MINDSim: User Simulator for News Recommenders

Xufang Luo
Microsoft Research Asia
xufluo@microsoft.com

Zheng Liu
Microsoft Research Asia
zheng.liu@microsoft.com

Shitao Xiao*
Beijing University of Posts and Telecommunications
stxiao@bupt.edu.cn

Xing Xie
Microsoft Research Asia
xing.xie@microsoft.com

Dongsheng Li
Microsoft Research Asia
dongsheng.li@microsoft.com

ABSTRACT

Recommender system is playing an increasingly important role in online news platforms nowadays. Recently, there is a growing demand for applying reinforcement learning (RL) algorithms to news recommendation aiming to maximize long-term and/or non-differentiable objectives. However, without an interactive simulated environment, it is extremely costly to develop powerful RL agents for news recommendation. In this paper, we build a user simulator, namely *MINDSim*, for news recommendation. Targeting at new user generation and corresponding behavior simulation, we first construct a hidden space for users using a generative adversarial network, so that new users can be generated by sampling from this hidden space. To capture complex and fast user interest drifts over time, we adopt an encoder-decoder architecture, which takes the clicked news during the simulation as input and outputs the new user interests for the next period of time. Finally, we build the *MINDSim* simulator using MIcrosoft News Dataset (MIND), and extensive experimental results on this large-scale real-world dataset demonstrate that *MINDSim* can simulate the behaviors of real users with high quality.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

news recommendation, user simulator, reinforcement learning

ACM Reference Format:
Xufang Luo, Zheng Liu, Shitao Xiao, Xing Xie, and Dongsheng Li. 2022. *MINDSim*: User Simulator for News Recommenders. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512080

*Work was done when the third author was the intern with Microsoft Research Asia.

1 INTRODUCTION

Recommender systems are becoming the vital channels of getting information, especially for some rapidly updated and plentiful information, e.g., news [25, 29]. Hence, a growing number of techniques are developed for news recommendations. Recently, applying reinforcement learning (RL) algorithms to recommenders draws much attention from both industry and academia [18, 19, 49], because RL algorithms are widely and successfully used in many decision-making tasks, such as video games [28] and locomotion [36], and they are naturally designed for optimizing some long-term [40] and non-differentiable objectives [32], such as dwell time and revisit, which are crucial for recommenders, but have not been extensively considered by conventional recommendation algorithms.

However, applying RL algorithms in real-world recommendation services is extremely costly [8, 38]. First, training RL agents requires an online interactive environment, where agents improve themselves via trial-and-error. But the low-quality recommendations caused by model training will lead to bad user experiences, making it prohibitive to directly train them in an online production environment. Second, deploying a new recommendation policy to a production service involves lots of engineering efforts, which are too costly for algorithmic experimentation. Previous research works [2, 8, 38] try to solve the above problems by creating an online environment via simulating users' behaviors, but the following two challenges in news recommendation cannot be effectively addressed by these methods.

First, recently proposed user simulators for other kinds of recommenders [35, 38, 47] cannot effectively handle the unique "new item" issue in news recommendation. Different from conventional recommenders (e.g., movie and product) in which new items are relatively rare, news recommenders need to deal with diverse contents generated almost every minute and then quickly become out-of-date within a few days or even hours [49]. This "new item" issue is a well-known challenge for news recommendation, which also raises challenges for the user simulator. Targeting at the "new item" issue of news recommendation, the simulator should be capable of modeling user interests over fast-changing and highly diverse news collections.

Second, although news recommendation algorithms can successfully handle the above "new item" issue, they are not capable of generating "new user" for simulation. Existing news recommendation algorithms can only predict the future behaviors of known users, i.e., they are not able to go beyond the fixed user set and thus cannot generate new users with similar behaviors as real users.

Adopting a news recommender as a simulator will prevent the simulator from covering more users with diverse interests, causing the trained RL agents to easily overfit. In addition, only simulating the behaviors of existing users may introduce modeling biases for the minority groups of users [23].

In this paper, we propose a user simulator, namely *MINDSim*, for news recommendation, which consists of the following two stages to tackle the above two challenges, respectively. 1) *MINDSim* constructs a hidden space for users, so that new users can be generated by sampling from this space. More specifically, *MINDSim* maps each user to the hidden space by encoding the user's history clicked news into a hidden vector using an auto-encoder, and the hidden vector can be used to represent this user's interests. Then, *MINDSim* trains a Generative Adversarial Network (GAN) using the real users' hidden vectors in the dataset to learn the distribution of user interests. Thus, the trained GAN can effectively model the hidden space of user interests and generate users by sampling from the hidden space. 2) *MINDSim* employs the encoder-decoder architecture to capture the fast user interest drifts over diverse newly appeared news. First, the encoder is used to aggregate the embeddings of the clicked news after the user is generated. Here, embeddings are generated by a large pre-trained language model with news articles as inputs. Thus, user interest changes caused by recent clicks can be reflected in the outputs of the encoder. Then, starting from the encoder output, the process that news is sequentially selected from a candidate news set is modeled as a decoding process. Besides, once the candidate news set is updated, clicked news in the last set is sent to the encoder, and then the decoding process is performed on the new set. Therefore, instant user interest changes within the current candidate news set are also considered.

To evaluate the performance, we build the *MINDSim* simulator using MIcrosoft News Dataset (MIND) [45], which is a large-scale news recommendation dataset collected from anonymized logs of real users. Extensive experimental results show that *MINDSim* can well simulate the behaviors of real users.

We summarize our contributions as follows:

- We propose a novel user simulator called *MINDSim*. To the best of our knowledge, this is the first user simulator tailored for online news recommendation, which provides an online interactive environment to train RL agents.
- We construct a hidden space for users using GAN, and thus can sample new users beyond the fixed dataset from the hidden space.
- We use an encoder-decoder architecture to effectively capture the complex and fast user interest drifts over time in news recommendation.
- We build the simulator with large-scale offline data. Extensive experiments demonstrate that *MINDSim* generates high-quality simulations of user behaviors.

