Recommender Systems: looking further into the future

Kyriakos Kyriakou

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2023

Abstract

Recommender systems play a crucial role in various sectors, from entertainment apps to e-commerce platforms, by offering personalized recommendations based on users' preferences and historical interactions. However, many of these systems primarily focus on predicting the immediate next action of users, potentially missing out on complex patterns in users' behaviors. This thesis presents an in-depth exploration of sequential recommendation systems, with the primary goal of improving the prediction of future user interactions. The research expands upon the work of Pancha et al. (2022), offering a divergent approach by focusing on the sequential order of future interactions, rather than their explicit temporal context. Using the SASRec model as a foundation, we introduce several training objectives and techniques, and evaluate their effectiveness on the widely-used MovieLens-1M dataset. Our findings reveal that our Integrated All Action Prediction modeling approach, combined with the Sampled Softmax Uniform loss, outperforms other methods in predicting future user interactions. This hybrid approach integrates the strengths of both Next Item Prediction and long-term All Action Prediction, achieving superior performance across several key metrics, including NDCG@10, Hit@10, and Kendall's Tau. The study also investigates the impact of various training techniques and the size of future prediction windows. Among these, Teacher Forcing particularly stands out for its proficiency in predicting a well-ordered sequence of future items. Additionally, extending the future prediction window size indicates potential long-term benefits in recommendation accuracy. Despite the significant contributions of this research, it opens new avenues for future work. These findings offer valuable insights and pave the way for the development of more effective and robust sequential recommendation systems.

Research Ethics Approval

Declaration

(*Kyriakos Kyriakou*)

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors David Wardrope, Timos Korres, Michael la Grange, and Thomas Baumhauer at Amazon. Their dedication, guidance, and invaluable insights have been instrumental in shaping this thesis. Their unwavering support and supervision throughout the past months have not only enriched my academic journey but also instilled in me a passion for continued learning and exploration.

I am also profoundly thankful to my family, whose unwavering faith in my abilities has been a constant source of strength and inspiration. Their love, encouragement, and sacrifices have been the bedrock upon which I built this work.

Additionally, a special note of gratitude goes to my partner, who has been a pillar of support and understanding throughout this endeavor. Her encouragement, patience, and unwavering belief in me have been a source of constant motivation.

To my friends, your camaraderie, encouragement, and belief in my vision have been invaluable. Your constant support, both in moments of triumph and in times of challenge, has been a beacon of hope and resilience.

In conclusion, I am reminded that the journey of completing this thesis was made possible by the collective efforts of many. I am forever indebted to all who played a part in this milestone of my academic journey.

Table of Contents

Chapter 1

Introduction

In recent years, recommender systems have gained significant importance across entertainment apps, e-commerce platforms, and other service domains [27, 30, 47]. These systems play a crucial role in offering personalized recommendations to users, leveraging their preferences, historical interactions, and behavioral data [24, 30]. The impact of these systems extends to enhancing customer satisfaction, driving subscription rates, and fueling business growth in diverse sectors like online retail (e.g., Amazon) and streaming services (e.g., Netflix) [24, 30]. However, it is worth noting that many existing recommendation systems predominantly focus on predicting users' next action based on the sequential patterns observed in their past behaviors, while largely neglecting the consideration of their further future behaviors [24, 17, 18, 19, 27, 47, 54]. This prompts the need for novel approaches that can account for both the historical sequential patterns and the potential sequential future behaviors of users.

By incorporating the sequential patterns of user behavior, an improvement can be achieved in the accuracy and timeliness of recommendations, particularly in contexts, such as e-commerce, where users often make purchases in a sequential manner [55, 24, 56, 54]. A key objective within the field of sequential recommender systems is to predict the ordered sequence of users' future interactions, as it holds immense potential in enhancing the quality of recommendations provided to users. For instance, observing that users who purchase phones often eventually acquire related accessories provides insights into their potential subsequent behaviors [24]. Gaining insights into users' future interactions and their sequential structure can significantly enhance offline recommendations, optimizing system performance and reducing the need for frequent model retraining and deployment [33].

In the context of sequential recommender systems, various approaches have been explored, ranging from simple Markov Chain models to advanced Transformer-based methods [4, 56, 24]. Notably, many of these methods mainly focus on predicting a user's immediate next behavior [47, 27, 54, 19, 24]. While effective in some scenarios, this approach might fall short in capturing the evolving preferences of users over longer time spans, and can demand significant computational resources due to frequent model retraining and deployment [19, 33, 24]. To address these limitations, Pancha et al. proposed a novel approach in their Pinnerformer paper for predicting long-term user engagement [33]. However, their approach was designed and evaluated within a specific data environment, which may limit its performance to other contexts [33]. Moreover, while their method captures long-term user engagement, it doesn't explicitly consider the sequential order of future interactions, which is a crucial aspect in understanding user behavior [33].

1.1 Research Goal

Building upon the insights from the Pinnerformer paper, our research aims to investigate the potential benefits of employing window-based training objectives in the context of sequential recommender systems. Our research takes a novel approach by not only predicting the next user interactions but also considering the order of all interactions within a specific K-interaction window. We also experiment with varying window sizes to understand their impact on the model's performance. To assess the ordering of the predictions, we employ the Kendall's Tau metric, a robust measure that evaluates the correspondence between predicted sequence and true ordered sequence of interactions within the K-interaction future window. In addition, we propose innovative training objectives, such as Integrated All Action prediction, and employ training techniques including auto-regressive to enhance the training process. While the Pinnerformer has provided valuable insights, it is important to note that its findings were limited to a specific data environment. To ensure a comprehensive understanding of these techniques, we recognize the need for evaluation in common data environments. Therefore, we conduct our experiments on the MovieLens 1M dataset, which offers a relatively dense representation compared to commonly used benchmark datasets for recommender system performance evaluation [19, 10]. However, it is important to acknowledge that the MovieLens dataset still exhibits sparsity when compared to the dataset specifically designed for evaluating Pinnerformer-based approaches [33].
In light of this, our research attempts to answer the following research questions:

[RQ1] *Can window-based training objectives improve the accuracy of predicting future user interactions and generate well-ordered future predictions within a given window in sequential recommender systems?*
[RQ2] *How do specific training techniques and temporal embeddings impact the accuracy and ordering of future predictions in sequential recommender systems?*
[RQ3] *How does the choice of window size impact the performance of the model in sequential recommender systems?*

Our findings highlight that our proposed Integrated All Action Prediction model, when combined with the Sampled Softmax Uniform loss, stands as a robust strategy for future user interaction predictions. This hybrid model outperforms others in our experiments, demonstrating superior performance in the $NDCG_{avg}@10$ and $Hit_{avg}@10$ (averaged over a future window size K), as well as the Kendall's Tau metrics, compared to a next-item training objective. Furthermore, among the various training techniques we explored, the Teacher Forcing technique exhibits potential. Notably, the hybrid model together with Teacher Forcing, outperforms all other models in the Kendall's Tau metric, suggesting it as a particularly promising direction for future research in predicting well-ordered sequences of future items. Lastly, our experiments reveal that expanding the future prediction window size can offer long-term benefits in recommendation accuracy, despite minor trade-offs in the immediate prediction performance. These insights collectively provide valuable guidance for future developments in the realm of sequential recommendation systems.

1.2 Report Structure

The report is structured as follows: Chapter 2 presents the background on traditional and deep learning-based recommendation systems. Chapter 3 details our methodology, including our model's architecture and learning approach, and the evaluation process. Chapter 4 discusses the experiments conducted and their results, addressing each research question. Finally, Chapter 5 concludes the report, summarizing the findings, limitations, and potential future work.

Chapter 2

Background

Recommendation systems have become a cornerstone of personalized services in the digital age, and their significance has been extensively studied in the literature [30, 6]. These systems aim to provide personalized recommendations by analyzing and understanding individual behaviors and preferences [30, 24].

Historically, recommendation systems primarily relied on either collaborative filtering or content-based approaches [56, 6, 42, 24]. Collaborative filtering operates on the principle of using the collective preferences or behaviors of a group of users to suggest items or make predictions for individual users [37, 42, 24]. Yet, as user preferences evolve over time, recommender systems need regular retraining to keep up with these changes [37, 42, 24]. On the other hand, content-based filtering recommends items similar to those an individual has shown interest in the past, utilizing both explicit (e.g., purchases) and implicit (e.g., clicks) interactions [42, 45, 24]. A common assumption in these systems is that all interactions have equal significance in understanding user preferences [6, 24]. However, in real-world scenarios, users' future behaviors are influenced not only by their long-term preferences but also by their current intentions [6, 56, 24].

Recognizing these challenges, modern research has pivoted towards sequential recommender systems. These systems draw on both long-term and short-term changes in user behavior, reflecting the dynamic nature of user preferences [30, 27, 33, 47, 6, 24]. By integrating temporal and sequential data, they delve deeper into user behavior patterns, leading to more precise and reliable recommendations [6, 24].

2.1 Traditional Approaches

In the realm of recommendation systems, certain methods have been foundational in shaping the field's early strategies. Two of the most prevalent approaches are collaborative filtering (CF) and Markov Chains (MC) [24]. As aforementioned, CF leverages the aggregated preferences of a group of users to generate recommendations for individual users [43]. To address evolving user behaviors, models are often re-trained to capture new preferences [37, 55, 43]. Bayesian Personalized Ranking (BPR), a commonly used CF method in the literature, employs matrix factorization to understand personalized rankings from user interactions [19, 27, 17, 34, 24]. However, CF methods can struggle to perform adequately due to issues such as data sparsity and the cold-start problem, which pertains to new users with limited data [24, 30, 37].

Markov Chain methods have been introduced as an alternative approach to capture sequential behaviors [30, 55, 4, 24]. Models based on MC can effectively predict the next item in a sequence by modeling short-term interests in sequential patterns using context information, as exemplified by the Fossil model [13, 55, 24]. However, MC models have their own limitations, as they only account for correlations between closely related elements and fail to capture long-term user preferences and temporal changes [55, 35, 24].

2.2 Deep Learning Approaches

Deep learning methods have recently been recognized as potential solutions for overcoming the limitations of traditional methods and enhancing personalization in the field of recommendation systems [47, 27, 30, 18, 19, 17, 33, 6, 24]. Techniques such as Multi-layer Perceptrons (MLPs) have been extensively employed [44], but their use is decreasing in favor of more specialized architectures that are better suited to handle complex patterns and address issues like the vanishing gradient problem [40, 55].

In the literature, Convolutional Neural Networks (CNNs) have also been used to enhance recommendation outcomes by extracting features from a range of data sources, including text, images, and audio [55, 44, 14, 9, 24]. The Caser model, introduced by Jiaxi Tang et al., is an example of a sequential recommendation system that utilizes CNNs to generate low-dimensional embeddings of user-item interactions [41, 24]. Nevertheless, it has been observed that the effectiveness of CNNs can be limited by insufficient data, which restricts their ability to capture significant information [55, 24].

Recurrent Neural Networks (RNNs) have proven to be highly effective in recommendation tasks, mainly due to their ability to handle dynamic series data [55, 5, 48, 58, 24]. In order to accurately capture and weight long-term preferences, variants of RNNs, such as Long Short-Term Memory (LSTM) [58, 57] and Gated Recurrent Units (GRU) [29], have been introduced, which utilize gating mechanisms [55, 24].

Hidasi et al. introduced GRU4Rec, a model that leverages GRU to effectively model the sequences of user-item interactions [17, 24]. The researchers subsequently proposed an enhanced system, GRU4Rec+ [16], which incorporated new loss functions called TOP1-max and BPR-max, along with an improved sampling strategy to increase the performance [56, 24]. Specifically, the authors successfully combined the Top1-max loss, designed to maximize the probability of predicting the correct item, and the BPR-max, aimed at maximizing the ranking of the correct item [17, 16, 24]. However, despite the demonstrated effectiveness of RNN-based models in recommendation tasks, they still carry significant limitations, such as their difficulty in understanding deeper patterns in longer sequences [55, 56, 24].

2.2.1 Attention mechanisms

Attention mechanisms have exhibited their effectiveness in various fields such as machine translation [3] and image captioning [52, 24]. The fundamental concept underlying these mechanisms is the identification of the most relevant components of the input that contribute to the final output [27]. Prior to the advent of the Transformer architecture, LSTM-based systems employed attention to enhance their performance, capturing relevant parts of a sequence without needing to rely on the Transformer's mechanisms [49]. However, the Transformer architecture, introduced by Vaswani et al., eliminated the need for sequential processing, allowing for better handling of long-term dependencies and parallel processing of data [46]. Attention mechanisms have since been incorporated into sequential recommender systems to discern the most relevant items based on users' historical behavior [27, 30, 55, 33, 24].

