On the User Behavior Leakage from Recommender System Exposure

XIN XIN, JIYUAN YANG, HANBING WANG, JUN MA, and PENGJIE REN,
Shandong University, China
HENGLIANG LUO and XINLEI SHI, Meituan Inc., China
ZHUMIN CHEN and ZHAOCHUN REN, Shandong University, China

Modern recommender systems are trained to predict users’ potential future interactions from users’ historical behavior data. During the interaction process, despite the data coming from the user side, recommender systems also generate exposure data to provide users with personalized recommendation slates. Compared with the sparse user behavior data, the system exposure data are much larger in volume since only very few exposed items would be clicked by the user. In addition, user historical behavior data are privacy sensitive and commonly protected with careful access authorization. However, the large volume of recommender exposure data generated by the service provider itself usually receives less attention and could be accessed within a relatively larger scope of various information seekers or even potential adversaries.

In this article, we investigate the problem of user behavior data leakage in the field of recommender systems. We show that the privacy-sensitive user past behavior data can be inferred through the modeling of system exposure. In other words, one can infer which items the user has clicked just from the observation of current system exposure for this user. Given the fact that system exposure data could be widely accessed from a relatively larger scope, we believe that user past behavior privacy has a high risk of leakage in recommender systems. More precisely, we conduct an attack model whose input is the current recommended item slate (i.e., system exposure) for the user while the output is the user’s historical behavior. Specifically, we exploit an encoder-decoder structure to construct the attack model and apply different encoding and decoding strategies to verify attack performance. Experimental results on two real-world datasets indicate a great danger of user behavior data leakage. To address the risk, we propose a two-stage privacy-protection mechanism that first selects a subset of items from the exposure slate and then replaces the selected items with uniform or popularity-based exposure. Experimental evaluation reveals a trade-off effect between the recommendation accuracy and the privacy disclosure risk, which is an interesting and important topic for privacy concerns in recommender systems.

CCS Concepts: • Information systems → Recommender systems; Information extraction; • Security and privacy → Privacy protections;

X. Xin and J. Yang contributed equally to this research.
This work was supported by the Natural Science Foundation of China (grant nos. 2272274, 62202271, 61972234, 61902219, 62072279, and 62102234), the Key Scientific and Technological Innovation Program of Shandong Province (grant no. 2019JZZY010129), the Fundamental Research Funds of Shandong University, Meituan, and the Tencent WeChat Rhino-Bird Focused Research Program (grant no. JR-WXG-2021411).

Authors’ addresses: X. Xin, J. Yang, H. Wang, J. Ma, P. Ren, Z. Chen, and Z. Ren (corresponding author), Shandong University, 72 Binhai Road, Qingdao, Shandong Province, China, 266237; emails: xinxin@sdu.edu.cn, {jiyuan.yang, hanbing.wang}@mail.sdu.edu.cn, {majun, renpengjie, chenzhumin, zhaochun.ren}@sdu.edu.cn; H. Luo and X. Shi, Meituan Inc., No. 4 Wangjing East Road, Chaoyang District, Beijing, China, 100102; emails: {luohengliang, shixinlei}@meituan.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
1049-8188/2023/02-ART57 $15.00
https://doi.org/10.1145/3568954

ACM Transactions on Information Systems, Vol. 41, No. 3, Article 57. Publication date: February 2023.
1 INTRODUCTION

Recommender systems have been widely applied in various online web services, for example, online shopping [22], video or music platforms [62], and news recommendations [41], to provide users with the most interesting items. Generally speaking, a recommender system aims to predict the user’s future behavior based on the user’s historical interactions. Then, the recommendation list is generated by selecting the most relevant items according to the predicted user behavior. User historical behavior data are crucial to providing effective recommendations. These data are also highly privacy sensitive since various types of user profile information, such as gender, age, and even political orientation, could be inferred from the user behavior data [61]. As a result, user behavior data are protected with strict department-specific access authorization or regulations, such as the General Data Protection Regulation (GDPR) [38].

During the interaction process between users and recommenders, despite the data coming from the user feedback, the system also generates personalized slates of items that are pushed to the user as the recommendation service. These system behavior data are also known as recommender system exposure. Figure 1 gives illustrative examples of the exposure data. Compared with the sparse user behavior data, the system exposure data are much larger in volume since only very few exposed items would be clicked by the user. In addition, although service providers could adopt different strategies to infuse items or advertisements into the exposed list, the system exposure data still reflect users historical behavior patterns to a large degree. However, the large volume of system exposure receives much less attention compared with user behavior data and could often be accessed in a relatively larger scope that contains various information seekers or even adversaries [48]. For example, different departments may share the system behavior logs to perform cross-domain collaboration. The authors of [24] show that there are security concerns regarding the protection of recommender exposure data.

In this article, we investigate the problem of user behavior leakage in the field of recommender systems. We aim to answer the following questions:

(1) Is there a risk that the user historical behavior privacy can be inferred from the system behavior (i.e., recommender exposure) data?
(2) If there is the risk, how should the attack model be conducted?
(3) How should a protection mechanism be designed to downgrade the user behavior leakage risk?

There are plenty of previous works focusing on the privacy concern in recommender systems. For example, the authors of [13, 29] use cryptography to mask the user profile and behavior data. The authors of [13, 29] introduce differential privacy [12, 46, 65] to prevent user profile leakage. The authors of [33, 41, 52, 55, 60, 64] utilize federated learning to perform local computing on edge devices. The authors of [2, 9] utilize adversarial learning to promote recommendation models security. The authors of [63] investigate the membership inference attack against recommender systems. However, these works focus only on the protection of user data. In this article, we target
On the User Behavior Leakage from Recommender Exposure

Fig. 1. Two illustrative examples of recommender exposure data. (a) The user interacts with two items in the exposed item slate. (b) There is no observed interactions between the exposed items and the user in this scenario.

a new privacy preserving scenario, which is how the system behavior data exposes user privacy. This scenario poses a new attack and protection view from the recommender perspective.

We propose a framework containing a black-box attack model [47] to infer user past behavior privacy from system exposure and a protection mechanism to downgrade the privacy leakage risk. Here, the black-box model means that we do not need to know or explicitly model the recommendation algorithm. For the attack model, we adopt an encoder-decoder architecture. The input of the attack model is the exposed items from the system over a period. Note that the input data comes from the recommender perspective only, which means that we do not need to know which items in the exposed list are accessed by the user. Then, the encoder is utilized to map the input system behavior data to a latent representation. Here, we utilize three different encoding strategies, including mean pooling-, max pooling-, and self-attention [51]– based encoding.\(^1\) Based on the latent representation of system behavior, we propose two decoding methods to infer the privacy of user past behavior: point-wise decoding and sequence-wise decoding. Point-wise decoding treats the inference as a multi-label classification task with each past interaction as a label. Sequence-wise decoding further considers the sequential order of user past behavior. We utilized three different sequential models to conduct sequence-wise decoding, including Gated Recurrent Units (GRUs), a Long-Short Term Memory Network (LSTM), and an attention-based transformer decoder. We conducted experiments on two real-world datasets and empirical results indicate a great danger of user behavior leakage. In other words, potential adversaries can infer which items the user has clicked before just from the observation of current system exposure for this user. To alleviate the leakage risk, we propose a two-stage protection mechanism that first selects a subset of items from the system exposure and then replaces the selected items with uniform or popularity-based

\(^1\)In this article, we utilize three simple encoding strategies to further demonstrate the risk of privacy leakage since the attack can be performed without trivial design and complex computation. We leave more advanced encoding methods for the attack model as one of our future works.

ACM Transactions on Information Systems, Vol. 41, No. 3, Article 57. Publication date: February 2023.
exposure. Experimental evaluation reveals a trade-off effect between the recommendation accuracy and the privacy disclosure risk. We hope this work can raise more community concern regarding the protection of system data other than just focusing on protection from the user perspective. To summarize, the main contributions of this work are as follows.

