Knowing your FATE: Friendship, Action and Temporal Explanations for User Engagement Prediction on Social Apps

Xianfeng Tang†, Yozen Liu‡, Neil Shah‡, Xiaolin Shi‡, Prasenjit Mitra†, Suhang Wang†,
The Pennsylvania State University†, Snap Inc.‡
{xut10, pum10, szw494}@psu.edu {yliu2, nshah, xiaolin}@snap.com

ABSTRACT
With the rapid growth and prevalence of social network applications (Apps) in recent years, understanding user engagement has become increasingly important, to provide useful insights for future App design and development. While several promising neural modeling approaches were recently pioneered for accurate user engagement prediction, their black-box designs are unfortunately limited in model explainability. In this paper, we study a novel problem of explainable user engagement prediction for social network Apps. First, we propose a flexible definition of user engagement for various business scenarios, based on future metric expectations. Next, we design an end-to-end neural framework, FATE, which incorporates three key factors that we identify to influence user engagement, namely friendships, user actions, and temporal dynamics to achieve explainable engagement predictions. FATE is based on a tensor-based graph neural network (GNN), LSTM and a mixture attention mechanism, which allows for (a) predictive explanations based on learned weights across different feature categories, (b) reduced network complexity, and (c) improved performance in both prediction accuracy and training/inference time. We conduct extensive experiments on two large-scale datasets from Snapchat, where FATE outperforms state-of-the-art approaches by ≈10% error and ≈20% runtime reduction. We also evaluate explanations from FATE, showing strong quantitative and qualitative performance.

ACM Reference Format:

1 INTRODUCTION
With rapid recent developments in web and mobile infrastructure, social networks and applications (Apps) such as Snapchat and Facebook have risen to prominence. The first priority of development of most Social Apps is to attract and maintain a large userbase. Understanding user engagement plays an important role for retaining and activating users. Prior studies try to understand the return of existing users using different metrics, such as churn rate prediction [38] and lifespan analysis [39]. Others model user engagement with macroscopic features (e.g., demographic information) [1] and historical statistic features (e.g., user activities) [19]. Recently, Liu et al. [20] propose using dynamic action graphs, where nodes are in-App actions, and edges are transitions between actions, to predict future activity using a neural model.

Despite some success, existing methods generally suffer from the following: (1) They fail to model friendship dependencies or ignore user-user interactions when modeling user engagement. As users are connected in social Apps, their engagement affects each other [32]. For example, active users may keep posting new contents, which attract his/her friends and elevate their engagement. Thus, it is essential to capture friendship dependencies and user interactions when modeling user engagement. (2) Engagement objectives may differ across Apps and even across features. For example, an advertising team may target prediction of click-through-rate, while a growth-focused team may care about usage trends in different In-App functions. Therefore, the definition of user engagement must be flexible to satisfy different scenarios. (3) Existing methods focus on the predicting user engagement accurately, but fail to answer why a user engages (or not). Explaining user engagement is especially desirable, since it provides valuable insights to practitioners on user priorities and informs mechanism and intervention design for managing different factors motivating different users’ engagement. However, to our knowledge, there are no explainable models for understanding user engagement.

To tackle the aforementioned limitations, we aim to use three key factors: friendship, in-App user actions, and temporal dynamics, to derive explanations for user engagement. Firstly, since users do not engage in a vacuum, but rather with each other, we consider friendships to be key in engagement. For example, many users may be drawn to use an App because of their family and friends’ continued use. Secondly, user actions dictate how a user uses different In-App functions. Therefore, the definition of user engagement activity using a neural model.

In this work, we first propose measurement of user engagement based on the expectation of metric(s) of interests in the future, which flexibly handles different business scenarios. Next, we formulate a prediction task to forecast engagement score, based on heterogeneous features identified from friendship structure, user actions, and temporal dynamics. Finally, to accurately predict future engagement while also obtaining meaningful explanations, we propose an end-to-end neural model called FATE (Friendship, Action and Temporal Explanations). In particular, our model is powered by (a)
a friendship module which uses a tensor-based graph convolutional network to capture the influence of network structure and user interactions, and (b) a tensor-based LSTM [9] to model temporal dynamics while also capturing exclusive information from different user actions. FATE’s tensor-based design not only improves explainability aspects by deriving both local (user-level) and global (App-level) importance vectors for each of the three factors using attention and Expectation-Maximization, but is also more efficient compared to classical versions. We show that FATE significantly outperforms existing methods in both accuracy and runtime on two large-scale real-world datasets collected from Snapchat, while also deriving high-quality explanations. To summarize, our contributions are:

- We study the novel problem of explainable user engagement prediction for social network applications;
- We design a flexible definition for user engagement satisfying different business scenarios;
- We propose an end-to-end self-explainable neural framework, FATE, to jointly predict user engagement scores and derive explanations for friendships, user actions, and temporal dynamics from both local and global perspectives; and
- We evaluate FATE on two real-world datasets from Snapchat, showing ≈10% error reduction and ≈20% runtime improvement against state-of-the-art approaches.

2 RELATED WORK

2.1 User Behaviour Modeling

Various prior studies model user behaviours for social network Apps. Typical objectives include churn rate prediction, return rate analysis, intent prediction, etc [2, 3, 13, 14, 17, 20, 21, 38] and anomaly detection [18, 29, 30]. Conventional approaches rely on feature-based models to predict user behaviours. They usually apply learning methods on handcrafted features. For example, Kapoor et al. [13] introduces a hazard based prediction model to predict user return time from the perspective of survival analysis; Lo et al. [21] extract long-term and short-term signals from user activities to predict purchase intent; Trouleau et al. [35] introduce a statistical mixture model for viewer consumption behavior prediction based on video playback data. Recently, neural models have shown promising results in many areas such as computer vision and natural language processing, and have been successfully applied for user modeling tasks [7, 20, 38]. Yang et al. [38] utilize LSTMs [11] to predict churn rate based on historical user activities. Liu et al. [20] introduce a GNN-LSTM model to analyze user engagement, where GNNs are applied on user action graphs, and an LSTM is used to capture temporal dynamics. Although these neural methods show superior performance, their black-box designs hinder interpretability, making them unable to summarize the reasons for their predictions, even when their inputs are meaningful user activities features.

