A Large-scale Friend Suggestion Architecture

1st Lin Zhang*
International Digital Economy Academy (IDEA)
Shenzhen, China
zhanglin@idea.edu.cn

2nd Rui Li*
Harbin Institute of Technology
Shenzhen, China
19s152089@stu.hit.edu.cn

***Abstract*—Online social as an extension of traditional life plays an important role in our daily lives. Users often seek out new friends that have significant similarities such as interests and habits, motivating us to exploit such online information to suggest friends to users. In this work, we focus on friend suggestion in online game platforms because in-game social quality significantly correlates with player engagement, determining game experience. Unlike a typical recommendation system that depends on item-user interactions, in our setting, user-user interactions do not depend on each other. Meanwhile, user preferences change rapidly due to fast changing game environment. There has been little work on designing friend suggestion when facing these difficulties, and for the first time we aim to tackle this in large scale online games. Motivated by the fast changing online game environment, we formulate this problem as friend ranking by modeling the evolution of similarity among users, exploiting the long-term and short-term feature of users in games. Our experiments on large-scale game datasets with several million users demonstrate that our proposed model achieves superior performance over other competing baselines.**

I. Introduction

With the rapid development of the World Wide Web, most of the human activities are migrated to the Internet, such as game [33] and social [15]. Online service providers often develop efficient toolboxes to improve our online experience by analyzing our activities. A popular practice is recommender systems [24], which analyze user behavior like viewing and purchasing, and then recommend the items to users so as to fit the obtained profiles of the individual users. Apart from the need of items, we naturally desire to make friends because friendship constitutes a critical part of humanity. In recent years, the development of recommendation systems is mostly concentrated in user-item setting [3], however, recommending strangers to make friends is also a desirable feature in many applications like players in games and dating applications. *How can we suggest new friends to users in an online setting when their interactions are sparse, mostly unavailable in reality?* In this work, we focus on online games and refer to this setting as friend suggestion [20].

From the theory of homophily in sociology [16], we know that people tend to associate with people that have similar behaviors to each other, and people that we friend with tend to behave like each other over time. This motivates us that the key for the friend suggestion task is to find people that are similar to each other, and therefore we need to address the problem of estimating the similarity between users. To be able to estimate similarity, information about user profiles [19], [18] and behaviors [5], [17] has to be taken into account. It provides two different perspectives to model user preferences on decision making, including long-term and short-term intents. Users from online games generate abundant data consisting of both perspectives. Specifically, individual game players generate rich online activities from time to time, and many of them are collected in a sequential order that reflects user dynamics, capturing users' short-term changes. By contrast, players' profiles provide a long-term perspective to describe players, since these contain slow change or fixed information such as gender and age, which often have a predominant influence on players' decisions. By fusing long-term and short-term information, we can obtain a more comprehensive representation of users and build a model on top of it to learn the similarity between users.

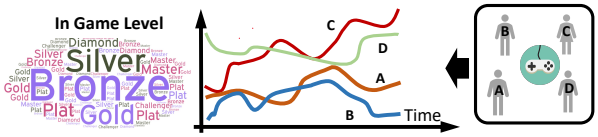


Fig. 1: Online games often provide fast updates to the social environment, and players tend to react quickly to changes in games. For example, one common feature of users is their game level, which changes rapidly and so that the preference on choosing friends changes accordingly. This phenomenon suggests that a good friend suggestion system should adapt to this fast changing preference as opposed to a static user preference in a user-item recommendation setting.

One popular way to exploit the aforementioned short-term and long-term features is by sequential recommendation [4] from traditional e-commerce recommendation settings, leveraging sequences of viewing or purchasing items in online shopping platforms. One may argue that we can collect user-user interaction sequences and apply existing sequential recommendation models [36]. However, the direct interaction behavior of users is often sparser than that of user-item interactions [32] because common online websites or mobile apps are often not designed for social interaction directly. Beyond the sparseness in interaction, users' behaviors in games may also be sparse due to the inactivity of large

*Authors are equally contributed.

proportion of users, which is a typical phenomenon in games. Particularly, interactions among users has significant difference to that of user-item setting. On one hand, a user-item interaction sequence has consecutive dependencies over each other, reflecting a user's sequential decisions [27]. Although user-user interactions happened sequentially, the consecutive interactions are not necessarily depending on each other as those in a user-item interaction sequence, suggesting that we can not simply treat them as a logical sequence and use these existing sequential recommendation models to address our problem. On the other hand, a user's preference over items changes slowly, since the general taste of a user is often static with no or slow change over time [30]. While this may not hold for players in games because of the fast updating environment in games, as shown in Fig. 1. In contrast to a real-world human that takes years to develop his/her taste, the role of a player in the game changes rapidly to adapt to the surrounding, making his/her preferences about the decision evolves quickly rather than static. The question then arises: *how can we deal with friend suggestion in game platforms where users' preferences change rapidly, along with non-logical and sparse user-user interactions?*

To address the aforementioned issue, we propose a novel end-to-end solution that leverages user behavior evolution with the proposed similarity evolution modeling, called Deep Similarity Evolutionary neural Network (DSEN). Specifically, we first learn a low-dimensional embedding from user features at each time point through a sequence sensitive model, fusing the long-term information and the short-term information as the unified embedding. We then characterize the similarity between users over time by computing they similarity through an efficient generalized dot product model. Next, we model the evolution of user preference over time using the obtained similarity along the time line implicitly, with the goal of inferring the current preference from the past.

To sum up, we have the following contributions:
1, **Novelty.** We develop a novel end-to-end friend suggestion framework by integrating both long-term and short-term information. To the best of our knowledge, this can be the first work on this task without relying on interactions among users, and also the first application to online game scenarios.
2, **Efficiency.** We evaluate our model on large scale data generated from a popular online game with several millions of users, and show superior results over competing methods.