- We point out a new privacy disclosure risk in recommender systems, that the user historical behavior privacy could be inferred from system exposure.
- We propose an attack model to perform user privacy inference. Experimental results on two real-world datasets indicate a great danger of privacy leakage.
- We propose a protection mechanism to alleviate the privacy risk. Empirical evaluation reveals a trade-off effect between recommendation accuracy and privacy leakage risk.

2 RELATED WORK

In this section, we provide a literature review of the research about recommendation, exposure data, and privacy concerns in recommender systems.

2.1 Recommendation

A recommender system is an information filtering system, aiming to predict user future preference based on historical user interactions with the system [63]. Collaborative filtering is one of the most successful recommendation methods, which is based on the similarity of users or items [31]. The key idea of collaborative filtering is that the user preference of an item can be predicted from similar user preferences over similar items [31]. Typical collaborative filtering methods include matrix factorization-based methods [26, 31, 37] or neighborhood-based methods [45]. Content-based recommendation [39], which utilizes the metadata of users and items (e.g., descriptions of item attributes and user profiles) to generate recommendation is also applied widely.

Recently, deep learning–based recommendation methods have become a hot research topic. The authors of [19] proposed neural collaborative filtering, which utilizes a multi-layer perceptron (MLP) to model high-order user-item interaction signals. Large numbers of studies have been proposed in this line of research, such as NFM [17], CDAE [57] and Wide&Deep [7]. The key idea is to use deep learning to increase model expressiveness. Graph neural networks (GNNs) also demonstrated their capability in recommendation since the user-item interactions can be naturally represented by interaction graphs. Plenty of graph recommendation models have emerged, such as HOP-Rec [59], NGCF [53], and LightGCN [18]. There are also other recommendation methods, such as causal inference–based methods and reinforcement learning–based methods [6, 58, 66]. We do not elaborate on all of them, only specific recommendation models, since this work focuses on the attack and protection of user past behavior privacy.

Another closely related sub-field is sequential recommendation, which aims to predict the user’s next interesting item from previous interacted items (in an interaction session). Early works for sequential recommendation are based on Markov chain [42] and factorization methods [21]. Recently, numerous deep learning–based recommendation models also emerged, including recurrent neural network (RNN)–based methods [20], convolutional neural network (CNN)–based methods [62], and the renowned attention-based methods [27, 49]. The main difference between our attack task and sequential recommendation lies in the following.

- The input of our attack model is the current system exposure data whereas the input for sequential recommendation is the previous user interactions.
- The output of our attack model is past user behavior whereas the output of sequential recommendation is predicted user future behaviors.
• The system usually exposes a slate of items simultaneously, which means that the input for our attack model could have no strong sequential signals.

2.2 Recommender Exposure

Compared with the sparse user feedback data, the large volume of exposure data generated by the recommender receives relatively less research attention. The exposed items are selected by the recommender and, thus, can be regarded as a kind of system behavior data. Existing research regarding the exposure data mainly focuses on negative sampling from exposed items or exposure debiasing [5].

To perform effective item ranking, negative training instances are necessary to provide comparison signals. To this end, the recommender exposure data contain rich information about the negative preference of users since only very few exposed items would be interacted by the user. The authors of [10] proposed to use the exposed but not interacted items as negative examples. The authors of [11, 54] proposed reinforced negative samplers, which can generate exposure-alike negative instances instead of directly choosing from exposure data. Generally speaking, the exposed but not interacted items can be seen as a kind of hard-negative example, which can help to boost the ranking performance. However, the authors of [10, 32] verified that both easy-negative examples and hard-negative examples are important for model training. Negative sampling based on only exposure data would also downgrade model performance. The mixture of exposed items and uniformly sampled items could be a good negative-sampling strategy.

Exposure bias, also known as “previous model bias” [34] since the exposure data generation is affected by previous recommendation policies, is one of the biggest sources of bias in recommendation [34]. Exposure bias happens since only parts of specific items are exposed by the system; thus, non-exposed and non-interacted items do not always represent user negative preference. More specifically, an unobserved interaction could only be attributed to the user unawareness of the item because the item is not exposed to the user. Therefore, regarding non-interacted items as negative samples could lead to misunderstanding of user true preference and suboptimal performance. There are works focusing on alleviating the effect of exposure bias. The authors of [43] proposed to use the missing-not-at-random assumption to perform debiasing. The authors of [25] proposed an exposure-based propensity matrix factorization framework to counteract exposure bias. The authors of [4] constructed a general debiasing framework and proposed an automatic debiasing method for a recommendation system based on meta learning. How to model exposure data distribution and conduct exposure debiasing has become an emerging hot topic.

Existing research regarding recommender exposure mainly focuses on negative sampling and debiasing. This work focuses on a new perspective, which is how system exposure data leaks user behavior privacy.

2.3 Privacy Security in Recommender Systems

Recent research shows that various types of user-sensitive information, such as age, gender, occupation, and even political orientation, can be inferred from user–item interactions [61]. As a result, numerous laws and regulations regarding privacy protection have been established for online web services, especially for recommender systems [38]. Privacy security has seen rapidly growing research interest in both academia and industry.

Existing privacy preserving recommendation methods can be categorized into cryptography-based methods, differential privacy-based methods, and federated learning-based methods. Cryptography methods attempt to mask user data, for example, through fully homomorphic encryption [29]. The authors of [13] presented a privacy recommendation algorithm in which the data on the server are encrypted and embedded to reduce the readability of the data. Differential
privacy-based methods aim to introduce random perturbations as the noisy signal to the user data \cite{12, 46}. For example, the authors of \cite{46} developed matrix factorization algorithms under local differential privacy. The authors of \cite{65} proposed differential private graph convolutional networks to protect users’ sensitive data against attribute inference attacks and provide high-quality recommendations at the same time. The authors of \cite{9} utilized gender obfuscation for user profiles to protect user privacy. The authors of \cite{2} conducted privacy-aware recommendation with adversarial training that aims to both protect users against attribute inference attacks and preserve the quality of recommendation. In recent years, federated learning–based methods have become a hot research topic with the prevalence of increasing numbers of edge devices with computation capacity. The key idea of federated learning–based methods is distributed model training \cite{60}. The original data are not transferred between the server and the device to avoid leakage. The transferred communication message is based on model gradients \cite{1, 60}. However, the model gradients can also be utilized to infer the original data. As a result, \cite{3, 33, 35} proposed to combine federated learning with cryptography methods or differential privacy. There is also research focusing on improving the efficiency of federated learning–based recommender systems \cite{28, 36}. Federated learning has also been applied to domain-specific recommendation tasks, such as news recommendation \cite{41}, graph-based recommendation \cite{55}, and sequential recommendation \cite{14}.

The authors of \cite{63} conducted research focusing on membership inference attacks against recommender systems. Their attack aims to determine whether a user’s behavior data are used by the targeted recommender. They then proposed a simple yet effective protection mechanism to address privacy concerns.

These existing privacy-preserving methods in recommender systems mostly focus on the protection of user behavior data. However, system behavior data (i.e., the system exposure data) are less explored regarding privacy concerns. Due to the fact that the interactions between users and recommenders naturally form a closed loop, we argue that system behavior data should also be investigated to protect user privacy, especially when there is data sharing between different recommender systems. To the best of our knowledge, our work is the first attempt that raises privacy concerns from the system perspective versus the commonly investigated user perspective.

3 THE ATTACK MODEL

In this section, we first present notations and task formulation of user behavior privacy attacks in this work. Then, we describe the encoding and decoding detail of the attack model. The task of the user behavior privacy attack is to infer user historical behavior given system exposure data. In this article, we exploit a simple encoder-decoder architecture to further demonstrate the risk of privacy leakage since the attack can be performed without trivial design and complex computation.

