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:
Xianfeng Tang[†], Yozen Liu[‡], Neil Shah[‡], Xiaolin Shi[‡], Prasenjit Mitra[†], Suhang Wang[†]. 2020. Knowing your FATE: Friendship, Action and Temporal Explanations for User Engagement Prediction on Social Apps. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403276

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 features, and hints at their reasons for using the App. Thirdly, user behavior changes over time, and often obey temporal periodicity [24]. Incorporating periodicity and recency effects can improve predictive performance.

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

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403276

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 explainablity 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 gain 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,

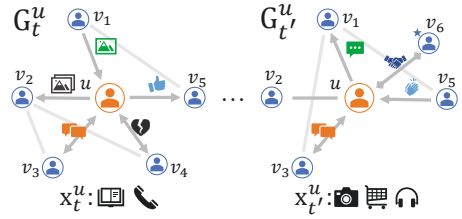


Figure 1: User graphs are temporal, and capture friendship structure, user actions (node features), and user-user interactions (edge features) over various in-App functions.

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 provides 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 $1, 2, \cdots, T$. 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 $\mathbf{G}_t^u = (\mathcal{V}_t^u, \mathcal{E}_t^u, \mathbf{X}_t^u, \mathbf{E}_t^u)$. Here $\mathcal{V}_t^u = \{u\} \cup \mathcal{N}_t(u)$ denotes the nodes in $\mathbf{G}_t^u$, where $\mathcal{N}_t(u)$ is a group of users related to u, the set of edges $\mathcal{E}_t^u$ represents friendships, nodal features $\mathbf{X}_t^u$ characterize user actions, and features on edges $\mathbf{E}_t^u$ 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

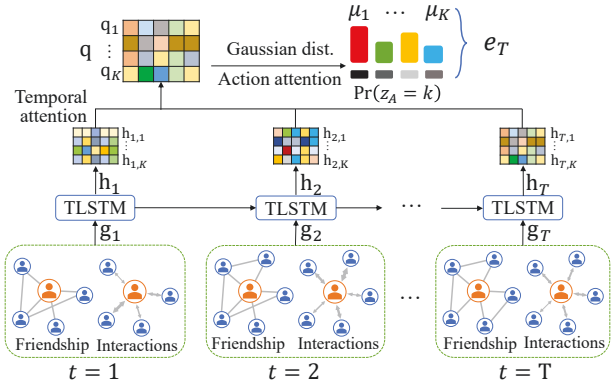


Figure 2: Overall framework of FATE: tGCN-based friendship modules capture local network structure and user interactions at each timestep, and tLSTM captures temporal dynamics for distinct user actions. Finally, an attention mixture mechanism governs user engagement prediction.

vector (i.e., a row of $\mathbf{X}_t$) is represented by $\mathbf{x}_t^u = [\mathbf{x}_{t,1}^u, \cdots, \mathbf{x}_{t,K}^u]$, where $\mathbf{x}_{t,k}^u \in \mathbb{R}^{d_k}$ is the k-th category of features, and $[\cdot]$ denotes concatenation alongside the row. There are many ways to define the graph structure. One example of selecting $\mathbf{G}$ is based on ego-networks, as shown in Figure 1; here, $\mathcal{N}_t(u)$ is the set of friends of u, which reduces the size of graph sharply 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 messages 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: $e_t^u = \mathbb{E}(\mathcal{M}(u,\tau)|\tau \in [t, t+\Delta t]))$, where $\mathcal{M}$ is the metric of interest, and Δt 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 following vectors: (1) User action importance $\mathbf{A}^u \in \mathbb{R}^K_{\geq 0}$, $\sum_{k=1}^K \mathbf{A}_k^u = 1$, which assigns each user action a score that reflects its contribution to user engagement. (2) Temporal importance $\mathbf{T}^u \in \mathbb{R}^{T\times K}_{\geq 0}$, $\sum_{t=1}^T \mathbf{T}_{tk}^u = 1$ for $k = 1, \cdots, K$, which identifies the importance of user actions over every time interval for the engagement; (3) Friendship importance $\mathbf{F}^u \in \mathbb{R}^{|t\times\mathcal{N}_t(u)|}_{\geq 0}$, $\sum_{v\in\mathcal{N}_t(u)} \mathbf{F}_{tv}^u = 1$ for $t = 1, \cdots, T$, 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