2.2 Explainable Machine Learning

Explainable machine learning has gained increasing attention in recent years [8]. We overview recent research on explainable GNN/RNN models, as they relate to our model design. We group existing solutions into two categories. The first category focuses on post-hoc interpretation for trained deep neural networks. One kind of model-agnostic approach learns approximations around the predictions, such as linear proxy model [27] and decision trees [28, 43]. Recently, Ying et al. [41] introduce a post-hoc explainable graph neural network to analyze correlations between graph topology, node attributes and predicted labels by optimizing a compact subgraph structure indicating important nodes and edges. However, post-analyzing interpretations are computationally inefficient, making it difficult to deploy on large systems. Besides, these methods do not help predictive performance. The second group leverages attention methods to generate explanations on-the-fly, and gained tremendous popularity due to their efficiency [6, 9, 25, 31, 37]. For example, Pope et al. [25] extend explainability methods for convolutional neural networks (CNNs) to cover GNNs; Guo et al. [9] propose an interpretable LSTM architecture that distinguishes the contribution of different input variables to the prediction. Despite these attention methods successfully provide useful explanations, they are typically designed for one specific deep learning architecture (e.g., LSTMs or CNNs). How to provide attentive explanations for hierarchical deep learning frameworks with heterogeneous input is yet under-explored.

3 PRELIMINARIES

First, we define notations for a general social network App. We begin with the user as the base unit of an App. Each user represents a registered individual. We use \( u \) to denote a user. We split the whole time period (e.g., two weeks) into equal-length continuous time intervals. The length of time intervals can vary from hours to days. The past \( T \) time intervals in chronological order are denoted as \( 1, 2, \ldots, T \). Users are connected by friendship, which is an undirected relationship. Namely, if \( u \) is a friend of \( v \), \( v \) is also a friend of \( u \). Note that friendship is time-aware, users can add new friends or remove existing friends at any given time. Users can also use multiple in-App features, like posting a video, chatting with a friend, or liking a post on Facebook; we call these various user actions. We use a time-aware feature vector to represent the user action for each specific user. A typical feature of social network Apps is in-App communication. By sending and receiving messages, photos, and videos, users share information and influence each other. We call these user interactions.

**User graph.** To jointly model user activities and social network structures, we define a temporal user graph for every user at time \( t \) as \( G_t^u = (V_t^u, E_t^u, X_t^u, E_t^u) \). Here \( V_t^u = \{u\} \cup N_t(u) \) denotes the nodes in \( G_t^u \), where \( N_t(u) \) is a group of users related to \( u \), the set of edges \( E_t^u \) represents friendships, nodal features \( X_t^u \) characterize user actions, and features on edges \( E_t^u \) describe user interactions. Note that we split nodal features into \( K \) categories, so that each category of features is aligned with a specific user action, respectively. Thus, both the topological structure and the features of user graphs are temporal. In particular, for any given node \( u \), its feature...
Interactions

\[ t = \ldots \]

\[ z^2 \]

Friendship

\[ T \]

\[ 2 \]

\[ c^1 \]

Interaction

\[ u \]

whole social network. Each individual can take different actions in

where

x

(1) User action importance

vector (i.e., a row of \( X_t \)) is represented by

\[ x^u_t = [x^u_{t,1}, \ldots, x^u_{t,K}] \]

where \( x^u_{t,k} \in \mathbb{R}^{d_k} \) is the k-th category of features, and \([\cdot]\) denotes concatenation along the row. There are many ways to define

how to represent the graph structure. One example of selecting \( G \) is based on ego-networks, as shown in Figure 1: here, \( N_i(u) \) is the set of friends of \( u \), which reduces the size of graph compared to using the

whole social network. Each individual can take different actions in
every time interval to control and use in-App functions.

Defining user engagement: Because of the dynamism of user activities, social network structure, and the development of the
App itself, the user engagement definition should be specified for
every user and every time interval. Besides, the primary focus of user engagement varies widely depending on the specific business
scenario. For example, Facebook may utilize login frequency to measure engagement, while Snapchat may use the number of mes-
sages sent. Thus, user engagement requires a flexible definition
which can meet different needs. To tackle above challenges, we
define user engagement score using the expectation of a metric of interest in the future, as:

\[ e^u_t = \mathbb{E}(M(u, t) | r \in \{t, t+\Delta t\}) \]

where \( M \) is the metric of interest, and \( \Delta t \) denotes a future time period. Both the metric and the time interval can be adjusted by scenario.

Explaining user engagement: We identify three key factors that highly impact the user engagement, including user action, temporal dynamics, and friendship. The interpretation is to derive importance/influence of these three factors for user engagement.

In particular, we aim at interpreting user engagement from both local (i.e., for individual users) and global (i.e., for the whole group of people, or even the entire App) perspectives. The local interpretations for individual users are formulated as followings:

(1) User action importance \( A^u \in \mathbb{R}^{K \times |\mathbb{R}^{d_k}|} \times k=1 \), which assigns each user action a score that reflects its contribution to user engagement.

(2) Temporal importance \( T^u \in \mathbb{R}^{T \times |\mathbb{R}^{d_k}|} \times t=1 \), which identifies the importance of user actions over every time interval for the engagement.

(3) Friendship importance \( F^u \in \mathbb{R}^{|V_u \times N_i(u)|} \times v \in V_u \), which characterizes the contributions of friends to user engagement of \( u \) over time. For user action and temporal dynamics, we also derive explanations from a global view since they are shared

by all users. Specifically, we formulate (1) global user action importance \( A^* \in \mathbb{R}^{K \times |\mathbb{R}^{d_k}|} \times k=1 \), which identifies the size of graph compared to using the whole social network. Each individual can take different actions in every time interval to control and use in-App functions. (2) Global temporal importance \( T^* \in \mathbb{R}^{T \times |\mathbb{R}^{d_k}|} \times t=1 \). Compared to local explanations which help understand individual user behaviors, global explanations inform overall App-level user behaviors.