2 SETTINGS AND FRAMEWORK

2.1 Interactions between Users and the News Recommender

We focus on a typical and real-world setting in news recommendation and describe the interactions between the users and the news recommender in this subsection. As shown in Fig. 1, once a user

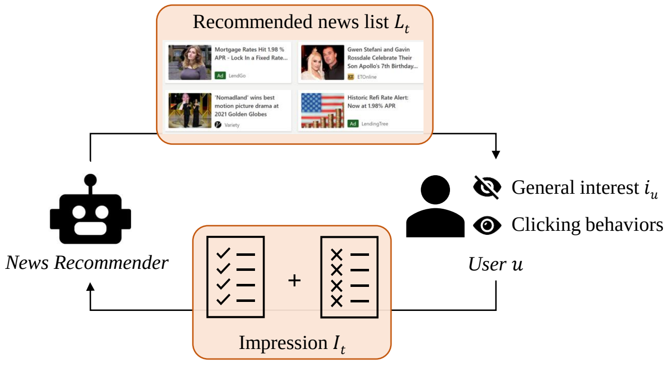

Recommended news list L_t
Mortgage Rates Hit 1.98 % APR - Lock In a Fixed Rate...
Ad LendGo
Gwen Stefani and Gavin Rossdale Celebrate Their Son Apollo's 7th Birthday...
ETOnline
'Nomadland' wins best motion picture drama at 2021 Golden Globes
Variety
Historic Refi Rate Alert: Now at 1.98% APR
Ad LendingTree
General interest i_u
Clicking behaviors
News Recommender
User u
+
Impression I_t

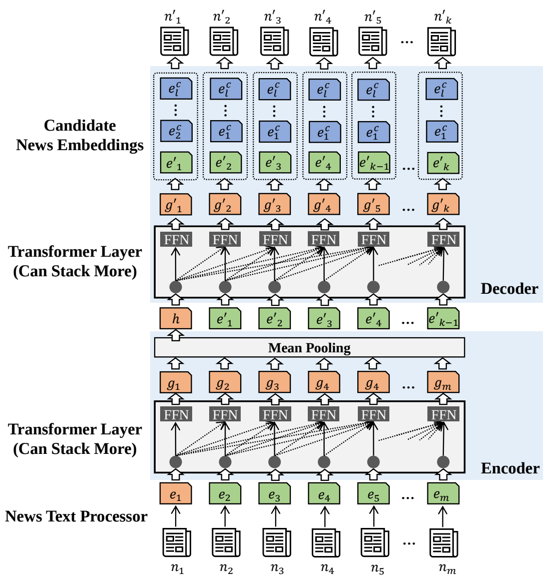
Figure 1: Interactions between the news recommender and the user u at time step t.

opens a news website or application, the news recommender will select a list of news from a large candidate set that is continually updated with the latest news, and present them to the user. Next, the user will selectively click and read them according to his/her interests. Note that users have personalized interests, which are unknown to the recommender. Then, the recommender will improve the selected news list by considering the clicked news in previous interactions, and present them to the user. The interaction goes on until the user leaves the news website or application. Here, a list of news with clicked/unclicked labels composes an *impression* $I = (n_1^+, \ldots, n_i^+, n_{i+1}^-, \ldots, n_{|I|}^-)$, where n^+ and n^- are clicked and unclicked news respectively, and $|\cdot|$ denotes the size of the list, and thus one round of interactions forms an impression.

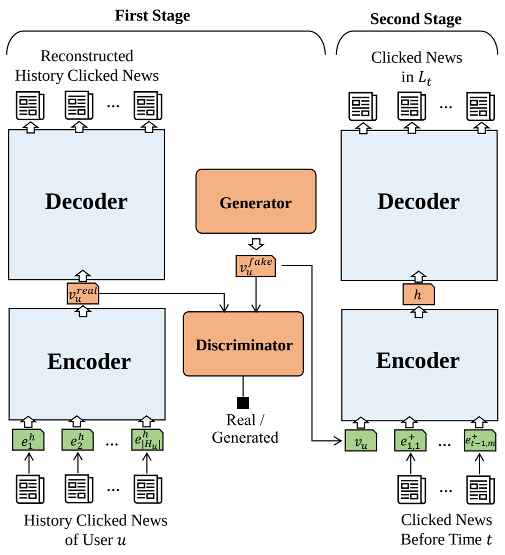
2.2 Partially Observable Markov Decision Process

We next give a mathematical framework to formulate the interaction between the user and the news recommender described above. First, since we try to apply RL algorithms to optimize the recommendation policy, the news recommender and users naturally correspond to the RL agent and the environment in general RL interaction system respectively. Then, under this setting, users with personalized interests lead to different state transition functions and Markov Decision Processes (MDPs), which raises great difficulties for recommendation policy learning at the agent side.

Here, *we propose to introduce the partially observable Markov decision process (POMDP) to solve this problem.* Particularly, we merge the user-specific information into states. Note that this information is used for modeling different users at the user side, and should be unknown for the recommender. Hence, it is appropriate to use POMDP rather than MDP to model the interaction and handle the unknown information. Specifically, POMDP is a generalization of the MDP, which emphasizes that states in the environment are not fully observable for the RL agent and the agent can only receive observations from the environment. Formally, it can be defined as a 7-tuple $\langle S, A, T, R, \Omega, O, \gamma \rangle$, where S is the state space, A is the action space, $T : S \times A \mapsto S$ is the state transition function, R is the reward function, Ω is the observation space, $O : S \mapsto \Omega$ is the observation function based on states, γ is the discount factor for



(a) Model Architecture.



(b) Two Stages.

Figure 2: Illustration of the proposed *MINDSim* simulator for news recommenders.

future rewards. Targeting at the news recommendation problem, we define the key elements in the 7-tuple as follows:

Observation o contains the information that the recommender can see and needs to be considered when taking actions. We design the observation as the impression sequence. At time step t, $o_t = (I_1, \dots, I_{|o_t|})$ denotes impressions regarding the specific user before this time step, and $o_{t+1} = o_t \oplus I'$, which means that the observation is updated with the newly generated impression I' at time step t. Observation o_1 can be empty if the targeted user is new.

State s contains full information at the user side. We assume that each user u has a general unchanged interest i_u, and state s is designed as the combination of the observation and the user interest, i.e., $s_t = (o_t, i_u)$. Note that for a new user, state $s_1 = (\emptyset, i_u)$, and user interest changes are modeled in the transition function.

Action a at time step t is a list of news $L_t = (n_{t,1}, \dots, n_{t,|L_t|})$ that the recommender presents to the user, i.e., an impression without clicked and unclicked labels.

State transition function T models clicking behaviours of users. It takes state s_{t-1} and action a_{t-1} as inputs. Since action is an impression without labels and the state is updated with the newly collected impression, the output of the transition function, i.e., the next state s_t, corresponds to the clicking behaviors of the user, providing each impression with clicked or unclicked labels.

2.3 Problem Definition

In this paper, we focus on building a user simulator for news recommendation, and the problem is defined as follows. Generally, given an offline dataset containing some users and their behaviors, we learn a state transition function in the POMDP, representing the environment side in the RL interaction system. Learning a reward function can be defined as another problem, and is not the focus of this paper. We use $\mathcal{U} = \{u_1, u_2, \dots, u_{|\mathcal{U}|}\}$ to denote the user set in the offline dataset, and a specific user in this set is denoted as u [1]. Each user has a clicking history sequence $H_u = (n_1^h, n_2^h, \dots, n_{|H_u|}^h)$ and an impression sequence $IMP_u = (I_1, I_2, \dots, I_{|IMP_u|})$ ordered in time, where n_*^h is one of the clicked news in the history and the impression is defined in Section 2.1.

For the state transition function learning problem, we notice that state $s_t = (o_t, u)$ changes over time and u is unchanged. Thus, this problem can be naturally decomposed into two sub-problems, i.e., general user interest modeling and clicking behavior simulation. They are handled in two stages respectively by *MINDSim* as follows.