II. Related work

We now brief review the related work in the literature. We can categorize the related into the following topics.

Friend suggestion: Existing friend suggestion methods often rely on mature social networks, assuming that users have solid and frequent connections with others, then learning features from this graph structure to measure the similarity between users, such as using mutual social influence [34], using scholar interaction [29], using social network [22]. In [10], authors developed a social community-based model to deal with friend suggestions by analyzing users' co-occurrences in neighbors. In [21], they proposed a friend suggestion algorithm that uses a user's implicit social graph to generate a friend group. Their method apply to any interaction-based social network. Despite its effectiveness, such methods may not fit in our setting from two perspectives: One is that we may not have access to stable social networks, and another is that user-user interaction is sparse in games. Some players even intentionally avoid engaging with friends and familiarity, thus making the social network useless. Meanwhile, Graph Neural Networks (GNNs) [1] are gaining more and more attention and have been applied to friend recommendation by supplying powerful embeddings [22]. Despite showing promising results, these methods rely on clear interactions and well-defined graphs. Existing methods may not be applicable when no solid social network is available.

Sequential recommendation system: One related topic is sequential recommendation, focusing on predicting the next user interaction with items [12], [26] by learning user interaction sequences in the history. Prior to the era of deep leaning, researchers have applied Markov chain theory to sequence modeling and make prediction on the chain [13]. Such models assume that the next actions depend on the most up-to-date actions, therefore these fail to capture long-term dependencies. More recently, models based on Recurrent Neural Networks (RNN) have overcome the limits of Markov chain and gained significant progress [31]. Beyond RNNs, its variants including Long Short-Term Memory (LSTM) [11] and Gated Recurrent Unit (GRU) [7] achieve further improvement by addressing issues in conventional RNNs. Beyond such recurrent-based models, recent advanced models like TransFormer and Bert show better ability to capture the user's short-term and long-term interests from a sequence of user-item interactions [14], [23]. Besides, researchers employ GNNs to address this problem [28], converting the user's sequence data into a sequence graph. Yet, as these sequences are fundamentally different from ours, their interaction between users and items is an individual's behavior.

User interest model: Another area of research that has attracted much attention is user interest modeling, that is, how to model user preferences based on the user's historical click sequence, which can be applied to click-through rate (CTR) estimation models. Deep learning based methods have achieved competitive results in CTR tasks, and recent works typically focus on modeling user latent interest from user historical click sequences [37], [35]. DIN [37] employees the attention mechanism to model target items capturing the differentiated interests of users. DIEN [35] proposes an interest extraction layer to capture the temporal interest from a historical behavior sequence and adopts an interest evolving layer to model the interest evolving process that is relative to the target item. Unlike user behavior sequences in E-commerce/advertising scenes, user interactive behaviors in online games are relatively sparse, but indeed there are relatively rich game behavior sequences. Such classical user interest CTR models cannot be directly applied to friend application scenarios. How to

TABLE I: Notation

Symbols	Definitions and Descriptions
p	rating prediction
$\mathbf{S}_{u_i}$	activity sequences of user i
$\mathbf{H^{(t)}}$	user's hidden state representation at first layer at time t
$\mathbf{R_{u_i}^{(t)}}$	user's profile feature at time t
$l_{i,j}$	pairwise link features between u_i and u_j
$e^{(t)}$	user's embedding after feature concatenation at time t
$M(\mathbf{e_i}, \mathbf{e_j})$	multi-view similarity between u_i and u_j
$g(\cdot)$	user-to-user top-level embedding

utilize sequences of matchmaking/combat behaviors in game scenarios to recommend strangers is the focus of our research in this paper.

III. Deep Neural Friend suggestion

A. Problem Formulation

In this section, we first introduce multiple concepts related to friend suggestions in an online game platform. In a large-scale multiplayer online game, individual user generates a series of their in-game behaviors (recharge, consumption, gaming and social interaction, etc.) are recorded and stored daily, to generate the users' personalization features. We model these data to produce users' profiles to express users' long-term personalization and short-term personalization. The short-term personalization is from the user's instant online activities. For players in a game, this contains their actions in a game, such as tactical skills, moving traces and so on. Formally, these can be represented as sequences that collecting from t time slots: $\mathbf{S}_i = [x^{(1)}, x^{(2)}, ..., x^{(t)}]$, capturing the evolution of users' actions over time. Whereas, long-term personalization often includes features that change is slow or constant over a long time period, such as profiles. In addition to the user's own characteristics, the static interaction relationship between users, such as the number of common friends, the number of games, and the common team, etc., can reflect the close relationship between pairs of users, called pairwise link features.

In particular, we know little about the social relations between users, and the dynamic interactions between users are often sparse in our setting, which is often the case of suggesting strangers to users in games. This setting is rarely considered in the literature, therefore existing solutions are often not applicable to this case. This work aims to address the friend suggestion problem in the just-mentioned setting with user features from i) long-term changing, ii) short-term changing perspectives, and iii) pairwise link features.

As mentioned in the last section, users' interaction behaviors play an importance role to enhance users' experience, increase the game time, and promote consumption. More specifically, we extract a pair of interacting users from a friend list directly, or read from the temporary interaction data: sending out and approving friend application. In our scenario, we recommend strangers to users as friend lists daily, and we employ the click-through rate of applications as the measurement.