We pose the following problem formalization:

Problem (Explainable Engagement Prediction). Build a framework that for every user \( u \), predicts the engagement score \( e^u_t \) with explanations \( A^u \), \( T^u \) and \( F^u \) based on the historical user graphs \( G^u_t, \ldots, G^u_{t+\Delta t} \) and (b) generates global explanations \( A^* \) and \( T^* \).

### 4 OUR APPROACH: FATE

We next introduce our proposed approach for explainable engagement prediction, FATE. Firstly, FATE leverages specific designed friendship modules (bottom of Figure 2) to model the non-linear social network correlations and user interactions from user graphs of a given user as input. The friendship modules aggregate user graphs and generate representations for user graphs accordingly. These graph representations preserve exclusive information for every time interval and every user action. Next, a temporal module based on tensor-based LSTM [9] (tLSTM, middle part of Figure 2) is utilized to capture temporal correlations from graph representations. Finally, a mixture of attention mechanisms (top of Figure 2) is deployed to govern the prediction of user engagement based on the output of tLSTM, while also jointly deriving importance vectors as explanations. An illustration of the framework is given in Figure 2. We discuss FATE in detail in the following text.

#### 4.1 Friendship Module

As shown in Figure 3, the goal of the friendship module is to model the non-linear correlation of social network structure and user interactions in every user graph \( G^u_t \). Naturally, graph neural networks (GNNs) [12, 22, 23, 33] can be applied to capture the dependencies of users. We choose the popular graph convolutional networks (GCNs) [16] as our base GNN model. A GCN takes a graph as input, and encodes each node into an embedding vector. The embedding for each node is updated using its neighbor information on each layer of a GCN as:

\[
\tilde{x}^u = \sigma \left( \sum_{v \in N_i(u)} x^v W \right),
\]

(1)

where \( x \) and \( \tilde{x} \) denote input feature and output embedding of the layer, respectively, \( W \) is a feature transformation matrix, and \( \sigma(\cdot) \) denotes a non-linear activation.

However, adopting vanilla GCN in our case is not ideal, because matrix multiplication in GCN mixes all features together. It is difficult to distinguish the importance of input features by looking
where \( \mathbf{x} \) is the symmetric normalized adjacency matrix derived from the input to the output space separately (as illustrated by different matrices in the middle part of Figure 3). Note that each element (e.g. row) of the hidden matrix in a tGCN layer encapsulates information exclusively from a certain category of the input, so that the following mixture attention can distinguish the importance of different user actions and mix exclusive information to improve prediction accuracy. A tGCN layer can be treated as multiple parallel vanilla GCN layers, where each layer is corresponding to one category of features that characterizes one user action. Given a user graph input, We adopt a two-layer tGCN to encode the friendship dependencies into node embedding:

\[
\mathbf{x}^u = \sigma \left( \sum_{v \in N(u)} \mathbf{x}^v \otimes \mathbf{W} \right),
\]

where \( \mathbf{W} = \{ \mathbf{W}_1, \cdots, \mathbf{W}_K \}, \mathbf{W}_k \in \mathbb{R}^{d_k \times d_f} \), is a set of \( K \) parameter matrices corresponding to each group of features, and \( \mathbf{x}^u \otimes \mathbf{W} = [\mathbf{x}_1^u \mathbf{W}_1, \cdots, \mathbf{x}_K^u \mathbf{W}_K] \in \mathbb{R}^{K \times d_f}, \mathbf{x}_k^u \mathbf{W}_k \in \mathbb{R}^{d_f} \) maps each category of features from the input to the output space separately (as illustrated by different matrices in the right part of Figure 3).

The learned node embedding vectors from the tGCN can be aggregated as a representation for the graph, such as using mean-pooling to average embedding vectors on all nodes. However, there is a significant disadvantage to such simple solution: namely, the closeness of friends is ignored. In reality, most users only have a few close friends; users with many friends may only frequently engage with one or few of them. To validate, we compute the friend communication rate of all Snapchat users from a selected city (obscured for privacy reasons). Specifically, we compute the percentage of friends that a user has directly communicated (Chat/Snap) with at least \( K \) times in a two-week span. As Figure 4 shows, most users mainly communicate with a small percentage (10-20%) of their friends, and don’t frequently contact the remaining ones. Therefore, friendship activeness is key in precisely modeling the closeness of users. To this end, we propose a friendship attention mechanism [36] to quantify the importance of each friend. Formally, a normalized attention score is assigned for each friend \( v \in \mathcal{N}(u) \):

\[
\alpha_{uv} = \frac{\exp (\phi (\mathbf{x}_v^u \otimes \mathbf{e}_v^u))}{\sum_{v \in \mathcal{N}(u)} \exp (\phi (\mathbf{x}_v^u \otimes \mathbf{e}_v^u))},
\]

where \( \mathbf{x}_v^u \) is the embedding vector of node \( v \) from the tensor-based GCN, \( \mathbf{e}_v^u \) is the edge feature on edge between \( u \) and \( v \), \( \otimes \) denotes concatenation, and \( \phi (\cdot) \) is a mapping function (e.g., a feed-forward neural network). Both user actions (preserved by node embedding vectors) and user interactions are considered by the friendship attention mechanism. To obtain graph representations, we first get the averaged embedding from all friend users weighted by the friendship attention score:

\[
\bar{x} = \sum_{v \in \mathcal{N}(u)} \alpha_{uv} \tilde{x}_v^u.
\]

Then we concatenate it with the embedding vectors on node \( u \) alongside each feature category to get the graph embedding:

\[
g^u = \bar{x} \oplus \hat{x} = [\tilde{x}_1^u \oplus \hat{x}_1, \cdots, \tilde{x}_K^u \oplus \hat{x}_K],
\]

as shown in the right part of Figure 3. Note that \( \tilde{x}_k^u \oplus \hat{x}_k \) is specifically learned from user action \( k \), and \( g^u \in \mathbb{R}^{K \times (2d)} \) preserves exclusive information for every user action. Given \( g_1^u, \cdots, g_K^u \) from \( T \) historical user graphs, the next step is to capture temporal dynamics using the temporal module.