In the 1st stage, we construct a hidden space for users, and each user u can be represented by a hidden vector v_u sampled from the space. Here, we first use the user's history clicks H_u to construct v_u, and thus v_u contains information regarding this user's general interests. Then, a generative model is learned to model the distribution of v_u, representing the hidden space for users, and user hidden vectors can be sampled using this generative model.

In the 2nd stage, we use an encoder-decoder architecture to take complex and fast user interest changes into consideration, and based on the sophisticated modeling for user interest changes, *MINDSim* can simulate clicking behaviors close to real users. Here, for time step t, inputs of model in this stage include the user-specified hidden vector v_u, impressions $(I_1, I_2, \dots, I_{t-1})$ collected before time step t, and candidate news list L_t provided by the recommender (i.e., all news in I_t without labels). The output of the model is the clicked news list, and thus an action L_t (recommended news list) is converted into an impression I_t (news list with clicked or unclicked labels) in the 2nd stage.

[1]In this paper, user index is omitted for simplify.

3 METHODOLOGY

In this section, we introduce *MINDSim* in detail. First, since both two stages can be generally regarded as sequence-to-sequence processes, we use the same network architecture for both stages with only minor differences, and we describe the architecture in the first subsection. Then, we show how this architecture can be applied to these two stages in the following two subsections separately. Finally, we summarize the training and inference process of *MINDSim*.

3.1 Model Architecture

Generally, we use two architectures in *MINDSim*. First, a large pre-trained language model is utilized as the news text processor, and every news n can be converted to a vector e using it. Second, sequences of news embedding vectors can be processed by the encoder-decoder. Both architectures are built upon transformer layers. As a result, we'll briefly introduce the transformer layer first; then, we'll introduce the detailed workflows of the text processor and the encoder-decoder architecture. Finally, we'll present the loss function used to train the encoder-decoder model.

3.1.1 Transformer Layer. Since transformer-based architectures show great power in various domains, we adopt the transformer layer as the basic component of the model architecture. We use similar architecture as in [41] and give a brief introduction here, with more details shown in Appendix B.

One transformer layer contains two successive modules, i.e., multi-head self-attention and position-wise feed-forward network, denoted as MultiHeadAtten and FFN, respectively. Multiple transformer layers can be stacked to enable learning a more complex relationship between inputs, and calculations in one layer are conducted as follows:

$$\begin{aligned} \hat{\mathbf{H}}_i &= \mathrm{LN}(\mathrm{MultiHeadAtten}(\mathbf{H}_i) + \mathbf{H}_i), \\ \mathbf{H}_{i+1} &= \mathrm{LN}(\mathrm{FFN}(\hat{\mathbf{H}}_i) + \hat{\mathbf{H}}_i), \end{aligned} \tag{1}$$

in which $\mathbf{H}_i$ and $\mathbf{H}_{i+1}$ are the input and output hidden states of the i-th layer; LN indicates the layer-norm. Here, $\mathbf{H}_1$ is the input embedding matrix, obtained by stacking the embedding sequence $(e_1 + p_1, e_2 + p_2, \dots, e_m + p_m)$, where e_i is the embedding of the i-th input (word or news), and p_i is the learnable positional embedding to address chronological sequence.

3.1.2 News Text Processor. We use a pre-trained language model, which is widely recognized for its high capability in natural language understanding, as the news text processor. Particularly, we leverage one of the SOTA BERT-like text encoders, UniLMv2-base [3] as our backbone network. In this model, the input word embeddings are iteratively processed by 12 cascaded Transformer layers. We perform mean-pooling over the last layer's hidden states ($\mathbf{H}_{13}$) and linearly transform the result into our news embedding:

$$e = \mathbf{W}^e \mathrm{MeanPooling}(\mathbf{H}_{13}), \tag{2}$$

where $\mathbf{W}^e$ is the linear transformation matrix.

3.1.3 Encoder-Decoder. As shown in Fig. 2(a), the proposed encoder-decoder architecture is also mainly composed of transformer layers. Generally, the encoder maps a sequence of news into a hidden vector, and the decoder generates a sequence of news starting from the hidden vector.

Encoder: The input sequence of news $(n_1, n_2, \dots, n_m)$ is first converted into embeddings $(e_1, e_2, \dots, e_m)$ by the text processor. Next, stacked transformer layers can learn complex transition patterns between them, and covert it to a hidden vector sequence, denoted as $(g_1, g_2, \dots, g_m)$. Then, we use a mean pooling operator to aggregate these hidden vectors, i.e.,

$$h = \mathrm{MeanPooling}(g_1, g_2, \dots, g_m). \tag{3}$$

Decoder: As a usual decoding process, the output news embedding e'_t at time step t is used as the input for the next time step. And here hidden vector h is taken as the input for the first time step. However, the output news cannot be directly chosen from a fixed candidate set via a softmax layer as common practice in NLP tasks, since the word set is usually fixed, but the candidate news set is continually updated with the latest news. Therefore, the output news is selected via ranking in *MINDSim*. More specifically, stacked transformer layers take the input news embedding sequence, and output hidden vectors $(g'_1, g'_2, \dots, g'_k)$. Meanwhile, the candidate news set $(n^c_1, n^c_2, \dots, n^c_l)$ is converted into candidate embeddings $(e^c_1, e^c_2, \dots, e^c_l)$ using the text processor. After that, we compute the score for each candidate news in every position via dot product between every hidden vector g' and every candidate embedding e^c:

$$\mathbf{M}^{score} = \mathbf{G}'(\mathbf{E}^c)^\intercal = \begin{bmatrix} g'_1 \\ g'_2 \\ \vdots \\ g'_k \end{bmatrix} \left[(e^c_1)^\intercal; (e^c_2)^\intercal; \dots; (e^c_l)^\intercal \right]. \tag{4}$$

where $\mathbf{G}' \in \mathbb{R}^{k \times d_e}$, $(\mathbf{E}^c)^\intercal \in \mathbb{R}^{d_e \times l}$, and d_e is the embedding size. Thus, we obtain scores for all candidate news (with index 1 to l) at each position (with index 1 to k). Then, the candidate news are ranked according to the matrix $\mathbf{M}^{score} \in \mathbb{R}^{k \times l}$. At the first position, news with the max score in this position (1st row of $\mathbf{M}^{score}$) is ranked as the first one in the final result. Next, for the second position, the one with the max score here (2nd row of $\mathbf{M}^{score}$) among the remaining candidates expect the first chosen one is ranked as the second one in the final result. Then, by following such a rule, we can obtain news embeddings at every position, forming $(e'_1, e'_2, \dots, e'_k)$, which is shifted and used as inputs for the decoder. Note that we use the teacher forcing strategy [22] in the training stage, and use outputted hidden vectors by the transformer layers as inputs during test time, since ground-truth is not available during test time. Finally, corresponding news sequence of the selected embeddings $(n'_1, n'_2, \dots, n'_k)$ are taken as decoded news.

By treating the clicked news sequence like a natural language sentence, information and relationships between previously clicked news can be effectively captured by the model to predict the next clicked news. In this way, the decoding process can model the transition dynamics among news in clicked sequences. Such sophisticated modeling can help *MINDSim* to have a precise understanding of user interest changes.