The Self-Attentive Sequential Recommendation (**SASRec**) model proposed by Kang et al., leverages the Transformer architecture and highlights the potential of attention mechanisms in sequential recommender systems [19]. This model, which is often utilized as an advanced baseline in the field's recent studies [27, 55, 33], tries to predict the next interactions of users, given their action sequences [19]. Their strategy with self-attention mechanism allows the model to balance users' short-term intentions and long-term preferences, thereby enhancing the model's ability to generate recommendations [19, 6, 24].

Another sequential recommendation model that has gain attention in the field is BERT4Rec [38]. The model also leverages the transformer architecture to elevate its ability to make accurate recommendations [24]. It utilizes a bidirectional model trained to handle sequential data using the **Cloze task** [38, 24]. The Cloze task is a technique where certain items in a sequence are masked, and the model is trained to predict these masked items, thereby improving the model's overall performance [38, 24].

Li et al. in TiSASRec study, improved upon SASRec by integrating time intervals between sequential items [27, 6, 28, 24]. Without this integration, all items would be treated as equidistant, an approach not suitable for recommendation systems where the timing of interactions can provide significant information [27, 6, 28, 24]. TiSASRec creates a relational matrix for items, predicated on the time intervals between each pair of items in a user's historical records [27, 6, 24]. Nevertheless, TiSASRec has limitations in comprehending variations between timestamps in different contexts and disregards the consistent patterns in similar timestamps [27, 56, 24]. To address this, Zhang et al. proposed TAT4SRec [56]. TAT4SRec adopts an encoder-decoder model that separates timestamps and interacted items to enhance temporal information processing [56]. Additionally, TAT4SRec utilizes two embedding modules, including a window-based function to maintain continuous dependencies in similar timestamps [56, 24]. To overcome the computational restrictions of RNN-based systems, Sun et al. [39] utilized a transformer model in their research to accelerate training time and identify the relationship between items irrespective of their distance [6, 24, 39]. Thereafter, Zhang et al. developed a Time-Aware Long- and Short-term Attention Network (TLSAN) model that outperformed several leading baselines [55]. The model incorporates personalized time-aggregation to encapsulate users' individual long-term habits and employs attention mechanisms to capture short-term behaviors of users [55, 24].

2.2.2 Window-based Predictor

Pinnerformer was proposed by Pancha et al., aiming to tackle the issue of handling changeable sequence interactions, and deploying large models in production [33, 24]. Building on their prior work in the Pinnersage study[32], the authors presented a novel window-based prediction approach. This research aligns closely with our goals, especially the aim to train models for predicting future user interactions over a K-sized window. Pancha et al. adopted a 28-day training window in their methodology, which proved optimal even when evaluation was conducted over a 14-day window [33]. They enhanced pin representations by incorporating time features using a combination of sine and cosine transformations with a fixed P periods, drawing inspiration from the Time2Vec approach [20]. Additionally, a logarithmic transformation of time was applied [33]. This results in $2P+1$ features, which are concatenated into a singular vector to create an input vector of dimension D_{in}, representing the comprehensive action's representation [33].

To optimize their representation, Pancha et al. utilized pairs of user embeddings and target Pin embeddings [33]. Their strategy revolved around [33]:

1. Selecting these pairs.

2. Determining negative examples for a given user-Pin pair.

3. Calculating the loss for each pair and its associated negative examples.

For the negative selection, two primary sources emerge:
In-batch negatives: Here, all positive examples within a batch are treated as potential negatives [33]. However, this method has the risk of potentially demoting popular Pins and may not reflect the true underlying distribution of Pins used during retrieval [33].
Random negatives: These are drawn uniformly from the entire corpus of Pins [33]. This method, if used alone, can lead to model collapse due to the simplicity of the negatives [33]. To harness the strengths of both methods, the authors merge both in-batch and random negatives, resulting in a balanced pool of negatives [33].

When it comes to the loss function, the authors employed a **Sampled Softmax** loss function (equation 3.9), which computes a weighted average loss for each user-positive embedding pair [24, 33]. For situations where negatives are not uniformly distributed, the LogQ correction term is utilized to account for the probability of a negative example surfacing in the batch, as shown in equation 3.10 in Chapter 3 [33].

The model relies on a transformer architecture to generate a sequence of user representations [33, 24]. Building upon the next item prediction strategy of SASRec, Pancha et al. introduced two new training objectives [33]. The first one called **All Action Prediction**, leverages the final user embedding to predict all positive engagements within the K-day window, as represented in Figure 2.1[33]. The objective of this approach is to force the model to learn long-term interests [33]. For this method, they randomly select 32 actions per user within their K day time window [33].

To further enhance this, they develop another training objective called **Dense All**

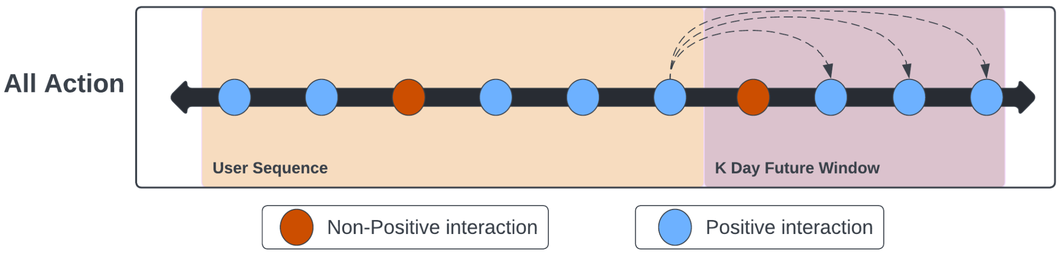

Figure 2.1: All Action Prediction training objective used in Pinnerformer [33]. The red circles indicate embeddings tied to actions considered non-positive, while the blue circles denote embeddings corresponding to actions deemed positive. From Pinnerformer paper, Pancha et al. [33].

Action Prediction [33]. Instead of using only the final embedding as in their all action prediction, they use both the final and intermediate user embeddings to predict future interactions within the K time window, as depicted in Figure 2.2 [33]. More specifically, in this approach, for each user embedding corresponding to a random chosen set of items, they try to predict a random positive action from all the future positive actions [33]. They only predict single positive actions with each embedding to mange the memory usage and model efficiency [33]. This objective enhancement proved to significantly improve the model's performance as the authors state [33].

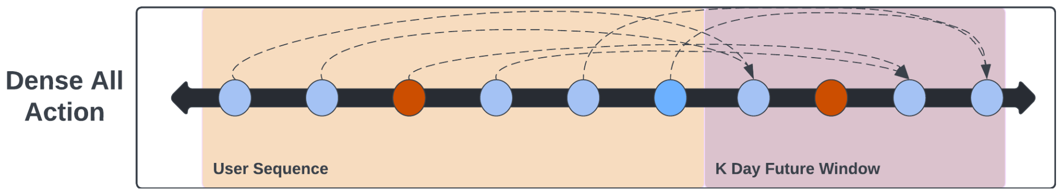

Figure 2.2: Dense All Action Prediction training objective used in Pinnerformer [33]. The figure illustrates just one example of manifestation as the connection pairings are randomly sampled. From Pinnerformer paper, Pancha et al. [33].

Chapter 3

Methodology

3.1 Problem Statement

In the domain of sequential recommendation, we can define $U = (u_1, u_2, ..., u_{|U|})$ as the collection of all users and $I = (i_1, i_2, ..., i_{|I|})$ as a compilation of items. We can express $S^u = (S^u_1, S^u_2, ..., S^u_{|S^u|})$ to represent the chronologically ordered sequence of actions by user $u \in U$, where S^u_i signifies the item that user u has interacted with at the i-th time step. The fixed length of the interaction sequence for user u is indicated by n. For each user, we consider the last n actions they performed. If a user has fewer than n actions, we pad the sequence, and if they have more than n, we truncate it. Given the interaction sequence S^u up to the current time step, the goal in sequential recommendation is to predict the user's next K interactions. As illustrated in figure 3.1, the task involves feeding the user sequence $S^u = (S^u_1, S^u_2, ..., S^u_{|S^u|-1})$ into the model, which then aims to produce a shifted sequence $S^u = (S^u_2, S^u_3, ..., S^u_{|S^u|})$. This shifted sequence provides predictions for the next interactions at each time step [19].

3.2 Model Architecture

The SASRec model's significant relevance and efficacy in sequential recommendation tasks have not only positioned it as a cornerstone model in our research but also influenced our choice of model architecture.

The SASRec model processes the training sequence $S^u = (S^u_1, S^u_2, ..., S^u_{n-1})$ into sequence $s = (s_1, s_2, ..., s_n)$ of fixed-length n [19]. The maximum sequence length that the model can handle is indicated by the sequence length n [19]. In alignment with the approach taken by Kang et al., we too consider the most recent n sequences if the

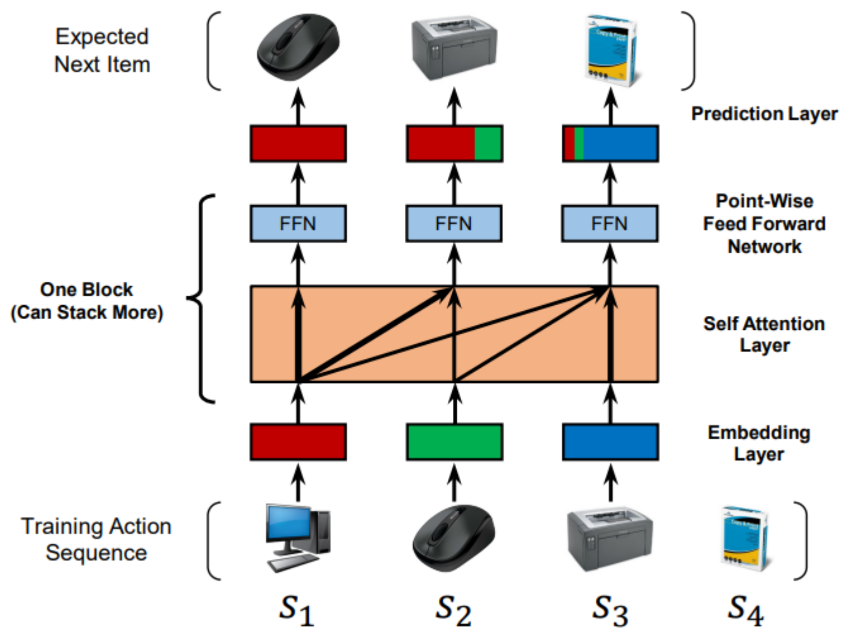

Figure 3.1: Training process of SASRec, the model takes into account all prior interactions at each time step and employs attention to understand items important to the subsequent action. From SASRec paper, Kang et al. [19].

sequences are longer, or pad the sequences with additional items on the left if they are shorter [19]. Like SASRec, our model utilizes an item embedding matrix $M \in R^{|I|xd}$, where d denotes the latent dimensionality [19]. Our model also generates an input embedding matrix $E \in R^{nxd}$ [19]. Here, s_i is an index used to fetch the corresponding embedding from matrix M, such that each E_i is equivalent to M_{s_i} [19]. Mirroring the original SASRec model, we incorporate a position embedding $P \in R^{nxd}$ into the input embedding to address the self-attention model's inherent lack of position awareness, resulting in an enhanced input embedding E_b, as follows:

$$E_b = \left(\sum_{i=1}^{n} M_{s_i} + P_i \right) \tag{3.1}$$

In our model, we've also sought to incorporate temporal information to enhance the learning process, much like the approach taken by Pinnerformer [33]. Specifically, we apply the **Time2Vec** (T2V) approach, which is designed to allow the model to autonomously learn time intervals that are of relevance to the task at hand [20, 33]. Unlike the Pinnerformer model, which uses fixed periods P, our approach allows the model to learn these periods itself [33]. This results in a more flexible representation, potentially capturing a wider variety of temporal patterns. Specifically, given a timestamp t, we obtain features using the sine and cosine transformation, as follows:

$$r(t)_{2i-1} = \cos\left(\frac{2\pi t}{p_i} + \phi_{2i-1}\right)$$

$$r(t)_{2i} = \sin\left(\frac{2\pi t}{p_i} + \phi_{2i}\right), \quad i = 1,\dots,P$$

$$r(t)_{2P+1} = \log(t)$$

where p_i are the periods and ϕ is a learned vector [33]. This transformation yields $2P+1$ features similarly to Pinnerformer [33]. After encoding the temporal features, we concatenate them with the original input vector, resulting in a longer input vector [33]. To match the dimensionality, we apply a multi-layer perceptron (MLP) to the concatenated vector. However, in our study, we find that adding temporal information (Time2Vec) to train the model doesn't enhance performance. This could be attributed to the structure of the tested dataset or the potential for overfitting due to the introduction of more parameters.