### 4.2 Temporal Module

As user activities and interactions evolve over time, modeling its temporal dynamics is a key factor of an accurate prediction for user engagement. Inspired by the success of prior studies for modeling sequential behavior data [20, 34, 38, 40] with recurrent neural networks, we utilize LSTM [11] to capture the evolution of dynamic user graphs. Specifically, we adopt tLSTM following Guo et al. [9]. Mathematically, the transformation at each layer of the tLSTM is as follows:

\[
\begin{align*}
& f_t = \sigma (g_t^u \otimes \mathbf{U}_f + h_{t-1} \otimes \mathbf{U}_f^b + b_f), \\
& i_t = \sigma (g_t^u \otimes \mathbf{U}_i + h_{t-1} \otimes \mathbf{U}_i^b + b_i), \\
& o_t = \sigma (g_t^u \otimes \mathbf{U}_o + h_{t-1} \otimes \mathbf{U}_o^b + b_o), \\
& c_t = f_t \odot c_{t-1} + i_t \odot \tanh (g_t^u \otimes \mathbf{U}_c + h_{t-1} \otimes \mathbf{U}_c^b + b_c), \\
& h_t = o_t \odot \tanh (c_t),
\end{align*}
\]

where \( \odot \) denotes element-wise multiplication, \( \mathbf{U}_f, \mathbf{U}_i, \mathbf{U}_o, \mathbf{U}_c \) and \( b_f, b_i, b_o, b_c \) are parameters. Similar to tGCN, tLSTM can also be considered as a set of parallelized LSTMs, where each LSTM is responsible for a specific feature group corresponding to its user action. Because the input graph embedding vectors \( g_1^u, \cdots, g_K^u \) to tLSTM are specific to each feature category (user action), tLSTM can capture the exclusive temporal dependencies of each user action separately. Similar to i, we define the hidden states of tLSTM as \( h_t = [h_{t,1}, \cdots, h_{t,K}] \) where \( h_{t,k} \) is exclusively learned for user action \( k \). We further use the hidden states to generate the engagement scores.

### 4.3 User Engagement Score Generation

As aforementioned, user action, temporal dynamics, and friendship are key factors to characterize and predict user engagement. We introduce three latent variables as \( z_A, z_I, z_F \) to represent different user actions (feature category), time intervals, and friends, respectively so that we can distinguish the influence of specific actions, time intervals, and friends. For example, different friends may contribute unequally to user engagement; and certain in-App functions could have higher contributions. Introducing latent variables also bridges the gap between learning explanations and predicting engagement. The desired explanations are importance vectors that constrain the posteriors of latent variables, and further govern the generating...
of user engagement scores (introduced in Section 4.4). Specifically, FATE generates user engagement predictions as follows:

\[
p(e_T|G_u) = \prod_{t=1}^{T} \prod_{k=1}^{K} \mathbb{P}(e_T, z_A = k, z_f = t, z_f = \emptyset|G_u)
\]

\[
= \prod_{t=1}^{T} \prod_{k=1}^{K} \mathbb{P}(z_f = \emptyset|z_f = t, z_A = k, G_u) \cdot \mathbb{P}(z_f = t|z_A = k, [h_{u,k}])
\]

\[
\cdot \mathbb{P}(z_f = k|[h_u])
\]

where \([h_u]\) denotes \([h_1 \ldots h_T]\), and \([h_{u,k}]\) denotes \([h_{1,k} \ldots h_{T,k}]\). The joint probability distribution is further estimated using the conditional probability of latent variables \(z_f, z_f, z_A\), which characterize how user engagement scores are affected by the friendship, temporal dynamics, and user actions accordingly. We keep designing FATE in accordance with the generation process in Eqn. 8. In particular, node embeddings are first computed exclusively for every friend, time interval, and user action with proposed tGCN. Next, friendship attention \(p(z_f = \emptyset|z_f = t, z_A = k, G_u)\) is estimated using Eqn. 4. The summation over \(\emptyset\) in Eqn. 8 is derived by graph representations from friendship modules. Then tLSTM encapsulates temporal dynamics of graph representation. The conditional probability of \(z_f\) is given as a temporal attention over \([h_u]\):

\[
\beta_{t,k} = p(z_f = t|z_A = k, [h_u]) = \frac{\exp(\psi_k([h_{u,k}]))}{\sum_{t'=1}^{T} \exp(\psi_k([h_{u,k}]))}
\]

where \(\psi_k(\cdot)\) is a neural network function specified for user action type \(k\). Using temporal attention, each user action is represented by its exclusive summarization over all past time intervals as

\[
a_k = \sum_{t=1}^{T} \beta_{t,k} h_{u,k}
\]

Finally, we approximate \(p(z_A = k|[h_u])\) as the user action attention with another softmax function:

\[
p(z_A = k|[h_u]) = \frac{\exp(\phi(\theta_k + h_{u,k}))}{\sum_{k'=1}^{K} \exp(\phi(\theta_k + h_{u,k}))}
\]

allowing parameter \(\phi(\cdot)\) is parameterized by a neural network.

To approximate the summation over all time intervals \((t = 1, \ldots, T)\) in Eqn. 8, we use Gaussian distributions to estimate the contribution of every user action to user engagement. Specifically, we use \(N(\mu_k, s\sigma_k)\) to parameterize the Gaussian distribution for user action \(k\). Here \(\psi_k(\cdot)\) is also a neural network. By integrating over all user actions, the user engagement score is derived as:

\[
p(e_T) = \prod_{k=1}^{K} N(\mu_k, s\sigma_k) \cdot p(z_A = k|[h_u])
\]

4.4 Explainable User Engagement

To explain the predicted user engagement, FATE learns the importance vectors as explanations. Similar to many previous studies \(\{6, 9, 26, 37\}\), the local explanations for individual users are directly derived from proposed mixture attentions. Specifically, the friendship attention, temporal attention and user action attention are acquired as importance vectors for friendship, temporal and user action, respectively. Because the computation of these attention vectors are included by FATE, it takes no extra cost to derive local explanations. Local explanations reflect specific characteristics and preferences for individual users, which can change dynamically for certain users.

