Performance Comparizon of the Hidden Size Parameter in the SASRec Algorithm on the Movielens 1M and Amazon Musical Instrument Dataset

Table 1. Average Recall@k and MRR@k Performances Influenced by *hidden_size*

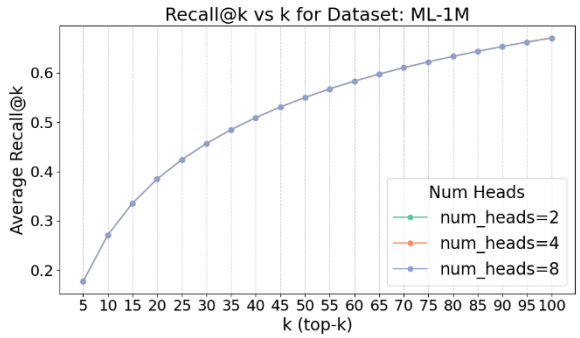
Dataset	*hidden_size*	Recall@k			MRR@k		
		k = 10	k = 20	k = 30	k = 10	k = 20	k = 30
Amazon Musical Instrument	64	0.0502	0.0687	0.0823	0.0214	0.0227	0.0232
Amazon Musical Instrument	128	0.0499	0.0687	0.0820	0.0211	0.0224	0.0229
Amazon Musical Instrument	256	0.0482	0.0662	0.0793	0.0206	0.0218	0.0224
ML-1M	64	0.2563	0.3687	0.4399	0.0976	0.1053	0.1082
ML-1M	128	0.2747	0.3889	0.4617	0.1082	0.1161	0.1190
ML-1M	256	0.2814	0.3963	0.4683	0.1146	0.1225	0.1254

The increase in Recall@k performance was also reflected in the MRR@k performance measurement. The influence of the *hidden_size* parameter on the Mean Reciprocal Rank (MRR@k) metric is shown in Figure 2 (b) and (d). On the ML-1M dataset, increasing the *hidden_size* value had a positive impact on the MRR value (Figure 2 (b)). Respectively, *hidden_size* values of (64, 128, 256) produced MRR@10 performances of (0.0976, 0.1082, 0.1146), MRR@20 of (0.1053, 0.1161, 0.1225), and MRR@30 of (0.1082, 0.1190, 0.1254). Conversely, on the Amazon Musical Instrument dataset (see Figure 2 (d)), the increase in *hidden_size* was not accompanied by an increase in MRR performance. In fact, there was a gradual decrease in the MRR value. Respectively, the *hidden_size* values of (64, 128, 256) produced MRR@10 performances of (0.0214, 0.02105, 0.02055), MRR@20 of (0.0227, 0.02235, 0.02180), and MRR@30 of (0.0232, 0.02285, 0.02235).

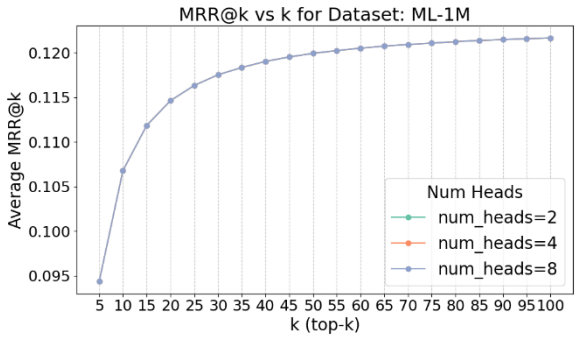
Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

4.2. The Influence of the *num_heads* Parameter

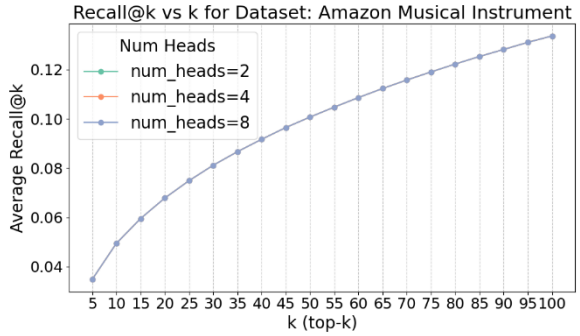
The influence of the *num_heads* parameter can be seen in Figure 3, which consists of four sub-figures. The experimental results using the Movielens 1M (ML-1M) dataset are presented in Figure 3 (a) which measures Recall@k, and Figure 3 (b) which measures MRR@k. The experimental results using Amazon Musical Instrument dataset are shown in Figure 3 (c) which measures Recall@k, and Figure 3 (d) which measures MRR@k. The detailed impach of the *num_heads* parameter on performance with k = {10, 20, 30} is presented in Table 2.