3.1 Notations and the Attack Task Formulation

Let $\mathcal{U}$ and $\mathcal{I}$ denote the user set and the item set, respectively. During the service of a recommender system, the user $u$ produces a series of user behaviors (e.g., view, clicks, or purchases) based on the items exposed by the system. We use $B^u = \{B^u_1, B^u_2, \ldots, B^u_{|B^u|}\}$ to denote the set of user behavior sequences of user $u$, where $B^u_i = (b_1, b_2, \ldots, b_M)$ with $b_j \in \mathcal{I}$ being a specific user behavior sequence. This is regarded as the user privacy in this work. The set of recommended item slates for user $u$ is represented by $E^u = \{E^u_1, E^u_2, \ldots, E^u_{|E^u|}\}$, where $E^u_i = (e_1, e_2, \ldots, e_N)$ with $e_j \in \mathcal{I}$ denotes a specific exposed item slate. These are the input data for the attack model of this work. As shown in Figure 2(b), the task of the attack model is to infer $B^u$ from the observation of $E^u_i$, which can be formulated as the estimation of

$$p(b_1, b_2, \ldots, b_M|e_1, e_2, \ldots, e_N).$$  

(1)
Fig. 2. A recommender system (a) aims to predict the user’s future behavior from the historical user interactions. In this article, the attack scenario (b) focuses on inferring the privacy of the user’s past behavior from the system behavior data.

Table 1. The Glossary Table

| Notations | Description |
|-----------|-------------|
| $\mathcal{U}, \mathcal{I}$ | user and item set |
| $B^u$ | user behavior sequence set of user $u$, $B^u = \{B^u_1, B^u_2, \ldots, B^u_{|B^u|}\}$ |
| $B^u_i$ | specific user behavior sequence of user $u$, $B^u_i = (b_1, b_2, \ldots, b_M)$ with $b_j \in \mathcal{I}$ |
| $E^u$ | set of exposed item slates for user $u$, $E^u = \{E^u_1, E^u_2, \ldots, E^u_{|E^u|}\}$ |
| $E^u_i$ | specific exposed item slate for user $u$, $E^u_i = (e_1, e_2, \ldots, e_N)$ with $e_j \in \mathcal{I}$ |
| $M$ | length of user behavior sequence |
| $N$ | size of exposed item slate |
| $d$ | item embedding size |

The common sequential recommendation task, which aims to predict the next interesting item from previous user interactions, can be formulated as the estimation of $p(b_{t+1}|b_1, b_2, \ldots, b_t)$ as shown in Figure 2(a). We can see that there are substantial differences between our attack task and the sequential recommendation task. The input and output of our attack task are from different information sources, systems and users, respectively. For sequential recommendations, the input and output focus only on the user perspective. As the input of our attack task, the system exposure data $E^u_i$ comes from the system side only, which means that the attack model does not know which items in $E^u_i$ were clicked by the user. Generally speaking, very few items in $E^u_i$ could be accessed by the user. Finally, the recommender usually exposes a slate of items simultaneously, which means that the items in $E^u_i$ could have no strong sequential orders.

Table 1 summarizes the important notations used in this article.

3.2 Privacy Attack

To perform the attack task, we exploit an encoder-decoder architecture. The encoder aims to convert the input of current system exposure $E^u_i$ into a latent representation. Then, the decoder attempts to infer the past user behavior privacy $B^u_i$ from the encoded representation. Figure 3 illustrates the overall architecture of the attack model.
Fig. 3. The overall attack model structure. The encoder aims to map the input of system exposure $E^u_i$ to a latent representation $c$. Then, the user privacy $B^u_i$ could be inferred through point-wise decoding or sequence-wise decoding.

3.2.1 Encoder. Due to the fact that the recommender system usually exposes a slate of items simultaneously, we do not consider the sequential order of items in $E^u_i$. In this work, we utilize three simple encoding strategies — mean pooling—, max pooling—, and self-attention—based encoding — to further demonstrate the risk of privacy leakage since the attack can be conducted without trivial design and complex computation. We leave more advanced encoding methods for the attack model as one of our future works.

Mean pooling—based encoding. The pooling technique has been widely used in the field of computer vision to produce a summary statistic of the input and reduce the spatial dimension. Given the input system exposure $E^u_i = (e_1, e_2, ..., e_N)$, we first embed each item $e_j$ into a dense representation $e_j \in \mathbb{R}^d$ where $d$ denotes the embedding size. This can be done through a simple embedding table lookup operation. Then, we utilize mean pooling to seize the mean effect of the recommender-exposed item feature vectors as the slate-level representation:

$$c_{mean} = \frac{1}{N} \sum_{j=1}^{N} e_j \in \mathbb{R}^d.$$  

Max pooling—based encoding. While the mean pooling—based encoding attempts to encode the mean effect of system behavior, the max pooling—based encoding forces the encoder to retain only the most useful exposed item features. We select the highest neuron activation value across the whole embedding space:

$$c_{max}(i) = \max_{j=1, \ldots, N} e_j(i), \quad i = 1, \ldots, d,$$

where $c_{max}(i)$ denotes the $i$-th element of the max pooling encoding representation $c_{max} \in \mathbb{R}^d$. $e_j(i)$ is the $i$-th element of the item embedding $e_j$.

Self-attention—based encoding. While mean pooling and max pooling are rather simple encoding methods, we also attempt to exploit the successful transformer encoder [51] to learn the latent
representation of the system exposure data. The transformer encoder is based on the self-attention mechanism, which is highly efficient and capable of uncovering semantic patterns of the input data. We illustrate the structure of the self-attention–based encoding in Figure 4. As discussed before, the recommender often exposes a slate of items simultaneously. Thus, we do not introduce the position embeddings in our attack encoder.

The self-attention–based encoder contains multi-head attention and feed-forward networks. Multi-head attention adopts scaled dot-product attention at each head to learn the importance of input items. Dot-product–based attention is formulated as

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

where $Q, K, V$ denote the queries, keys, and values, respectively. The attention operation computes a weighted sum of values according to the weights calculated from the correlations between the query and the key. The scale factor $\sqrt{d}$ is used to normalize the computed correlations to avoid overly large inner product. Self-attention uses the same objects as queries, keys, and values. Then, the multi-head attention on the input item embeddings is formulated as

$$\text{MHA}(E) = \text{concat}\{\text{head}_1, \text{head}_2, \ldots, \text{head}_h\}W^O,$$

where

$$\text{head}_i = \text{Attention}(W^Q_i E, W^K_i E, W^V_i E),$$

$E = [e_1, e_2, \ldots, e_N] \in \mathbb{R}^{N \times d}$ is the stacked embedding matrix of the exposed items, $W^Q_i, W^K_i, W^V_i$ and $W^O$ are trainable parameters, and $h$ is the number of heads.

To avoid overfitting and enable more stable learning without vanishing or exploding gradient issues, we also introduce dropout layers, residual connections, and layer normalization. The representation after multi-head attention is formulated as

$$\tilde{E} = E + \text{dropout}(\text{MHA}(\text{LayerNorm}(E))) \in \mathbb{R}^{N \times d}.$$
Next, a two-layer feed-forward network (FFN) is utilized to increase the encoder capacity. Then, the latent representation after self-attention–based encoding is formulated as

$$C_{\text{att}} = \hat{E} + \text{dropout}(\text{FFN}(\text{LayerNorm}(\hat{E}))) \in \mathbb{R}^{N \times d}. \quad (7)$$

In addition, we insert a CLS token into the original input of $(e_1, e_2, .. e_N)$. Then, the corresponding vector of the CLS token in $C_{\text{att}}$ can also be regarded as the final encoded vector representation of system exposure.

3.2.2 Decoder. Based on the encoded latent representation of system behavior, we propose two decoding strategies to infer the privacy of user past behavior: point-wise decoding and sequence-wise decoding. For sequence-wise decoding, we utilize three notable sequential models: GRU, LSTM, and the attention-based transformer decoder.

**Point-wise decoding.** Point-wise decoding treats inference as a multi-label classification task with each past interaction as a label. The decoder is composed of a fully connected layer and the classification probability can be defined as

$$q = [q_1, q_2, .. q_{|I|}] = \text{softmax} (\sigma (W_d c + b)) \in \mathbb{R}^{|I|}, \quad (8)$$

where $q$ denotes the classification probability, $\sigma$ is the activation function, $c$ is $c_{\text{mean}}, c_{\text{max}}$ or the CLS token vector in $C_{\text{att}}$ according to different encoding methods. $W_d \in \mathbb{R}^{d \times |I|}$ and $b \in \mathbb{R}^{|I|}$ are trainable parameters.