Friend suggestion: Given a user u_i and his/her long-term features R_{u_i} and short-term features S_{u_i}, link features $l_{i,j}$, aiming at learning a function $f(S_{u_i}, R_{u_i}, l_{i,j}) \rightarrow p_{ij}$ for each candidate user u_j where p_{ij} denotes the estimated strength of the potential interaction between u_i and u_j. This estimated strength represents the likelihood that user u_i tends to interact with user u_j in the future. Thus, the higher the strength, the more likely user u_i is to make a friend request to user u_j. Then, we utilize the estimations to rank the target in the candidate set so as to suggest the proper friends to the user.

B. Methodology

We now present the details of our proposed framework, Deep Similarity Evolutionary neural Network (DSEN), for friend suggestions, which exploits users' long-term and short-term information. The proposed model has the following main components, including the structured sequence embedding layer, the multi-view similarity modeling layer, and the similarity evolution layer.

1) General Framework: In this paper, we propose a novel friend suggestion framework by ranking people in the candidate set after retrieval through a multi-stage deep structure. An overview of our proposed model is shown in Figure II. The embedding layer takes user's action sequences as input and projects them into a dense vector for each time point, which uses hidden states as output. Then the output embedding of each time slot is concatenated with the user's static profile vector, which can comprehensively capture user's long-term and short-term personality. Subsequently, the obtained dense features are fed into the next layer to obtain users' historical interests from multi-view viewpoints. Finally, we apply an evolutionary similarity layer to output a score to determine the similarity.

2) Structured sequence embedding layer.: Individual players constantly output a series of actions in the game, providing rich information to understand the players' state over time. The friend suggestion setup needs to understand the recent changes made by the user. We therefore start by modeling activity sequences which has directly or indirectly related to the user's friend application preference. First, we transform the user's activity sequences $\mathbf{S}(x^{(1)}, x^{(2)}, ..., x^{(t)})$ into a dense representation via an embedding layer, where we take the most up-to-date t time steps from the user's records and pad the sequence with zeros if one does not have enough data. We know that RNN and its variants have the ability to capture inherent sequential dependency, hence we implement this style-based embedding layer to extract features preserving user characteristics. We adopt GRU [8] here because it alleviates the vanishing gradient problem and does not yield more learnable parameters as in RNNs, which can be stated as

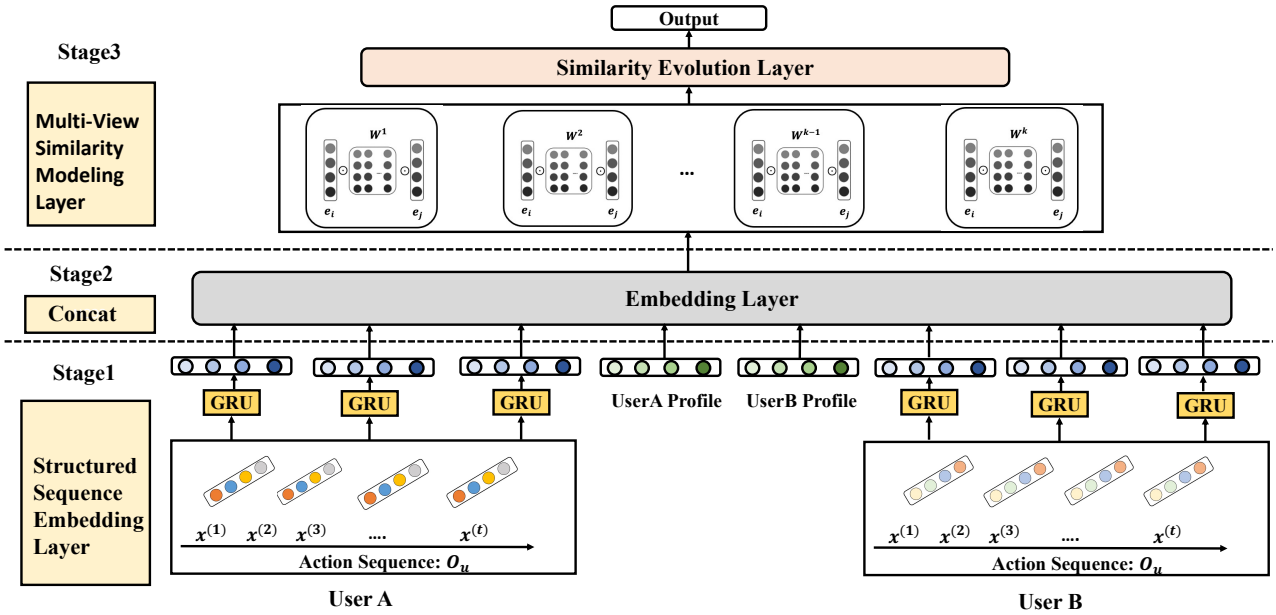


Fig. 2: An overview of the proposed architecture (DSEN). It consists of three stages. First, in structured sequence Embedding layer, it embeds a series of user action sequences into low-dimensional vectors. This is then used to concatenate with users' profile vectors to obtain users' embeddings at each time point. At the third stage, we introduce a multi-view similarity model to capture the evolution process of user-user similarity from the obtained users' representations. In this layer, $W_i, \forall i \in 1, 2, ..k$ are all learnable parameters, where k is the number of views in similarity vector. Here, e_i, e_j are the hidden layer embeddings of source and target users, respectively. We then employ the recurrent structure to understand the similarity between users, which is designed to capture the relevance of similarity from this evolutionary process.