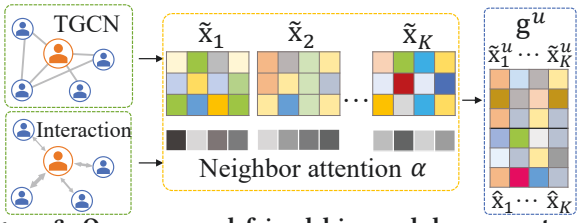


Figure 3: Our proposed friendship module uses a tensor-based GCN with neighbor attention to generate user graph embeddings jointly from ego-networks and interactions.

by all users. Specifically, we formulate (1) global user action importance $\mathbf{A}^* \in \mathbb{R}^K_{\geq 0}$, $\sum_{k=1}^K \mathbf{A}_k^* = 1$ and (2) global temporal importance $\mathbf{T}^* \in \mathbb{R}^{T\times K}_{\geq 0}$, $\sum_{t=1}^T \mathbf{T}_{tk}^* = 1$ for $k = 1, \cdots, K$. 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 (a) for every user u, predicts the engagement score e_T^u with explanations $\mathbf{A}^u$, $\mathbf{T}^u$ and $\mathbf{F}^u$ based on the historical user graphs $\mathbf{G}_1^u, \cdots, \mathbf{G}_T^u$, and (b) generates global explanations $\mathbf{A}^*$ and $\mathbf{T}^*$.*

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 $\mathbf{G}_t^u$. 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:

$$\tilde{\mathbf{x}}^u = \sigma\left(\sum_{v\in\mathcal{N}(v)} \mathbf{x}^v\mathbf{W}\right), \tag{1}$$

where $\mathbf{x}$ and $\tilde{\mathbf{x}}$ denote input feature and output embedding of the layer, respectively, $\mathbf{W}$ 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

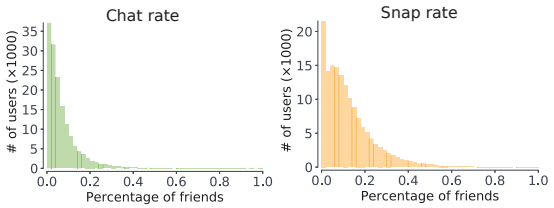


Figure 4: Most users communicate frequently only with a subset (≤20%) of their friends, making careful aggregation important when considering influence from neighbors.

at the output of a GCN layer. To tackle this limitation, we propose a *tensor-based GCN* (tGCN), which uses a tensor of learnable parameters. The updating rule of one tGCN layer is:

$$\tilde{\mathbf{x}}^u = \sigma\left(\sum_{v\in\mathcal{N}(v)} \mathbf{x}^v \otimes \mathcal{W}\right), \tag{2}$$

where $\mathcal{W} = \{\mathbf{W}_1, \cdots, \mathbf{W}_K\}$, $\mathbf{W}_k \in \mathbb{R}^{d_k \times d'}$, is a set of K parameter matrices corresponding to each group of features, and $\mathbf{x}^v \otimes \mathcal{W} = [\mathbf{x}_1^v\mathbf{W}_1, \cdots, \mathbf{x}_K^v\mathbf{W}_K] \in \mathbb{R}^{K\times d'}$, $\mathbf{x}_k^v\mathbf{W}_k \in \mathbb{R}^{1\times d'}$ maps each category of features 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:

$$\tilde{\mathbf{X}} = \sigma\left(\hat{\mathbf{A}}\sigma\left(\hat{\mathbf{A}}\mathbf{X} \otimes \mathcal{W}_0\right) \otimes \mathcal{W}_1\right), \tag{3}$$

where $\hat{\mathbf{A}}$ is the symmetric normalized adjacency matrix derived from the input user graph, $\mathbf{X}$ are nodal features, and $\mathcal{W}_*$ are parameters. As input features describe user actions, their exclusive information is preserved in the output of tGCN as $\tilde{\mathbf{X}} = [\tilde{\mathbf{X}}_1, \cdots, \tilde{\mathbf{X}}_K] \in \mathbb{R}^{K\times d'\times(|\mathcal{N}(v)|+1)}$, which will be used later for generating engagement predictions and explanations.