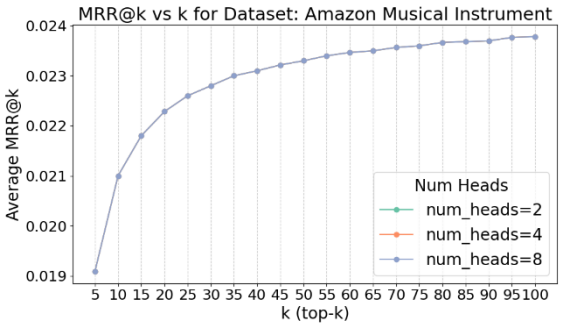
(a)Recall@k over ML-1M with diffferent num_heads



(b) MRR@k over ML-1M with diffferent num_heads



(c) Recall@k over Amazon Musical Instrument with diffferent num_heads



(d) MRR@k over Amazon Musical Instrument with diffferent num_heads

Figure 3. Performance Comparizon of the *num_heads* Parameter in the SASRec Algorithm on the Movielens 1M and Amazon Musical Instrument Dataset

Table 2. Average Recall@k and MRR@k Performances Influenced by *num_heads*

		Recall@k			MRR@k		
Dataset	*num_heads*	k = 10	k = 20	k = 30	k = 10	k = 20	k = 30
Amazon Musical Instrument	2	0.0494	0.0679	0.0812	0.0210	0.0223	0.0228
Amazon Musical Instrument	4	0.0494	0.0679	0.0812	0.0210	0.0223	0.0228
Amazon Musical Instrument	8	0.0494	0.0679	0.0812	0.0210	0.0223	0.0228
ML-1M	2	0.2709	0.3846	0.4566	0.1068	0.1146	0.1175
ML-1M	4	0.2709	0.3846	0.4566	0.1068	0.1146	0.1175
ML-1M	8	0.2709	0.3846	0.4566	0.1068	0.1146	0.1175

On both datasets, ML-1M and Amazon Musical Instrument, the experimental results showed that variations in the *num_heads* value (i.e., 2, 4, and 8) did not yield significant performance differences in the average Recall@k. This was indicated by the overlapping graphs in Figure 3 (a) and (c). The average Recall@10, Recall@20, and Recall@30 values on ML-1M were all the same, namely 0.2708, 0.3846, and 0.4566 for all *num_heads* values. The same pattern also occurred on the Amazon dataset, where the

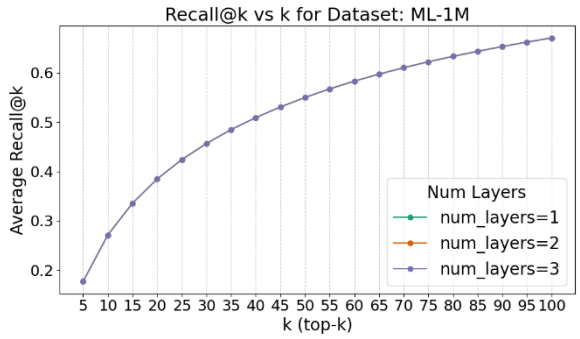
Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

average Recall@10, Recall@20, and Recall@30 were consistently 0.0494, 0.0678, and 0.08115 for all *num_heads* values.

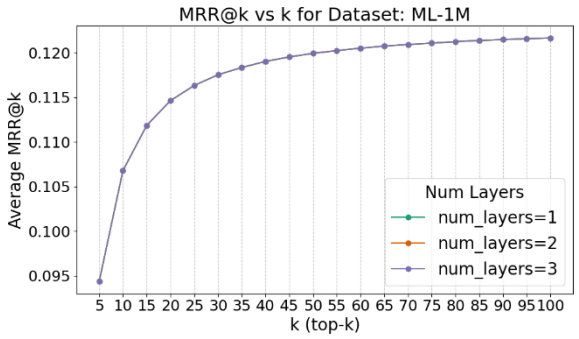
The same results were also shown for the influence of *num_heads* value variations (2, 4, and 8) on MRR@k performance. This influence occurred on both the ML-1M and Amazon Musical Instrument datasets. On the ML-1M dataset, the MRR@10, MRR@20, and MRR@30 values remained identical. It can be seen in Figure 3 (b) and (d)), that the average MRR values appeared to overlap for all *num_heads* value configurations. The average MRR@10, MRR@20, and MRR@30 values on ML-1M were all the same, namely 0.1068, 0.1146, and 0.1175 for all *num_heads* configurations. The same pattern also occurred on the Amazon dataset, where the average MRR@10, MRR@20, and MRR@30 values were consistent at 0.0210, 0.0223, and 0.0228 for all *num_heads* values. The test results indicated that the number of attention heads in the self-attention mechanism did not affect SASRec's ability to recognize sequential recommendation patterns. Considering the increased computation for a larger number of *num_heads*, a value of *num_heads*=2 could be used as the default value.