3.1.4 Model Learning. We apply cross entropy loss to train the model. Suppose that there are $|D|$ sequence pairs in the dataset, then the loss function is defined as follows:

$$\mathcal{L} = \sum_{a=1}^{|D|} \sum_{b=1}^{k} \left(- \sum_{c=1}^{l} y_{b,c}^{(a)} \log p_{a,b,c}^{score} \right)$$
$$p_{a,b}^{score} = \text{softmax}\left(m_{a,b}^{score} \right), \tag{5}$$

where $m_{a,b}^{score}$ is the b-th row of the a-th score matrix $\mathbf{M}_a^{score}$, and $y_{a,b}$ is the corresponded one-hot vector of the ground-truth label.

We now summarize the encoder-decoder architecture used in *MINDSim*. First, the input news sequence is converted to embedding vectors by the text processor. Next, we use stacked transformer layers followed by a mean pooling operator to encode information from inputs into an internal hidden vector. After that, the decoding process starts from the internal hidden vector, and a sequence of hidden vectors is generated by stacked transformer layers in the decoder. Finally, news in the candidate set are ranked according to scores calculated by the dot product of generated hidden vectors and candidate news embeddings, and the model is optimized via the cross entropy loss. Note that we use separate parameters for the encoder and decoder to address that they have different effects.

3.2 1st Stage: User Hidden Space Construction

As shown in Fig. 2(b), in this stage, the user hidden space is learned by the following two steps.

3.2.1 Reconstruction. Given a user u and the corresponding clicking history sequence $H_u = (n_1^h, n_2^h, \ldots, n_{|H_u|}^h)$, the user hidden vector v_u is generated by the encoder and learned via the reconstruction task. Specifically, both the input and output sequence to the encoder-decoder are H_u during training. The candidate news set consists of H_u as clicked news and unclicked news sampled from other users' histories. Thus, after the mean pooling, the hidden vector h_u contains information of all news in H_u. Moreover, to control the scale of values in the hidden vector, we further use a tanh activation function, i.e., $v_u = \tanh(h_u)$. user hidden vector v_u is taken as the start vector for the decoding process.

Therefore, v_u aggregates information from the user's historical clicked news, and can reflect the user's general interest i_u. So far, we find a way to map a user to an example in the hidden space. Next, the hidden space is constructed via learning the distribution of v_u.

3.2.2 GAN. We use Wasserstein GAN [1] to learn the distribution of v_u and then generate new user's hidden vectors by sampling from the distribution. Real users' hidden vectors generated with user histories in the dataset are denoted as v_u^{real}, while those generated by the generator G are denoted as v_u^{fake}. And the discriminator D tries to classify real or fake hidden vectors. Then, we have the following learning objective:

$$\min_G \max_D \mathop{\mathbb{E}}_{v_u^{real} \sim \mathbb{P}_{real}} \left[D(v_u^{real}) \right] - \mathop{\mathbb{E}}_{v_u^{fake} \sim \mathbb{P}_{fake}} \left[D(v_u^{fake}) \right], \tag{6}$$

where $\mathbb{P}_{fake}$ is the model distribution implicitly defined by $v_u^{fake} = G(z)$ and z is the noise vector. Besides, we also adopt gradient penalty to improve GAN training [13]. After training, the user hidden space is constructed and we can use the learned generator to sample user hidden vectors beyond the offline dataset.

3.3 2nd Stage: User Behavior Prediction

Based on the learned general user interests, we next use the model to predict user behaviors with user interests changes modeled via the encoder-decoder architecture.

For time step t, user u, and its hidden vector v_u, the collected impressions of this user before time step t is denoted as $(I_1, I_2, \ldots, I_{t-1})$, and current recommended news list is denoted as L_t. Every impression contains clicked news n^+ and unclicked new n^-, and news in L_t are unlabeled. The encoder takes the general user interest and clicked news before t into consideration, and accordingly gathers information in them. Hence, the input sequence in this stage is $(n_{1,1}^+, n_{1,2}^+, \ldots, n_{2,1}^+, n_{2,2}^+, \ldots, n_{t-1,1}^+, n_{t-1,2}^+, \ldots)$. The decoding process simulates user clicking behaviors, so the clicked news in L_t consist of the output sequence, denoted as $(n_{t,1}^+, n_{t,2}^+, \ldots, n_{t,k}^+)$, and L_t is the candidate news set, since user can only choose from recommended news. Therefore, candidate news set L_t is converted into an impression I_t, and the observation is updated accordingly.

In this stage, *MINDSim* models different kinds of user interest changes via the encoder-decoder architecture, to address complex and fast changes in the news recommendation scenario. First, clicked news after the user is generated is considered by the encoder, and thus the hidden vector after the mean pooling always contains information regarding updated user interests. Then, for recommended news in the current time step, user behavior, in which news is clicked one by one, is modeled as a decoding process. Here, previously clicked news can influence the next click. For instance, if a user's interest is the same as before, the user may keep reading the sports news, or if the user's interest changes quickly, he/she may read entertainment news after reading sports news. Various kinds of changes are modeled as transitions between candidate news via the decoding process, and transition patterns can be learned from the data.

3.4 Summary

The training process in *MINDSim* is summarized here. First, using H_u of all users, a reconstruction model is learned to map H_u to v_u. Then, based on v_u generated from existing users in the dataset, the encoder-decoder model in the 2nd stage is learned using impression sequences of those users. Besides, we build a GAN based on v_u of users in the dataset to enable *MINDSim* to generate new users.

During inference, we first use GAN to generate a user hidden vector, and then use the model in the 2nd stage to encode newly clicked news and predict clicks via decoding. In this way, users are generated using GAN, and their behaviors are obtained using the encoder-decoder in the 2nd stage. Hence, *MINDSim* can simulate users and be used as the environment in the interaction in Fig. 1.

4 EXPERIMENTS

In this section, we conduct detailed experiments to answer the following research questions (RQs):

RQ1: Can *MINDSim* model user behaviors appropriately and accurately?

RQ2: Can *MINDSim* simulate users whose behaviors are similar to real users?

RQ3: Can *MINDSim* go beyond the fixed offline dataset?

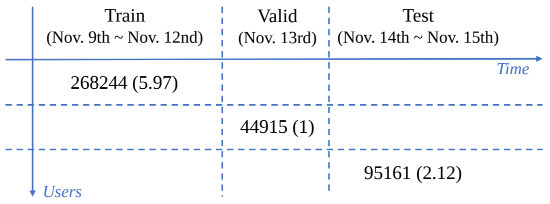


Figure 3: Separation of the dataset. Number of users are the proportion are labelled.

4.1 Data Processing and Statistics

We use a large-scale English news recommendation dataset, i.e., MIND, in the experiments. MIND was collected from anonymized behavior logs of real users during a week (from Nov. 9th, 2019 to Nov. 15th, 2019) and serves as a benchmark dataset for news recommendation in many research works [9, 42, 44]. Note that we only use impression and news data in MIND dataset, while entities and relations are ignored in this work since we don't focus on language understanding. We also include some discussions on news recommendation datasets in Appendix C.1.