follows,

$$\begin{cases} \mathbf{u^{(t)}} = \sigma\left(g\left(\left[\mathbf{X^{(t)}}; \mathbf{H^{(t-1)}}\right] + b_u\right)\right) \\ \mathbf{r^{(t)}} = \sigma\left(g\left(\left[\mathbf{X^{(t)}}; \mathbf{H^{(t-1)}}\right] + b_r\right)\right) \\ \mathbf{C^{(t)}} = \tanh\left(g\left(\left[\mathbf{X^{(t)}}; \mathbf{r^{(t)}} \cdot \mathbf{H^{(t-1)}}\right] + b_c\right)\right) \\ \mathbf{H}^{(t)} = \mathbf{u}^{(t)} \odot \mathbf{H}^{(t-1)} + \left(1 - \mathbf{u}^{(t)}\right) \odot \mathbf{C}^{(t)} \end{cases} \tag{1}$$

where $\mathbf{X}^{(t)}$ and $\mathbf{H}^{(t)}$ represent the input and output at time step t; $g(\cdot)$ signifies the fully connected function; σ denotes the sigmoid activation function; $\odot$ is the Hadamard product operation. Besides, $\mathbf{u^{(t)}}$ and $\mathbf{r^{(t)}}$ are the update gate at t time step and reset gate at t time step, respectively. $\mathbf{C}^{(t)}$ is candidate activation, which is used to receive the information from input state and hidden state.

In this layer, each encoder unit extracts the feature vectors of an input sequence and projects the temporal dependency into a hidden representation. Therefore, each user has a unique representation vector at each time slot, which can reflect activity such as skill dynamics in a game.

After obtaining the embedding for each time slot, we combine it with the long-term features of the players, i.e., profiles. The underlying assumption is that one intermediate action of a player may deviate from his/her normal patterns and mislead the similarity estimation process. Meanwhile, the long-term information alone may not always reflect the player's current preferences. It is therefore necessary to combine these two together as an input for computing the similarity at each time step. For each user, the hidden state representation is $\mathbf{H^{(t)}}$ and the user's profile features is $\mathbf{R^{(t)}}$, and the concatenation output is $\mathbf{e^{(t)}}$.

3) Multi-view similarity modeling layer: From the previous step of our model, we obtain a time-aware representation of the players. We now introduce a new simple yet effective method for computing the similarity between users. Often, recommendation systems use dot product to get the similarity between input embedding pairs $(\mathbf{e_i}, \mathbf{e_j})$, denoting $\mathbf{e_i^T e_j}$. With simply linear algebra knowledge, we know that this score function can be formulated as a bilinear form of $\mathbf{e_i^T I e_j}$, where $\mathbf{I}$ is an identity matrix. However, this undermines the importance of the similarity function since the similarity measure and embedding should be mutually enforced to achieve recommendation success. This abandons the discriminative power of the similarity function by using the identity matrix, making the prediction models fully dependent on the embedding. To overcome this limit, we choose to learn a parametric similarity subspace, $\mathbf{W} \in \mathbb{R}^{m \times m}$, to better fit the data, resulting in a general similarity measurement as $M(\mathbf{e_i}, \mathbf{e_j}) = \mathbf{e_i^T W e_j}$. Also, learning a single similarity matrix may not be able to fully capture the relations between input feature-pair embeddings. Inspired by ensemble learning theory, we aim to learn a group similarity function to determine similarity collectively. More specifically, we can formulate ensemble similarity learning as

$$M(\mathbf{e_i}, \mathbf{e_j}) = \sum_k \mathbf{e_i^T W^k e_j} \tag{2}$$

where $\mathbf{W^k}$ denotes the kth similarity matrix. To implement this, we can further rewrite this bilinear into a linear format

like the following:

$$M(\mathbf{e_i}, \mathbf{e_j}) = \sum_k \mathbf{e_i^T W^k e_j} = \sum_k \mathbf{W^k} \otimes \left(\mathbf{e_i^T e_j}\right) = \sum_k \sum_{pq} \mathbf{W^k_{pq} e_{ip} e_{jq}} \tag{3}$$

By vectorization, we have $\mathbf{z} = (\mathbf{e_{i1}} * \mathbf{e_{j1}}, ..., \mathbf{e_{im}} * \mathbf{e_{jm}})^T$ and $\mathbf{v^k} = \left(\mathbf{W^k_{11}}, \mathbf{W^k_{21}} ..., \mathbf{W^k_{mm}}\right)^T$, and reformulate the similarity measurement as follows,

$$M(\mathbf{e_i}, \mathbf{e_j}) = \sum_k \mathbf{e_i^T W^k e_j} = \left[\mathbf{v^1}, \mathbf{v^2}, ..., \mathbf{v^k}\right]^T \mathbf{z} \tag{4}$$

This can be implemented as a standard Multi-Layer Perceptron (MLP) network having

$$\mathbf{g} = u\left(\mathbf{V}^T \mathbf{z} + b\right) \tag{5}$$

where b is the bias term and $\mathbf{V} = \left[\mathbf{v^1}, \mathbf{v^2}, ..., \mathbf{v^k}\right]$. Here, $u()$ is activation function, and we use ReLU to promote sparsity in outputs.

4) Similarity evolution modeling layer.: So far we have obtained the similarity embedding at each time point for a given user-user pairs. The goal of this layer is to produce user-user pair similarity based on the output from the multi-view similarity modeling layer. As stated before, the similarity between a user-user pair will evolve over time. For example, online game players will change their strategy of adding strangers according to their own level changes, and naturally the similarity between users will evolve over time. The historical similarities will have different relevance effects on the current time slot, therefore the problem now translates into capturing this underlying relevance. While memory mechanism has been shown to be an effective way of capturing relevant information, we therefore employ it to learn the underlying similarity evolution automatically. Specifically, we implement our similarity evolution modeling layer by leveraging LSTM [6], in which the similarity sequence obtained from the multi-view similarity modeling layer is used as input to the LSTM. Each LSTM unit receives a similarity vector and transforms the user-pair temporal dependency similarity into a hidden representation. In this way, our similarity evolution layer can capture the inherent relations between historical behavior sequences.