4.3. The Influence of the n_layers Parameter

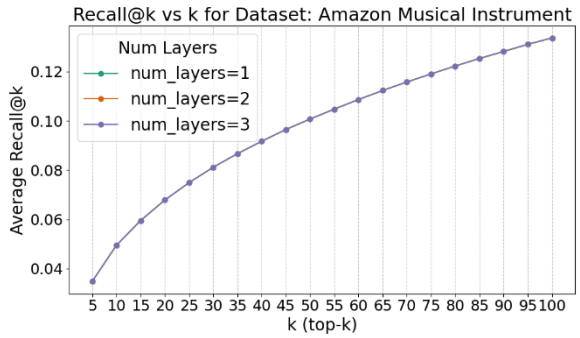
The influence of the *n_layers* parameter can be seen in Figure 4, which consists of four sub-figures. The experimental results using the Movielens 1M (ML-1M) dataset are presented in Figure 4 (a) which measures Recall@k, and Figure 4 (b) which measures MRR@k. The experimental results using Amazon Musical Instrument dataset are shown in Figure 4 (c) which measures Recall@k, and Figure 4 (d) which measures MRR@k. The detailed impach of the *n_layers* parameter on performance with k = {10, 20, 30} is presented in Table 3.



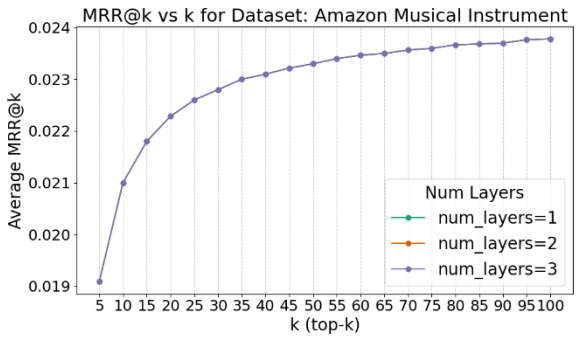
(a) Recall@k over ML-1M with diffferent n_layers



(b) MRR@k over ML-1M with diffferent n_layers



(c) Recall@k over Amazon Musical Instrument with diffferent n_layers



(d) MRR@k over Amazon Musical Instrument with diffferent n_layers

Figure 4. Performance Comparizon of the *n_layers* Parameter in the SASRec Algorithm on the Movielens 1M and Amazon Musical Instrument Dataset

On both datasets, ML-1M and Amazon Musical Instrument, the experimental results showed that variations in the n_layers value (i.e., 1, 2, and 3) did not provide significant performance differences in the average Recall@k. This was shown by the overlapping graphs in Figure 4 (a) and (c). The average Recall@10, Recall@20, and Recall@30 values on ML-1M were all the same, namely 0.2708, 0.3846,

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

and 0.4566 for all n_layers configurations. The same pattern also occurred on the Amazon dataset, where the average Recall@10, Recall@20, and Recall@30 values were consistent at 0.0494, 0.0678, and 0.0749 for all n_layers values.

Table 3. Average Recall@k and MRR@k Performances Influenced by *num_layers*

		Recall@k			MRR@k		
Dataset	*num_layers*	k = 10	k = 20	k = 30	k = 10	k = 20	k = 30
Amazon Musical Instrument	1	0.0495	0.0678	0.0812	0.0210	0.0223	0.0228
Amazon Musical Instrument	2	0.0495	0.0678	0.0812	0.0210	0.0223	0.0228
Amazon Musical Instrument	3	0.0495	0.0678	0.0812	0.0210	0.0223	0.0228
ML-1M	1	0.2708	0.3846	0.4566	0.1068	0.1146	0.1175
ML-1M	2	0.2708	0.3846	0.4566	0.1068	0.1146	0.1175
ML-1M	3	0.2708	0.3846	0.4566	0.1068	0.1146	0.1175

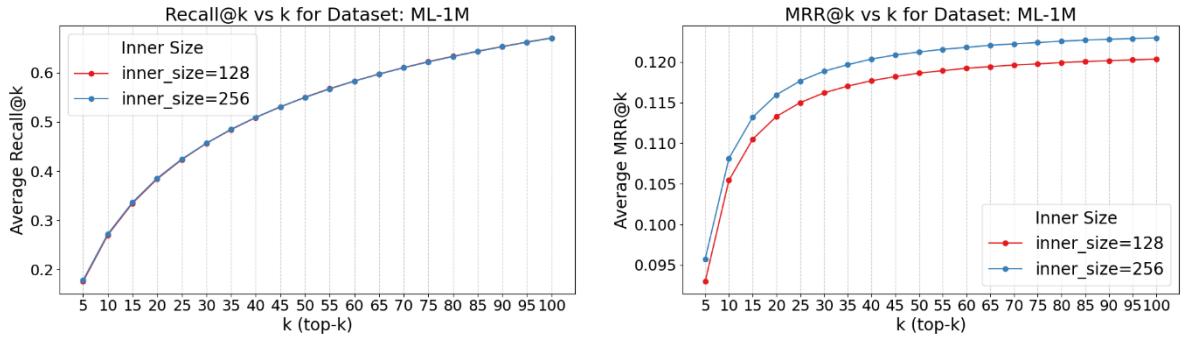
The same results were also shown for the influence of n_layers value variations (1, 2, and 3) on MRR@k performance. This influence occurred on both the ML-1M and Amazon Musical Instrument datasets. On the ML-1M dataset, the MRR@10, MRR@20, and MRR@30 values remained identical. As can be seen in Figure 4 (b) and (d)) the average MRR values appeared to overlap for all n_layers value configurations. The average MRR@10, MRR@20, and MRR@30 values on ML-1M were all the same, namely 0.1068, 0.1146, and 0.1175 for all n_layers configurations. The same pattern also occurred on the Amazon dataset, where the average MRR@10, MRR@20, and MRR@30 values were consistent at 0.0210, 0.0223, and 0.0228 for all n_layers values. The test results showed that the n_layers parameter did not affect SASRec's ability to recognize sequential recommendation patterns. Considering the increased computation for a larger number of n_layers, a value of n_layers=1 should be used as the default value.