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 once 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 $v \in \mathcal{N}(u)$:

$$\alpha_v = \frac{\exp\left(\phi\left(\tilde{\mathbf{x}}^v \oplus \mathbf{e}^v\right)\right)}{\sum_{\nu\in\mathcal{N}(u)} \exp\left(\phi\left(\tilde{\mathbf{x}}^\nu \oplus \mathbf{e}^\nu\right)\right)}, \tag{4}$$

where $\tilde{\mathbf{x}}^v$ is the embedding vector of node v from the tensor-based GCN, $\mathbf{e}^v$ is the edge feature on edge between u and v, $\oplus$ 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:

$$\hat{\mathbf{x}} = \sum_{v\in\mathcal{N}(u)} \alpha_v \tilde{\mathbf{x}}^v. \tag{5}$$

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

$$\mathbf{g}^u = \tilde{\mathbf{x}}^u \oplus \hat{\mathbf{x}} = \left[\tilde{\mathbf{x}}_1^u \oplus \hat{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_K^u \oplus \hat{\mathbf{x}}_K\right], \tag{6}$$

as shown in the right part of Figure 3. Note that $\tilde{\mathbf{x}}_k^u \oplus \hat{\mathbf{x}}_k$ is specifically learned from user action k, and $\mathbf{g}^u \in \mathbb{R}^{K\times(2d')}$ preserves exclusive information for every user action. Given $\mathbf{g}_1^u, \cdots \mathbf{g}_T^u$ 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 evolvement 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:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{g}_t^u \otimes \mathcal{U}_f + \mathbf{h}_{t-1} \otimes \mathcal{U}_f^{\mathbf{h}} + \mathbf{b}_f\right),\\
\mathbf{i}_t &= \sigma\left(\mathbf{g}_t^u \otimes \mathcal{U}_i + \mathbf{h}_{t-1} \otimes \mathcal{U}_i^{\mathbf{h}} + \mathbf{b}_i\right),\\
\mathbf{o}_t &= \sigma\left(\mathbf{g}_t^u \otimes \mathcal{U}_o + \mathbf{h}_{t-1} \otimes \mathcal{U}_o^{\mathbf{h}} + \mathbf{b}_o\right),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh\left(\mathbf{g}_t^u \otimes \mathcal{U}_c + \mathbf{h}_{t-1} \otimes \mathcal{U}_c^{\mathbf{h}} + \mathbf{b}_c\right),\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right),
\end{aligned} \tag{7}$$

where $\odot$ denotes element-wise multiplication, $\mathcal{U}_*$, $\mathcal{U}_*^{\mathbf{h}}$ and $\mathbf{b}_*$ 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 $\mathbf{g}_1^u, \cdots, \mathbf{g}_T^u$ to tLSTM are specific to each feature category (user action), tLSTM can capture the exclusive temporal dependencies of each user action separately. Similar to $\mathbf{x}$, we define the hidden states of tLSTM as $\mathbf{h}_t = [\mathbf{h}_{t,1}, \cdots, \mathbf{h}_{t,K}]$ where $\mathbf{h}_{t,k}$ 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_J, z_I 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:

$$
\begin{aligned}
p\left(e_T|\{\mathbf{G}_*\}\right) &= \sum_{k=1}^{K}\sum_{t=1}^{T}\sum_{v=1}^{|\mathcal{N}(u)|} p\left(e_T, z_A=k, z_J=t, z_I=v|\{\mathbf{G}_*\}\right) \\
&= \sum_{k=1}^{K}\sum_{t=1}^{T}\sum_{v=1}^{|\mathcal{N}(u)|} \underbrace{p\left(e_T|z_A=k, z_J=t, z_I=v; \tilde{\mathbf{x}}^v\right)}_{\text{node embedding}} \\
&\cdot \underbrace{p\left(z_I=v|z_J=t, z_A=k, \mathbf{G}_t\right)}_{\text{friendship attention}} \cdot \underbrace{p\left(z_J=t|z_A=k, \{\mathbf{h}_{*,k}\}\right)}_{\text{temporal attention}} \\
&\cdot \underbrace{p\left(z_A=k|\{\mathbf{h}_*\}\right)}_{\text{user action attention}},
\end{aligned} \tag{8}
$$