Then, we use label smoothing regularization [16, 50] to generate the ground-truth label for the multi-label classification task. Label smoothing prevents the network from becoming over-confident. It encourages the fully connected layer to make a finite output and generalize better. Assume that the ground-truth user behavior sequence is $B^u_i = (b_1, b_2, .., b_M)$; the ground-truth probability after label smoothing is defined as

$$y_j = \begin{cases} 
(1-\varepsilon)/M & \text{if } j \in B^u_i \\
\varepsilon/(|I|-M) & \text{otherwise},
\end{cases} \quad (9)$$

where $\varepsilon$ is a small constant.

Finally, the training loss function for point-wise decoding is formulated as

$$\mathcal{L}(y, q) = - \sum_{j=1}^{|I|} y_j \log q_j. \quad (10)$$

Empirically, we found that label smoothing dramatically downgrades the risk of model overfitting.

**Sequence-wise decoding.** Point-wise decoding does not consider the sequential order of user past behaviors. In this subsection, we describe in detail how to use sequence-wise decoding to infer user behavior privacy in reversed order (i.e., from the most recently $b_M$ to the earliest $b_1$). In Section 3.2.1, we present a latent representation for the system exposure data. We then use three different sequential models to perform the inference: GRU [8], LSTM [44], and the transformer decoder [51].

During the training process, we process the user behavior sequence $B^u_i = (b_1, b_2, .., b_M)$ reversely and complement $B^u_i$ with two special tokens as $(<\text{start}>, b_M, .., b_2, b_1, <\text{end}>)$. Then, we put $B^\mu_i = (<\text{start}>, b_M, .., b_2, b_1)$ as the input of the decoder and the output is shifted to $(b_M, .., b_2, b_1, <\text{end}>).$ We use $Q \in \mathbb{R}^{(M+1)\times |I|}$ to denote the output classification probability of the decoder as

$$Q = \text{SeqDecoder}(C, B^\mu_i), \quad (11)$$
where \( \text{SeqDecoder} \) represents the three different sequential models. \( C \in \mathbb{R}^{N \times d} \) is the stacked \( c_{\text{mean}}, c_{\text{max}}, \) or \( C_{\text{att}} \) according to different encoding methods. In the following, we briefly introduce the three sequential decoding models.

1. **LSTM-based decoder.** The LSTM is a famous sequence modeling neural network. We utilize the most common LSTM unit, which is composed of a cell, an input gate, an output gate, and a forget gate. The cell can remember values over arbitrary time intervals and these three gates regulate the flow of information into and out of the cell [44]. For each timestep \( t(1 \leq t \leq M + 1) \), the output classification probability \( q_t \in \mathbb{R}^{|J|} \) from the LSTM-based decoder is formulated as

\[
\begin{align*}
  i_t &= \sigma(W_i[h_{t-1}; b_{t-1}] + p_i) \\
  f_t &= \sigma(W_f[h_{t-1}; b_{t-1}] + p_f) \\
  \tilde{g}_t &= \tanh(W_g[h_{t-1}; b_{t-1}] + p_g) \\
  g_t &= i_t \odot \tilde{g}_t + f_t \odot g_{t-1} \\
  o_t &= \sigma(W_o[h_{t-1}; b_{t-1}] + p_o) \\
  h_t &= o_t \odot \tanh(g_t) \\
  q_t &= \text{softmax}(\text{FFN}(h_t)),
\end{align*}
\]

where \( b_{t-1} \) is the embedding of the \((t-1)\)-th item (i.e., \( b_{t-1} \)) in the decoder. \( h_{t-1} \) denotes the output hidden state of the \((t-1)\)-th step. The \( i_t, f_t, o_t \) represent the three gates’ activation vector, respectively. \( g_t \) and \( \tilde{g}_t \) are cell state vectors. \( W_i, W_f, W_g, W_o, p_i, p_f, p_g, \) and \( p_o \) are trainable parameters. At the first training time step, the input \( b_0 \) is the embedding of the special token \(<\text{start}>\).

2. **GRU-based decoder.** The GRU is also a notable gating-based recurrent neural network. Compared with LSTM, it is relatively simpler, with fewer parameters. For each timestep \( t(1 \leq t \leq M+1) \), the output classification probability \( q_t \in \mathbb{R}^{|J|} \) from the GRU-based decoder is formulated as

\[
\begin{align*}
  z_t &= \sigma(W_z[h_{t-1}; b_{t-1}] + p_z) \\
  r_t &= \sigma(W_r[h_{t-1}; b_{t-1}] + p_r) \\
  \tilde{h}_t &= \tanh(W_h[r_t \odot h_{t-1}; b_{t-1}] + p_h) \\
  h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \\
  q_t &= \text{softmax}(\text{FFN}(h_t)),
\end{align*}
\]

where \( b_{t-1} \) and \( h_{t-1} \) are similar to the LSTM-based decoder. \( z_t \) and \( r_t \) represent the “update gate” vector and the “reset gate” vector, respectively. \( W_z, W_r, W_h, p_z, p_r, \) and \( p_h \) are learnable parameters. Similar to the LSTM-based decoder, the input \( b_0 \) is the embedding of the special token \(<\text{start}>\) at the first training timestep.

3. **Attention-based transformer decoder.** We do not elaborate the mathematical detail again since the attention-based transformer decoder shares similar calculations with the attention-based encoding method. However, we still need to claim the following differences.

- Unlike the encoder, position embeddings are introduced in the decoder to distinguish the order of user past behaviors and introduce the sequential signals.
- Due to the nature of attacking user past behavior sequences, the model should consider only the last \( j \) items when inferring the \((j-1)\)-th item. As a result, a casual mask is introduced in the decoder to modify the attention computation as masked multi-head attention.

In contrast to point-wise decoding, in sequence-wise decoding we define a cross-entropy loss for each output position to perform the classification. The final loss function for sequence-wise
decoding can be formulated as

\[ \mathcal{L}(Q, Y) = - \sum_{m=1}^{M} \sum_{j=1}^{\lfloor I \rfloor} y_{mj} \log q_{mj}, \]  

(14)

where \( y_{mj} \) and \( q_{mj} \) are ground-truth probability and the \((m, j)\)-th entry of the computed probability matrix \( Q \) (i.e., the stack of \( q_t \)). Note that in the inference stage, we cannot obtain the ground-truth user behaviors, which was possible during the training process. As a result, we use the inference results of previous steps as the input for the next inference step.

4 PRIVACY PROTECTION

In the previous section, we describe the attack model to infer user past behavior privacy from system exposure data. Here, we describe a two-stage protection mechanism that aims to alleviate the privacy risk. The reason that user behavior can be inferred from system exposure data lies in the fact that the items exposed by the system have inherent relationships with the previous interacted items of the user (e.g., similarity or sequential connections). A simple solution is to infuse random items as the noisy signal into the system exposure to obfuscate the relationships. The key idea is similar to differential privacy [48]. The difference is that existing differential privacy–based methods add noise to the user data while our protection targets introducing noise into the system exposure.

The proposed protection mechanism consists of two stages: position selection and item replacement. At the first stage, we decide which exposure positions would be replaced using a random-based or similarity-based position selection method. We then utilize a uniform-based or popularity-based replacement strategy to sample items to replace the exposed items on corresponding selected positions. Figure 5 illustrates the protection mechanism.

4.1 Position Selection

At this stage, we design two methods, the random-based method and the similarity-based method, to choose which positions in the exposed item slates would be replaced, as shown on the left-hand side of Figure 5.

**Random-based position selection.** In this method, we randomly select positions of the exposed item slate according to a uniform distribution. Given the exposed item slate \( E^u_i = (e_1, e_2, \ldots, e_N) \) and a replacement proportion \( L \), we randomly select \( m = \lfloor N \times L \rfloor \) positions as \( p = \{p_1, p_2, \ldots, p_m\} \), where \( p_i \leq N \).