However, local explanations could only help us understand user engagement from individual level. Taking user action as an example, the distribution of its importance vector could vary a lot among different users (see experiments in Section 5.5.1 as an example). Because some functions of the App cannot be personalized for every user, it is necessary to interpret their contributions from a global view. For example, when distributing a new feature in an A/B test, it is more reasonable to understand the impact of the feature globally. Under such circumstances, we formulate the global interpretation of user engagement as a learning problem, where the global importance vectors are jointly learned with the model. Taking the global importance vector for user action \(A^*\) as an example, we adopt the Expectation–Maximization (EM) method to learn \(A^*\) jointly with the optimization of model parameters \(\theta\):

\[
L(\theta, A^*) = -\sum_{u \in S} \mathbb{E}_{q_A^u} \left[ \log p(e_T^u | z_A^u; (G_u)) \right] - \mathbb{E}_{q_A^u} \left[ \log p(z_A^u | (A_u^*)) \right] + \mathbb{E}_{q_A^u} \left[ \log p(z_A^u | (A_u^*)) \right] - \mathbb{E}_{q_A^u} \left[ \log p(z_A^u | (G_u)) \right]
\]

where the summation \(\sum\) is applied over all training samples \(S\), and \(q_A^u\) denotes the posterior distribution for \(z_A^u\):

\[
q_A^u (z_A^u) = p(z_A^u | (G_u), e_T^u, \theta) = p(z_A^u | (G_u), e_T^u) \cdot p(z_A^u | (G_u))
\]

\[
\approx p(z_A^u | (G_u), e_T^u) \cdot p(e_T^u | z_A^u)
\]

The last term in Eqn. 13 serves as a regularization term over the posterior of \(z_A^u\). Note that the posterior of \(z_A^u\) governs the user action attention. Consequently, the regularization term encourages the action importance vectors of individual users to follow the global pattern parameterized by \(A^*\). Moreover, we can derive the following closed-form solution of \(A^*\) as:

\[
A^* = \frac{1}{|S|} \sum_{u \in S} q_A^u
\]

which takes both user action attention and the prediction of user engagement into consideration. The learning of user action importance relies on the estimation of posterior \(q_A^u\). During training stage, network parameters \(\theta\) and the posterior \(q_A^u\) are estimated alternatively. Namely, we first freeze all parameters \(\theta\) to evaluate \(q_A^u\) over the batch of samples, then use the updated \(q_A^u\) with gradient descent to update \(\theta\) by minimizing 13. Similarly for the global temporal importance, we derive the following closed-form solution:

\[
T_{e,k} = \frac{1}{|S|} \sum_{u \in S} \beta_{t,k}
\]

4.5 Complexity Analysis

The proposed tGCN and adopted tLSTM \(\{9\}\) are more efficient than their vanilla versions. Specifically, we have:

Theorem 4.1. Let \(d_{in}\) and \(d_{out}\) denote input and output dimensions of a layer. The tensor-based designs for GCN and LSTM reduce network complexity by \((1 - 1/K)d_{in} \cdot d_{out} + 4(1 - 1/K)(d_{in} + d_{out})d_{out}\) trainable parameters, and reduce the computational complexity by \(O((d_{in} \cdot d_{out}) + O((d_{in} + d_{out})d_{out})\), respectively.

Proof. We provide the proof in Appendix A.1. □
As a result, the proposed designs accelerate the training and inference of FATE, and produce a more compact model. Appendix A.2 shows that FATE’s tensor-based design reduces training and inference time by ≈20% compared to using the vanilla version (GCN/LSTM).

5 Evaluation

In this section, we aim to answer the following research questions:

- **RQ1**: Can FATE outperform state-of-the-art alternatives in the user engagement prediction task?
- **RQ2**: How does each part/module in FATE affect performance?
- **RQ3**: Can FATE derive meaningful explanations for friendships, user actions, and temporal dynamics?
- **RQ4**: Can FATE flexibly model different engagement metrics?

5.1 Datasets and Experiment Setup

We obtain two large-scale datasets from Snapchat. Each dataset is constructed from all users that live in a different city (on two different continents), we filter out inactive/already churned users. We follow previous studies on Snapchat [20] and collect 13 representative features for user actions on Snapchat, normalizing to zero mean and unit variance independently before training. Table 5 in Appendix provides each feature. We consider 1-day time intervals over 6 weeks. We use the 3 weeks for training, and the rest for testing. We use 2 weeks of user graphs as input to predict engagement in the following week (i.e., $\Delta t = 7d$).

To show that FATE is general for multiple prediction scenarios, we evaluate on two notions of user engagement. The first metric considers user session time in hours (winsorized to remove extreme outliers). The second metric considers user related activities, which are core functions of Snapchat. We aggregate and average four normalized snap related features, including send, view, create and save, as the measurement for user engagement. The prediction of user engagement scores based on two different metrics is denoted by Task 1 and Task 2, respectively. We choose $\Delta t = 7d$.

To show that FATE is general for multiple prediction scenarios, we evaluate on two notions of user engagement. The first metric considers user session time in hours (winsorized to remove extreme outliers). The second metric considers user related activities, which are core functions of Snapchat. We aggregate and average four normalized snap related features, including send, view, create and save, as the measurement for user engagement. The prediction of user engagement scores based on two different metrics is denoted by Task 1 and Task 2, respectively. We choose $\Delta t = 7d$.

5.2 Compared Methods

To validate the accuracy of user engagement prediction, we compare FATE with the following state-of-the-art methods:

- **Linear Regression (LR)**: We utilize the averaged feature vectors of each node in $G_r$ as a representation for time interval $t$, and concatenate the vectors over all past time intervals as the input.
- **XGBoost (XGB)** [4]: We adopt the same preprocessing steps of LR as input for XGBoost.
- **MLP** [10]: We experiment on a two-layer MLP with the same input features to LR and XGB.
- **LSTM** [11]: LSTM is a popular RNN model for various sequential prediction tasks. We implement a two-layer LSTM which iterates over historical user action features. The final output is fed into a fully-connected layer to generate prediction.