where $\{\mathbf{h}_*\}$ denotes $\{\mathbf{h}_1 \dots \mathbf{h}_T\}$, and $\{\mathbf{h}_{*,k}\}$ denotes $\{\mathbf{h}_{1,k} \dots \mathbf{h}_{T,k}\}$. The joint probability distribution is further estimated using the conditional probability of latent variables z_I, z_J, 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 $p(z_I = v|z_J = t, z_A = k, \mathbf{G}_t)$ is estimated using Eqn. 4. The summation over v in Eqn. 8 is derived by graph representations from friendship modules. Then tLSTM encapsulates temporal dynamics of graph representation. The conditional probability of z_J is given as a temporal attention over $\{\mathbf{h}_{*,k}\}$:

$$
\beta_{t,k} = p\left(z_J=t|z_A=k, \{\mathbf{h}_{*,k}\}\right) = \frac{\exp\left(\varphi_k\left(\mathbf{h}_{t,k}\right)\right)}{\sum_{\tau=1}^{T}\exp\left(\varphi_k\left(\mathbf{h}_{\tau,k}\right)\right)}, \tag{9}
$$

where $\varphi_k(\cdot)$ 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

$$
\mathbf{a}_k = \sum_{t=1}^{T}\beta_{t,k}\mathbf{h}_{t,k} \tag{10}
$$

Finally, we approximate $p(z_A = k|\{\mathbf{h}_*\})$ as the user action attention with another softmax function:

$$
p\left(z_A=k|\{\mathbf{h}_*\}\right) = \frac{\exp\left(\phi\left(\mathbf{a}_k \oplus \mathbf{h}_{T,k}\right)\right)}{\sum_{\kappa=1}^{K}\exp\left(\phi\left(\mathbf{a}_\kappa \oplus \mathbf{h}_{T,\kappa}\right)\right)}, \tag{11}
$$

where $\phi(\cdot)$ is parameterized by a neural network.

To approximate the summation over all time intervals ($t = 1, \cdots, T$) in Eqn. 8, we use Gaussian distributions to estimate the contribution of every user action to user engagement. Specifically, we use $N(\mu_k, sd_k) = \psi_k(\mathbf{a}_k \oplus \mathbf{h}_{T,k})$ 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:

$$
p\left(e_T\right) = \sum_{k=1}^{K} N\left(\mu_k, sd_k\right) \cdot p\left(z_A=k|\{\mathbf{h}_*\}\right). \tag{12}
$$

4.4 Explainable User Engagement

To interpret the predicted user engagement, FATE learns the importance vectors as explanations. Similar to many previous studies (e.g., [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 scores 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 $\mathbf{A}^*$ as an example, we adopt the Expectation–Maximization (EM) method to learn $\mathbf{A}^*$ jointly with the optimization of model parameters θ:

$$
\begin{aligned}
\mathcal{L}(\theta, \mathbf{A}^*) = &-\sum_{u\in\mathcal{S}} \mathop{\mathbb{E}}_{q_A^u}\left[\log p\left(e_T^u|z_A^u; \{\mathbf{G}_*^u\}\right)\right] \\
&- \mathop{\mathbb{E}}_{q_A^u}\left[\log p\left(z_A^u|\{\mathbf{h}_*^u\}\right)\right] - \mathop{\mathbb{E}}_{q_A^u}\left[\log p\left(z_A^u|\mathbf{A}^*\right)\right],
\end{aligned} \tag{13}
$$

where the summation $\sum$ is applied over all training samples $\mathcal{S}$, and q_A^u denotes the posterior distribution for ${z_A}^u$:

$$
\begin{aligned}
q_A^u &= p\left(z_A^u|\{\mathcal{G}_*^u\}, e_T^u, \theta\right) \propto p\left(e_T^u|z_A^u, \{\mathcal{G}_*^u\}\right) \cdot p\left(z_A^u|\{\mathcal{G}_*^u\}\right) \\
&\approx p\left(e_T^u|z_A^u, \mathbf{q}_k^u \oplus \mathbf{h}_{T,k}^u\right) \cdot p\left(z_A^u|\{\mathbf{h}_*^u\}\right).
\end{aligned} \tag{14}
$$

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 $\mathbf{A}^*$. Moreover, we can derive the following closed-form solution of $\mathbf{A}^*$ as:

$$
\mathbf{A}^* = \frac{1}{|\mathcal{S}|}\sum_{u\in\mathcal{S}} q_A^u, \tag{15}
$$

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 θ and the posterior q_A^u are estimated alternatively. Namely, we first freeze all parameters θ to evaluate q_A^u over the batch of samples, then use the updated q_A^u with gradient descent to update θ by minimizing 13. Similarly for the global temporal importance, we derive the following closed-form solution:

$$
\mathbf{T}_{t,k}^* = \frac{1}{|\mathcal{S}|}\sum_{u\in\mathcal{S}} \beta_{t,k}. \tag{16}
$$

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 $(1 - 1/K)d_{in} \cdot d_{out}$ and $4(1 - 1/K)(d_{in} + d_{out})d_{out}$ trainable parameters, and reduce the computational complexity by $O\left(d_{in} \cdot d_{out}\right)$ and $O\left(\left(d_{in} + d_{out}\right)d_{out}\right)$, 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 explains 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 *snap* 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 root mean square error (RMSE), mean absolute percentage error (MAPE), and mean absolute error (MAE) as our evaluation metrics. We run all experiments 10 times and report the averaged results. Other technical details are discussed in Appendix B. Our code is publicly available on **Github**[1].

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 $\mathbf{G}_t$ 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 prepossessing steps of LR as input for XGBoost.
- **MLP** [10]: We experiment on a two-layer MLP with the same input features to LR and XGBoost.
- **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.

[1] https://github.com/tangxianfeng/FATE

Table 1: FATE consistently outperforms alternative models in prediction error metrics on both Task 1 and Task 2, and both datasets Region 1 and Region 2.

		Region 1			Region 2		
		RMSE	**MAPE**	**MAE**	**RMSE**	**MAPE**	**MAE**
Task 1	**LR**	.188±.001	.443±.001	.153±.000	.183±.000	.375±.001	.151±.000
	XGB	.141±.000	.260±.000	.101±.000	.140±.000	.224±.001	.098±.000
	MLP	.139±.003	.233±.007	.094±.004	.125±.005	.238±.011	.095±.004
	GCN	.131±.012	.228±.019	.094±.007	.128±.008	.242±.010	.101±.003
	LSTM	.121±.005	.221±.003	.093±.003	.122±.002	.213±.005	.095±.004
	TGLSTM	.114±.002	.215±.005	.088±.000	.122±.005	.201±.004	.093±.002
	FATE	**.109±.003**	**.204±.001**	**.081±.001**	**.118±.002**	**.196±.003**	**.088±.000**
Task 2	**LR**	.201±.000	.674±.001	.160±.000	.190±.000	.553±.000	.151±.000
	XGB	.100±.000	.347±.000	.078±.001	.134±.000	.337±.000	.089±.001
	MLP	.088±.003	.288±.006	.066±.003	.101±.002	.261±.005	.075±.000
	GCN	.094±.006	.294±.008	.069±.004	.100±.002	.257±.013	.072±.003
	LSTM	.080±.002	.249±.005	.059±.002	.097±.002	.235±.003	.070±.002
	TGLSTM	.079±.001	.241±.006	.058±.000	.095±.001	.239±.003	.070±.001
	FATE	**.072±.001**	**.213±.003**	**.053±.000**	**.093±.000**	**.224±.002**	**.066±.000**

- **GCN** [16]: We combine all historical dynamic friendship graphs into a single graph. For each user, we concatenate action features over the observed time period into a new nodal feature vector.
- **Temporal GCN-LSTM** (TGLSTM) [20]: TGLSTM is designed to predict future engagement of users, and can be treated as current state-of-the-art baseline. TGLSTM first applies GCN on action graph at each time interval, then leverage LSTM to capture temporal dynamics. We adopt the same design following Liu et al.[20] and train TGLSTM on our data to predict the engagement score.