Our architecture, following the authors of the SASRec model, features a **self-attention block** that employs the scaled dot-product attention mechanism, defined as [19, 46, 3]:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3.2}$$

In the formula, Q represents queries, K keys, and V values, where each row signifies an item [19, 46]. The authors state that the attention layer, intuitively, computes a weighted total of all values, where the weight between query i and value j corresponds to the interaction between query i and key j [19]. Similar to Kang et al., we employ self-attention operations that use linear projections to convert the enhanced input embedding into three matrices [19]:

$$S = \text{SA}(E_b) = \text{Attention}(E_b W_b^Q, E_b W_b^K, E_b W_b^V) \tag{3.3}$$

Moreover, to maintain causality, which is a key feature of the SASRec model, we ensure that when predicting the $(t+1)$-th item, only the first t items are considered [19]. We implement this by blocking all links between Q_i and K_j, where $j > i$ [19]. Following the SASRec model's design, we also employ a **point-wise two-layer feed-forward network** to all S_i identically, ensuring there's no interaction between S_i and S_j where $i \neq j$ [19]:

$$F_i = \text{FFN}(S_i) = \text{ReLU}(S_i W^{(1)} + b^{(1)})W^{(2)} + b^{(2)} \tag{3.4}$$

Furthermore, our architecture, inspired by SASRec, uses **stacked self-attention blocks** to capture complex item transitions [19]. Each block outputs a hierarchically processed version of the input, which feeds into the next block [19, 53]. To counterbalance the potential drawbacks of deep networks, we implement strategies such as residual connections, layer normalization, and dropout. **Residual connections** allow the output of one layer to be added to the output of a layer several steps further into the network [11]. This helps propagate features from lower layers to to higher, effectively capturing important sequential dynamics [11, 19]. **Layer normalization** and **dropout** are used to stabilize training and prevent overfitting [2, 19, 36].

The **output layer** uses the final self-attention block's outputs, combined with an item embedding matrix N, to estimate the relevance of the item i given the t items [19]:

$$r_{i,t} = F_t^{(b)} N_i^T, \tag{3.5}$$

Finally, to make the model smaller and more resistant to overfitting, we employ as Kang et al., a **shared item embedding** M [19]:

$$r_{i,t} = F_t^{(b)} M_i^T, \tag{3.6}$$

3.3 Model Learning

In this section, we delve into the core elements of our learning process. We start by discussing the methods employed for data partitioning, which essentially set the stage for our model training. Subsequently, we shed light on the different loss functions used to measure the accuracy of our model and to direct the learning process. Moving forward, we present our various training objectives, each representing a distinct approach towards predicting item interactions. Lastly, we outline the different techniques adopted to enhance further the learning process.

3.3.1 Data Partitioning

Various data partitioning techniques, such as temporal split, random split, and user-based split, are prevalent in the domain of sequential recommendation research [31]. The choice of a particular partitioning strategy often depends on the dataset's nature, the model's requirements, and the specific objectives of the study [31]. Our data partitioning techniques, outlined below, stem from existing methodologies used in sequential recommendation research, particularly those detailed in [19, 27, 13, 12, 35]. We opted for these approaches based on their proven efficacy in prior works, their alignment with our research goals, and the nature of our dataset. While our dataset does contain timestamps, the temporal windows for users are relatively narrow, typically occurring within brief periods. This sparsity in temporal information precluded an effective temporal split. As a result, we chose to focus on predicting future interactions, rather than attempting a multi-day prediction as in Pinnerformer [33]. As part of our preprocessing steps, we do not use users and items associated with less than five interactions, ensuring sufficient interaction data for meaningful pattern extraction [19]. We consider each user-item interaction as an event in a sequence, ordered according to their timestamps. From this cleaned and ordered data, we implement two distinct partitioning strategies, each tailored to a unique set of training objectives. These strategies guide our model learning process and establish the trajectory of our research.

3.3.1.1 Next Item Prediction Partitioning

The first partitioning approach, geared towards the next item prediction objective, adopts a traditional train-validation-test split on a per-user basis. In essence, for each user, we chronologically divide their interactions in an 80-10-10 ratio as shown in Figure 3.2. Specifically, 80% of their interactions are designated for training, 10% for validation, and the remaining 10% (which constitutes the most recent interactions) for testing. This follows the methodology outlined in the original SASRec paper [19], with the exception that our validation and test sets are window-based, enabling further future interaction predictions.

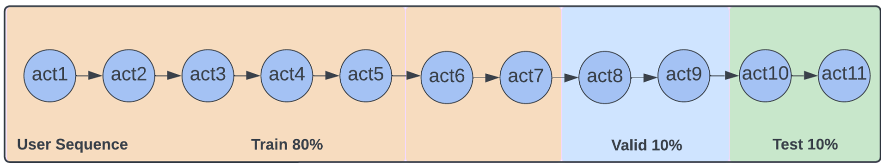

Figure 3.2: Data Partitioning for Next Item Prediction.

3.3.1.2 Window-based Training Partitioning

Building upon the traditional user-based partitioning, this alteration caters specifically to our window-based training objectives. As depicted in Figure 3.3, we initially divide the data into three sets: Train, Valid, and Test. However, the Train set is further split into Input Train and Target Train subsets. The Input Train subset is used to train the model, while the Target Train subset is used as the target output for the model during the window-based training. This partitioning scheme allows our model to learn from a specific window of sequences (Input Train) and predict the subsequent window of sequences (Target Train).

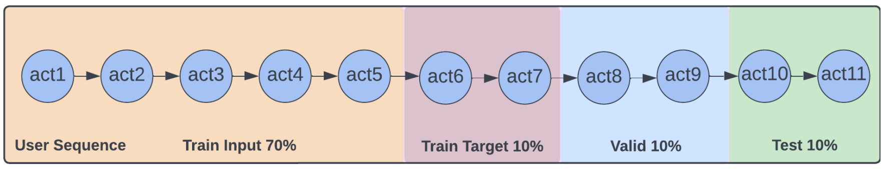

Figure 3.3: Data Partitioning for Window-based Prediction.

3.3.1.3 Incorporation of Time2Vec

In scenarios where we incorporate Time2Vec into our model, we ensure our partitioning strategy continues to uphold the chronological integrity of interactions [20]. The timestamps associated with each interaction are also included in our input sequence now, forming user-item-timestamp triplets, which allows the model to preserve the temporal dynamics.

3.3.2 Loss Functions

For our study, we employ four types of loss functions for training the model.

1. Binary Cross Entropy (BCE): This loss function is utilized for the next item prediction objective, as illustrated in the original SASRec paper [19]. Positive examples stem from actual user interactions. For negative sampling, in each epoch, one negative item (denoted as j) is randomly generated for each timestep in each sequence [19]. We also employ this function later in our masking training technique (Section 3.3.4.1) to calculate the loss of the masking sequence. The formula is given as:

$$L_{BCE} = -\sum_{Su \in S} \sum_{t=1,2,\ldots,n} \left[\log(\sigma(r_{o,t+k})) + \sum_{j \notin S^u} \log(1 - \sigma(r_{j,t}) \right] \tag{3.7}$$

2. Window-Based Binary Cross Entropy (BCE_{WB}): This is a slightly modified version of BCE with the only difference being that it calculates the loss at every future position up to K and then averages over the window. The formula for this loss function is:

$$L_{BCE_{WB}} = -\frac{1}{K} \sum_{k=1}^{K} \sum_{Su \in S} \sum_{t=1,2,\ldots,n} \left[\log(\sigma(r_{o,t+k})) + \sum_{j \notin S^u} \log(1 - \sigma(r_{j,t})) \right] \tag{3.8}$$

3. Sampled Softmax Uniform (SS-U): This loss function, used by Pinnerformer [33], computes a weighted average loss for each user-interaction embedding pair. The function is defined as:

$$L(u_i, p_i) = -\log \frac{e^{s(u_i,p_i)}}{e^{s(u_i,p_i)} + \sum_{j=1}^{N} e^{s(u_i,n_j)}} \tag{3.9}$$

where u_i is the user, p_i is the item the user will interact with, n_j are the negative sampled items (items that the user will not interact with) uniformly distributed, $s(u_i, p_i)$ is the predicted score of user interacting with the item, and N is the number of negative samples [33].

4. Sampled Softmax with LogQ (SS-LogQ): This variant of Sampled Softmax, also used by Pinnerformer [33], includes a correction term $log(Q_i)$ to account for the probability of a negative example appearing in the batch . It is defined as:

$$L(u_i, p_i) = -\log \frac{e^{s(u_i,u_i)} - \log(Q_i(p_i))}{e^{s(u_i,u_i)} - \log(Q_i(p_i)) + \sum_{j=1}^{N} e^{s(u_i,n_j)} - \log(Q_i(n_j))} \tag{3.10}$$

where $(Q_i(p_i)$ and $(Q_i(n_j)$ are the sampling probabilities for actual interaction with item and negative items respectively [33].
In addition to our loss functions, our model is optimized by *Adam* optimizer [22].

3.3.3 Training Objectives

Our primary task centers around predicting the future interactions a user will undertake. The ”training objectives” define precisely what the model aims to achieve or optimize during its learning phase. In the following sub-sections, we present the specific training objectives we’ve designed to steer our model’s learning. Each objective represents a different perspective on how the model should ideally predict future user-item interactions, providing a variety of approaches for model training and evaluation.

3.3.3.1 Next Item Prediction

Our initial training objective mirrors the conventional next item prediction method, akin to the SASRec model [19]. The aim is to predict the next interaction at each step across the entire training sequence. We employ our first data partitioning scheme (Figure 3.2) to support this, structuring user-item interactions chronologically for effective training. Notably, during this process, our attention is not limited to a specific K interactions future window but extends to the complete training sequence, as represented in Figure 3.4.

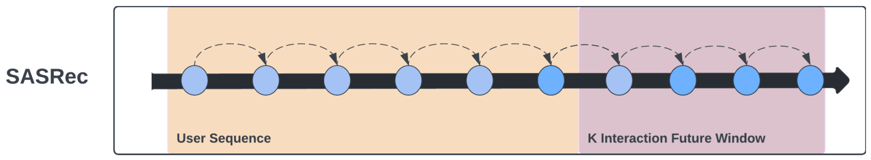

Figure 3.4: Next Item Prediction training. Notably, this objective trains on the whole train sequence without considering the K interaction window.

3.3.3.2 Skip Item Prediction

This training objective is an innovative adaptation of the conventional next item prediction training. This approach, specifically designed to augment our future predictor, employs the future window data partition method, as illustrated in Figure 3.3. The Skip Item Prediction strategy generates multiple unique training sequences for each user by appending individual items, one by one, from the target sequence (K interaction window) to the input sequence. To exemplify, consider a train input sequence for a user as $\{1,2,3,4,5,6\} \in S^u_{train}$ and the corresponding train target as $\{7,8,9\} \in S^u_{train}$. This approach then constructs new sequences for user as follows:

$$\begin{bmatrix} \{1,2,3,4,5,6,\textcolor{red}{7}\} \\ \{1,2,3,4,5,6,\textcolor{red}{8}\} \\ \{1,2,3,4,5,6,\textcolor{red}{9}\} \end{bmatrix} \quad \in S^u_{\text{train}}$$

Here, the highlighted (red) items are those appended from the target sequence. This strategy allows for the creation of new item embeddings during training in a next item prediction fashion. The resultant model is equipped with the capability to capture patterns better, thereby improving the prediction of future items.

3.3.3.3 Incremental Item Prediction

Our third training objective, is yet another innovative enhancement of the baseline next item prediction. The key distinction between this approach and the conventional next-item prediction lies in the construction of training sequences. While traditional next-item prediction uses a single input sequence for each user, the Incremental Item Prediction approach generates multiple sequences for each user, each with a distinct target appended at the end. This strategy employs the future window data partition method (Figure 3.3). The construction of sequences involves incremental appending of target items to the input sequence, echoing the 'teacher forcing' concept in machine learning [26]. For illustration, consider an input sequence for a user as $\{1,2,3,4,5,6\} \in$

S^u_{train} and the corresponding target as $\{7,8,9\} \in S^u_{train}$. This approach results in the creation of new sequences for user as follows:

$$\begin{bmatrix} \{1,2,3,4,5,6,\textcolor{red}{7}\} \\ \{1,2,3,4,5,6,\textcolor{green}{7},\textcolor{red}{8}\} \\ \{1,2,3,4,5,6,\textcolor{green}{7},\textcolor{green}{8},\textcolor{red}{9}\} \end{bmatrix} \in S^u_{\text{train}}$$

In this illustration, the red-highlighted items are those newly appended from the target sequence, while the green-highlighted items are those previously appended. This method enables the generation of unique item embeddings during next item prediction training, potentially leading to more accurate future item predictions.