5) Prediction layer: In the similarity evolution layer, we obtain a representation vector $\mathbf{g}$ that characterizes the evolution of the users similarity. We then adopt a single-layer feed-forward neural network to obtain the similarity scores between user-pairs at the current time step: $p = \sigma\left(\mathbf{W}^T \mathbf{g} + b\right)$, where p is the rating prediction, $\mathbf{W}$ is the learnable parameters, b is the bias term, $\sigma()$ is the activation function and we choose Sigmoid in this work.

6) Loss function.: To predict whether a user-pair will establish a friend-request connection, we model it as a binary classification task, and use sigmoid function as the output unit. In this work, we use the cross-entropy loss to train our model:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \left(y \log p(\theta) + (1-y) \log(1-p(\theta))\right) \tag{6}$$

Dataset	Section-1	Section-2	Section-3	Section-4
# train user-user pairs	4M	11.4M	4.1M	14.5M
# train friendship pairs	85k	2.7M	80k	4.3M
# test user-user pairs	16.1M	4.6M	17.3M	14.6M
# sequences features	55	55	55	55
# profiles features	68	68	68	68

TABLE II: The statistics of Dataset I.

Dataset	Section-5	Section-6
# train user-user pairs	5.0M	9.1M
# train friendship pairs	1.1M	1.9M
# test user-user pairs	4.9M	7.1M
# sequences features	55	55
# profiles features	68	68

TABLE III: The statistics of Dataset II.

where $\mathcal{D}$ denotes all the training samples and $y \in \{0, 1\}$ is the ground truth label whether user A has sent a friend request to user B, $p(\theta)$ is the output of our framework which represents the prediction probability.

7) Sampling strategy: In our scenario, we recommend each user a list of candidate, also called the exposed list. The positive samples are defined as these pairs that users click send the friend request to others. All these pairs have no request action are viewed as negative samples, and we sample from these due to reduce the computational cost and balance the positive samples. In general, we keep the ratio between positive instances and negative instances as $1:4$. Although the friendship behavior in the game is sparse, we can due to the large number of active users in our game, a certain amount of friend request logs are still generated daily. Thus, in our industrial settings, such samples are commonly extracted by routine batch jobs and populated in an upstream database at regular time intervals, to facilitate efficient model training.

8) Candidate retrieval.: In our scenario, friend suggestion includes two key sections: the candidate retrieval and the friend ranking, which is coarse to refine process. Candidate retrieval aims to generate a list of top-N (e.g., N = 1000) potential friends out of dozens millions users. Then friend ranking is to implement fine-grained re-ranking from the generated candidates, delivering top-n (e.g., n = 100) users to end users. We follow popular strategy in this domain to do the candidate retrieval, which is multiple way retrieval. Furthermore, in our scenario, we only suggest 100 strangers for each user per day, so there is no need to support real-time retrieve and online inference.

IV. Experiment

A. Experimental settings

1) Datasets: We evaluate DSEN on several large-scale game datasets, shown as Table. II. Our data is all collected from a large-scale online game from a major online game company in the world that contains about six millions of players. And we divide the data into different sections (Section1-6) based on logging time and user area. We employ in-game user profiles and players' actions as long-term and short-term

TABLE IV: In game player features description

Feature type	Description	Number
Profile (long-term)	Registration information	13
	Game mode	14
	Activation summary	28
	Consumption summary	13
Behavior (short-term)	Score and award count	10
	Equipment count	4
	Tactical skills	37
	Team statistics	4

features, respectively. More specifically, we have 68 features from profiles and 55 features from action features, as shown in Table. IV, where all action features are collected daily from the past 15 days. All features are pre-processed with zero-mean and unit-variance normalization. We used 80% as training set for learning parameters and 20% as validation set for hyper-parameter tuning for each training dataset. Test dataset is the one collected from the day next to date in training dataset, ensuring no label exposure.

Four types of datasets are used for the comparison, where all these are collected from the same game but different sections. The datasets in Table. III can generate stable social networks, which have fewer cold start users than those in Table. II. We can therefore implement experiments for graph-based models using datasets in Table. III.

2) Baselines: We compared our proposed method to four baselines which are all deep neural network based methods for performance evaluation. Since our proposed methods aim to address friend suggestion in online game settings, we do not have any other models can implement this case directly, therefore we compare with models formulating from popular deep neural models such as MLP and GRU.

- **GRU**: On the source user tower side and the target user tower side, we adopt Gated Recurrent Unit to extract the short-term interest of the user's active sequence, and the top layer outputs the rating score through a single-layer fully connected network. [6].
- **MLP**: Multi-Layer Perceptron with fully-connected layers and ReLU activations to learn user representations [9]. This shares the same ranking loss with the proposed method DSEN.
- **self-Attention**: Using a self-attention module to capture the hidden temporal dependency in the user's active sequence, then calculating the current prediction score based on the similarity evolutionary process learned the user-user pairs embedding [25].
- **GraFRank**: Learning user representation from social networking, integrating with multi-modal user feature, for friend suggestion. [22].

Note that GraFRank explicitly depends on the social graph while the features introduced in this paper are independent of the structure of the network. A direct comparison between our proposed model and GraFRank is therefore unfair. We report GraFRank results on a few datasets using the social network. But such comparisons merely suggest that building a friend suggestion model based on individual independent features is a promising direction as a surrogate of social network design, especially when social network information is insufficient.

3) Evaluation Metrics.: To evaluate friend recommendation, we use two popular metrics: Hit-Rate (HR@K), Normalized Discounted Cumulative Gain (NDCG@K), where K denotes the top-chosen number. Large values of Hit-Rate and NDCG indicate better predictive performance. Compared with the HR@K, NDCG@K is more sensitive to the order and position of the recommended list, which can reflect whether the relevance ranking conforms to the target user's real preference priority. It is worth noting that the number of users is often very large for online applications, **small improvements to these metrics can have significant impact on overall performance**, thus improving online user experience.