4.4. The Influence of the *inner_sizes* Parameter

The influence of the *inner_sizes* parameter can be seen in Figure 5, which consists of four sub-figure. The experimental results using the Movielens 1M (ML-1M) dataset are presented in Figure 5 (a) which measures Recall@k, and Figure 5 (b) which measures MRR@k. The experimental results using Amazon Musical Instrument dataset are shown in Figure 5 (c) which measures Recall@k, and Figure 5 (d) which measures MRR@k. The detailed impach of the *inner_sizes* parameter on performance with k = {10, 20, 30} is presented in Table 4. In Table 4, the green-colored cells indicate the best performance obtained from each parameter combination.

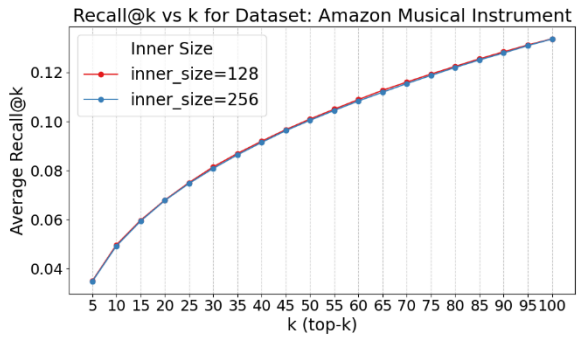
On both datasets, ML-1M and Amazon Musical Instrument, the experimental results showed that variations in the *inner_sizes* value (i.e., 128 and 256) did not provide significant performance differences in the average Recall@k. This was indicated by the overlapping graphs in Figure 5 (a) and (c). The average recall@k performance on the ML-1M dataset with an *inner_sizes* value of 128 was: Recall@10=0.2698, Recall@20=0.3838, and Recall@30=0.4564. The average recall@k performance increased with an increase in the *inner_sizes* value to 256, with: Recall@10=0.2718, Recall@20=0.3854, and Recall@30=0.4567. The opposite occurred on the Amazon Musical Instrument dataset. The average recall@k performance with an *inner_sizes* value of 128 was: Recall@10=0.0496, Recall@20=0.0679, and Recall@30=0.0815. The average recall@k performance decreased with an increase in the *inner_sizes* value to 256, with: Recall@10=0.0492, Recall@20=0.0677, and Recall@30=0.0808. Although there were increases and decreases in the average Recall@k, the performance changes were not very large.

Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

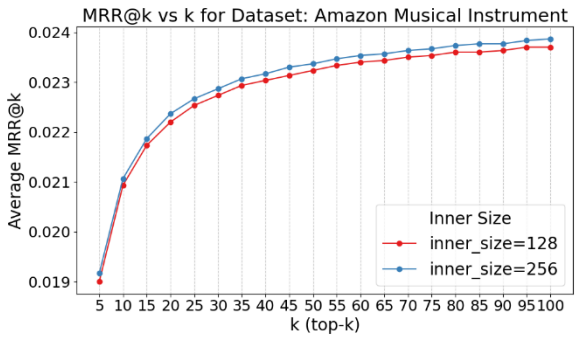


(a) Recall@k over ML-1M with diffferent Inner Size

(b) MRR@k over ML-1M with diffferent Inner Size



(c) Recall@k over Amazon Musical Instrument with diffferent Inner Size



(d) MRR@k over Amazon Musical Instrument with diffferent Inner Size

Figure 5. Performance Comparizon of the *inner_size* Parameter in the SASRec Algorithm on the Movielens 1M and Amazon Musical Instrument Dataset

Table 4. Average Recall@k and MRR@k Performances Influenced by *inner_size*

		Recall@k			MRR@k		
Dataset	*inner_size*	k = 10	k = 20	k = 30	k = 10	k = 20	k = 30
Amazon Musical Instrument	128	0.0496	0.0679	0.0815	0.0209	0.0222	0.0227
Amazon Musical Instrument	256	0.0492	0.0677	0.0808	0.0211	0.0224	0.0229
ML-1M	128	0.2698	0.3838	0.4564	0.1054	0.1133	0.1162
ML-1M	256	0.2718	0.3854	0.4567	0.1081	0.1160	0.1189