3.3.3.4 All Action Prediction

For our next training objective, and as the first window-based training approach, we use All Action Prediction, adopted from the Pinnerformer paper [33]. However, we adapt the approach to our specific needs, focusing on predicting all future interactions within a given window, rather than predicting actions within a certain time frame [33]. Given a user input train sequence $\{S^u_1, S^u_2, ..., S^u_{S^u_n}\}$, the final user embedding $e_{s^u_n}$ is used to predict all actions within a given window, $\{S^u_{n+1}, S^u_{n+2}, ..., S^u_{n+K}\}$, where n is the fixed length of the sequence and K is the length of the prediction window [33]. In this context, the 'all actions' refer to the interactions in the target train sequence window (Figure 3.3). With this objective, we train the model to discern and learn longer-term interaction patterns. Due to the dataset's nature, target train sequences often fall short of the predefined K-prediction window. Adapting the Pinnerformer's method, we fill sequences to a consistent length of K, by randomly sampling from the target sequence, aiding the model in grasping longer-term patters [33]. To ensure balance, a 1:1 ratio for positive to negative samples is maintained.

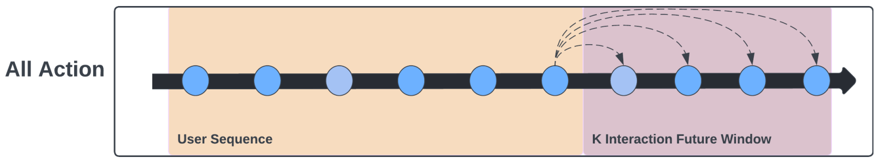

Figure 3.5: All Action Prediction training adopted from Pinnerformer [33].

3.3.3.5 Dense All Action Prediction

The fifth training objective, referred to as Dense All Action Prediction, is an evolution of the All Action Prediction objective. This method is adapted from the one presented in the Pinnerformer paper, with modifications to better suit our interaction-based prediction context [33]. This objective aims to increase the robustness of the prediction signal by utilizing not just the final user embedding, but using all the input train sequence user embeddings $e_{s_i^u}$ [33]. For each of these selected embeddings, we strive to predict a single randomly chosen interaction from the target train sequence window set $\{S^u_{n+1}, S^u_{n+2}, \ldots, S^u_{n+K}\}$. Figure 3.6 represents this training objective.

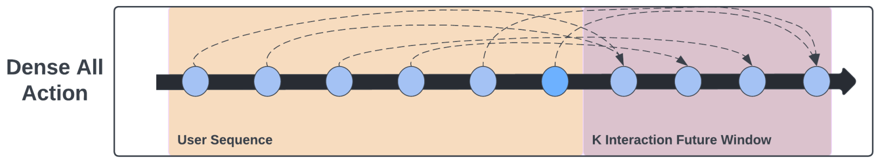

Figure 3.6: Dense All Action Prediction Training adopted from Pinnerformer [33].

3.3.3.6 Super Dense All Action Prediction

Building upon the Dense All Action Prediction method, we introduce a training objective called Super Dense All Action Prediction. This approach aims to further boost the learning signal by harnessing more information from the user's interaction sequence. Under the Super Dense All Prediction objective, for each embedding $e_{s_i^u}$ in the input sequence, we aim to predict not just one, but all positive interactions from the set of future interactions within the target train sequence window $\{S^u_{n+1}, S^u_{n+2}, \ldots, S^u_{n+K}\}$. This training objective pushes the model to learn more granular and complex patters from the user's interaction history, enhancing its ability to predict a broader range of future interactions.

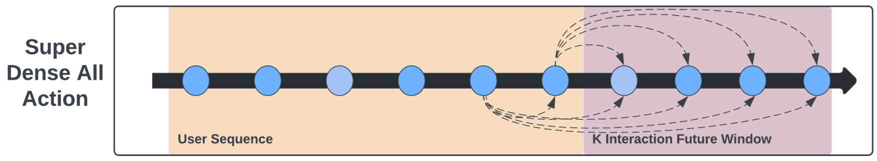

Figure 3.7: Super Dense All Action Prediction training. While only two user embeddings are depicted here for visual clarity, in reality, all embeddings participate in predicting future interactions.

3.3.3.7 Rolling Future Window Prediction

Expanding upon the All Action Prediction method, we introduce a novel training objective, Rolling Future Window Prediction. This approach seeks to enhance the

predictive capability of the model by creating multiple target windows for each position in the training sequence. For a given input training sequence $\{S_1^u, S_2^u, ..., S_n^u\}$, and a target sequence $\{S_{n+1}^u, S_{n+2}^u, ..., S_{n+K}^u\}$, the Rolling Future Window method works as follows:
For each embedding $e_{s_i^u}$ in the training sequence, the model predicts a window of K items starting from S_{i+1}^u to S_{i+K}^u. The last embedding $e_{s_n^u}$ in the sequence extents into the target sequence as All Action Prediction does (Figure 3.5). This approach not only

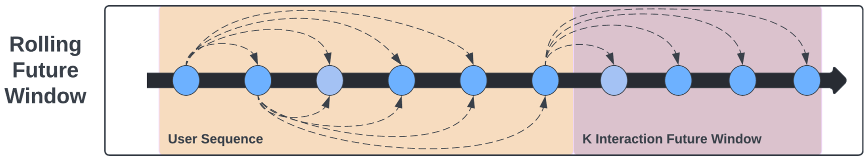

Figure 3.8: Rolling Future Window Prediction training. While only three user input embeddings are depicted here for visual clarity, in actuality, all item embeddings predict future interactions from their position.

allows the model to capture the continuity in the user's sequence of interactions but also recognizes the potential impact of individual interactions on shaping the user's future interactions.

3.3.3.8 Integrated All Action Prediction

Our final training objective, named Integrated All Action Prediction, seeks to harmonize the strengths of both "Next Item Prediction" and "All Action Prediction". This approach operates on the premises that combining these two distinct yet complementary training objectives can potentially provide a more comprehensive training signal for the model.

In the "Next Item Prediction", the model is trained to predict the immediate next item in the sequence at every step, which allows the model to capture short-term dependencies and recent user behaviors. However, this strategy might overlook longer-term user preferences and the overall temporal dynamics of user interactions.

On the other hand, the "All Action Prediction" objective, which predicts all future actions within a certain window, focuses more on the user's longer-term interests. Yet, it may not be as sensitive to the most recent user interactions.

By integrating these two objectives in a hybrid way, we aim to capture both recent and longer-term user behaviors, thereby providing a richer and more balanced learning signal. This combined strategy aids the model in learning the dynamics of user interactions over time, leading to potentially more accurate and diverse recommendations. The

Figure 3.9 below, illustrates graphically the integrated objective.

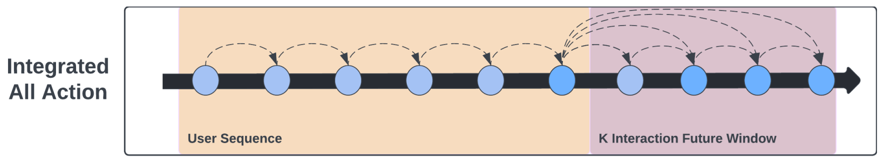

Figure 3.9: Integrated All Action Prediction training.

3.3.4 Training Techniques

While setting the right objectives is fundamental, the methodology adopted to attain those objectives is equally pivotal. The ”training techniques” specify how the model is trained to meet the objectives, determining the training regimen the model undergoes. In our research, we’ve incorporated three techniques: Masking, Auto-regressive learning, and Teacher Forcing. Each of these techniques shapes our model’s learning trajectory in unique ways, enabling it to discern intricate patterns and dependencies in user-item sequences. The following subsections delve deeper into each technique, shedding light on their significance and their roles in augmenting the efficacy of our sequential recommendation model.

3.3.4.1 Masking

The first technique implemented is Masking, drawing inspiration from the masking technique utilized in BERT4Rec by Sun et al. [38]. This technique involves creating masked version of the input training sequence where a certain percentage ρ of items are randomly replaced with a mask token of zero, such as $\{S_1^u, S_2^u, 0, S_4^u, S_5^u\}$.

We apply two types of loss during the training process: a target loss and a masked loss. The target loss is computed normally, predicting the items in the target sequence. In contrast, the masked loss is computed based on the model’s prediction of the masked items. We pass both the original and masked sequences into the model and attempt to predict the masked items. The final loss used to optimize the model is a combination of these two losses, which encourages the model to understand the patterns in the user-item interactions while also predicting the masked items.

$$\boxed{\begin{array}{l}\textbf{Input: } \{S_1^u, S_2^u, S_3^u, S_4^u, S_5^u\} \xrightarrow{\text{randomly mask}} \{S_1^u, 0_1, S_3^u, S_4^u, 0_2\} \\ \textbf{Labels: } 0_1 = S_2^u, 0_2 = S_5^u\end{array}}$$

In this example, the model takes in the input sequence with masked items and aims to predict the original values S_2^u and S_5^u at the masked positions, which will potentially encourage the model to learn deeper patters and help predict future interactions.

3.3.4.2 Auto-Regressive

The Auto-Regressive technique (AR) is another training technique implemented to enhance our future predictor-based training objectives in making future correctly predictions. This method is employed to teach the model to predict the next future item based on the current input sequence and then update the input sequence with the predicted target item.

Let's consider an input sequence $S^u = \{S_1^u, S_2^u, S_3^u, S_4^u, S_5^u\}$ and a corresponding target sequence $S^{uTarg} = \{S_{n+1}^u, S_{n+2}^u, S_{n+3}^u\}$. In auto-regressive training setup, the model first predicts the next target item S_{n+1}^u based on the input sequence S^u [8]. Then, the predicted target item $\hat{S}_{n+1}^u$ is appended to the input sequence, forming a new sequence $S^{u'} = \{S_1^u, S_2^u, S_3^u, S_4^u, S_5^u, \hat{S}_{n+1}^u\}$ which will be the new item embedding fed to the model. The model then generates a new item embedding and predicts the next target item S_{n+2}^u based on this updated sequence $S^{u'}$. This process is repeated for each target item in the sequence. However, since the length of the input sequence embedding is fixed to n, for each appended item, the oldest interaction is removed from the sequence. This ensures that the length of the sequence remains constant while the model is trained to predict subsequent target items based on an increasingly updated sequence of interactions. Especially early in the training process, AR techniques can sometimes lead to an accumulation of errors; if one prediction is incorrect, subsequent predictions can be adversely affected. The auto-regressive update of the input sequence can be expressed as follows:

$$S^{u'} = [S_{n-i+1}^u, S_{n-i+2}^u, \ldots, S_n^u, \hat{S}_{n+1}^u, \ldots, \hat{S}_{n+i}^u] \quad \text{if } \hat{S}_{n+i}^u \text{ is predicted for } i = 1, 2, \ldots, K$$

and the system of equations is defined as:

$$S^{u'} = \begin{cases} [S_2^u, S_3^u, \ldots, S_n^u, \hat{S}_{n+1}^u] & \text{if } S_{n+1}^u \text{ is predicted} \\ [S_3^u, S_4^u, \ldots, S_n^u, \hat{S}_{n+1}^u, \hat{S}_{n+2}^u] & \text{if } S_{n+2}^u \text{ is predicted} \\ \vdots & \vdots \\ [S_{n-K+1}^u, S_{n-K+2}^u, \ldots, S_n^u, \hat{S}_{n+1}^u, \ldots, \hat{S}_{n+K}^u] & \text{if } S_{n+K}^u \text{ is predicted} \end{cases}$$

Where n is the length of the original sequence, K is the length of the target window sequence, and $\hat{S}_{n+i}^u$ is the predicted target item at the i-th step.

3.3.4.3 Teacher Forcing

The Teacher Forcing technique (TF) is a variant of the Auto-Regressive approach, with the key difference being that instead of appending the predicted target item to the input sequence, we append the actual target item [7, 26, 50]. Similar to auto-regressive technique, it can be applied to the future-based training objectives. It can be particularly effective when the model's predictions are consistently incorrect, leading to a cascading effect of errors in an Auto-Regressive setup.

Using the same example as before, with an input sequence $S^u = \{S^u_1, S^u_2, S^u_3, S^u_4, S^u_5\}$ and a corresponding target sequence $S^{uTarg} = \{S^u_{n+1}, S^u_{n+2}, S^u_{n+3}\}$, in Teacher Forcing, the model still predicts the next target item S^u_{n+1} based on the input sequence S^u. However, regardless of the model's prediction, the actual target item S^u_{n+1} is appended to the input sequence for the new item embedding. This process is repeated for each target item in the sequence, while maintaining a fixed sequence length n by removing again the oldest interaction for each appended item. The teacher forcing update of the input sequence is expressed as:

$$S^{u'} = [S^u_{n-i+1}, S^u_{n-i+2}, \ldots, S^u_n, S^u_{n+1}, \ldots, S^u_{n+i}] \quad \text{if } S^u_{n+i} \text{ is predicted for } i = 1, 2, \ldots, K$$

This technique allows for more robust training, as the model is guided by the actual target items during the training process, thereby reducing the accumulation of prediction errors [50].