**Similarity-based position selection.** The above random-based position selection could lead to a situation in which potential future interacted items are switched out, thus, downgrading recommendation accuracy. In similarity-based position selection, we select positions according to the similarity between a specific exposed item and the overall slate representation (i.e., the representation center of all exposed items within a slate). Given the exposed item slate \( E^u_i \), we utilize the item embeddings obtained from the recommendation model (i.e., SASRec [27]) pretrained using user historical behavior data and perform mean pooling to seize the mean effect of the user historical behavior. The user behavior representation of user \( u \) is defined as

\[ b_u = \frac{1}{\lvert B_u \rvert} \sum_{j=1}^{\lvert B_u \rvert} b^u_j \in \mathbb{R}^d. \]  

(15)
Privacy protection includes two stages: position selection and item replacement. Position selection decides which exposure positions would be replaced using random-based or similarity-based strategies. Item replacement replaces the exposed items on previously selected positions with uniform or popularity sampled items.

Then, we calculate the softmax cosine similarity between the user behavior representation \( b_u \) and each exposed item \( e_i \) as

\[
s(u, i) = \text{softmax} \left( \frac{e_i \cdot b_u}{ \max(\|e_i\|_2 \cdot \|b_u\|_2, \omega)} \right),
\]

where \( s(u, i) \) represents the probability similarity between \( e_i \) and the user behavior, and \( \omega \) is a small constant.

Since \( b_u \) denotes the overall user historical behavior representation, which can be further regarded as the user preference, we believe that replacing dissimilar positions could help us to maintain good recommendation performance. As a result, we assign the probability of selecting a position to be replaced according to the value of \( s(u, i) \) and positions with smaller \( s(u, i) \) are more likely to be replaced. Finally, we select \( m = \lceil N \cdot L \rceil \) positions as \( p = \{p_1, p_2, \ldots, p_m\} \) according to the calculated similarity.

**4.2 Item Replacement**

After position selection, we investigate two item replacement strategies based on uniform exposure and popularity exposure, as shown on the right-hand side of Figure 5.

**Uniform replacement.** In this strategy, we randomly sample items from the whole item set according to a uniform distribution to replace the exposed items of the system. For the selected positions \( p = \{p_1, p_2, \ldots, p_m\} \) discussed in Section 4.1, we uniformly sample a replacement item set \( \{r_{p_1}, r_{p_2}, \ldots, r_{p_m}\} \). Then, the system exposure data can be updated as

\[
E^*_i = (e^*_1, \ldots, e^*_j, \ldots, e^*_N), e^*_j = \begin{cases} r_{p_f} \sim \text{Unif}(I) & \text{if } j = p_f, \\ e_j & \text{otherwise,} \end{cases}
\]

where \( f \leq m \). We take \( E^*_i \) as the input data for the trained attack model to perform a new attack inference to verify how the replacement affects the attack model performance. The new recommendation accuracy can be defined as

\[
\text{acc}^*_u = \frac{|E^*_u \cap B^u|}{|E^*_u|}.
\]

Obviously, the uniform replacement would also affect recommendation accuracy.
### Table 2. Dataset Statistics

| Dataset | Zhihu | MIND |
|---------|-------|------|
| #users  | 7,963 | 94,057 |
| #items  | 64,573 | 34,376 |
| #impressions | 1,000,026 | 8,584,442 |
| #clicks | 271,725 | 347,727 |

The impressions can be seen as system behavior data whereas the clicks can be regarded as the user behavior privacy.

**Popularity replacement.** In this strategy, we sample the replacement item set according to item popularity. As discussed before, given the exposed item slate $E_u^i$ and the replacement proportion $L$, we select the positions as $p = \{p_1, p_2, \ldots, p_m\}$ as described in Section 4.1. Then, we sample a replacement item set $\{r_{p_1}, r_{p_2}, \ldots, r_{p_m}\}$ according to item popularity. Here, we adopt two kinds of popularity, the overall popularity and the in-batch popularity. Overall popularity means that the popularity distribution is computed over all items whereas in-batch popularity means that the popularity is computed over the items in the current inference batch. Then, the system exposure data can be updated as

$$e_j^* = \begin{cases} 
  r_{p_f} \sim \text{Pop}(I) \text{ or Pop}(I_b) & \text{if } j = p_f, \\
  e_j & \text{otherwise},
\end{cases}$$

(19)

where $f \leq m$, and $I_b$ denotes the item set in the batch. Similarly, we take $E_u^i$ as the input data for the trained attack model to perform a new attack inference to verify how popularity-based replacement affects attack model performance and recommendation accuracy.

### 5 EXPERIMENTS

In this section, we conduct experiments to verify the user behavior privacy leakage risk in recommender systems.\textsuperscript{2} We aim to answer the following research questions:

- **RQ1** How does the attack model perform? Is there is a substantial user behavior leakage risk in recommender systems?
- **RQ2** How does the number of exposed items affect attack performance? What is the proper setting to conduct the attack?
- **RQ3** How does the protection mechanism affect attack performance and recommendation accuracy?

#### 5.1 Dataset Description

The experiments were conducted on two real-world datasets: Zhihu\textsuperscript{3} \cite{zhihu_dataset} and MIND\textsuperscript{4} \cite{mwindataset}. Both datasets contain user behavior data (e.g., clicks) and system behavior data (e.g., exposed impressions). Table 2 summarizes the statistics of the two datasets.

**Zhihu.** This dataset is collected from a large-scale knowledge-acquisition platform. The original dataset contains question information, answer information, and the user profile. Here, we focus on the answer recommendation scenario. In the serving period of the recommender, a slate of answers

\textsuperscript{2}We release our code at https://github.com/nancheng58/On-the-User-Behavior-Leakage-from-Recommender-System-Exposure.

\textsuperscript{3}https://github.com/THUIR/Zhihu-Dataset.

\textsuperscript{4}https://msnews.github.io/.
On the User Behavior Leakage from Recommender Exposure

(i.e., the items in our setting) are exposed to the user. This can be seen as the system behavior data. The user may click some of the answers for more detailed information, which can be seen as the user behavior data. The dataset contains the show time and click time of all answers (0 for non-click answers). More precisely, given a timestamp $t$ and a user $u$, the $M$ answer clicks just before $t$ are seen as the past user behavior privacy. The $N$ system exposure impressions just following the timestamp $t$ are regarded as the input for the attack model. Finally, the dataset contains 1,000,026 system impressions and 271,725 user clicks over 64,573 items of 7,963 users. With the default setting of $M = 5$ and $N = 10$, we can get a total of 213,172 pairs of $[B^u_i|E^u_i]$. 

**MIND.** This dataset focuses on the news recommendation scenario, which is collected from the system logs of a news website. The dataset we use contains 8,584,442 recommended impressions for 94,057 users over 34,376 news events with 347,727 total user clicks. Each impression log contains a slate of recommended news events and historical user click behaviors of this user before the impression. We sort the historical user clicks according to the timestamp. Then, the historical clicked news events are regarded as the user behavior privacy and the news events in the recommended impressions are seen as the system behavior data. We get a total of 291,595 pairs of $[B^u_i|E^u_i]$ with the default setting of $M = 5$ and $N = 10$.

### 5.2 Evaluation Protocols

**5.2.1 Attack Evaluation.** We adopt cross-validation to evaluate the performance of the attack model. The ratio of training, validation, and test set is 8:1:1. We randomly sample the data of 80% of the users as the training set. The data of 10% of the users is used for validation and the remaining 10% of the users are regarded as the test users. Such user-based data splits can effectively avoid the potential information shortcuts between training and test. For validation and test, the evaluation is done by providing the attack model with item slates exposed by the system and then checking the rank of the $M$ ground-truth items in the inference results. We adopt Recall to evaluate attack performance. Let $\hat{B}^u_i@k$ denote the top-$k$ inference output of the attack model. Recall@$k$ measures how many ground-truth user behaviors are included in $\hat{B}^u_i@k$, which is formulated as

$$
\text{Recall}@k = \frac{|\hat{B}^u_i@k \cap B^u_i|}{M}.
$$

We then report the average Recall@$k$ across the whole test user set as the final results. We also report normalized discounted cumulative gain (NDCG) and mean reciprocal rank (MRR), which are weighted versions of Recall assigning higher weights to the top-ranked positions of the inference result lists.