In contrast to the Recall@k performance, for the average MRR@k measurement, an increase in the *inner_sizes* value was found to improve the MRR@k performance. This was shown in the graphs in Figure 5 (b) and (d). The impact of the *inner_size* parameter performance on the Mean Reciprocal Rank (MRR@k) metric is shown in Figure 5 (b) and (d). On the ML-1M and Amazon Musical Instrument datasets, an increase in the *inner_size* value had a positive impact on the MRR value. On the ML-1M dataset, an *inner_sizes* value of 128 produced average MRR@k values of: MRR@10=0.1054, MRR@20=0.1133, and MRR@30=0.1162. Meanwhile, an *inner_sizes* value of 256 produced average MRR@k values of: MRR@10=0.1081, MRR@20=0.1160, and MRR@30=0.1189. An increase in the average MRR@k performance also occurred on the Amazon Musical Instrument dataset. On the Amazon Musical Instrument dataset, an *inner_sizes* value of 128 produced average MRR@k values of: MRR@10=0.0209, MRR@20=0.0222, and MRR@30=0.0227. Meanwhile, an *inner_sizes* value of 256 produced average MRR@k values of: MRR@10=0.0211, MRR@20=0.0224, and MRR@30=0.0229. From the test results, it was shown that the *inner_sizes* value had an influence on the MRR@k value, but not much influence on the Recall@k value. Considering the performance increase and the existing computational requirements, the appropriate *inner_sizes* value for the case at hand needs to be carefully considered.

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

In general, the performance achieved by the SASRec algorithm on the Movielens 1M (ML-1M) dataset is higher compared to the Amazon Musical Instrument dataset. This can be observed in Table 1, Table 2, Table 3, and Table 4. This phenomenon can be observed when comparing the values of MRR@k and Recall@k under the same hyperparameter settings, namely hidden size, inner size, number of attention heads (*num_heads*), and number of layers (*num_layers*). The higher performance on the MovieLens 1M (ML-1M) dataset is influenced by its characteristics. In sequential recommendation tasks, predicting the next item becomes more difficult when the number of candidate items is large, compared to cases with fewer candidate items. In the MovieLens 1M dataset, there are 4,000 movies as candidate items, resulting in a probability of 1/4,000 for correctly predicting the next item. In contrast, the Amazon Musical Instrument dataset contains 213,600 candidate items, which reduces the probability of a correct prediction to 1/213,600. This condition explains why the overall Recall@k and MRR@k values on the Amazon Musical Instrument dataset are consistently lower compared to those on the MovieLens 1M dataset.

4.5. Performance Comparison with Other Models

To evaluate the capability of the SASRec algorithm in solving sequential recommendation tasks, we conducted a performance comparison with two other algorithms, namely BERT4Rec and GRU4Rec. These three algorithms were selected because they belong to the same group, i.e., Pure ID-Based Sequential Recommendation. To ensure fairness, we used consistent parameters across all models: (a) number_of_epochs = 100, (b) max_item_list_length = 50, and c. *learning_rate* = 0.001. Since SASRec was evaluated using 54 hyperparameter combinations, the same number of combinations was also applied to BERT4Rec and GRU4Rec. Given that GRU4Rec has fewer hyperparameters, we included the learning_rate as a variable hyperparameter with values [0.0005, 0.001, 0.0015]. The performance was then compared based on the average Recall@20 and MRR@20. In addition, we also reported the best Recall@20 and MRR@20 values achieved by each model. The results are presented in Table 5.

Table 5. Comparative Performance Analysis of SASRec, BERT4Rec, and GRU4Rec

		Movielens 1M			Amazon Musical Instrument		
		BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec	SASRec
Average	recall@10	0.3021	0.2871	0.2708	0.0237	0.0358	0.0494
	recall@20	0.4175	0.3989	0.3846	0.0356	0.0521	0.0678
	recall@30	0.4900	0.4700	0.4566	0.0446	0.0643	0.0811
Best	recall@10	0.3198	0.3080	0.2827	0.0260	0.0429	0.0506
	recall@20	0.4362	0.4246	0.3985	0.0393	0.0610	0.0689
	recall@30	0.5085	0.4965	0.4708	0.0492	0.0747	0.0824
Average	MRR@10	0.1261	0.1195	0.1068	0.0089	0.0147	0.0210
	MRR@20	0.1341	0.1272	0.1146	0.0097	0.0159	0.0223
	MRR@30	0.1370	0.1301	0.1175	0.0101	0.0163	0.0228
Best	MRR@10	0.1367	0.1315	0.1153	0.0100	0.0188	0.0214
	MRR@20	0.1446	0.1393	0.1232	0.0108	0.0200	0.0227
	MRR@30	0.1475	0.1422	0.1261	0.0112	0.0205	0.0232

Table 5 presents the performance comparison among SASRec, BERT4Rec, and GRU4Rec. The table contains two groups of rows: the Average group, which reports the mean performance across 54 hyperparameter combinations, and the Best group, which reports the best performance obtained among these 54 trials. From the Table 5, it can be observed that on the MovieLens 1M dataset, BERT4Rec outperforms both GRU4Rec and SASRec. This superiority is evident across all performance

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

measurements, with average Recall@10 = 0.3021, average Recall@20 = 0.4175, average Recall@30 = 0.4900, average MRR@10 = 0.1261, average MRR@20 = 0.1341, and average MRR@30 = 0.1370.