3.4 Evaluation Process

In this section, we outline the evaluation process that provides insights into the effectiveness of our proposed sequential recommendation system. The performance of our model is assessed using a suite of well-established metrics, each offering unique perspectives on model's predictive capabilities. Specifically, while multiple metrics gauge the model's accuracy in predicting future items, Kendall's Tau is employed to assess its proficiency in structured output prediction, capturing the model's ability to predict not only the forthcoming interactions but also their order. In our evaluation process, it's worth noting that users having fewer than K (window size) items in their validation/test sets are not evaluated. Finally, when evaluating on the test set, a concatenation of train and valid set is used as the user input sequence.

3.4.1 Normalized Discounted Cumulative Gain

The first evaluation metric we adopt is Normalized Discounted Cumulative Gain (NDCG@N), which is a widely-used metric in the field, capturing both accuracy and rank of the recommended items [19, 15]. This metric is particularly relevant to our study as it allows us to evaluate the quality of our model's prediction over the K future window. The NDCG at the i-th future position is calculated based on the top-N recommendation list and then averaged across all users:

$$\text{NDCG@N}_i = \frac{1}{|U|} \sum_{u \in U} NDCG(u) \tag{3.11}$$

where $NDCG(u)$ is the normalized discounted cumulative gain for user u at the i-th future position, and $|U|$ is the total number of users. Subsequently, we compute the average NDCG over the future window size K:

$$\text{NDCG}_{\text{avg}} = \frac{1}{K} \sum_{i=1}^{K} \text{NDCG@N}_i \tag{3.12}$$

Furthermore, we adopt an approach that is followed in studies [19, 15, 23] in order to reduce the computing demands associated with evaluating over all user-items. Specifically, we randomly choose 99 negative items and rank them next to the actual item for each user u at each position i into the future [19]. NDCG then, is evaluated based on 100 items, providing an efficient and robust evaluation of the performance [19].

3.4.2 Hit Rate

Our second evaluation metric is Hit Rate (Hit@N), a simple measure of predictive accuracy [19]. For this metric, we check whether the actual item is within the top-N recommendations [19, 12, 15]. The Hit Rate at the i-th future position is calculated by determining if the actual item is in the top-N recommendations and then averaging these results across all users. Subsequently, the average Hit Rate over the future window size K is computed:

$$\text{Hit@N}_i = \frac{1}{|U|} \sum_{u \in U} hit(u) \qquad \text{Hit}_{\text{avg}} = \frac{1}{K} \sum_{i=1}^{K} \text{Hit@N}_i \tag{3.13}$$

where $hit(u)$ is 1 if the actual item is in the top-N list at the i-th future position, and 0 otherwise. Much like with NDCG, we use the same efficient approach of sampling a set of negative items for the calculation [19].

3.4.3 Kendall's Tau

Finally, our third evaluation metric is Kendall's Tau (τ). This metric is a statistical measure employed to assess how effectively our model orders the future interactions. Specifically, it compares the order of items predicted by our model with the actual order in which users interacted with items in the K future window [21]. The metric ranges from -1 to 1: a value of 1 signifies perfect agreement, -1 indicates complete disagreement, and 0 suggests no correlation [1, 21, 25]. In our study, a higher value indicates a closer match between the predicted and actual order. The essence of this metric lies in its ability to capture the model's proficiency in structured output prediction. In structured output prediction, the model's task isn't just to predict individual items but to predict sequences or structures of items where the order matters.

In our evaluation setup, for each user $u \in U$, the model produces a sequence of predictions for the next K items. We directly compare this predicted sequence with the true sequence of the next K items the user interacted with. Suppose the true future interactions for a user are the items $\{A,B,C\}$, and our model predicts the sequence $\{A,D,C\}$. Kendall's Tau will assess these sequences, providing a score that quantifies the similarity in their order.

Formally, given the true sequence of interactions $true\_sequence$ and the sequence predicted by our model $predicted\_sequence$ for a user u, the Kendall's Tau is given by:

$$\tau_u = \tau(true\_sequence, predicted\_sequence) \tag{3.14}$$

Where τ $(true\_sequence, predicted\_sequence)$ is the standard Kendall's Tau function that computes the metric by comparing all possible pairs of items in the sequences [21]. It evaluates the number of concordant pairs (pairs with the same order in both sequences) and discordant pairs (pairs with different orders) [1, 21, 25]. The final value for τ is then averaged across all users:

$$\text{Kendall's } \tau = \frac{1}{|U|} \sum_{u \in U} \tau_u \tag{3.15}$$

In our implementation, Kendall's Tau is calculated based on the top-1 predictions of the model, making it challenging, yet informative, metric to understand the model's ability to not only predict relevant items but also to predict them in an order that aligns with the actual future interactions of users.

Chapter 4

Experimental Analysis

4.1 Experiments

4.1.1 Dataset

We conduct our experimental analysis on the MovieLens-1M dataset [10]. Compared to alternatives such as the Steam and Amazon datasets, the MovieLens-1M dataset is denser [19]. Given its density and frequent use, it offers a robust benchmark in the field of recommender systems. Despite its dense nature, the MovieLens dataset does exhibit a high degree of sparsity when contrasted with the Pinnerformer dataset specifically designed on evaluating their training objectives [33]. A summary of the dataset statistics is presented in Table 4.1. It's worth mentioning that the user activity in this dataset is mainly concentrated within brief periods, adding a layer of complexity to our model's task of leveraging temporal information for accurate recommendations. Further insights on the dataset are provided through an exploratory data analysis in Appendix A.

#users	**#items**	**#actions**	**avg. actions/ user**	**avg. actions/ item**	**#users w/ single 7-day wind.**	**avg. 7-day wind./user**
6040	3416	1.0M	163.5	289.1	4243	14.4

Table 4.1: Preprocessed MovieLens-1M statistics.

4.1.2 Experimental Methodology

The methodology we follow for out study, is structured around three main experiments, each designed to address one of the three research questions posed in this study.

For **RQ1**, our focus is on evaluating different training objectives, as outlined in Section 3.3.3. Essentially, this experiment investigates how different objectives influence the model's learning when given certain pairs of user-item interactions. The baseline for these comparisons is the Next Item Training, which is mirroring the basic SASRec, but with the capability of making future predictions [19]. In this experiment, we test four different loss functions, namely Sampled Softmax Uniform (SS-U), Sampled Softmax with LogQ correction (SS-LogQ), Binary Cross Entropy (BCE), and Window-based BCE, as detailed in Section 3.3.2. We use a window size K for predictions of 7.

Our second experiment addresses **RQ2**. Here, we take the best-performing window-based prediction objective from the first experiment and set it against the baseline. This experiment delves into evaluating various techniques for creating pairs of user-item interactions. Specifically, we assess the impact of different training techniques detailed in Section 3.3.4 and also explore the influence of encoding temporal information using Time2Vec (T2V) [20]. For this experiment, we use only the BCE/BCE$_{WB}$ and Sampled Softmax Uniform loss functions, since preliminary findings from the first experiment suggest that these functions outperform Sampled Softmax with LogQ. Like in the first experiment our window size K is set to 7. In the cases involving masking, we apply a 15% masking rate to the action sequence.

Finally, to address **RQ3**, we conduct experiments with different future window sizes K. Specifically, we use the model that demonstrated the best performance in the previous experiments, as well as the baseline model, and we apply the SS-U and BCE/BCE$_{WB}$ loss functions. The objective here is to observe how performance varies with window sizes. Specifically, we experiment with window sizes K of $\{3, 7, 10, 14\}$ to evaluate the model's ability to generalize across different prediction horizons.

Each experiment is designed to incrementally build upon the findings of the previous one, allowing us to holistically evaluate the effectiveness of our proposed training objectives and techniques. We use Hit Rate@10, NDCG@10, and Kendall's Tau metrics to capture the performance at inference (see Section 3.4 for metrics information).

4.1.3 Experimental Setup

The experimental setup for this study remains consistent across all experiments to ensure a fair comparison of performance. The implementation details are as follows: We set the maximum length of a user sequence to 200 and use a dropout rate of 0.2 to prevent overfitting. Each model is trained for a total of 200 epochs, with a batch size of 128, as in SASRec and Pinnerformer papers [33, 19]. The learning rate is set to 0.001, which is a commonly used value for a steady optimization process. Following the practice of Kang et al. [19], we employ 2 self-attention blocks in our model architecture. Lastly, we use the Adam optimizer for our training optimization process [22].

4.2 Results

In this section, we present the empirical results of our experiments, systematically addressing each of our research questions. The results are organized according to the three major areas of investigation - performance comparison of different training objectives, the impact of various training techiniques and temporal information encoding, and the influence of different future window sizes on the model's performance.

4.2.1 Performance of Training Objectives

Models	NDCG$_{avg}$@10			Hit$_{avg}$@10			Kendall's Tau		
	BCE	SS-U	SS-LogQ	BCE	SS-U	SS-LogQ	BCE	SS-U	SS-LogQ
Baseline - Next Item	0.5556	0.5557	0.5548	0.8164	0.8055	0.8030	0.2433	0.2610	0.2472
Skip Item	0.5226	0.5609	0.5535	0.7929	0.8115	0.8064	0.2009	0.2487	0.2455
Incremental Item	**0.5582**	0.5660	0.5513	**0.8171**	0.8161	**0.8159**	**0.2525**	0.2634	**0.2590**
All Action	0.4233	0.4329	0.4285	0.6843	0.6908	0.6876	0.1449	0.1564	0.1433
Dense All Action	0.2853	0.3077	0.296	0.5184	0.5478	0.5299	0.0693	0.0763	0.0725
Super Dense All Action	0.2961	0.3055	0.2980	0.5397	0.5468	0.5454	0.0682	0.0641	0.0626
Rolling Future Window	0.2711	0.2827	0.2801	0.5077	0.5274	0.5084	0.0657	0.0695	0.0579
Integrated All Action	0.4958	**0.5669**	**0.5625**	0.7591	**0.8184**	0.8105	0.1914	**0.2641**	0.2558

Table 4.2: Performance scores for different training objectives using BCE (used as BCE_{WB} for window-based objectives), SS-U, and SS-LogQ losses. Bold indicates the best performance model for each metric and loss and underline the second best. Double underline indicates best performance for each metric regardless of the loss.

The first research question investigates the performance of the different training objectives described in Section 3.3.3. The results of this comparison are shown in the table 4.2 above.

The baseline model, which corresponds to the Next Item Training objective, achieves a robust performance across all three loss functions, setting a very strong baseline for other objectives to exceed. Specifically, it achieves its highest NDCG@10 score of 0.5557 when using SS-U loss, and a high Hit@10 of 0.8164 with BCE. Interestingly, it also reached a competitive Kendall's Tau score of 0.2610, suggesting a relatively good level of correlation between true and predicted sequences.

The Skip Item Prediction training objective, though performing lower than the baseline on BCE and SS-LogQ, it's worth noting that it's performance is competitive with SS-U loss, outperforming the baseline on NDCG@10 with 0.5609. Yet, its Kendall's Tau was noticeably lower, suggesting that is not strong in making correctly ordered predictions. It's essential to note that this training objective takes over twice the training time compared to other methods (except Incremental Item Prediction), as expected from creating a much larger number of training sequences for each user.

Meanwhile, the All Action Prediction model, our first window-based predictor, showed a significant drop in performance across all metrics. This could be attributed to the model's broad focus on predicting all future actions, which might dilute the signal for the immediate next items and lead to lower NDCG, Hit rate, and Kendall's Tau. Furthermore, the Dense All Action and Super Dense All Action models fared even lower than the All Action model across all metrics. The complexity of predicting multiple future items for each embedding might have led to this performance drop.

Interestingly, these results contrast the findings in the Pinnerformer paper [33], where the All Action and Dense All Action models performed well. The discrepancy could be attributed to different factors. In the Pinnerformer study, they used already learned embeddings called PinSage, while in our case we learn them during training the model from scratch [32, 51]. Furthermore, their dense specific dataset with many pins, differs significantly from the MovieLens dataset used in this study [33].

Our evolved and more complex all action modeling named Rolling Future Window Prediction, recorded the lowest scores across all metrics in our experiments, showing that it's a poor learning objective and likely too complex for our data environment.