To measure the explainability of FATE, we compare with the feature importance of XGB, and LSTM with temporal attention. After the boosted trees of XGB are constructed, the importance scores for input features are retrieved and reshaped as an explanation for temporal importance. For LSTM, we compute attention scores over all hidden states as an explanation for time intervals.

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. Secondly, tGCN and tLSTM 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 follow: **(1)** FATE$_{ts}$: 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$_{ts}$. We use the last output from LSTM to predict user engagement score. **(2)** FATE$_{fnd}$: We then study the effectiveness of the friendship module. We apply tLSTM on raw features to create FATE$_{fnd}$. **(3)** FATE$_{tmp}$: Next we study the

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

		Region 1			Region 2		
		RMSE	MAPE	MAE	RMSE	MAPE	MAE
Task 1	FATE_{ts}	.112±.002	.213±.004	.085±.001	.120±.000	.199±.001	.093±.000
	FATE_{fnd}	.119±.002	.218±.002	.089±.002	.121±.000	.199±.001	.090±.001
	FATE_{tmp}	.126±.001	.221±.003	.097±.002	.123±.002	.220±.002	.097±.000
	FATE_{int}	.112±.001	.208±.001	.086±.002	.119±.002	.198±.002	.091±.000
	FATE	**.109±.003**	**.204±.001**	**.081±.001**	**.118±.002**	**.196±.003**	**.088±.000**
Task 2	FATE_{ts}	.078±.001	.233±.004	.057±.002	.095±.001	.238±.003	.070±.002
	FATE_{fnd}	.076±.003	.228±.002	.057±.002	.094±.002	.231±.001	.068±.000
	FATE_{temp}	.083±.004	.240±.005	.061±.002	.102±.003	.253±.003	.071±.001
	FATE_{int}	.075±.000	.219±.001	.055±.000	.094±.001	.227±.002	.068±.002
	FATE	**.072±.001**	**.213±.003**	**.053±.000**	**.093±.000**	**.224±.002**	**.066±.000**

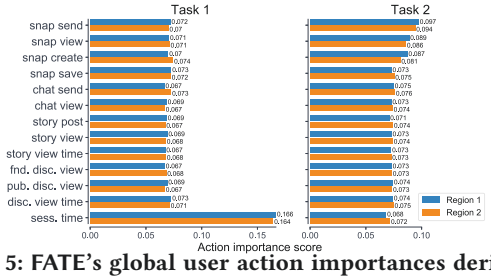


Figure 5: FATE's global user action importances derived on Task 1 and Task 2 correctly infer past *session time* and *snap*-related actions as the most important for prediction.

contribution from the temporal module. FATE_{tmp} first concatenate outputs from all friendship modules, then apply a fully-connected layer to generate user engagement score. **(4)** FATE_{int}: To analyze the contribution of explicitly modeling user interactions, we remove this part to create the last ablation FATE_{int}. The performance of all variations are reported in Table 2. FATE_{ts} performs worse when compared to FATE because it fails to extract exclusive information from each user action. However, it still outperforms TGLSTM, since user interactions enhance the modeling of friendship relation. The comparisons among FATE_{fnd}, FATE_{tmp} and FATE indicate the effectiveness of modeling friendship and temporal dependency for predicting user engagement. The comparison between FATE_{int} and FATE highlights the contribution of user interactions, which help FATE filter inactive friends and pinpoint influential users.

5.5 Explainability Evaluation

To answer the third research question, we first analyze the explanations derived from FATE. Then we compare the results with explanations from baseline methods.