On the other hand, in the Amazon Musical Instrument dataset, SASRec achieves better performance compared to the two baseline algorithms, BERT4Rec and GRU4Rec. This superiority is consistent across all performance measurements, with average Recall@10 = 0.0494, average Recall@20 = 0.0678, average Recall@30 = 0.0811, average MRR@10 = 0.0210, average MRR@20 = 0.0223, and average MRR@30 = 0.0228. Considering the dataset characteristics, particularly the high sparsity level of the Amazon Musical Instrument dataset, SASRec demonstrates greater robustness in handling sparse data.

5. DISCUSSION

5.1. Interpretation of Experimental Results

Based on the experimental results, the *hidden_size* parameter has an influence on the model's performance. A higher *hidden_size* value helps SASRec capture semantic relationships in sequential recommendation. This effect is clearly seen in the ML-1M dataset, where a *hidden_size* value of 256 produces better Recall and MRR performance. With a higher *hidden_size* value, the ML-1M dataset, which has lower sparsity, allows sequential interaction patterns to be learned more easily. Conversely, in the Amazon Musical Instrument dataset with higher sparsity, increasing the *hidden_size* can decrease performance. The same phenomenon is observed in the *inner_size* parameter. In SASRec, *inner_size* represents the capacity of the feed-forward network in the Transformer block. A moderate increase in *inner_size* can improve performance. This can be seen in the ML-1M dataset, where the *inner_size* value can capture sequential interaction relationships between items. On the other hand, the *num_heads* and *num_layers* parameters do not show a significant influence on sequential recommendation performance. Considering these results, increasing the values of *num_heads* and *num_layers* in the SASRec algorithm does not need to be performed, as it will only increase computational complexity.

5.2. Effect of Dataset Sparsity on Performance

In this experiment, we used two datasets with different levels of sparsity. The Movielens 1M dataset has a lower level of sparsity compared to the Amazon Musical Instrument dataset. In the experimental results, the performance of the SASRec algorithm was influenced by the different levels of sparsity. In the ML-1M dataset with low sparsity, the model was able to learn patterns more effectively. Sequential interaction patterns could be learned more easily, both for short-term and long-term dependencies. Conversely, in the Amazon Musical Instrument dataset with high sparsity, the small number of interactions made it difficult for the SASRec algorithm to capture sequential interaction patterns. Nevertheless, based on the comparison with other algorithms, the SASRec algorithm was relatively more stable than the BERT4Rec and GRU4Rec algorithms when applied to datasets with a high level of sparsity.

5.3. Comparison of SASRec with Other Algorithms

Based on the comparison results of the SASRec algorithm with BERT4Rec and GRU4Rec (see Table 5), it was found that the performance of the algorithms was influenced by the level of data sparsity. The BERT4Rec algorithm achieved the best performance on datasets with a low level of sparsity, such as ML-1M. The bidirectional self-attention mechanism in BERT4Rec allows the algorithm to understand interaction patterns from two directions in sequential interaction. The high density level in ML-1M provides sufficient contextual information for BERT4Rec. Nevertheless, on datasets with high sparsity such as Amazon Musical Instrument, the performance of BERT4Rec was lower compared to the SASRec algorithm. In the SASRec algorithm, performance was better on data with high sparsity.

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

The self-attention architecture allows the SASRec algorithm to capture information more efficiently. Meanwhile, GRU4Rec consistently showed moderate performance across different levels of sparsity.

6. CONCLUSION

This research evaluates the influence of hyperparameter configurations in the SASRec algorithm on sequential recommendation performance, measured by Recall@k and MRR@k metrics. Based on the experimental results on two different datasets (ML-1M and Amazon Musical Instrument), we obtained the following conclutions:

a. The hidden size has a significant influence on performance, especially on a dense dataset like ML-1M. Increasing the hidden size from 64 to 256 consistently improves Recall and MRR. However, on a dataset with high sparsity like Amazon, performance tends to stagnate or even decrease, which indicates potential overfitting.
b. The inner size parameter shows a similar pattern, where increasing the capacity of the feedforward network provides a slight performance improvement, especially on the ML-1M dataset.
c. Conversely, parameters such as *num_heads* and *num_layers* do not show a significant contribution to the recommendation quality on either dataset. This indicates that tuning these parameters is non-sensitive for the experiment.
d. Compared to BERT4Rec and GRU4Rec, the SASRec algorithm shows superior performance on datasets with high sparsity, such as Amazon Musical Instrument, with an average Recall@20 of 0.0678 and an average MRR@20 of 0.0223.