Among the tested training objectives, the Incremental Item Prediction model demonstrated the best overall performance across all metrics when used with BCE loss function, outperforming, but slightly the baseline model. We can also observe that for the Uniform Sampled Softmax, the model outperforms even more the baseline on the metrics NDCG@10 and Kendall's with 0.5660 and 0.2634 respectively. This suggests that the Incremental Item Prediction approach, which trains the model by incorporating more sequences that emphasize future window interactions, indeed contributes to enhancing the model's ability to generate more accurate and better-ordered interactions. However, this method is more resource-intensive than others, including Skip Item Prediction, due to the longer training times from both the increased number and length of training sequences.

Finally, the Integrated All Action model showcased a very promising performance, especially with the SS-U loss, where it achieved the highest NDCG@10 of 0.5669, the highest Hit@10 of 0.8184, and the highest Kendall's Tau score of 0.2641 among all models and losses tested. This demonstrates the effectiveness of an integrated approach that can harness the strengths of both Next Item and All Action modellings. By employing this hybrid model, we are not only able to learn better item representations but also gain insights into the user's future intentions, offering a more comprehensive understanding of user behaviour. This suggests that a combination of short-term and long-term prediction strategies can indeed enhance the accuracy and ordering of our recommendations. While the Integrated All Action model outperforms the baseline in SS-U and achieves the best overall performance, it's worth noting that its performance is close to that of the Incremental Item Prediction, but less resource-intensive.

4.2.2 Effects of Training Techniques and Temporal Information

Models	NDCG$_{avg}$@10		Hit$_{avg}$@10		Kendall's Tau	
	BCE	SS-U	BCE	SS-U	BCE	SS-U
Baseline - Next Item	**0.5556**	0.5557	**0.8164**	0.8055	**0.2433**	0.2610
Integrated All Action	0.4958	**0.5669**	0.7591	**0.8184**	0.1914	<u>0.2641</u>
Integrated All Action + T2V	0.0876	0.1210	0.1859	0.2510	0.0239	0.0006
Integrated All Action + MASK	0.4913	0.5429	0.7577	0.7947	0.1916	0.2578
Integrated All Action + AR	0.4928	0.5556	0.7609	0.8072	0.1806	0.2551
Integrated All Action + TF	<u>0.5183</u>	<u>0.5626</u>	<u>0.7800</u>	<u>0.8096</u>	<u>0.2108</u>	**0.2669**
Integrated All Action + MASK + TF	0.5024	0.5469	0.7727	0.7990	0.2102	0.2621

Table 4.3: Performance scores for different training techniques and temporal encoding, using BCE (used as BCE_{WB} for window-based objectives) and SS-U losses.

Having analyzed the performance of various training objectives in addressing RQ1, we now turn our focus to **RQ2** where we aim to evaluate the impact of different training techniques and temporal encoding on our model's performance to see if it actually improves in giving more accurate results. For this experiment, we select the best performing window-based prediction objective which is the Integrated All Action Prediction model from RQ1, and explore it under different setups. The results of this experiment are illustrated on Table 4.3.

First, we start with incorporating Time2Vec into the Integrated All Action model. As we can see, this led to a considerable drop in performance across all metrics and losses. This decrease in performance may be attributed to a combination of overfitting, due to the introduction of more parameters from Time2Vec, and the inherent nature of the MovieLens-1M dataset. In the latter case, user interactions predominantly occur within short timeframes, limiting the potential to extract meaningful temporal patterns. This was expected after carrying out a dataset analysis as can be seen on table 4.1 and Appendix A.

Furthermore, applying Masking to the Integrated All Action model showed a slight drop in performance for both losses, but the model still able to maintain a relatively high level of performance. This suggests that introducing an auxiliary task to predict masked items has probably added more complexity to the learning, and therefore has not significantly enhanced the model's capabilities in this context.

Introducing Auto-Regressive (AR) training to our best modelling approach, resulted again, in a slight decreased in the metrics for SS-U. However, for the BCE loss, the performance stayed relatively consistent with a slight increase in Hit@10 compared to the normal Integrated All Action. That being said, it's still clear that the baseline model with BCE outperforms. This indicates, that the AR approach, which iteratively updates the input sequence with predicted items is not leading in capturing better dependencies between behaviors as desired.

Another tested experiment, was applying Teacher Forcing (TF) to our model. With a first view, the technique had mixed effects on the results. While it led to an increase over normal Integrated All Action in all the metrics for BCE loss, it causes a slight decrease for NDCG@10 and Hit@10 metrics in SS-U. Notably, the TF technique demonstrated the highest overall Kendall's Tau score of 0.2669 across all metrics and losses, signifying its potential as a promising approach to enhance the model's ability to accurately predict the order of future items. Moreover, the results show that the teacher forcing technique has the most stable performance when changing between the two losses as it gets second and first place performance across all tasks.

Lastly, we wanted to test the model's recommendation performance using the TF technique, since it showed the most promising results, along with our masking technique.

Combining those two approaches with the integrated modeling, led to a slight drop in performance across all metrics and losses compared to baseline and simple integrated model approach, showing that it does not help significantly with the learning task.

The results captured, highlight the complexity of sequential recommendation tasks. Despite the various techniques evaluated, the base Integrated All Action objective with Sampled Softmax Uniform loss remained competitive, achieving the highest scores in NDCG@10 and Hit@10 metrics, while the Integrated All Action objective with the Teacher Forcing technique excelled slightly in Kendall's Tau.

4.2.3 Impact of Different Window Sizes

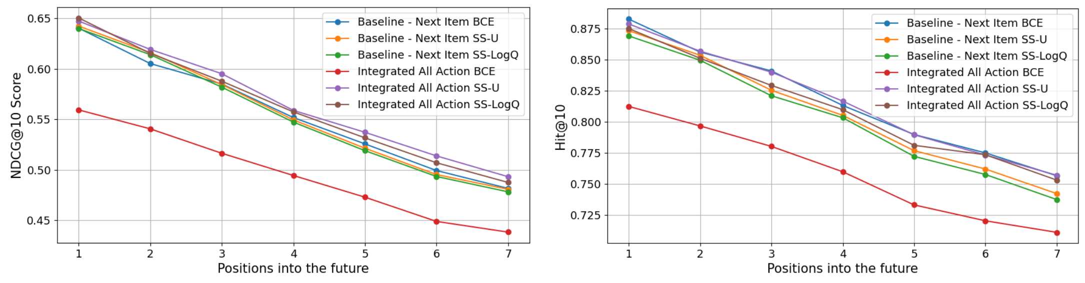

Figure 4.1: NDCG@10 and HR@10 score with window 7 for Baseline Next Item Training and Integrated All Action Prediction Models using Sampled Softmax Uniform, Sampled Softmax with LogQ Correction, and Binary Cross Entropy losses.

In this section, we delve into the results obtained for our third research question (**RQ3**), which investigates how varying the future window size impacts the performance of our model. Figures 4.1 and 4.2, present a graphical representation of the trends as these sizes change. The table recording the average results, along with the tables containing the data used to generate the graphs, are stated in Appendix B.

Firstly, we examine the results for a window size of 7, which was also tested in the prior experiments. Thus, this includes SS-LogQ loss results as well. Here, the Integrated model with BCE underperforms across the metrics. The Integrated All Action model shines in terms of NDCG@10, showing superior performance across most future positions, closely followed by the Integrated model with SS-LogQ. Other configurations yield marginally lower results. In terms of Hit@10, The Integrated model with SS-U is leading, closely trailed by the Baseline with BCE. Then, the integrated with SS-LogQ takes third place, and the remaining models achieving lower results. These results, as seen in the previous experiments, set the Integrated All Action model with SS-U to be effective for larger window sizes.

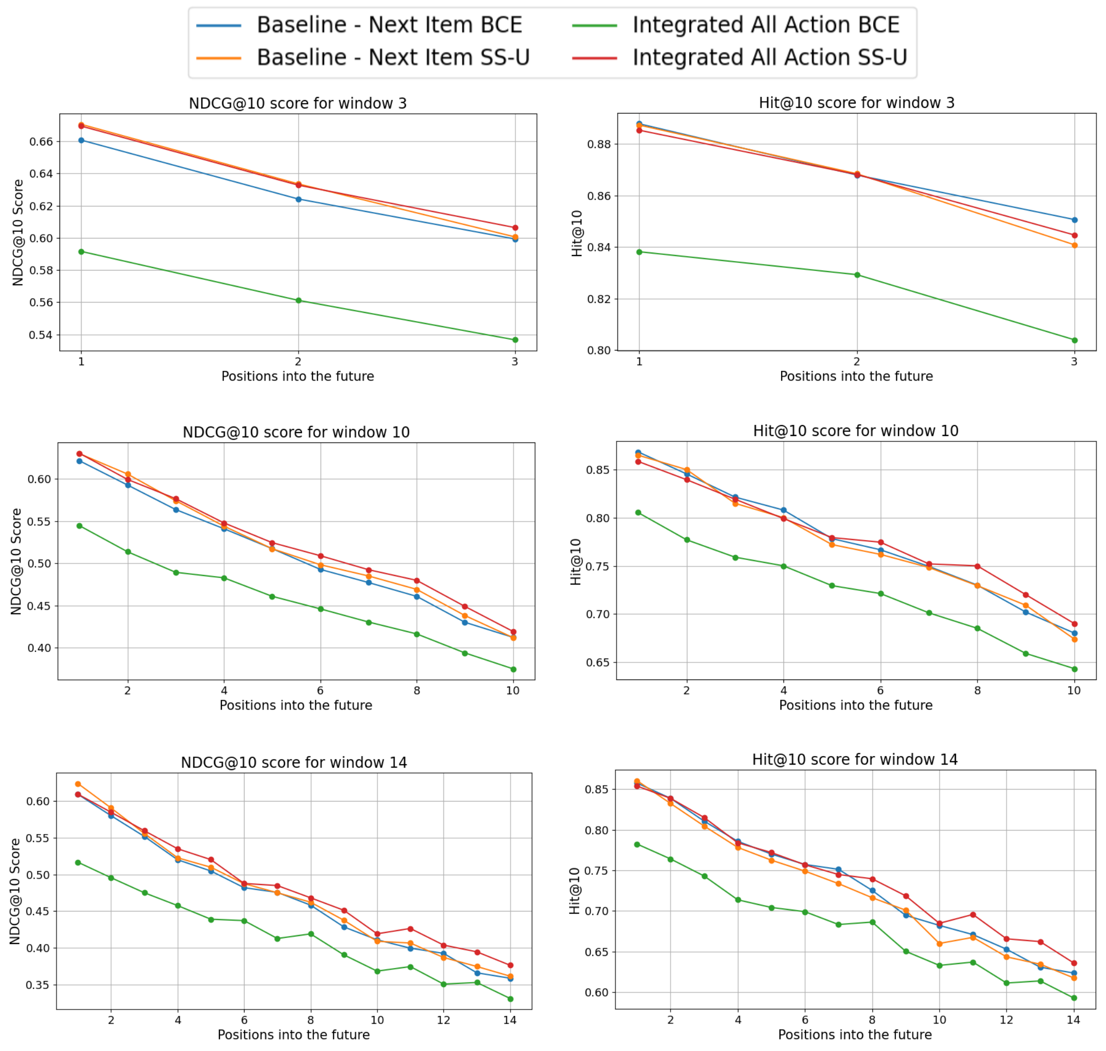

Figure 4.2: NDCG@10 and HR@10 scores for different windows for Baseline Next Item Training and Integrated All Action Prediction Models using SS-U and BCE losses.

Following, we focus on the experiment with a window size of 3. Upon close observation, it is evident that the Integrated model with BCE exhibits again the lowest performance among the tested configurations. For NDCG@10, it can be discerned from the graph that the performances of the Integrated and Baseline models with SS-U are almost indistinguishable, with the former showing slightly better accuracy at the 3^{rd} position into the future. The Baseline BCE model, on the other hand, begins with a slightly lower performance, but catches up with the Baseline SS-U model by the 3^{rd} position. When it comes to Hit@10, the Baseline BCE model manages to outperform the others, surpassing the Integrated SS-U and Baseline SS-U models at the 3^{rd} position. However, for the first and second positions, the performances are almost identical across the configurations. This suggests that the choice of loss function and training objective has a small impact on the model's performance, when the future window size for predictions is small.

Looking at the results for a window size of 10, it can be seen that the Integrated model with BCE continues to underperform. For NDCG@10, the Integrated objective with SS-U loss leads across almost all future positions, with the Baseline SS-U model coming in second. The baseline BCE model holds the third position, but from 5^{th} position onward, it almost matches the performance of the baseline with SS-U, and they both reach the same NDCG score at the 10^{th} future position. The results for Hit@10, showed that the three models (Integrated SS-U and both Baselines) share almost identical scores for the initial positions. However, starting from the 8^{th} position, the Integrated model begins to stand out and maintains a higher performance up to the final future position, indicating a better robustness when looking further into the future.