We process the original MIND dataset by following steps to make it suitable for building a user simulator. First, we group impressions in MIND by the user id. In this way, we obtain a large user set, and for every user, the history clicked news before the collection week, and related impressions during the collection week are listed. Each impression contains clicked news and unclicked news, and impressions are sorted by time. Then, we utilize some filters to remove some unqualified data. Specifically, users with less than two historical clicked news or less than two impressions are removed from the dataset. Finally, because we cannot access true labels of MIND's testing data, we merge users in training and validation data, and create a new separation for them. As shown in Fig. 3, besides users, the impression time also has a disjoint separation to avoid information leakage, since news is extremely time-sensitive. More specifically, if #impressions of the user in the training data duration (i.e., Nov. 9th to Nov. 12nd) is highest compared with other durations, this user will be regarded as training data. Then, according to this rule, 268244 users with their impressions from Nov. 9th to Nov. 12nd are used as training data, while their impressions at other time are discarded. This process is also applied to validation and testing data, and thus we obtain a new separation for MIND dataset, which is used in all experiments in this paper.

The average #news per impression in the processed dataset is 46.6, which is also the average size of the changing action space. Besides, the average #clicked news per impression is 1.7. Such sparse labels indicate the difficulties for algorithms to handle.

4.2 Predictive Performance

To clarify that the utilized encoder-decoder architecture is predictive for user behaviors and can model user behaviors appropriately and accurately, we conduct the following experiments. First, we train the auto-encoder in the 1st stage by reconstructing users' history clicks in the training data. Then, we use the learned user hidden vectors to train the encoder-decoder in the 2nd stage. Finally, we run the model trained in the 2nd stage on the testing data.

Table 1: Predictive performance on testing data.

Method	nDCG			MRR			Precision		
	@1	@5	@10	@1	@5	@10	@1	@5	@10
FM	0.310	0.419	0.448	0.185	0.314	0.339	0.185	0.121	0.088
DeepFM	0.313	0.424	0.452	0.190	0.320	0.344	0.190	0.123	0.088
BPR	0.314	0.423	0.453	0.189	0.319	0.344	0.189	0.123	0.088
NCF	0.317	0.425	0.454	0.194	0.323	0.347	0.194	0.122	0.088
SASRec	0.309	0.416	0.444	0.184	0.310	0.335	0.184	0.118	0.085
GAUM	0.298	0.405	0.434	0.168	0.294	0.320	0.168	0.115	0.085
MINDSim	**0.320**	**0.429**	**0.458**	**0.198**	**0.327**	**0.352**	**0.198**	**0.124**	**0.089**

To assess if the model can predict real user behaviors accurately, performances are evaluated via ranking metrics described below. Note that we do not incorporate GAN here, since the ground-truth clicked news for newly generated users is unknown. Moreover, due to the limitation of computational resources, we cannot fine-tune the news text processor (i.e., the large pre-trained language model) for every method below. Hence, we first fine-tune the text processor using SpeedyFeed [46] on the training data, and then fix its weights for all methods. The news embedding size is 64.

4.2.1 Compared Baselines. We use multiple widely used state-of-the-arts as baselines:

- Factorization Machines (**FM**) [33]: FM is a general predictor for recommendation, which combines the advantages of SVM with factorization models.
- Deep Factorization Machines (**DeepFM**) [14]: DeepFM leverages deep neural networks for feature learning to make FM more powerful.
- Bayesian Personalized Ranking (**BPR**) [34]: BPR uses a generic optimization criterion for personalized ranking to maximize posterior.
- Neural Collaborative Filtering (**NCF**) [15]: NCF replaces the inner product in matrix factorization with neural networks to learn the user-item interaction function.
- Self-Attentive Sequential Recommendation (**SASRec**) [21]: SASRec is a self-attention based sequential model which seeks to identify relevant items from user's history clicks for predicting the next item.
- Generative Adversarial User Model (**GAUM**) [8]: GAUM is an imitation learning based user simulator that tries to learn user behaviors in a generative adversarial way.

For FM, DeepFM, BPR, and NCF, user embeddings are required and here we calculate the user embedding via mean pooling over embeddings of user's clicked news. And to make the model be aware of new information, the user embedding is updated once the new impression is generated. Besides, since weights of the news embedding model are fixed, we further introduce two additional linear layers, whose hidden dimension is the same as the embedding dimension, serving as embedding layers for users and items respectively in these four baselines.

4.2.2 Evaluation Metrics. We adopt three metrics to evaluate the performance of methods. The first one is normalized Discounted Cumulative Gain (nDCG), which takes true and predicted relevance scores as inputs to measure ranking quality. Here, suppose there are N clicked news, the relevance score of the news ranked at m is set to $N - m + 1$. Then, nDCG@k is calculated as:

$$\text{nDCG@k} = \frac{\text{DCG}_\text{k}}{\text{IDCG}_\text{k}} = \frac{\sum_{i=1}^{k} \frac{r_i^{pred}}{\log(1+i)}}{\text{IDCG}_\text{k}}, \tag{7}$$

where r_i^{pred} is the predicted relevance scores, and IDCG_k can be calculated as DCG_k using r_i^{true}. The second used metric is mean reciprocal rank (MRR), which is calculated as:

$$\text{MRR} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{1}{rank_i}, \tag{8}$$

where $|D|$ is the number of sequences in the dataset, and $rank_i$ is the rank position of the first news in the ground-truth ranking list of the ith sequence. The third metric is Precision. With these three metrics, we can have a reasonable and comprehensive evaluation of the ranking quality of methods.

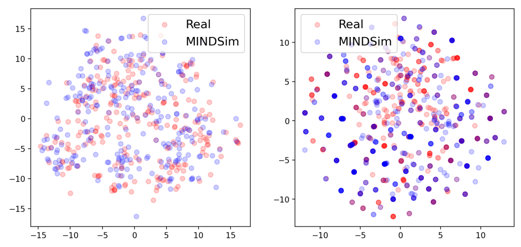
4.2.3 Implementation Details. Our implementation is based on RecBole [43], which is a unified recommendation library and provides many existing models. Hyper-parameters are set to be the same across methods, and can be found in Appendix C.

4.2.4 Results. Results of all methods are listed in Table 1. We can observe that *MINDSim* outperforms baselines, which demonstrates that the encoder-decoder architecture in *MINDSim* can accurately predict user's clicks and help to appropriately model user behaviors with high quality. Results also show that *MINDSim* has strength over other user simulator, which indicates the difficulty of building a simulator for news recommendation, and the importance of capturing fast and complex user interest changes in this task.

4.3 Simulation Results

We assess the simulation quality of *MINDSim* and analyze the similarities between generated users and real users in this experiment.