5.3 User Engagement Prediction Performance

To answer the first research question, we report user engagement prediction accuracy of above methods in Table 1. As we can see, FATE achieves best performance in both tasks. As expected, FATE significantly out-performs two feature-based methods LR and XGB since it captures friendship relation and temporal dynamics. Deep-learning based methods MLP, GCN, and LSTM achieves similar performance. However, FATE surpasses them with tremendous error reduction. Moreover, FATE outperforms state-of-the-art approach TGLSTM, by at most 10%. There are two potential reasons. First, FATE additionally captures friendship relation by explicitly modeling user-user interaction. Second, ICN and LSTM maintain independent parameters to capture exclusive information for every user actions, which enhances the predicting accuracy.

5.4 Ablation Study

To answer the second question, we design four variations of FATE as follows: (1) FATE$_{1T}$: We first evaluate the contribution of tensor-based design. To this end, we employ the original GCN [16] and LSTM [11] to create the first ablation FATE$_{1T}$. We use the last output from LSTM to predict user engagement score. (2) FATE$_{pred}$: We then study the effectiveness of the friendship module. We apply TGLSTM on raw features to create FATE$_{pred}$. (3) FATE$_{temp}$: Next we study the

---

1 https://github.com/tangxianfeng/FATE
Table 2: All components help FATE: Removing (a) tGCN/tLSTM, (b) friendship module, (c) temporal module or (d) user interactions hurts performance.

<table>
<thead>
<tr>
<th>Region 1</th>
<th>RMSE</th>
<th>MAPE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATE&lt;sub&gt;A&lt;/sub&gt;</td>
<td>112±0.02</td>
<td>213±0.04</td>
<td>0.85±0.01</td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Ind&lt;/sub&gt;</td>
<td>119±0.02</td>
<td>218±0.02</td>
<td>0.89±0.02</td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Temp&lt;/sub&gt;</td>
<td>126±0.01</td>
<td>221±0.03</td>
<td>0.97±0.02</td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Int&lt;/sub&gt;</td>
<td>112±0.01</td>
<td>208±0.01</td>
<td>0.86±0.02</td>
</tr>
<tr>
<td>FATE</td>
<td>109±0.03</td>
<td>204±0.01</td>
<td>0.81±0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region 2</th>
<th>RMSE</th>
<th>MAPE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATE&lt;sub&gt;A&lt;/sub&gt;</td>
<td>120±0.00</td>
<td>199±0.01</td>
<td>0.93±0.00</td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Ind&lt;/sub&gt;</td>
<td>121±0.00</td>
<td>199±0.01</td>
<td>0.99±0.00</td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Temp&lt;/sub&gt;</td>
<td>220±0.02</td>
<td>0.97±0.00</td>
<td></td>
</tr>
<tr>
<td>FATE&lt;sub&gt;Int&lt;/sub&gt;</td>
<td>19±0.02</td>
<td>0.99±0.00</td>
<td></td>
</tr>
<tr>
<td>FATE</td>
<td>196±0.03</td>
<td>0.88±0.00</td>
<td></td>
</tr>
</tbody>
</table>

5.5.1 User Action Importance. We first study the global user action importance. Figure 5 illustrates the importance score of different user actions, where a larger value indicates higher importance for user engagement.

Since the objective of Task 1 is to predict a session time-based engagement score, the importance of historical app usage length is significantly higher. This indicates that historical session time is the key factor for user engagement (defined by the expectation of session time in the future), as user actions usually follow strong temporal periodicity. Remaining user actions play similar roles in extending session time, which is intuitive, because on the entire

5.5.2 Temporal Importance. Figure 7 displays the overall temporal importance of user actions across time (i.e., past 14 days). Darker hue indicates higher importance to user engagement. For Task 1, SessionTime has strong short-term importance in both cities. Temporally close SessionTime (later days) data contributes to user engagement more. On the contrary, other user actions show long-term importance. For example, SnapView and ChatView show relatively higher importance on the first day. In addition to long/short-term characteristics, we see the importance of most user actions showing strong periodicity in a weekly manner. Similar conclusions can also be drawn from Task 2, where SnapView, SnapCreate, and SnapSave show longer-term correlation to user engagement. SnapSend on the other hand demonstrates a
5.5.4 Baseline comparisons on explainability. Feature importance from XGBoost can be used as a temporal importance explanation. As in Figure 10, results from XGBoost are very sparse, where most user actions receive an unnatural, near-0 importance score, likely because feature importance is only a byproduct of the training of XGBoost. Unlike FATE, the XGBoost objective is purely defined on prediction accuracy, failing to learn explanations for user actions over time. Figure 10 shows the temporal attention from LSTM. There are two weaknesses of using LSTM for explanation: (1) it is unable to capture the importance of each user action; (2) compared to FATE, the temporal attention fails to capture periodicity of user actions, which naive LSTM mixes and cannot separate. Comparatively, FATE derives richer and more fine-grained explanations.

6 Practical Applications

Our framework is designed with practical applications in mind. State-of-the-art in engagement prediction improves temporally-aware estimation of overall demand and key metrics, which offers flexible use in many forecasting and expectation-setting applications. Explainability in the model helps quantify both global and local factors in user engagement, and how they motivate users to engage with the platform. Moreover, it paves roads for personalized interventions and nudges to users to realize in-app value, stay in touch with their best friends and retain. Finally, our choices around tensor-based modeling improve efficiency by reducing parameters and decreasing training time. Yet, GNN training/inference is still a challenge for multi-million/billion-scale workloads, especially considering dynamism of the underlying data, temporality of predictions, and frequent model updation needs in practice, though new work in GNN scalability offers some promising inroads [5, 42]. In the future, we plan to develop automated and recurrent training and inference workflows which can handle these issues to gracefully scale FATE to production workloads larger than those we experimented on.