For our last experiment, we set the future interactions window size K to 14. We can observe that both Hit@10 and NDCG@10 metrics show that the Integrated model with Sampled Softmax Uniform, and both Baseline models yield quite similar performance in the first ten future positions. There doesn't appear to be any significant difference between these models within this range. Nevertheless, from the 10^{th} position onward, the Integrated model with SS-U begins to differentiate itself by delivering consistently higher performance up to the last 14^{th} future interaction. The Baseline models perform slightly worse that the leader. As in the previous experiments, the Integrated model with BCE trails behind. Testing on this window size still suggests that the Integrated All Action model maintains its better performance even for larger future windows.

Upon examining the NDCG@10 performance of the Integrated All Action modeling approach with SS-U loss over different future window sizes, we see some interesting trends. We show these comparisons in Figure 4.3.

In the first comparison between window sizes 3 and 7, we find that although the model with a window size of 3 starts with a significantly higher performance for the first future position, this advantage drops quickly. By the third future position, the performance gap between the two models is much more closed, indicating that a window size of 7 might be a more effective choice for longer-term prediction. The mean NDCG@10 scores for windows 3 and 7 for the model are 0.6362 and 0.5669 respectively.

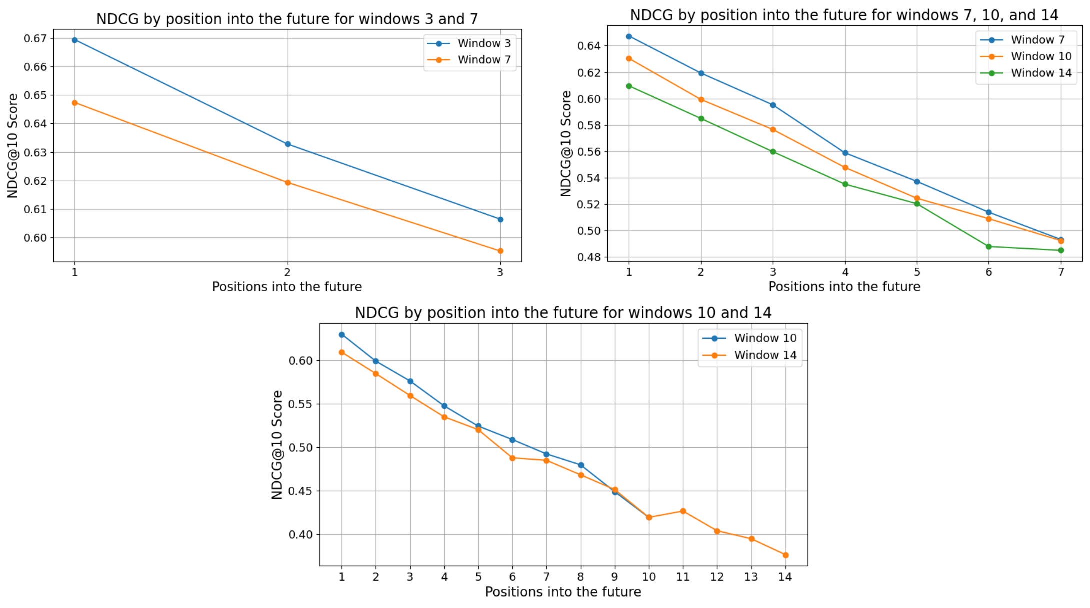

Figure 4.3: Comparison of different window sizes based on NDCG@10 for the Integrated All Action Prediction Model using Sampled Softmax Uniform loss.

When comparing larger window sizes of 7, 10, and 14, we notice that the performance for the first future positions is much closer across these models. Notably, the model with a window size of 14 displays the shallowest decline in performance across all future positions. It does, however, experience the most significant drop between the 5^{th} and 6^{th} future positions, after which its performance stabilizes.

Interestingly, the model with a window size of 10 starts to see a decrease in its performance drop by the end of its prediction window at position 7, matching the performance of the model with a window size of 7. This suggests that although the model with a window size of 10 may initially sacrifice some performance in its immediate predictions, it begins to reap benefits by the 7^{th} future position.

In the final comparison between window sizes 10 and 14, we observe that the model with a window size of 14 catches up to the model with a window size of 10 by the 9^{th} future position and continues to hold relatively steady. The average NDCG@10 scores across the window for these models are 0.5227 and 0.4801, respectively, indicating a consistent linear performance drop across future positions when training to predict further into the future.

These results suggest that the benefit of predicting further into the future does not lead to a dramatic drop in performance, and instead, the performance decline is stable and linear across the NDCG, Hit, and Kendall's Tau metrics. This insight can help guide the choice of future window size for optimal performance in long-term predictions.

Chapter 5

Conclusions

In this thesis, we have presented an in-depth study of sequential recommendation systems, aiming to enhance the understanding and development of models capable of predicting future user interactions. Our research was built upon the widely-recognized SASRec model, which served as the main architecture in our experiments [19]. Drawing inspiration from the work of Pancha et al. [33], we expanded their work by introducing several training objectives and techniques, each carrying its distinct characteristics. We have conducted a series of experiments to evaluate the effectiveness of these methodologies. While Pancha et al.'s research utilized a temporal window for predictions, our approach diverges by predicting future interactions without explicit temporal context [33]. Instead, we place a stronger emphasis on the sequential order of these interactions, recognizing that the chronology of user behavior carries significant value for recommendation systems. Our work has been guided by three main research questions, each of which has contributed to our understanding of the complexities and potential solutions in the realm of sequential recommendation systems.

In response to **RQ1**, our experimental results reveal that Integrated All Action Prediction modeling, when coupled with Sampled Softmax Uniform loss, demonstrates superior performance in predicting future user interactions. This integrated approach combines the strengths of Next Item Prediction and All Action Prediction, effectively leveraging the advantages of both short-term and long-term predictions, while mitigating their individual limitations. This hybrid model, when tested on a window size of 7, attained an average NDCG_{avg}@10 score of 0.5669, a Hit_{avg}@10 score of 0.8184, and a Kendall's Tau score of 0.2641, reflecting the accurate ordering of the recommendations.

Our findings for **RQ2** highlight the challenges associated with the implementation of different training techniques in enhancing the model's ability to generalize. While none of the proposed training techniques demonstrated significant improvements in this study, the Teacher Forcing technique exhibited some potential. Despite not showing immediate improvements in NDCG and Hit Rate compared to the base Integrated All Action, this approach hinted at a promising avenue for future research regarding the ordering of future items. Notably, it surpassed all other models by achieving a Kendall's Tau score of 0.2669. With more data or under different conditions, this technique might enable the model to more accurately predict target items based on an increasingly updated sequence of interactions. The incorporation of temporal information, however, did not yield the expected benefits like it did for Pancha et al., underscoring the complexity of the task when using a smaller and perhaps sparser dataset.

Finally, the answer to **RQ3** provides insightful observations on the impact of the size of the future prediction window. Our results suggest that while an increase in the window size might result in some sacrifice in the model's immediate prediction performance, it appears to offer long-term benefits by the end of the prediction window. Moreover, the performance decline is not dramatic and tends to stabilize, suggesting a linear relationship with the increase in future window size.

Although our research provides a significant contribution to the field, it also raises new questions and opens avenues for future work. One limitation of our study is the use of a single dataset, MovieLens-1M. While this dataset is widely used and dense, compared to other public datasets [19], it may not fully represent the diversity of user behavior in other domains or larger scales. In fact, testing our proposed methodologies on larger datasets, such as MovieLens-20M, could provide more insights, given its closer resemblance to the datasets used in Pancha et al.'s Pinnerformer paper [33]. Hence, future studies could validate and extend our findings by applying the proposed methodologies to such datasets and under different contextual conditions.

Another potential area for further exploration lies in the field of transfer learning. Given that the Integrated All Action Prediction modeling we employed combines Next Item Prediction with All Action Prediction, a possible direction could be to first train the item embeddings with Next Item Prediction, thereby learning the immediate item representations, and then proceed to All Action Prediction training. This idea is somewhat inspired by the approach taken in the Pinnerformer paper, where the authors used already learned item embeddings called PinSage [51, 33]. We had to learn these embeddings from scratch in our study, indicating the potential benefits of a two-step process, with each step focused on different aspects of the prediction task. Such a strategy could potentially lead to enhanced performance, and exploring it could be a promising direction for research to improve future recommendations.

In conclusion, our research contributes to the understanding of sequential recommendation systems and highlights promising directions for future exploration. We believe that our findings provide valuable insights for researchers and practitioners in the field, paving the way for the development of more effective and robust sequential recommendation systems.

Bibliography

[1] Alan Agresti. *Analysis of ordinal categorical data*, volume 656. John Wiley & Sons, 2010.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[5] Maarten De Rijke. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, 2017.

[6] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *ACM Transactions on Information Systems (TOIS)*, 39(1):1–42, 2020.

[7] Yang Feng, Shuhao Gu, Dengji Guo, Zhengxin Yang, and Chenze Shao. Guiding teacher forcing with seer forcing for neural machine translation. *CoRR*, abs/2106.06751, 2021.

[8] Aditya Grover. Lecture notes based on stanford university cs236, taught by stefano ermon and aditya grover, and have been written by aditya grover, with the help of many students and course staff. `https://deepgenerativemodels.github.io/notes/`.

[9] Pei-Yi Hao, Weng-Hang Cheang, and Jung-Hsien Chiang. Real-time event embedding for poi recommendation. *Neurocomputing*, 349:1–11, 2019.

[10] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 161–169, 2017.

[13] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 191–200. IEEE, 2016.

[14] Ruining He and Julian McAuley. Vbpr: visual bayesian personalized ranking from implicit feedback. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[16] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 843–852, 2018.

[17] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[18] Wendi Ji, Keqiang Wang, Xiaoling Wang, Tingwei Chen, and Alexandra Cristea. Sequential recommender via time-aware attentive memory network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 565–574, 2020.

[19] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pages 197–206. IEEE, 2018.

[20] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[21] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[24] Kyriakos Kyriakou. Informatics project proposal. University of Edinburgh, 2023.

[25] Kleanthi Lakiotaki, Stelios Tsafarakis, and Nikolaos Matsatsinis. Uta-rec: a recommender system based on multiple criteria analysis. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 219–226, 2008.

[26] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[27] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th international conference on web search and data mining*, pages 322–330, 2020.

[28] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1831–1839, 2018.

[29] Anjing Luo, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Zhixu Li, Lei Zhao, Victor S Sheng, and Zhiming Cui. Adaptive attention-aware gated recurrent unit for sequential recommendation. In *Database Systems for Advanced Applications: 24th International Conference, DASFAA 2019, Chiang Mai, Thailand, April 22–25, 2019, Proceedings, Part II 24*, pages 317–332. Springer, 2019.

[30] Julian McAuley. *Personalized Machine Learning*. Cambridge University Press, 2022.

[31] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Exploring data splitting strategies for the evaluation of recommendation models. *CoRR*, abs/2007.13237, 2020.

[32] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. PinnerSage. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, aug 2020.

[33] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. Pinnerformer: Sequence modeling for user representation at pinterest. *arXiv preprint arXiv:2205.04507*, 2022.

[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[35] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820, 2010.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[37] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[38] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations

from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[39] Shiming Sun, Yuanhe Tang, Zemei Dai, and Fu Zhou. Self-attention network for session-based recommendation with streaming data input. *IEEE Access*, 7:110499–110509, 2019.

[40] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[41] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 565–573, 2018.

[42] Poonam B Thorat, Rajeshwari M Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.

[43] Lyle Ungar and Dean P Foster. A formal statistical approach to collaborative filtering. *CONALD'98*, 1998.

[44] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. *Advances in neural information processing systems*, 26, 2013.

[45] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, volume 30, pages 47–56. Barcelona, 2000.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[47] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun. Sequential recommender systems: challenges, progress and prospects. *arXiv preprint arXiv:2001.04830*, 2019.

[48] Yaqing Wang, Caili Guo, Yunfei Chu, Jenq-Neng Hwang, and Chunyan Feng. A cross-domain hierarchical recurrent model for personalized session-based recommendations. *Neurocomputing*, 380:271–284, 2020.

[49] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.

[50] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[51] Jiajing Xu, Andrew Zhai, and Charles Rosenberg. Rethinking personalized ranking at pinterest: An end-to-end approach. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 502–505, New York, NY, USA, 2022. Association for Computing Machinery.

[52] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[53] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[54] Yongfu Zha, Yongjian Zhang, Zhixin Liu, and Yumin Dong. Self-attention based time-rating-aware context recommender system. *Computational Intelligence & Neuroscience*, 2022.