**5.2.2 Protection Evaluation.** To verify the effect of the proposed protection mechanism, we first compute $E^u_i$ for user $u$ in test users according to Equations (17) or (19). Then, we feed $E^u_i$ to the pretrained attack model to calculate the new attack metrics (i.e., Recall, NDCG, and MRR). We also compute the new recommendation accuracy for user $u$ according to Equation (18). We then report the overall recommendation accuracy as the average of all users in the test set.

### 5.3 Hyperparameter Settings

We conduct our experiments with a batch size of 400 pairs of system exposure data and user behavior data (i.e., $[B^u_i|E^u_i]$). The sizes of $B^u_i$ and $E^u_i$ are set as $M = 5$ and $N = 10$, respectively,

---

$\hat{B}^u_i@k$ contains $k \times M$ items. For point-wise decoding, $\hat{B}^u_i@k$ is the top-$(k \times M)$ items of the inference results since point-wise decoding does not consider the sequential order of user behavior. For sequence-wise decoding, $\hat{B}^u_i@k$ is composed of the top-$k$ items of all $M$ inference positions.
Table 3. Attack Performance with Point-wise Decoding

| Datasets | Encoder | Rec@5 | NDCG@5 | MRR@5 | Rec@10 | NDCG@10 | MRR@10 | Rec@20 | NDCG@20 | MRR@20 |
|----------|---------|-------|--------|-------|--------|--------|-------|--------|--------|-------|
| Zhihu    | Mean    | 0.0673| 0.0403 | 0.0315| 0.1282 | 0.0597 | 0.0394| 0.2372 | 0.0870 | 0.0467|
|          | Max     | 0.0652| 0.0393 | 0.0309| 0.1236 | 0.0580 | 0.0385| 0.2318 | 0.0850 | 0.0458|
|          | Att     | 0.0737| 0.0439 | 0.0342| 0.1364 | 0.0640 | 0.0424| 0.2493 | 0.0922 | 0.0500|
| MIND     | Mean    | 0.3998| 0.2416 | 0.1898| 0.6684 | 0.3282 | 0.2255| 0.8828 | 0.3829 | 0.2408|
|          | Max     | 0.3984| 0.2406 | 0.1889| 0.6631 | 0.3259 | 0.2241| 0.8762 | 0.3804 | 0.2394|
|          | Att     | 0.3977| 0.2399 | 0.1883| 0.6683 | 0.3271 | 0.2242| 0.8818 | 0.3817 | 0.2395|

Boldface denotes the highest score. Rec is short for Recall. “Mean,” “Max,” and “Att” denote three encoding strategies of mean pooling–, max pooling–, and self-attention–based encoding, respectively.

Table 4. Attack Performance with LSTM-Based Decoders

| Datasets | Encoder | Rec@5 | NDCG@5 | MRR@5 | Rec@10 | NDCG@10 | MRR@10 | Rec@20 | NDCG@20 | MRR@20 |
|----------|---------|-------|--------|-------|--------|--------|-------|--------|--------|-------|
| Zhihu    | Mean    | 0.0916| 0.0579 | 0.0470| 0.1634 | 0.0809 | 0.0563| 0.2868 | 0.1118 | 0.0647|
|          | Max     | 0.1102| 0.0690 | 0.0556| 0.1930 | 0.0955 | 0.0664| 0.3301 | 0.1299 | 0.0757|
|          | Att     | 0.1198| 0.0744 | 0.0596| 0.2063 | 0.1021 | 0.0709| 0.3495 | 0.1380 | 0.0806|
| MIND     | Mean    | 0.5861| 0.3591 | 0.2832| 0.8772 | 0.4541 | 0.3249| 0.9809 | 0.4811 | 0.3327|
|          | Max     | 0.5657| 0.3435 | 0.2711| 0.8719 | 0.4432 | 0.3127| 0.9837 | 0.4724 | 0.3212|
|          | Att     | 0.5335| 0.3353 | 0.2707| 0.8158 | 0.4269 | 0.3087| 0.9606 | 0.4644 | 0.3194|

Boldface denotes the highest score. Rec is short for Recall. “Mean,” “Max,” and “Att” denote three encoding strategies of mean pooling–, max pooling–, and self-attention–based encoding, respectively.

Table 5. Attack Performance with GRU-Based Decoders

| Datasets | Encoder | Rec@5 | NDCG@5 | MRR@5 | Rec@10 | NDCG@10 | MRR@10 | Rec@20 | NDCG@20 | MRR@20 |
|----------|---------|-------|--------|-------|--------|--------|-------|--------|--------|-------|
| Zhihu    | Mean    | 0.0875| 0.0542 | 0.0434| 0.1578 | 0.0767 | 0.0526| 0.2841 | 0.1084 | 0.0612|
|          | Max     | 0.1009| 0.0628 | 0.0504| 0.1792 | 0.0878 | 0.0606| 0.3114 | 0.1210 | 0.0695|
|          | Att     | 0.0833| 0.0512 | 0.0408| 0.1523 | 0.0733 | 0.0497| 0.2744 | 0.1038 | 0.0580|
| MIND     | Mean    | 0.5384| 0.3265 | 0.2575| 0.8429 | 0.4253 | 0.2987| 0.9739 | 0.4596 | 0.3086|
|          | Max     | 0.5586| 0.3471 | 0.2782| 0.8506 | 0.4422 | 0.3178| 0.9748 | 0.4745 | 0.3271|
|          | Att     | 0.5414| 0.3415 | 0.2763| 0.8110 | 0.4291 | 0.3126| 0.9561 | 0.4665 | 0.3233|

Boldface denotes the highest score. Rec is short for Recall. “Mean,” “Max,” and “Att” denote three encoding strategies of mean pooling–, max pooling–, and self-attention–based encoding, respectively.

without special mention. The item embedding size is set as \( d = 128 \). We train all models with the Adam optimizer \([30]\). The learning rate is set as 0.001. The dropout rate is tuned to 0.1. For attention-based encoding and transformer-based decoder, the hyperparameters of the multi-head self-attention are set as 2 heads with a total of 128 hidden neurons. The hidden size of the FFN is also set as 128. For the LSTM-based decoder and GRU-based decoder, the depth of recurrent layer is set to 1. We utilize the weight sharing technique \([23, 40]\) to tie the weights of the encoder item embedding and softmax layer item embedding in the decoder. For label smoothing, the \( \epsilon \) is set to the \( 1/|I| \). Each experiment is conducted 3 times and the average result is reported.

5.4 Attack Performance (RQ1)

Table 3 shows the attack performance with point-wise decoding on the two datasets. We can see that for point-wise decoding, the self-attention–based encoding achieves much better attack performance than mean pooling– and max pooling–based encoding on the Zhihu dataset. In contrast, for the MIND dataset, the three different encoding methods achieve similar attack performances.