The suggestion from this research is that the selection of hyperparameter configurations in SASRec should be adapted to the dataset's characteristics. For dense datasets, a configuration with a larger hidden size and inner size can provide better performance. Conversely, on datasets with high sparsity, using a model with low complexity, such as a small hidden size and inner size, can be more efficient without sacrificing accuracy. Furthermore, since increasing the number of layers and attention heads does not show a significant improvement, selecting the minimum values (such as *num_layers*=1 and *num_heads*=2) can be used to save computation time and resources without sacrificing performance. In the future, exploration of SASRec adaptation on sparse data as well as the integration of regularization techniques or pretrained embeddings can be a direction for further research.

CONFLICT OF INTEREST

The authors declares that there is no conflict of interest between the authors or with research object in this paper.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the Direktorat Penelitian dan Pengabdian kepada Masyarakat (DPPM) Universitas Telkom for the financial support through the Penelitian Internal Tahun 2025 Universitas Telkom grant scheme, under the contract number NO.228/LIT06/PPM-LIT/2025. This support has significantly contributed to the completion of this research.

REFERENCES

[1] X. Bai, H. Peng, Y. Huang, J. Wang, Q. Yang, and A. Ramírez-De-Arellano, “A graph attention network integrated with gated spiking neural P systems for session-based recommendation,” *Expert Syst Appl*, vol. 286, Aug. 2025, doi: 10.1016/j.eswa.2025.128029.

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

[2] D. B. Rajesh and A. Kumar, "Collaborative filtering models an experimental and detailed comparative study," *Sci Rep*, vol. 15, no. 1, p. 31667, Aug. 2025, doi: 10.1038/s41598-025-15096-4.

[3] H. Vaghari, M. Hosseinzadeh Aghdam, and H. Emami, "Group attention for collaborative filtering with sequential feedback and context aware attributes," *Sci Rep*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-94256-y.

[4] M. B. S. Siddik and A. T. Wibowo, "Collaborative Filtering Based Food Recommendation System Using Matrix Factorization," *JURNAL MEDIA INFORMATIKA BUDIDARMA*, vol. 7, no. 3, p. 1041, Jul. 2023, doi: 10.30865/mib.v7i3.6049.

[5] H. U. Khan, A. Naz, F. K. Alarfaj, and N. Almusallam, "A transformer-based architecture for collaborative filtering modeling in personalized recommender systems," *Sci Rep*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-08931-1.

[6] G. D. S. P. Moreira, S. Rabhi, J. M. Lee, R. Ak, and E. Oldridge, "Transformers4Rec: Bridging the Gap between NLP and sequential/session-based recommendation," in *RecSys 2021 - 15th ACM Conference on Recommender Systems*, Association for Computing Machinery, Inc, Sep. 2021, pp. 143–153. doi: 10.1145/3460231.3474255.

[7] H. Fan *et al.*, "TiM4Rec: An efficient sequential recommendation model based on time-aware structured state space duality model," *Neurocomputing*, vol. 654, p. 131270, Nov. 2025, doi: 10.1016/j.neucom.2025.131270.

[8] Z. Wang, B. Liu, W. Huang, T. Hao, H. Zhou, and Y. Guo, "Leveraging multimodal large language model for multimodal sequential recommendation," *Sci Rep*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-14251-1.

[9] Q. Tan *et al.*, "Sparse-Interest Network for Sequential Recommendation," in *WSDM 2021 - Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, Association for Computing Machinery, Inc, Aug. 2021, pp. 598–606. doi: 10.1145/3437963.3441811.

[10] Z. Fan *et al.*, "Sequential Recommendation via Stochastic Self-Attention," in *WWW 2022 - Proceedings of the ACM Web Conference 2022*, Association for Computing Machinery, Inc, Apr. 2022, pp. 2036–2047. doi: 10.1145/3485447.3512077.

[11] H. Wang, Y. Fan, Y. Du, X. Li, and X. Wang, "Improving contrastive learning with explanation method for sequential recommendation," *Expert Syst Appl*, vol. 291, Oct. 2025, doi: 10.1016/j.eswa.2025.128534.

[12] X. Du *et al.*, "Frequency Enhanced Hybrid Attention Network for Sequential Recommendation," in *SIGIR 2023 - Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Association for Computing Machinery, Inc, Jul. 2023, pp. 78–88. doi: 10.1145/3539618.3591689.

[13] A. Klenitskiy and A. Vasilev, "Turning Dross Into Gold Loss: is BERT4Rec really better than SASRec?," in *Proceedings of the 17th ACM Conference on Recommender Systems*, New York, NY, USA: ACM, Sep. 2023, pp. 1120–1125. doi: 10.1145/3604915.3610644.