6 Conclusion

In this paper, we explore the problem of explainable user engagement prediction for social network Apps. Given different notions of user engagement, we define it generally as the future expectation of a metric of interest. We then propose an end-to-end neural framework, FATE, which models friendship, user actions and temporal dynamics, to generate accurate predictions while jointly deriving local and global explanations for these key factors. Extensive experiments on two datasets and two engagement prediction tasks from Snapchat demonstrate the efficiency, generality and accuracy of our approach: FATE improves accuracy compared to state-of-the-art methods by ≈10% while reducing runtime by ≈20% owing to its use of proposed tensor-based GCN and LSTM components. We hope to continue to improve scaling aspects of FATE to deploy it for recurrent auto-training and inference at Snapchat. While FATE is designed with Snapchat in mind, our core ideas of engagement definition, contributing factors, and technical contribution in neural architecture design offer clear applications to other social Apps and online platforms.
Figure 9: FATE’s local friendship importance captures asymmetric influence of friends: the user has similar session time behaviors (right) as their highest-importance friends (blue and orange lines are close); session time spikes coincide with high temporal importances (left) of those friends (dark hues).

Figure 10: Comparisons of explainability.

ACKNOWLEDGEMENT

This material is based upon work supported by, or in part by, the National Science Foundation (NSF) under grant #1909702. Any opinions, findings, and conclusions in this material are those of the authors and do not reflect the views of the NSF.

REFERENCES

[38] Carl Yang, Xiaolin Shi, Luo Jie, and Jiawei Han. 2018. I Know You’ll Be Back: Interpretable New User Clustering and Churn Prediction on a Mobile Social Application. In KDD. ACM, 914–922.
A COMPLEXITY OF FATE

A.1 Theoretical Analysis

In this section, we analyze the complexity of FATE. In particular, we focus on the complexity reduction from the tensor-based designs of GCN and LSTM over the standard ones. Without loss of generality, we use \( d_{in} \) and \( d_{out} \) to denote the dimensions of input and output of a neural network layer (e.g., GCN, LSTM, etc.). We use the number of learnable parameters (neurons in the network) to measure the network complexity as follows:

**Theorem A.1.** By replacing the standard GCN and LSTM layers with corresponding tensor-based versions, the network complexity is reduced by \( (1 - \frac{1}{K})d_{in} \cdot d_{out} \) and \( 4(1 - \frac{1}{K})(d_{in} + d_{out})d_{out} \) number of trainable parameters, respectively.

**Proof.** The number of trainable parameters for the GCN layer is \( d_{in} \cdot d_{out} \) (see Eqn. 1), and that for the tensor-based GCN layer is \( K \cdot (d_{in} \cdot d_{out}) = \frac{d_{in} \cdot d_{out}}{K} \) (see Eqn. 2, assume they are equally divided into each category of user action features). Therefore, tensor-based GCN reduces network complexity by \( (1 - \frac{1}{K})d_{in} \cdot d_{out} \) number of parameters. Similarly, the standard LSTM layer has \( 4(d_{in} + d_{out}) \) trainable parameters (corresponding to the input transition, hidden state transition, and the bias); while the tensor-based LSTM layer only maintains \( 4 \left( \frac{d_{in} \cdot d_{out}}{K} + d_{in}^2 + d_{out} \right) \) number of parameters (for \( \mathcal{U}_i \), \( \mathcal{U}_i^h \) and \( b \), in Eqn. 7). As a result, the total number of parameters is reduced by \( 4(1 - \frac{1}{K})(d_{in} + d_{out})d_{out} \) when adopting the tensor-based LSTM over the standard one.

The computational complexity comes from multiplications. The reduction of computational complexity is analyzed through Theorem A.2.

**Theorem A.2.** The tensor-based GCN and the tensor-based LSTM reduce the computational complexity by \( O(d_{in} \cdot d_{out}) \) and \( O((d_{in} + d_{out})d_{out}) \), respectively.

**Proof.** Let \( N \) denote the number of nodes in the ego-network. Using Eqn. 1 and 2, the computational complexity of a GCN layer and a tensor-based GCN layer are \( N^2 \cdot d_{in} \cdot d_{out} \) and \( N^2 \cdot \frac{d_{in} \cdot d_{out}}{K} \cdot K + N \cdot \frac{d_{in} \cdot d_{out}}{K} \cdot K + \frac{d_{in} \cdot d_{out}}{K} \cdot K = N^2 \cdot d_{in} \cdot d_{out} \cdot \frac{1}{K} \), respectively. The reduction is then \( N(1 - \frac{1}{K})d_{in} \cdot d_{out} = O(d_{in} \cdot d_{out}) \). For an LSTM layer (Eqn. 7), it takes \( 4(d_{in} \cdot d_{out} + d_{in}^2) + 3d_{out} \) multiplications to update its hidden and gate, while the tensor-based LSTM layer takes only \( 4 \left( \frac{d_{in} \cdot d_{out}}{K} + d_{in}^2 + d_{out} \right) + 3d_{out} = 4 \left( \frac{d_{in} \cdot d_{out}}{K} + \frac{d_{in}^2}{K} + \frac{d_{out}}{K} \right) + 3d_{out} \) multiplications. Thus, the reduction of computational complexity by the tensor-based LSTM is \( O((d_{in} + d_{out})d_{out}) \).

Note that for FATE, it adopts multiple friendship modules with the tensor-based LSTM. Therefore, FATE is significantly benefited from the tensor-based design, reducing both network size and computational complexity sharply. However, the overall improvement over complexity does not exactly align with these tensor-based designs, due to costs from extra components in FATE such as the computation of attention scores. Therefore, we also analyze the real-world running time of FATE quantitatively in the following experiment section.

A.2 Experimental Results

We study the runtime complexity of FATE. We compare the runtime of FATE\(_s\) and FATE, to demonstrate the improvement by using tensor-based designs for FATE over a non tensor-based model FATE\(_s\). Both training and testing (inference) run times are reported in 3. We can see that training FATE takes significantly less time than FATE\(_s\) by an average of 20%. In addition, inference speed of FATE is also faster. Therefore, it is beneficial to adopt tensor-based designs when constructing the framework. Note that our implementation uses PyTorch Geometric\(^2\) as the underlying message passing framework.

**Table 3: Comparisons of Runtime (min).** FATE reduced 20% of runtime on average comparing with non-tensor-based FATE\(_s\).