The attack performance of the three sequence-wise decoding methods are shown in Tables 4–6. We can see from Table 4 that for the LSTM-based decoder, the self-attention–based encoding achieves the highest attack performance on the Zhihu dataset. On the MIND dataset, the mean pooling–based encoding achieves much better performance than max pooling– and
Table 6. Attack Performance with Transformer Decoders

| Datasets | Encoder | Rec@5 | NDCG@5 | MRR@5 | Rec@10 | NDCG@10 | MRR@10 | Rec@20 | NDCG@20 | MRR@20 |
|----------|---------|-------|--------|-------|--------|---------|--------|--------|---------|--------|
|          | Mean    | 0.1453 | 0.0946 | 0.0780 | 0.2359 | 0.1235 | 0.0898 | 0.3748 | 0.1585 | 0.0992 |
| Zhihu    | Max     | 0.1426 | 0.0920 | 0.0755 | 0.2339 | 0.1213 | 0.0874 | 0.3725 | 0.1561 | 0.0968 |
|          | Att     | 0.1344 | 0.0850 | 0.0699 | 0.2293 | 0.1161 | 0.0822 | 0.3721 | 0.1520 | 0.0919 |
| MIND     | Mean    | 0.5742 | 0.3714 | 0.3051 | 0.8232 | 0.4524 | 0.3369 | 0.9549 | 0.4864 | 0.3485 |
|          | Max     | 0.5758 | 0.3729 | 0.3066 | 0.8266 | 0.4536 | 0.3398 | 0.9519 | 0.4864 | 0.3494 |
|          | Att     | 0.5647 | 0.3639 | 0.2983 | 0.8119 | 0.4443 | 0.3316 | 0.9498 | 0.4798 | 0.3417 |

Boldface denotes the highest score. Rec is short for Recall. “Mean,” “Max,” and “Att” denote three encoding strategies of mean pooling–, max pooling- and self-attention–based encoding, respectively.

self-attention–based encoding. For GRU-based decoder performance shown in Table 5, we can see that the max-pooling encoding performance is better than the other two encoding methods on both datasets. The reason could be that the max-pooling encoding can select the most important features for the GRU-based decoder. Table 6 shows the attack results of the transformer decoder. We can see that for the transformer decoder, mean-pooling encoding achieves the best result on the Zhihu dataset. On the MIND dataset, max-pooling encoding achieves the highest attack performance.

Through the comparison between point-wise decoding (i.e., Table 3) and sequence-wise decoding (i.e., Tables 4–6), we can see that sequence-wise decoding achieves much better performance than point-wise decoding with the combination of all three encoders. These results indicate that considering the sequential order of user behaviors could further improve attack model performance.

Regarding the encoding methods, we can see that on the Zhihu dataset, self-attention–based encoding achieves the best performance when using point-wise decoding and the LSTM-based decoder. For the MIND dataset, mean-pooling encoding achieves the highest attack scores with point-wise decoding and the LSTM-based decoder. This demonstrates that for different datasets, we should choose different encoders to conduct the attack. We can also see that compared with the powerful self-attention–based encoding, the much simpler mean-pooling and max-pooling can still achieve comparable and even better attack performance. It further demonstrates the risk of privacy leakage since the attack can be performed without trivial and complex encoding methods.

In addition, we can see that the attack model performs much better on MIND than on Zhihu. The reason could be that the Zhihu dataset comes from an answer recommendation scenario. Each answer has an underlying latent question. As a result, the attack model could encounter difficulty in inferring the change of underlying questions, leading to relatively lower attack model performance than the MIND dataset.

Taking an overall look at the results of Tables 3–6, we can see that on the Zhihu dataset, the attack model can achieve the highest recall in the top-10 list of inference results: 23.59%. It indicates a great danger that more than 20% of user privacy can be exactly inferred from top-10 attack results. Regarding the MIND dataset, we can see that the privacy risk leakage further increases to the highest recall@10, 87.72%, and recall@20, 98.09%.

To validate this common risk in the recommendation, we do not focus on the trivial model design and, even though utilizing the simple encoder-decoder architecture, can also achieve great attack performance. The reason is that exposure data include enough information to infer the user’s historical behavior; even the mean-pooling encoder can learn the representation of exposure data easily.

To conclude, we can see that there is substantial user privacy leakage in recommender systems. A large amount of user past behavior privacy can be inferred from system exposure data through the simple encoder-decoder–based attack model.
5.5 Hyperparameter Study (RQ2)

In this subsection, we conduct experiments to see how the number of exposed items affect attack model performance. We illustrate the results when using the self-attention–based encoding method. The results of the other two encoding methods show a similar trend.

Figures 6(a), 6(b), 6(c), and 6(d) show the attack performance (Recall@10) of point-wise decoding, LSTM-based decoders, GRU-based decoders, and transformer decoders on Zhihu, respectively. We can see that when using point-wise decoding, different exposed item numbers lead to similar attack performance. However, when using sequence-wise decoding, the attack performance is much better and varies much more with different $N$. The reason could be that the recommendation scenario of Zhihu is answer recommendation. Each answer belongs to a latent question. Due to the fact that point-wise decoding does not model the sequential order of user behavior, point-wise decoding cannot capture the change of underlying questions of the answers, leading to a more flat and lower attack performance. However, sequence-wise decoding performs the inference based on the previous inference results. This method could have some capability to learn the change of latent underlying questions from the previous inference results. As a result, sequence-wise decoding achieves a much higher attack performance with the proper setting of $N$.

Figures 7(a), 7(b), 7(c), and 7(d) show the attack performance (Recall@10) of point-wise decoding and three different sequence-wise decoding methods on the MIND dataset, correspondingly. We can see that on the news recommendation dataset, both the point-wise decoding and sequence-wise decoding methods have more variation in attack performance with the different settings of $N$. 

ACM Transactions on Information Systems, Vol. 41, No. 3, Article 57. Publication date: February 2023.
In addition, we can see that on both datasets, the best attack performance is achieved when \( N = 10 \). This observation indicates that with fewer exposed items, the attack model may not learn strong signals to infer user behavior privacy. However, a too large number of exposed items could also introduce extra noise, which further confuses the model and downgrades the attack performance. \( N = 10 \) could be a proper setting for the attack.

### 5.6 Effect of Protection (RQ3)

In this subsection, we conduct experiments to verify the effect of the protection mechanism. We show the attack performance regarding recall and NDCG\(^6\); we also report the new recommendation accuracy. Figures 8 and 9 illustrate the results with different item replacement ratios using random-based position selection at the first stage. The results using similarity-based position selection with different item replacement ratios are shown in Figures 10 and 11. We can see that with the increase of the replacement proportion \( L \), attack performance drops dramatically. This

\(^6\)Results of MRR show the same trend with NDCG.
Fig. 10. Effect of the protection mechanism with similarity-based position selection on Zhihu. L is the proportion of replacement. Rec*, NDCG*, and Acc* denote the new attack recall, new attack NDCG, and new recommendation accuracy, respectively.

Fig. 11. Effect of the protection mechanism with similarity-based position selection on MIND. L is the proportion of replacement. Rec*, NDCG*, and Acc* denote the new attack recall, new attack NDCG, and new recommendation accuracy, respectively.

observation demonstrates that random exposure can help to alleviate user privacy leakage risk. However, we can also see from Figures 8(c), 9(c), 10(c), and 11(c) that recommendation accuracy decreases. It indicates a trade-off effect between recommendation accuracy and leakage risk. We can see from Figure 8 that the random-based position selection method with the uniform-based replacement mechanism alleviate about 40% leakage risk while maintaining about 80% recommendation accuracy when setting the replacement ratio $L$ to 0.2 in the Zhihu dataset.

Given the fact that users would click very few items in the exposed list, the designed similarity-based position selection method, which can keep the items that the user would like to click and then replace the other items with random or more diverse items, achieves better recommendation accuracy. We can see from the comparison between Figures 8(c) and 10(c) (also, Figures 9(c) and 11(c)) that compared with the random-based position selection method, the similarity-based selection method maintains higher recommendation accuracy with the increase of the replacement ratio $L$ when using uniform-based item replacement.

Figures 12 and 13 show the results of using similarity-based position selection at the first stage and uniform-based replacement at the second stage on two datasets. In contrast to the method
of replacing the dissimilar position items (i.e., similarity-based position selection method with uniform-based replacement in Figures 10 and 11), the similar position selection method selects positions with items matching well with user preference. The results show that it can alleviate user privacy leakage risk at the cost of recommendation accuracy degradation compared with replacing the dissimilar positions items. The reason could be that more similar positions items reveal much more user behavior information.

Regarding the item replacement strategy, we can see that uniform-based replacement is the most effective strategy to downgrade attack performance whereas in-batch popularity replacement and overall popularity replacement have a similar effect on attack performance. For recommendation accuracy, in-batch popularity replacement achieves the minimum accuracy sacrifice.