[14] D. Tikhonovich *et al.*, "eSASRec: Enhancing Transformer-based Recommendations in a Modular Fashion," in *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, New York, NY, USA: ACM, Sep. 2025, pp. 1175–1180. doi: 10.1145/3705328.3759317.

[15] A. Volodkevich, D. Gusak, A. Klenitskiy, A. Pembek, and A. Vasilev, "Autoregressive generation strategies for Top-K sequential recommendations," *User Model User-adapt Interact*, vol. 35, no. 3, p. 13, Sep. 2025, doi: 10.1007/s11257-025-09433-5.

[16] U. Javed, K. Shaukat, I. A. Hameed, F. Iqbal, T. M. Alam, and S. Luo, "A Review of Content-Based and Context-Based Recommendation Systems," *International Journal of Emerging Technologies in Learning*, vol. 16, no. 3, pp. 274–306, 2021, doi: 10.3991/ijet.v16i03.18851.

Vol. 7, No. 2, April 2026, Page. 1714-1731
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5158

[17] L. Pan, W. Pan, M. Wei, H. Yin, and Z. Ming, "A Survey on Sequential Recommendation," *Front Comput Sci*, pp. 1–45, Mar. 2025, doi: 10.1007/s11704-025-41329-w.

[18] W. Li *et al.*, "Collaborative local–global context modeling for session-based recommendation," *Inf Process Manag*, vol. 62, no. 5, p. 104196, Sep. 2025, doi: 10.1016/j.ipm.2025.104196.

[19] A. H. J. P. Juni Permana and Agung Toto Wibowo, "Movie Recommendation System Based on Synopsis Using Content-Based Filtering with TF-IDF and Cosine Similarity," *International Journal on Information and Communication Technology (IJoICT)*, vol. 9, no. 2, pp. 1–14, Dec. 2023, doi: 10.21108/ijoict.v9i2.747.

[20] W. Gao, "Research on optimization of library book recommendation system based on the collaborative fusion of transformer architecture and adaptive extreme learning machine," *Systems and Soft Computing*, vol. 7, Dec. 2025, doi: 10.1016/j.sasc.2025.200287.

[21] W. Zhang, Z. Chen, H. Zha, and J. Wang, "Learning from Substitutable and Complementary Relations for Graph-based Sequential Product Recommendation," *ACM Trans Inf Syst*, vol. 40, no. 2, Apr. 2022, doi: 10.1145/3464302.

[22] X. Huang, J. Sang, J. Yu, and C. Xu, "Learning to Learn a Cold-start Sequential Recommender," *ACM Trans Inf Syst*, vol. 40, no. 2, pp. 1–25, Apr. 2022, doi: 10.1145/3466753.

[23] Y. Ding, Y. Ma, W. K. Wong, and T. S. Chua, "Leveraging two types of global graph for sequential fashion recommendation," in *ICMR 2021 - Proceedings of the 2021 International Conference on Multimedia Retrieval*, Association for Computing Machinery, Inc, Aug. 2021, pp. 73–81. doi: 10.1145/3460426.3463638.

[24] X. Chen and Q. Li, "Causality-driven User Modeling for Sequential Recommendations over Time," in *WWW 2024 Companion - Companion Proceedings of the ACM Web Conference*, Association for Computing Machinery, Inc, May 2024, pp. 1400–1406. doi: 10.1145/3589335.3651896.

[25] M. Ma *et al.*, "Improving Transformer-based Sequential Recommenders through Preference Editing," *ACM Trans Inf Syst*, vol. 41, no. 3, pp. 1–24, Jul. 2023, doi: 10.1145/3564282.

[26] W.-C. Kang and J. McAuley, "Self-Attentive Sequential Recommendation," in *2018 IEEE International Conference on Data Mining (ICDM)*, IEEE, Nov. 2018, pp. 197–206. doi: 10.1109/ICDM.2018.00035.

[27] K. Zhou, H. Yu, W. X. Zhao, and J. R. Wen, "Filter-enhanced MLP is All You Need for Sequential Recommendation," in *WWW 2022 - Proceedings of the ACM Web Conference 2022*, Association for Computing Machinery, Inc, Apr. 2022, pp. 2388–2399. doi: 10.1145/3485447.3512111.

[28] J. Ni, J. Li, and J. McAuley, "Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 188–197. doi: 10.18653/v1/D19-1018.

[29] F. M. Harper and J. A. Konstan, "The MovieLens Datasets," *ACM Trans Interact Intell Syst*, vol. 5, no. 4, pp. 1–19, Jan. 2016, doi: 10.1145/2827872.

[30] F. Sun *et al.*, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *International Conference on Information and Knowledge Management, Proceedings*, Association for Computing Machinery, Nov. 2019, pp. 1441–1450. doi: 10.1145/3357384.3357895.

[31] C. Chen, F. Liu, Y. Li, C. Wu, T. Qi, and Q. Liu, "User Behavior Modeling with Self-Supervised Contrastive Learning for Recommendation," in *Proceedings of the Web Conference 2021 (WWW)*, 2021.