[55] Jianqing Zhang, Dongjing Wang, and Dongjin Yu. Tlsan: Time-aware long-and short-term attention network for next-item recommendation. *Neurocomputing*, 441:179–191, 2021.

[56] Yihu Zhang, Bo Yang, Haodong Liu, and Dongsheng Li. A time-aware self-attention based neural network model for sequential recommendation. *Applied Soft Computing*, 133:109894, 2023.

[57] Chuanchuan Zhao, Jinguo You, Xinxian Wen, and Xiaowu Li. Deep bi-lstm networks for sequential recommendation. *Entropy*, 22(8):870, 2020.

[58] Yuwen Zhou, Changqin Huang, Qintai Hu, Jia Zhu, and Yong Tang. Personalized learning full-path recommendation model based on lstm neural networks. *Information sciences*, 444:135–152, 2018.

Appendix A

Dataset Analysis

We present a series of graphical analyses performed on the MovieLens-1M dataset. These visualizations provide a deeper understanding of the underlying patterns and characteristics of the data, such as the distribution of user-item interactions, the activity levels of users, and the ratings data.

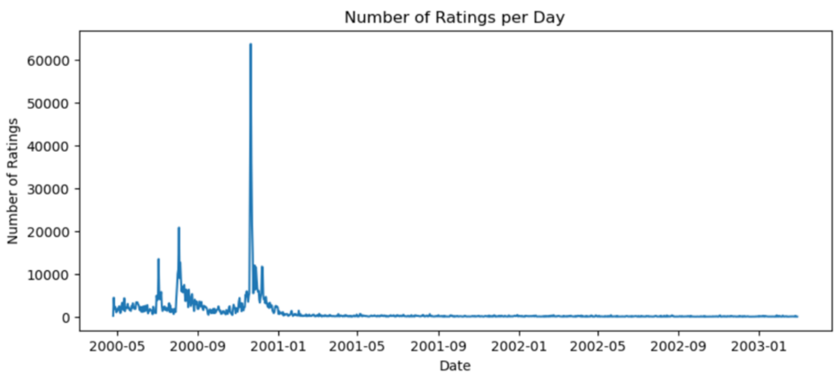

Figure A.1: Ratings-Date distribution.

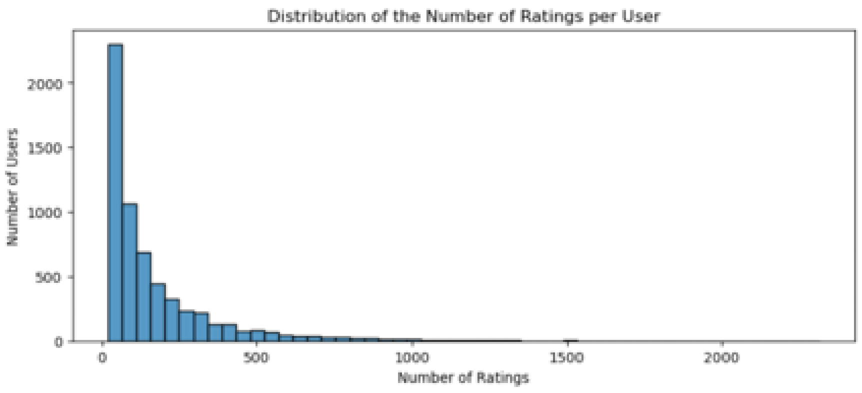

Figure A.2: User-Ratings distribution.

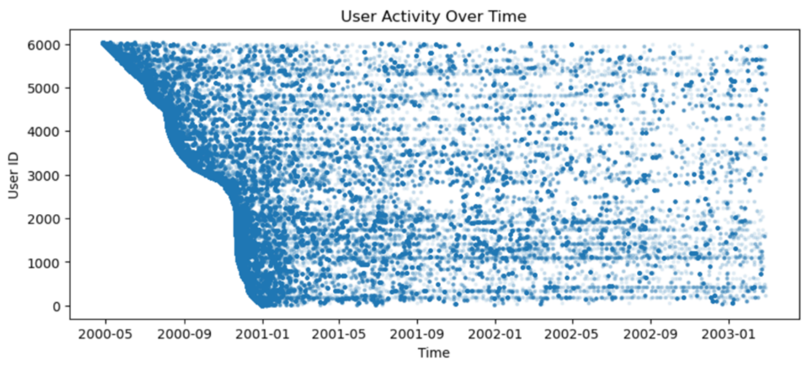

Figure A.3: User activity over time.

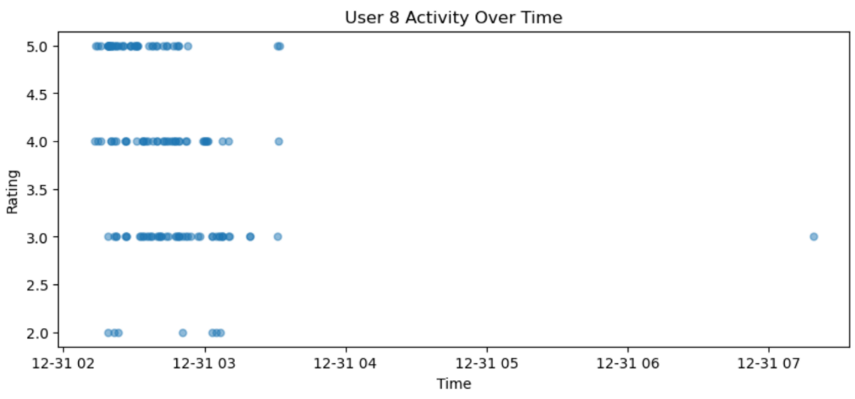

Figure A.4: User activity over time.

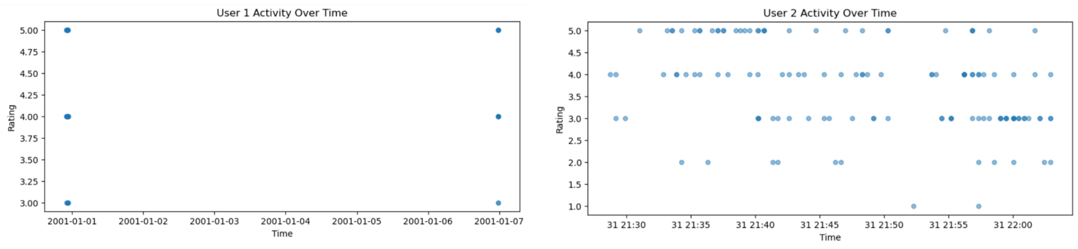

Figure A.5: Activities for specific users.

Despite the dataset density, the user activity predominantly occurs within short timeframes. This condensation of interactions poses a challenge for our model's ability to leverage temporal information, limiting the potential to extract meaningful patterns for recommendations.

Appendix B

Window Sizes Impact

The table below illustrates the average results across all metrics when model trained and evaluated on different window sizes.

Window Sizes	NDCG$_{avg}$@10	Hit$_{avg}$@10	Kendall's Tau
Window 3	0.6362	0.8661	0.3125
Window 7	0.5669	0.8184	0.2641
Window 10	0.5227	0.7782	0.2289
Window 14	0.4801	0.7405	0.1882

Table B.1: Performance comparison on different window sizes for the Integrated All Action Prediction Model using Sampled Softmax Uniform loss.

The tables below presents the results of the graph with a Window size of 7 for both the baseline and integrated all action prediction objectives. We consider three different loss functions: Sampled Softmax with LogQ, Sampled Softmax Uniform, and Binary Cross Entropy:

Position	Baseline BCE		Baseline SS-U		Baseline SS-LogQ	
	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10
1	0.6404	0.8827	0.6424	0.8734	0.6400	0.8692
2	0.6053	0.8562	0.6162	0.8536	0.6141	0.8496
3	0.5850	0.8410	0.5845	0.8254	0.5819	0.8210
4	0.5518	0.8132	0.5495	0.8052	0.5473	0.8033
5	0.5257	0.7898	0.5214	0.7768	0.5190	0.7722
6	0.4993	0.7750	0.4953	0.7620	0.4934	0.7576
7	0.4816	0.7566	0.4807	0.7422	0.4782	0.7374

Table B.2: NDCG@10 and HR@10 scores for Baseline with Window 7.

Position	Integrated BCE		Integrated SS-U		Integrated SS-LogQ	
	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10
1	0.5594	0.8124	0.6474	0.8788	0.6505	0.8751
2	0.5406	0.7967	0.6193	0.8569	0.6154	0.8512
3	0.5164	0.7802	0.5952	0.8400	0.5878	0.8293
4	0.4944	0.7598	0.5589	0.8167	0.5574	0.8098
5	0.4729	0.7331	0.5372	0.7896	0.5317	0.7811
6	0.4489	0.7203	0.5138	0.7733	0.5070	0.7735
7	0.4383	0.7110	0.4932	0.7570	0.4875	0.7531

Table B.3: NDCG@10 and HR@10 scores for Integrated All Action with Window 7.

The following tables show the results used for plotting the Window 3 graphs.

Position	Baseline BCE		Baseline SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.6608	0.8879	0.6706	0.8874
2	0.6242	0.8680	0.6336	0.8685
3	0.5993	0.8507	0.6006	0.8409

Table B.4: NDCG@10 and HR@10 scores for Baseline with Window 3.

Position	Integrated BCE		Integrated SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.5916	0.8382	0.6695	0.8854
2	0.5612	0.8293	0.6328	0.8682
3	0.5366	0.8040	0.6064	0.8447

Table B.5: NDCG@10 and HR@10 scores for Integrated with Window 3.

Next, we illustrate the tables of the Window 10 graphs:

Position	Baseline BCE		Baseline SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.6215	0.8685	0.6299	0.8651
2	0.5927	0.8456	0.6057	0.8501
3	0.5636	0.8215	0.5741	0.8150
4	0.5409	0.8080	0.5440	0.8001
5	0.5173	0.7783	0.5172	0.7721
6	0.4927	0.7667	0.4982	0.7619
7	0.4771	0.7494	0.4849	0.7483
8	0.4607	0.7299	0.4690	0.7294
9	0.4300	0.7019	0.4380	0.7090
10	0.4121	0.6801	0.4118	0.6739

Table B.6: NDCG@10 and HR@10 scores for Baseline with Window 10.

Position	Integrated BCE Loss		Integrated SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.5446	0.8054	0.6304	0.8586
2	0.5136	0.7771	0.5993	0.8396
3	0.4892	0.7590	0.5765	0.8193
4	0.4827	0.7500	0.5478	0.7992
5	0.4608	0.7294	0.5244	0.7794
6	0.4459	0.7212	0.5089	0.7746
7	0.4303	0.7011	0.4923	0.7520
8	0.4162	0.6850	0.4797	0.7500
9	0.3937	0.6589	0.4488	0.7200
10	0.3748	0.6431	0.4191	0.6900

Table B.7: NDCG@10 and HR@10 scores for Integrated with Window 10.

Lastly, we display below the tables of the Window 14 graphs:

Position	Baseline BCE		Baseline SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.6095	0.8574	0.6242	0.8604
2	0.5802	0.8386	0.5907	0.8323
3	0.5519	0.8105	0.5560	0.8045
4	0.5201	0.7857	0.5228	0.7782
5	0.5048	0.7700	0.5099	0.7625
6	0.4820	0.7572	0.4875	0.7490
7	0.4756	0.7512	0.4751	0.7336
8	0.4582	0.7253	0.4622	0.7163
9	0.4285	0.6946	0.4376	0.7009
10	0.4109	0.6822	0.4089	0.6600
11	0.3995	0.6709	0.4065	0.6675
12	0.3924	0.6529	0.3868	0.6435
13	0.3659	0.6308	0.3745	0.6345
14	0.3585	0.6236	0.3613	0.6176

Table B.8: NDCG@10 and HR@10 scores for Baseline with Window 14.

Position	Integrated All Action BCE Loss		Integrated with SS-U	
	NDCG@10	HR@10	NDCG@10	HR@10
1	0.5168	0.7824	0.6096	0.8537
2	0.4956	0.7640	0.5849	0.8386
3	0.4751	0.7430	0.5597	0.8150
4	0.4576	0.7137	0.5351	0.7839
5	0.4390	0.7043	0.5203	0.7719
6	0.4371	0.6991	0.4878	0.7568
7	0.4127	0.6833	0.4849	0.7448
8	0.4190	0.6863	0.4682	0.7396
9	0.3904	0.6503	0.4512	0.7186
10	0.3682	0.6330	0.4191	0.6848
11	0.3744	0.6371	0.4264	0.6957
12	0.3505	0.6113	0.4037	0.6657
13	0.3527	0.6139	0.3945	0.6623
14	0.3309	0.5929	0.3762	0.6360

Table B.9: NDCG@10 and HR@10 scores for Integrated with Window 14.