We use the *MINDSim* model obtained from experiments in the last sub-section and conduct the following steps. First, generated hidden vectors by the auto-encoder in the 1st stage are used to train the GAN. Then, we use the trained GAN and the encoder-decoder in the 2nd stage to generate users and their clicking behaviors. In our problem, the recommender provides recommended news list L to users, and then users' clicking behaviors can be obtained based on L. Thus, the recommender should keep the same for fairness when comparing the behaviors of real users and the simulator. However, in the dataset, recommended news lists are provided by Microsoft News service, so we cannot effortlessly access this full policy and use it to interact with newly generated users by *MINDSim*. Here, we solve this problem by using the testing data to approximately recover the recommendation policy as follows. First, users in the testing data are mapped to the hidden space via the trained auto-encoder. Then, when a new user hidden vector is generated by the GAN, we compute the Euclidean distance between the generated user and all user vectors in the testing data, and recommended lists of the user with minimum distance are chosen as data used for the newly generated user. Hence, the clicking behaviors of real users and the simulator can be compared under the same recommender.



(a) User hidden vectors. (b) Clicked news embeddings.

Figure 4: Visualization of simulation results.

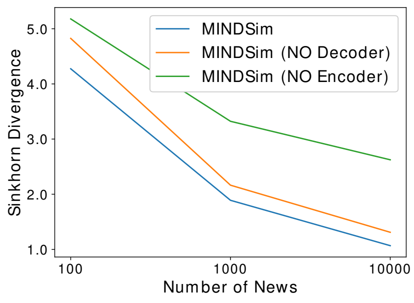


Figure 5: Sinkhorn Divergences of different models.

4.3.1 Visualization. As shown in Fig. 4, we provide two-dimensional t-SNE visualizations for both user hidden vectors generated by the GAN, and clicked news embeddings generated by the GAN and encoder-decoder. Fig. 4(a) shows that the auto-encoder in the 1st stage can successfully map users to the hidden space, and the GAN can learn the distribution of user hidden vectors. Hence, new users can always be sampled from the space via the trained GAN. Fig. 4(b) shows that the clicked news of generated users is similar to real users in the dataset. This result demonstrates that *MINDSim* can generate users whose behaviors are similar to real users, and thus the core problem in building a user simulator for news recommendation is addressed.

4.3.2 Quantitative Comparison. We further provide some quantified comparisons to assess the simulation quality and analyze the importance of every component in *MINDSim*. Generally, besides GAN, *MINDSim* consists of two parts, i.e., encoder and decoder. Hence, we set two baselines here:

- MINDSim (NO Encoder): The encoder is used for aggregating relevant information in impressions before the current time step. Here, the encoder is removed and thus, such information will not be considered by the model.
- MINDSim (NO Decoder): The decoder trained in the 2nd stage is removed here, and outputs of the encoder (i.e., hidden vector h after the mean pooling operator) are directly used to calculate the score for candidate news via dot product.

Two baseline models are trained using the training data via identical steps as *MINDSim*. And clicking behaviors are generated using testing data as before.

Table 2: Performances in generalization experiments. Improvements are emphasized in italic with the "+" sign.

Method	Data	nDCG		MRR		Precision	
		@1	@10	@1	@10	@1	@10
FM	Original	0.291	0.424	0.156	0.307	0.156	0.083
	Augmented	*0.295 +*	*0.430 +*	*0.162 +*	*0.313 +*	*0.162 +*	*0.084 +*
DeepFM	Original	0.300	0.434	0.170	0.320	0.170	0.084
	Augmented	*0.303 +*	*0.438 +*	*0.175 +*	*0.326 +*	*0.175 +*	*0.085 +*
BPR	Original	0.300	0.434	0.170	0.319	0.170	0.084
	Augmented	*0.302 +*	*0.437 +*	*0.172 +*	*0.323 +*	*0.172 +*	*0.085 +*
NCF	Original	0.303	0.438	0.174	0.326	0.174	*0.085 +*
	Augmented	*0.305 +*	*0.439 +*	*0.177 +*	*0.327 +*	*0.177 +*	0.084
SASRec	Original	0.293	0.423	0.158	0.305	0.158	0.080
	Augmented	*0.300 +*	*0.430 +*	*0.169 +*	*0.315 +*	*0.169 +*	*0.082 +*

We use the unbiased Sinkhorn divergence [12] to quantify the similarity between behaviors of generated users and real users. Sinkhorn divergences, used in many generation tasks [30, 31], are positive and definite approximations of Optimal Transport distances, and can be estimated by samples. Since user behaviors are samples generated from some underlying distributions, smaller Sinkhorn divergences indicate more similar behaviors.

We calculate Sinkhorn divergences with the different number of samples (i.e., news) to eliminate sampling bias in estimating, and show them in Fig. 5. We can observe that: 1) User behaviors generated by *MINDSim* have much lower divergences than baselines regardless of the number of samples, which demonstrates that *MINDSim* can simulate users and their behaviors better, and indicates the importance of both the encoder and the decoder; 2) MINDSim (NO Encoder) performs much worse than other two methods, which shows that taking information of continuously generated impressions into consideration is important for behavior prediction, and is necessary for modeling fast user interest changes in building the user simulator of news recommendation.

4.4 Generalization Performances

We design the following experiments to show that *MINDSim* can generate new users beyond the fixed offline dataset, which also answers RQ3 at the beginning of the experiment section.

First, we randomly sample a small set of users (i.e., 2000 users) from the training data. Next, we augment this subset with retrained *MINDSim*. Specifically, we generate 1000 user hidden vectors using GAN, representing 1000 new users. Their clicking behaviors are simulated using the same way as visualization, i.e., we find the user with the smallest Euclidean distance among 2000 users, and use the corresponding impressions as recommended lists for the new user. Therefore, we obtain 1000 users whose hidden vectors and clicking behaviors are generated by *MINDSim*. Then, we use the original dataset containing 2000 users and the augmented dataset containing 3000 users (i.e., 2000 real users and 1000 simulated users) to train some recommendation models (i.e., FM, DeepFM, BPR, NCF, and SASRec) respectively. Finally, model performances are assessed on the testing data using ranking metrics.

We list the ranking metrics of all methods in Table 2. Results show *MINDSim* can help to promote almost every metric for all recommendation methods on the testing data. The promotion on generalization performance demonstrates that *MINDSim* can go beyond and enhance the dataset by simulating new users and their clicking behaviors, which are similar but different from the fixed dataset. Hence, the core priority in building a user simulator is effectively tackled by *MINDSim*.

5 RELATED WORK

News Recommendation Algorithms: Online news platforms have become important media of information access. Considering that there are massive volumes of news articles on the news platforms and fresh content are being generated at a rapid speed, it is critical to make personalized news recommendation based on each user's individual interest. In recent years, a great deal of works has been dedicated to the development of news recommendation systems [24, 26, 29, 49]. Our work differs from these works in that we focus on building the user simulator for news recommenders, rather than the recommender itself.