5.5.1 User Action Importance. We first study the global user action importance $\mathbf{A}^*$. 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 activities usually follow strong temporal periodicity. Remaining user actions play similar roles in extending session time, which is intuitive, because on the entire

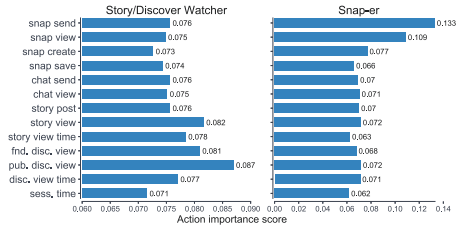


Figure 6: Two sample local user action importances: Users with different dominating engagement behaviors exhibit different user action importances.

App level, all the represented in-App functions are heavily consumed. However, we see that `Snap`-related actions are relatively more important than others. A potential reason is that sending and receiving Snaps (images/videos) are core functions which distinguish Snapchat from other Apps and define product value.

For predicting user engagement defined on normalized `Snap`-related actions in Task 2, we see that `SnapSend`, `SnapView`, and `SnapCreate` play the most important role. `SnapSend` contributes more to user engagement comparing with `SnapView`, as sending is an active generation activity while viewing is passively receiving information. Similarly, `SnapCreate` is more important than `SnapSave`, for the reason that creating a Snap is the foundation of many content generation activities, whereas Snap-saving is infrequent. Besides `Snap`-related actions, `ChatSend` is the most important, which makes sense given that private Chat messaging is the next most common usecase after Snaps on Snapchat, and users often respond to Snaps with Chats and vice-versa.

Next, we analyze user action importance for individual users. We take Task 2 as an example, and select two random users from Region 1. To help understand user preference and characteristics, we query an internal labeling service that categorizes users according to their preferences for different Snapchat features. The service, built on domain knowledge, is as independent from FATE. Generally, a "Snap-er" uses `Snap`-related functions more frequently, while a "Story/Discover Viewer" is more active on watching friend/publisher Story content on Snapchat. As illustrated in Figure 6, the importance scores of `Snap`-related user actions of a Snap-er are significantly higher than that of remained user actions. However, for Story/Discover Viewers, other actions (`StoryView`, `Public-DiscoverView`) contribute more. This shows the diversity of action importance for individual users, as the distribution of importance scores changes according to user characteristics.

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

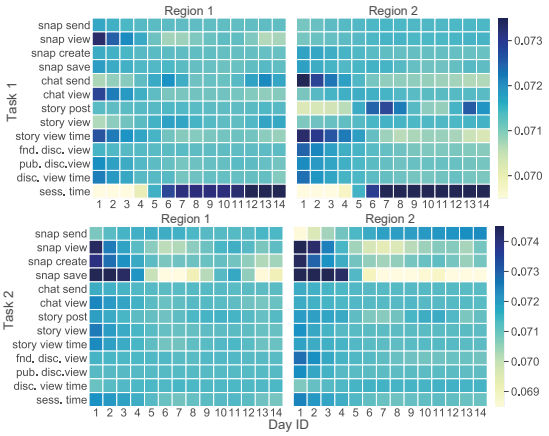


Figure 7: FATE's global temporal importances show long and short-term action importances over time.

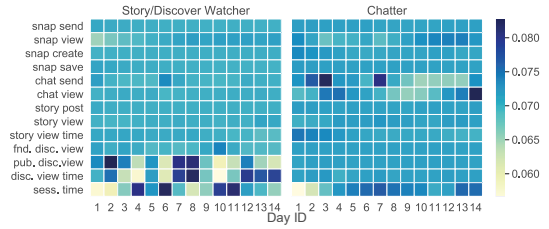


Figure 8: FATE can capture diverse local level temporal importance for users of different persona.

short-term correlation. The periodicity of temporal importance is also relatively weaker compared to Task 1.