<table>
<thead>
<tr>
<th>Task</th>
<th>Region 1</th>
<th>Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Test</td>
</tr>
<tr>
<td>Task 1</td>
<td>FATE(_s)</td>
<td>216.96</td>
</tr>
<tr>
<td></td>
<td>FATE</td>
<td>181.35</td>
</tr>
<tr>
<td>Task 2</td>
<td>FATE(_s)</td>
<td>207.23</td>
</tr>
<tr>
<td></td>
<td>FATE</td>
<td>172.00</td>
</tr>
</tbody>
</table>

B IMPLEMENTATION DETAILS

B.1 Experimental Environment

Our experiments are conducted on a single machine on Google Cloud Platform\(^3\), with a 16-core CPU, 60GB memory and 2 Nvidia P100 GPUs.

B.2 Data Preprocessing

We select two geographic regions, one from North America and the other from Europe, to compile two datasets. We set the time period from 09/16/2019 to 10/27/2019, with a one-day time interval length. There are totally 42 days in the time period (6 weeks). For each dataset, we first query all users whose locations are within the corresponding region. Users who spend less than one minute (session time) on a daily average are treated as extremely inactive and filtered. We then obtain the friendship of these users as our social network and historical user action records in each day. Detailed features and descriptions for user actions are reported in Table 5. Besides, we also acquire user-user commutation as features for user interaction, including chat, snap, and story. These features are constructed from the aggregation of each type of interaction. Table 4 details both datasets.

\(^2\)https://github.com/rusty1s/pytorch_geometric
\(^3\)https://cloud.google.com

**Table 4: Statistics of Datasets.**

<table>
<thead>
<tr>
<th>Region 1</th>
<th>Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Time period</td>
<td>09/16/2019 - 10/27/2019</td>
</tr>
<tr>
<td>Avg. # users</td>
<td>153006</td>
</tr>
<tr>
<td>Avg. node degree</td>
<td>51.58</td>
</tr>
<tr>
<td># node features</td>
<td>13</td>
</tr>
<tr>
<td># edge features</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 5: Selected features for user actions on Snapchat.

<table>
<thead>
<tr>
<th>In-App function</th>
<th>Feature name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snap</td>
<td>SnapSend</td>
<td># of snaps sent to friends.</td>
</tr>
<tr>
<td></td>
<td>SnapView</td>
<td># of snaps viewed from friends.</td>
</tr>
<tr>
<td></td>
<td>SnapCreate</td>
<td># of snaps created by the user.</td>
</tr>
<tr>
<td></td>
<td>SnapSave</td>
<td># of snaps saved to the memory/smartphone.</td>
</tr>
<tr>
<td>Chat</td>
<td>ChatSend</td>
<td># of text messages sent to friends.</td>
</tr>
<tr>
<td></td>
<td>ChatView</td>
<td># of received text messages.</td>
</tr>
<tr>
<td>Story</td>
<td>StoryPost</td>
<td># of videos posted to the user’s page</td>
</tr>
<tr>
<td></td>
<td>StoryView</td>
<td># of watched story videos posted by others.</td>
</tr>
<tr>
<td></td>
<td>StoryViewTime</td>
<td>Total time spent for watching stories.</td>
</tr>
<tr>
<td>Discover</td>
<td>FriendDiscoverView</td>
<td># of watched videos posted by friends on Discover page</td>
</tr>
<tr>
<td></td>
<td>PublisherDiscoverView</td>
<td># of watched videos posted by publisher on Discover page</td>
</tr>
<tr>
<td></td>
<td>DiscoverViewTime</td>
<td>Total time spent for watching videos on Discover page</td>
</tr>
<tr>
<td>Misc.</td>
<td>SessionTime</td>
<td>Total time spent on Snapchat.</td>
</tr>
</tbody>
</table>

B.3 Model Implementations

We implement all compared baseline methods in Python 3.7. Linear Regression is adopted from scikit-learn\(^4\). We use XGBoost\(^4\) from the official package\(^5\) with its recommended setting and parameters. We implement the GCN model with PyTorch Geometric. We set up a two layer GCN, with the hidden size of 128, using ELU as the activation function. Similarly, we build the LSTM model as a two-layer LSTM using PyTorch\(^6\). The hidden size is 128. We set the dropout rate to 0.3 for the second layer. ELU is used as the activation. We following the original settings for TGLSTM as introduced in the paper \(^20\). We implement FATE with PyTorch and PyTorch Geometric. Friendship modules contain two-layer tGCN. The dimension of output embedding for all feature categories is set to 32. The design of tLSTM is inspired by IMV-LSTM\(^7\). We use two layers of tLSTM for FATE. Our code is available on Github\(^8\).

For LR and Xgboost, we train until convergence. For neural network models, we set the batch size to 256 and the max number of epoch to 10. All models are optimized by Adam algorithm \(^15\), with a learning rate of 0.001. They are trained until reaching the max epoch or early-stopped on the validation set. The validation set contains 10% samples randomly selected from the training set. All methods are trained and tested 10 times to get averaged results.

B.4 Evaluation Metrics

Three common metrics Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE) are used to evaluate the performance of all methods. The detailed definitions of these metrics are stated as below:

\[
\text{RMSE} = \sqrt{\frac{1}{|S|} \sum_{u \in S} (e_u - \hat{e}_u)^2},
\]

\[
\text{MAPE} = \frac{1}{|S|} \sum_{u \in S} \frac{|e_u - \hat{e}_u|}{\hat{e}_u},
\]

\[
\text{MAE} = \frac{1}{|S|} \sum_{u \in S} |e_u - \hat{e}_u|,
\]  \hspace{1cm} (17)

where \(\hat{e}_u\) denotes the ground truth of predicted user engagement score \(e_u\).

While RMSE and MAE receive higher penalties from larger values, MAPE focuses on the prediction error of samples with smaller engagement scores. Therefore, combining these metrics leads to more comprehensive conclusions.

\(^4\)https://scikit-learn.org
\(^5\)https://xgboost.readthedocs.io/
\(^6\)https://pytorch.org/
\(^7\)https://github.com/KurochkinAlexey/IMV_LSTM
\(^8\)https://github.com/tangxianfeng/FATE