4) Parameter Settings.: We implement our model based on Tensorflow2 [2]. The whole model is learnt by optimizing the log loss of Eq 6. For the embedding size d, we tested the value in the set of $[8, 16, 32, 64, 128, 256]$. We search the batch size and learning rate in $[1024, 2048, 4096, 8192, 16384]$ and $[0.001, 0.005, 0.01, 0.05, 0.1]$, respectively. An early stopping strategy was performed, where we stopped training if the training AUC on the validation set has no increase for two consecutive epochs of training. For all learnable parameters in the network, we use random initialization with variance scaling. Moreover, we adopt Adam optimizer to train all learnable parameters. For the baseline algorithms, the parameters were initialized as in the corresponding papers, and were then carefully tuned to obtain the optimal performance. We also present the values of the key parameters used for the final results at Table VIII. On average, under this setting, the running time for training is $3795.12s$ per epoch and $0.043ms$ for inference per sample.

B. Experimental Results

1) Friend suggestion: We first present the friend suggestion performance of all methods in Table. V, including the Hit rate and NDCG values on these testing datasets. All these methods are built on same user features, including users' profiles, users' in game behavior sequences and pairwise link features? Generally speaking, the performance shows a similar trend in these datasets that DSEN $\geq$ self-Attention $\geq$ GRU $\geq$ MLP. We find that MLP always gets the worst performance because it does not consider the temporal dependencies in data, thus fails to capture the dependencies in user sequence features. As a comparison, GRU gains decent improvement over MLP since it captures dependencies in sequence by memorizing information. These two models both obtain the similarity by directly learning embeddings from raw features. This can simplify the similarity between users due to the evolving user preferences. Thanks to the modeling similarity evolution, our method DSEN consistently outperforms base-

TABLE V: Friend suggestion performance comparison between ours and the competing techniques on four large datasets

Dataset	models	K=10		K=20		K=50		K=100	
		HIT@K	NDCG@K	HIT@K	NDCG@K	HIT@K	NDCG@K	HIT@K	NDCG@K
	MLP	0.0688	0.0334	0.1063	0.0429	0.1877	0.0589	0.2627	0.0711
	GRU	0.1547	0.0987	0.2039	0.1111	0.2733	0.1249	0.3252	0.1333
	self-Attention	0.1662	0.1050	0.2189	0.1183	0.2897	0.1324	0.3459	0.1415
Section-1	**DSEN**	**0.1680**	**0.1071**	**0.2222**	**0.1207**	**0.2986**	**0.1359**	**0.3532**	**0.1448**
	MLP	0.1780	0.1059	0.2358	0.1204	0.3149	0.1362	0.3773	0.1463
	GRU	0.1784	0.1109	0.2328	0.1246	0.3142	0.1407	0.3728	0.1502
	self-Attention	0.1881	0.1130	0.2464	0.1278	0.3312	0.1446	0.3888	0.1540
Section-2	**DSEN**	**0.1914**	**0.1167**	**0.2499**	**0.1315**	**0.3421**	**0.1498**	**0.4050**	**0.1600**
	MLP	0.1796	0.1108	0.2334	0.1244	0.3062	0.1389	0.3615	0.1479
	GRU	0.1840	0.1173	0. 2330	0.1296	0.3069	0.1444	0.3587	0.1528
	self-Attention	0. 1787	0.1119	0.2286	0.1244	0.3005	0.1387	0.3538	0.1474
Section-3	**DSEN**	**0.1846**	**0.1159**	**0.2363**	**0.1289**	**0.3117**	**0.1439**	**0.3679**	**0.1531**
	MLP	0.1296	0.0812	0.1634	0.0898	0.2134	0.0997	0.2435	0.1046
	GRU	0.1305	0.0819	0.1656	0.0907	0.2132	0.1002	0.2455	0.1054
	self-Attention	0.1291	0. 0873	0.1644	0.0.0929	0.2144	0.1008	0.2461	0.1049
Section-4	**DSEN**	**0. 1320**	**0.0.0829**	**0.1698**	**0.0924**	**0.2191**	**0.1022**	**0.2527**	**0.1077**

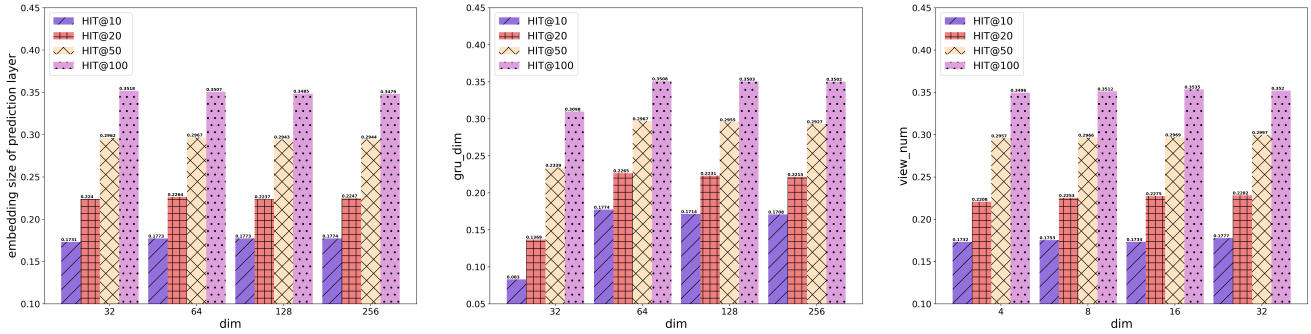


Fig. 3: Configuration Analysis of HIT@K, including embedding size of the prediction layers, GRU dimension and the number of views in similarity.