6 CONCLUSIONS AND FUTURE WORK

In this article, we have investigated the risk of user behavior privacy leakage in recommender systems. We have focused on answering the question of whether user past behavior privacy can
be inferred from system behavior data. We have conducted an attack model through an encoder-decoder architecture. We have proposed to utilize three different encoding methods to encode the system exposure data. We then presented point-wise decoding and three sequence-wise decoders to infer user past behaviors from the encoded representation. Experimental results on two real-world datasets have verified a great danger of privacy leakage in recommender systems. To alleviate the risk, we have proposed a two-stage protection mechanism based on infusing random items into the exposed item sets. We first select exposure positions based on random or item similarity at the first stage, and then replace exposed items on the corresponding positions with uniform or popularity sampled items. Experimental results have demonstrated a trade-off effect between the recommendation accuracy and the privacy leakage risk.

We hope that this work raises more community concerns regarding the protection of recommender system behavior data beyond focusing only on user perspectives. Compared with sparse user historical behavior, the large volume of system exposure data receives relatively less research attention. This work has proposed a new perspective regarding the attack and protection of system behavior data.

Future work includes investigating more advanced encoding and decoding methods to conduct attacks. In addition, exposure data are highly affected by various kinds of biases (e.g., the popularity bias). Thus, how to conduct a debiased attack model would also be a research direction. More importantly, a promising future direction is the design of more advanced protection methods with little sacrifice of recommendation accuracy. Finally, exploring the relationship across recommendation accuracy, privacy leakage risk, and recommendation diversity or novelty would be an interesting direction.

ACKNOWLEDGMENTS

All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Muhammad Ammad-ud din, Elena Ivannikova, A. Suleiman Khan, Were Oyomno, Qiang Fu, Eeik Kuan Tan, and Adrian Flanagan. 2019. Federated collaborative filtering for privacy-preserving personalized recommendation system. *arXiv: Information Retrieval* (2019).

[2] Ghazaleh Beigi, Ahmadreza Mosallanezhad, Ruocheng Guo, Hamidreza Alvari, Alexander Nou, and Huan Liu. 2020. Privacy-aware recommendation with private-attribute protection using adversarial learning. In *13th ACM International Conference on Web Search and Data Mining (WSDM’20), Houston, TX, February, 2020* (2020), 34–42.

[3] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. 2021. Secure federated matrix factorization. *IEEE Intell. Syst.* 36, 5 (2021), 11–20.

[4] Jiawei Chen, Hande Dong, Yang Qiu, Xiangnan He, Xin Xin, Liang Chen, Guli Lin, and Keping Yang. 2021. AutoDebias: Learning to debias for recommendation. In *SIGIR*. ACM, 21–30.

[5] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and debias in recommender system: A survey and future directions. *ACM Trans. Inf. Syst.* Just Accepted (October 2022). https://doi.org/10.1145/3564284

[6] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. 2019. Top-\(k\) off-policy correction for a REINFORCE recommender system. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 7–10.

[8] Kyunghyun Cho, Bart van Merrienboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*. ACL, 1724–1734.
[9] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. 2021. A survey on adversarial recommender systems: from attack/defense strategies to generative adversarial networks. Computing Surveys (2021). https://doi.org/10.1145/3497729

[10] Jingtao Ding, Fuli Feng, Xiangnan He, Guanhui Yu, Yong Li, and Depeng Jin. 2018. An improved sampler for Bayesian personalized ranking by leveraging view data. In Companion Proceedings of the The Web Conference 2018 (Lycon, France) (WWW ’18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 13–14. https://doi.org/10.1145/3184558.3186905

[11] Jingtao Ding, Yuan Hu, Xiangnan He, Yong Li, and Depeng Jin. 2019. Reinforced negative sampling for recommendation with exposure data. In IJCAI. ijcai.org, 2230–2236.

[12] Chen Gao, Chao Huang, Dongsheng Lin, Depeng Jin, and Yong Li. 2020. DPLCF: Differentially private local collaborative filtering. In SIGIR. ACM, 961–970.

[13] Zhengqi Jiang, Xinyu Liu, Qiang Li, Yu Li, and Dong Guo. 2021. PrivItem2Vec: A privacy-preserving algorithm for top-N recommendation. International Journal of Distributed Sensor Networks 17, 12 (2021). https://doi.org/10.1177/15501477211061250

[14] Jiali Han, Yun Ma, Qiaozhu Mei, and Xuanzhe Liu. 2021. DeepRec: On-device deep learning for privacy-preserving sequential recommendation in mobile commerce. In Proceedings of the Web Conference 2021. 900–911.

[15] Bin Hao, Min Zhang, Weizhi Ma, Shaoyun Shi, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2021. Large-scale rich context query and recommendation dataset in online knowledge-sharing. arXiv:2106.06467 [cs.IR]

[16] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. Bag of tricks for image classification with convolutional neural networks. In CVPR. Computer Vision Foundation/IEEE, 558–567.

[17] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. 355–364.

[18] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGC N: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 639–648.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web. 173–182.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In ICLR (Poster).

[21] Balázs Hidasi and Domonkos Tikk. 2016. General factorization framework for context-aware recommendations. Data Mining and Knowledge Discovery 30, 2 (2016), 342–371.

[22] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 368–377.

[23] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In ICLR (Poster). OpenReview.net.

[24] Arjan J. P. Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, Reginald L. Lagendijk, and Qiang Tang. 2013. Privacy in recommender systems. In Social Media Retrieval. Springer, 263–281.

[25] Ray Jiang, Silvia Chiappa, Tor Lattimore, András György, and Pushmeet Kohli. 2019. Degenerate feedback loops in recommender systems. In Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES’19) (Honolulu, HI). ACM, New York, NY, 383–390. https://doi.org/10.1145/3306618.3314288

[26] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored item similarity models for top-N recommender systems. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 659–667.

[27] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In 2018 IEEE International Conference on Data Mining (ICDM’18). IEEE, 197–206.

[28] Farwa K. Khan, Adrian Flanagan, Kuan Eeik Tan, Zareen Alamgir, and Muhammad Ammad-Ud-Din. 2021. A payload optimization method for federated recommender systems. In Recsys. 432–442.

[29] Jinsu Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In Recsys. 432–442.

[30] Jianxun Lian, Fuzheng Zhang, Min Hou, Hongwei Wang, Xing Xie, and Guangzhong Sun. 2017. On the User Behavior Leakage from Recommender Exposure 57:23

[31] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. Computer 42, 8 (2009), 30–37.

[32] Jianxun Lian, Fuzheng Zhang, Min Hou, Hongwei Wang, Xing Xie, and Guangzhong Sun. 2017. Practical lessons for job recommendations in the cold-start scenario. In Proceedings of the Recommender Systems Challenge 2017 (Como, Italy) (RecSys Challenge’17). ACM, New York, NY, Article 4, 6 pages. https://doi.org/10.1145/3124791.3124794
[58] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. 2020. Self-supervised reinforcement learning for recommender systems. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 931–940.

[59] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: High-order proximity for implicit recommendation. In Recsys. 140–144.

[60] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. 2019. Federated learning. Synthesis Lectures on Artificial Intelligence and Machine Learning 13, 3 (2019), 1–207.

[61] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-efficient transfer from sequential behaviors for user modeling and recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1469–1478.

[62] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In Proceedings of the 12th ACM International Conference on Web Search and Data Mining. ACM, 582–590.

[63] Minxing Zhang, Zhaochun Ren, Zihan Wang, Pengjie Ren, Zhumin Chen, Pengfei Hu, and Yang Zhang. 2021. Membership inference attacks against recommender systems. In CCS. ACM, 864–879.

[64] Shijie Zhang and Hongzhi Yin. 2022. Comprehensive privacy analysis on federated recommender system against attribute inference attacks. CoRR abs/2205.11857 (2022).

[65] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Lizhen Cui, and Xiangliang Zhang. 2021. Graph embedding for recommendation against attribute inference attacks. In WWW. ACM/IW3C2, 3002–3014.

[66] Yang Zhang, Fuli Feng, Xiangnan He, Tianxin Wei, Chonggang Song, Guohui Ling, and Yongdong Zhang. 2021. Causal intervention for leveraging popularity bias in recommendation. In SIGIR. ACM, 11–20.

Received 31 May 2022; revised 15 August 2022; accepted 4 October 2022