RL for Recommendation: Research works regarding RL for recommendation can be generally categorized into two groups. In the first one, to optimize some long-term objectives, researchers try to apply RL algorithms to different recommendation scenarios and problems that have not been applied before. These scenarios include E-commerce [4, 18], online personalized news recommendation [49], video recommendation [6] and so on. The second type of research works target on solving some problems when RL is applied to the scenario. Problems include but are not limited to class imbalance (negative feedback is much more than positive ones) [48], dynamic and unstable environments with high-variance and distribution shifting [7], extremely large action spaces [5]. The target of these works is different from the focus of this paper, since we try to build a user simulator for news recommendation. Learning RL agents using the simulator is the future work.

User Simulators for Recommendation: This line of works is mainly related to model-based RL algorithms in the RL taxonomy, where the dynamics model is learned and used to interact with the RL agent to improve sample efficiency. Some works put the dynamics model learning and the RL agent learning problem into one framework. For example, [2, 8, 38] use the generative framework based on Generative Adversarial Imitation Learning [17]. [50] uses a World Model as the customer simulator, and use Dyna-Q [39] based algorithm to train the recommendation policy.

Some other works only focus on building the simulator or environment reconstruction. [47] proposes a user simulator for evaluating conversational recommenders. [35] develops RecoGym, which is an environment for product recommendation in online advertising, but it is not a full RL set-up. [37] proposes an deconfounded environment reconstruction method for multi-agent RL based recommender. [20] develops a recommender simulation platform, which emphasizes configurability and targets on general recommenders.

Our work differs from these works in two aspects. First, we focus on the news recommendation problem, which has not been addressed by previous works. And we further introduce transformer layers and the encoder-decoder architecture to handle fast and complex user interest changes. Second, we use a POMDP to model the recommendation policy learning problem and handle users with different interests, and thus building the user simulator can be regarded as a supervised learning problem.

6 CONCLUSION

We propose a user simulator, namely *MINDSim*, for news recommenders. Different from previous works for user simulators, *MINDSim* designs two stages to tackle the "new item" issue in news recommendation and the "new user" problem in building the simulator, respectively. In the 1st stage, *MINDSim* constructs a hidden space for users, so that new users can be generated by sampling from this hidden space. In the 2nd stage, *MINDSim* leverages an encoder-decoder architecture to capture the fast user interest drifts due to instantly appearing new items. Extensive experiments on the large-scale MIND dataset well demonstrate that *MINDSim* can simulate behaviors of real users with high quality.

REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *ICML*.

[2] Xueying Bai, Jian Guan, and Hongning Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. *NeurIPS* (2019).

[3] Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Jianfeng Gao, Songhao Piao, Ming Zhou, et al. 2020. Unilmv2: Pseudo-masked language models for unified language model pre-training. In *ICML*.

[4] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for e-commerce. In *WWW*.

[5] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *AAAI*.

[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 *WSDM*.

[7] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *KDD*.

[8] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML*.

[9] Jamell Dacon and Haochen Liu. 2021. Does Gender Matter in the News? Detecting and Examining Gender Bias in News Articles. In *Companion Proceedings of the Web Conference 2021*. 385–392.

[10] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *ACL*.

[11] Linhao Dong, Shuang Xu, and Bo Xu. 2018. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *ICASSP*.

[12] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouve, and Gabriel Peyré. 2019. Interpolating between Optimal Transport and MMD using Sinkhorn Divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 2681–2690.

[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved Training of Wasserstein GANs. In *NeurIPS*.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.

[16] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[17] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *NeurIPS*.

[18] 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 *KDD*.

[19] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SlateQ: A tractable decomposition for reinforcement learning with recommendation sets. *arXiv preprint* (2019).

[20] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. *arXiv preprint arXiv:1909.04847* (2019).

[21] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*.

[22] John F Kolen and Stefan C Kremer. 2001. *A field guide to dynamical recurrent networks*. John Wiley & Sons.

[23] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-Bandit-Based News Article Recommendation Algorithms. In *WSDM*.

[24] Jianxun Lian, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2018. Towards Better Representation Learning for Personalized News Recommendation: a Multi-Channel Deep Fusion Approach.. In *IJCAI*. 3805–3811.

[25] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. 2010. Personalized news recommendation based on click behavior. In *Proceedings of the 15th international conference on Intelligent user interfaces*. 31–40.

[26] Zheng Liu, Yu Xing, Fangzhao Wu, Mingxiao An, and Xing Xie. 2019. Hi-Fi Ark: Deep User Representation via High-Fidelity Archive Network.. In *IJCAI*.

[27] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML*.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[29] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *KDD*.

[30] Giorgio Patrini, Rianne van den Berg, Patrick Forre, Marcello Carioni, Samarth Bhargav, Max Welling, Tim Genewein, and Frank Nielsen. 2020. Sinkhorn autoencoders. In *UAI*.

[31] Gabriel Peyré, Marco Cuturi, et al. 2019. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning* 11, 5-6 (2019), 355–607.

[32] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732* (2015).

[33] Steffen Rendle. 2010. Factorization machines. In *ICDM*. IEEE.

[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.

[35] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720* (2018).

[36] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *ICML*. PMLR.

[37] Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. 2019. Environment reconstruction with hidden confounders for reinforcement learning based recommendation. In *KDD*.

[38] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *AAAI*.

[39] Richard S Sutton. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*. Elsevier, 216–224.

[40] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *NeurIPS* (2017).

[42] Sanne Vrijenhoek, Mesut Kaya, Nadia Metoui, Judith Möller, Daan Odijk, and Natali Helberger. 2021. Recommenders with a mission: assessing diversity in news recommendations. In *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*. 173–183.

[43] et al. Wayne Xin Zhao. 2020. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. *arXiv preprint arXiv:2011.01731* (2020).

[44] Chuhan Wu, Fangzhao Wu, Yongfeng Huang, and Xing Xie. 2020. Neural news recommendation with negative feedback. *CCF Transactions on Pervasive Computing and Interaction* 2, 3 (2020), 178–188.

[45] Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, and Ming Zhou. 2020. MIND: A Large-scale Dataset for News Recommendation. In *ACL*. https://msnews.github.io/.

[46] Shitao Xiao, Zheng Liu, Yingxia Shao, Tao Di, and Xing Xie. 2021. Training Large-Scale News Recommenders with Pretrained Language Models in the Loop. *arXiv preprint arXiv:2102.09268* (2021).

[47] Shuo Zhang and Krisztian Balog. 2020. Evaluating Conversational Recommender Systems via User Simulation. In *KDD*.
[48] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *KDD*.
[49] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *WWW*.
[50] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation. In *WSDM*.

A NOTATIONS

Notations used in this paper are summarized in Table 3.

Table 3: Notations used in this paper.