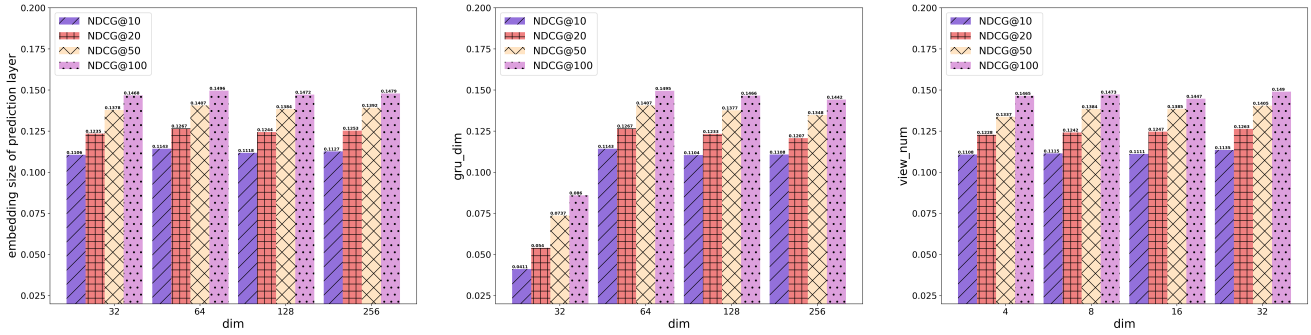


Fig. 4: Configuration Analysis of NDCG@K, including embedding size of the prediction layers, GRU dimension and the number of views in similarity.

line methods, achieving around 3%- 35% gains over baselines. When removing this similarity evolution as self-Attention model, we observe noticeable performance decline, indicating the expressiveness of similarity evolution modeling.

We also compared the performance of our DSEN with GraFrank on the two datasets Section-5 and Section-6, and the results are reported in Table VI. As we mentioned before, this is not a fair comparison, so this part is to provide another angle to understand the directions of addressing friend suggestion, using social networks or not. Here, we follow the feature usage in the original GraFrank paper, and use the user profile as the input. We observe a significant performance gap between

TABLE VI: The performance comparison between DSEN and GNN.

Dataset	models	K=10		K=100	
		HIT@K	NDCG@K	HIT@K	NDCG@K
Section-5	GraFRank	0.0035	0.0182	0.1666	0.0432
	DSEN	**0.1568**	**0.0954**	**0.3269**	**0.1296**
Section-6	GraFRank	0.051	0.024	0.2328	0.0595
	DSEN	**0.1976**	**0. 1225**	**0.4095**	**0.1657**

TABLE VII: We perform ablation analysis on two datasets. Attention-based model has inferior performance than the LSTM-based one.

Dataset	models	K=10		K=100	
		HIT@K	NDCG@K	HIT@K	NDCG@K
Section-2	DSEN-ATT	0.1767	0.0712	0.3688	0.1062
	DSEN	**0.1914**	**0.1167**	**0.4050**	**0.1600**
Section-3	DSEN-ATT	0.1633	0.0675	0.3370	0.0991
	DSEN	**0.1846**	**0.1159**	**0.3679**	**0.1531**

ours and GraFrank. Apart from being different in the use of features, GNN-based model has the built-in drawback of learning interactions in features, thus it is hard to obtain the similarity between users in our setting.

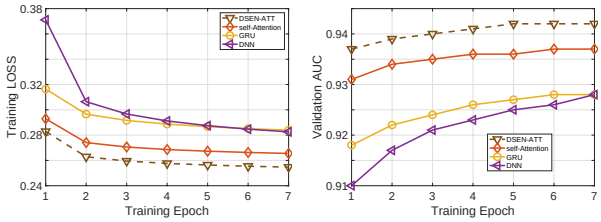
2) Ablation study: Now, we present an ablation study to analyze the architectural choice of the model. As our goal is to capture the evolution of similarity, LSTM is one immendiate choice and the other is an attention model such as TransFormers [25]. For comparison, we now implement a version of the TransFormers model,, called as DSEN-ATT. The results of both methods are presented in Table VII. Interestingly, we find that DSEN-ATT performs much worse than DSEN. One possible explanation is that the similarity depends more on the most up-to-date activities as opposed to the entire global sequence, so that as the global capture ability becomes a setback of Transformer. LSTMs can capture this contextual relationship more accurately, especially when the sequence similarity is short, which is generally the case in our setting.

3) Hyperparameter analysis: In this subsection, we aim to analyze the effect of a few important hyperparameter of the DSEN, including the embedding size of prediction layer, GRU output dimensions, and the number of views in similarity computation. The results of different values are presented at Figure. 4 and Figure.3, showing Hit rate and NDCG, respectively.

Effects of the embedding size of prediction layer: We first analyze the impact of varying the dimension of the top-level embedding in the prediction layer, and report the results in the first column of Figure. 3 and Figure. 4. We observe that with an increase in the dimension of the prediction layer, it promotes performance in HIT@K and NDCG@K. These dimensions determine the expressive capability of the model, but an over-sized dimension will significantly increase the learning cost and cause over-fitting in the test data. So we observe a diminishing returns when the dimension is greater than 64.

TABLE VIII: The parameter settings

name	value
Training batch size	16384
Learning rate	0.01
The top-level embedding size	64
GRU hidden size	64
The number of views in similarity vector	32
The number of GRU layers	2
Epochs	8



(a) Training loss (b) Validation AUC

Fig. 5: The analysis of training loss and validation AUC: DSEN converges faster than other baselines and reaches a better optimization minima w.r.t AUC.