Notation	Meaning
$u, \mathcal{U}$	User, User set
i_u	Unchanged general interest of user u
v_u	Hidden vector for user u
L	Recommended news list
I	Impression
H_u	News sequence for clicking history of user u
n, n^+, n^-	News, Clicked news, Unclicked news
n^c, n^h	Candidate news, Historical news
e	Embedding vector of the news

B TRANSFORMER LAYER

Given the input sequence of embedding $(e_1, e_2, \ldots, e_m)$, where $e_i \in \mathbb{R}^d$ is embedding vector of the i-th item and m is the sequence length, we first stack them and obtain an embedding matrix $\hat{\mathbf{E}} \in \mathbb{R}^{m\times d}$. Next, following [21], we add a learnable positional embedding to address that items in the sequence are chronological. Hence, we can obtain the embedding matrix $\mathbf{E}$ as follows:

$$\mathbf{E} = \begin{bmatrix} e_1 + p_1 \\ e_2 + p_2 \\ \vdots \\ e_m + p_m \end{bmatrix} \in \mathbb{R}^{m\times d},$$

where p_i is generated by a learnable positional embedding look-up table $\mathbf{P} \in \mathbb{R}^{m\times d}$ with the position index i as the input. After that, the embedding matrix $\mathbf{E}$ is sent to the transformer layer, which is consisted of two sub-layers as follows.

Multi-Head Self-Attention: Self-attention mechanism has been successfully applied in many tasks with sequential inputs [11, 41]. It automatically learns weights to aggregate the input sequence, and emphasizes important information for the task via adjusting weights. Besides, self-attention mechanism has shown strong power in handling sequential inputs with variable lengths [10]. Hence, we applied it in *MINDSim* to process news sequences of different users with variable lengths, and use it to learn news embeddings. More specifically, self-attention operation process input embedding matrix $\mathbf{E}$ as follows:

$$\begin{aligned} &\text{SelfAttention}\,(\mathbf{E}) \\ &= \text{Attention}\left(\mathbf{E}\mathbf{W}^Q, \mathbf{E}\mathbf{W}^K, \mathbf{E}\mathbf{W}^V\right) \\ &= \text{softmax}\left(\frac{\left(\mathbf{E}\mathbf{W}^Q\right)\left(\mathbf{E}\mathbf{W}^Q\right)^T}{\sqrt{d}}\right)\left(\mathbf{E}\mathbf{W}^V\right), \end{aligned} \tag{9}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d\times d}$ are learnable matrices to project embedding matrix $\mathbf{E}$ into query, key and value, respectively. Besides, in the proposed encoder-decoder architecture, we also use a left-to-right uni-directional attention mask to take the natural of sequence into consideration.

Moreover, the multi-head attention is used as follows:

$$\begin{aligned}\mathbf{F} &= \text{MultiHead}\left(\mathbf{E}\right)\\ &= \left[\text{head}_1; \text{head}_2; \ldots; \text{head}_\text{h}\right]\mathbf{W}^O,\end{aligned} \tag{10}$$

where each head is a self-attention operator in Equation (9), and $\mathbf{W}^O \in \mathbb{R}^{hd \times d}$ is another learnable projection matrix for the concatenation of all attention outputs.

Position-Wise Feed-Forward Network: After multi-head self-attention layer, a position-wise feed-forward network is applied to endow the model with nonlinearity and interactions between different dimensions as follows:

$$g_i = \text{FFN}(f_i) = \text{GELU}\left(f_i\mathbf{W}^{(1)} + b^{(1)}\right)\mathbf{W}^{(2)} + b^{(2)}, \tag{11}$$

where f_i is the i-th output of the multi-head self-attention layer, GELU is a Gaussian Error Linear Unit (GELU) activation [16], $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times 4d}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{4d \times d}$ are weight matrices, $b^{(1)} \in \mathbb{R}^{4d}$, $b^{(2)} \in \mathbb{R}^{d}$ are bias. Weights and bias are learnable, and shared across all positions.

For the output of each sub-layer, we employ some additional operations, which have been proved to be effective in deep neural networks. First, dropout is applied, and followed by a residual connection. After that, layer normalization is also utilized to stabilize training. Moreover, two sub-layers above together with these additional operations can stack more to enable learning a more complex relationship between inputs. Finally, a hidden vector sequence $(g_1, g_2, \ldots, g_m)$, where $g_i \in \mathbb{R}^d$, is obtained after staked transformer layers.

C EXPERIMENT DETAILS

C.1 Discussions on the Dataset

We discuss existing new recommendation datasets, and elaborate the reason for choosing MIND dataset in experiments in this subsection.

First, this paper targets at building the first user simulator tailored for online news recommendation, to the best of our knowledge, MIND is the only publicly available dataset with enough information that can satisfy unique requirements for reaching the target of this paper. Other datasets cannot meet requirements for the following reasons. 1) Building a user simulator requires organizing the dataset by users, and hence, user id is the necessary information, but it is not provided in some datasets (e.g., Yahoo News). 2) To provide essential information for interactions between users and the news recommender, impressions must contain clicked news, as well as unclicked ones presented to users, and then the decoding process in the 2$^{\text{nd}}$ stage is performed on the candidate news set consisting of those presented news. Although some datasets (e.g., SmartMedia Adressa in Norwegian, Kaggle dataset from Globo.com in Portuguese) contain clicked news, unclicked but presented news are missed in these datasets. Besides, rich historical clicked news, used for constructing hidden vectors for users, are also missed in these datasets. 3) Some other news recommendation datasets focus on different aspects, rather than online personalized news recommendation. For example, some datasets (e.g., BuzzFeed News, Fakeddit, ...) focus on fake news detection. Reuters Corpora and 20 Newsgroups are used for news classification. Hacker News contains news and comments regarding IT industry, but sequential impressions with clicking behaviors are not shown in this dataset. YOW Dataset focuses on explicit and implicit feedbacks, but it only contains 383 articles and 25 users, and detailed information about sequential impressions is also missing. 4) Some other datasets are not available now, such as the plista dataset. A related challenge (i.e., CLEF-NewsREEL Challenge) to the plista dataset is obsolete now. Besides, the language used in this dataset (not English) and the lack of rich historical clicks for users also obstruct the usage of this dataset for building the user simulator.

Second, we would like to highlight that MIND dataset is representative for news recommendation. It contains about 160k English news articles and more than 15 million impression logs generated by 1 million users. Each log contains clicked and unclicked news, and historical news of this user. Such a large-scale dataset with abundant information is a convincing benchmark to test the performance of algorithms related to news recommendation.

C.2 Hyper-parameters

Hyper-parameters are set to be same across methods. We set initial learning rate to 0.001, and use Adam to adaptively change the learning rate during training. Batch size is set to 128. The training process is ended when the early stopping criteria is met, and the step for early stopping is set to 10. The model with the best performance on validation data is chosen and we report its results on testing data. For GAUM, we follow their open source implementation, and use the same model architecture and configurations. For *MINDSim* and SASRec, the number of transformer layers is 3 and the number of head is 2, and the dropout probability is set to 0.5.

C.3 Implementation Details of GAN

Some implementation details of GAN in *MINDSim* are listed here. The generator and discriminator of the GAN are both composed by 4 linear layers, with layer normalization and leaky ReLU [27] added between two layers. The size of the noise vector z is 100, and dimensions of layers in the generator and discriminator are 100-128-256-128-64 and 64-128-256-128-1, respectively. GAN is trained using Adam optimizer with initial learning rate 0.0001.

C.4 Computational Costs

Experiments are run on a single 6-core machine with one V100 GPU, and each run costs about 2 hours.