Effects of GRU dimensions: We now investigate the influence of the value of GRU dimensions on the behavior sequence extraction layer. In the second column in Figure. 3 and Figure. 4, we present the performance comparison w.r.t the GRU dimension. Generally, with the increase of GRU dimension, the performance of small dimensions shows significant increase while that of large dimensions has little change. This parameter affect the amount of information that characterize the user's interest, but a too large dimensions may introduce the noise during feature interaction.

Effects of the number of views in similarity: Recall that we developed a multi-view model for similarity computation. The key hyperparameter is the number of similarity matrices as in Eq. 2. This adds a large level of flexibility and non-linearity to the similarity measurement. So, we vary this number to understand its effect on the performance. From the results, we can observe that large $\mathbf{k}$ tends to obtain better performance, about 1%-2% improvement when varying $\mathbf{k}$ from 4 to 32.

4) Model Training Analysis.: We now analyze the training of the model by examining the training loss and AUC performance in Figure. 5 (a) and (b), respectively. As expected, experiments on the train loss and AUC both demonstrate that DSEN has better training efficiency. DSEN shows faster convergence over baselines which take about 3 epochs to reach its optimal state. The empirical evidence shows a flat loss landscape that leads to better generalization. For this experiment, we observe a similar scene of DSEN, achieving the lowest training loss and having results on the test datasets.

V. Conclusion

We study the friend suggestion problem in large scale online game scenarios, which appears to be the first work on this setting. We formulate this as a similarity measurement problem, motivated by our empirical experience, and address it via the proposed similarity evolution model. To understand user preference more accurately, we employ both long-term features and short-term features of players in games, which helps us determine the user's current intentions. We develop a deep neural framework that tackles the aforementioned features, and learns the preference evolution process from a proposed similarity measurement model. We run our experiments on several million user datasets from the a major online game company in the world, and our proposed approach consistently outperforms the competing baselines.

References

[1]

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[4] Q. Chen, H. Zhao, W. Li, P. Huang, and W. Ou. Behavior sequence transformer for e-commerce recommendation in alibaba. *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 2019.

[5] X. Chen, J. Pang, and R. Xue. Constructing and comparing user mobility profiles. *ACM Trans. Web*, 8(4), Nov. 2014.

[6] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Computer Science*, 2014.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[9] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Acm Conference on Recommender Systems*, pages 191–198, 2016.

[10] A. Epasto, S. Lattanzi, V. Mirrokni, I. Sebe, A. Taei, and S. Verma. Ego-net community mining applied to friend suggestion. *Proceedings of the VLDB Endowment*, 9:324–335, 12 2015.

[11] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[12] X. Guo, C. Shi, and C.-M. Liu. Intention modeling from ordered and unordered facets for sequential recommendation. pages 1127–1137, 04 2020.

[13] R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation, 2016.

[14] W.-C. Kang and J. McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018.

[15] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 591–600, New York, NY, USA, 2010. Association for Computing Machinery.

[16] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[17] T. Nguyen, D. Q. Tran, G. Dam, and M. H. Nguyen. Estimating the similarity of social network users based on behaviors. *Vietnam Journal of Computer Science*, 5, 05 2018.

[18] O. Peled, M. Fire, L. Rokach, and Y. Elovici. Entity matching in online social networks. In *2013 International Conference on Social Computing*, pages 339–344, 2013.

[19] E. Raad, R. Chbeir, and A. Dipanda. User profile matching in social networks. In *2010 13th International Conference on Network-Based Information Systems*, pages 297–304, 2010.

[20] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom. Suggesting friends using the implicit social graph. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, page 233–242, New York, NY, USA, 2010. Association for Computing Machinery.

[21] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom. Suggesting friends using the implicit social graph. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 233–242, 2010.

[22] A. Sankar, Y. Liu, J. Yu, and N. Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, WWW '21, page 2535–2546, New York, NY, USA, 2021. Association for Computing Machinery.

[23] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[24] J. Tang, X. Hu, and H. Liu. Social recommendation: A review. *Social Network Analysis and Mining*, 3:1113–1133, 12 2013.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv*, 2017.

[26] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 403–412, New York, NY, USA, 2015. Association for Computing Machinery.

[27] S. Wang, L. Hu, Y. Wang, L. Cao, Q. Z. Sheng, and M. Orgun. Sequential recommender systems: Challenges, progress and prospects. *arXiv preprint arXiv:2001.04830*, 2019.

[28] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 346–353, 2019.

[29] Y. Xu, D. Zhou, and J. Ma. Scholar-friend recommendation in online academic communities: An approach based on heterogeneous network. *Decision Support Systems*, 119:1–13, 2019.

[30] S. Yakhchi. Learning complex users' preferences for recommender systems. *arXiv preprint arXiv:2107.01529*, 2021.

[31] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A dynamic recurrent model for next basket recommendation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 729–732, New York, NY, USA, 2016. Association for Computing Machinery.

[32] W. Zeng, A. Zeng, M. Shang, and Y. Zhang. Information filtering in sparse online systems: Recommendation via semi-local diffusion. *PLoS ONE*, 8, 2013.

[33] Y. Zhang and Y. Zhang. Top-k influential nodes in social networks: A game perspective. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1029–1032, New York, NY, USA, 2017. Association for Computing Machinery.

[34] H. Zheng and J. Wu. Friend recommendation in online social networks: Perspective of social influence maximization. *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2017.

[35] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai. Deep interest evolution network for click-through rate prediction. 2018.

[36] G. Zhou, N. Mou, Y. Fan, Q. Pi, and K. Gai. Deep interest evolution network for click-through rate prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:5941–5948, 2019.

[37] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1059–1068, 2018.