We then study the temporal importance for individual users. Similar to action importance, we randomly select two users from Region 1, and plot temporal importance scores when predicting user engagement score in Task 1. As shown in Figure 8, users with different dominant behaviors exhibit different temporal importance score distributions. The temporal importance scores of `Publisher-DiscoverView` and `DiscoverViewTime` are relatively higher for the Story/Discover Watcher, with clear periodicity effects (importance in day 1-2, and then in 7-8, and again in 13-14, which are likely weekends when the user has more time to watch content). The Chatter has higher score for `Chat`-related features, with more weight on recent `ChatViews` (days 12-14). Our results suggest that explanations learned by FATE coincide with past understanding of temporal influence in these behaviors.

5.5.3 Friendship Importance. We next validate the learned (local) friendship importance. Figure 9 demonstrates two example users selected from Region 1, for Task 1. The heatmaps illustrate the importance scores of their friends. Clearly, friendship importance scores are not uniformly distributed among all friends. Some friends hold higher importance scores to the selected user, while others have relatively lower scores. This is potentially due to low similarity in user activities, or two friends being independently active (but not jointly interactive). To verify this assumption and interpret friendship importance scores, we compare user activeness (session time) of the selected user with their most important friends and their least importance friends (measured by the sum of scores over 14 days). As Figure 9 shows, the both users follow a pattern similar to their most important friends and unlike the least important ones. Moreover, temporal importance (darker hue) of the highest-importance friend coincides in the temporal heatmaps (left) and the session time activity plots (right) for both users in (a) and (b).

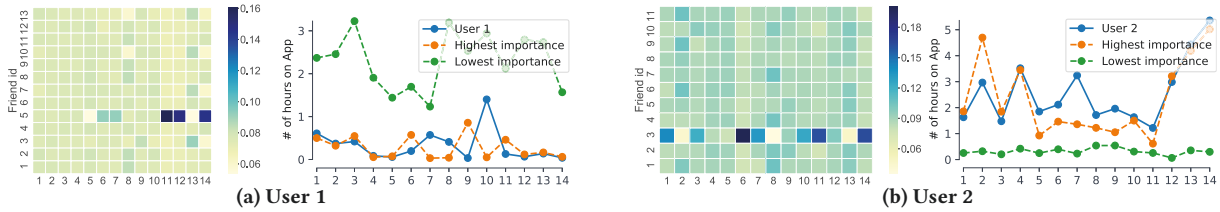
5.5.4 Baseline comparisons on explainability. Feature importance from XGBoost can be used as a temporal importance explanation. As in Figure 10, results from XGBoost are very sparse, where most user actions receive an unnatural, near-0 importance score, likely because feature importance is only a byproduct 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 weakness 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 naïve LSTM mixes and cannot separate. Comparatively, FATE derives richer and more fine-grained explanations.

5.6 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.

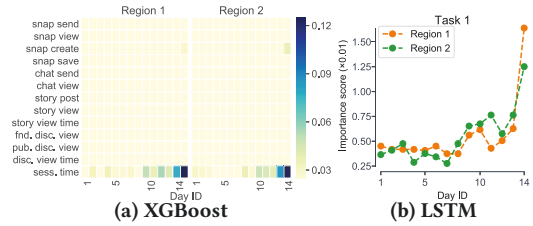
6 CONCLUSION

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.



(a) User 1 (b) User 2

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).



(a) XGBoost (b) LSTM

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

[1] Tim Althoff and Jure Leskovec. 2015. Donor retention in online crowdfunding communities: A case study of donorschoose. org. In *WWW*. 34–44.
[2] Wai-Ho Au, Keith CC Chan, and Xin Yao. 2003. A novel evolutionary data mining algorithm with applications to churn prediction. *TEC* 7, 6 (2003), 532–545.
[3] Austin R Benson, Ravi Kumar, and Andrew Tomkins. 2016. Modeling user consumption sequences. In *WWW*. 519–529.
[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *KDD*. ACM, 785–794.
[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*.
[6] Heeyoul Choi, Kyunghyun Cho, and Yoshua Bengio. 2018. Fine-grained attention mechanism for neural machine translation. *Neurocomputing* 284 (2018), 171–176.
[7] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW*.
[8] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. 2018. Explaining explanations: An overview of interpretability of machine learning. In *DSAA*. IEEE, 80–89.
[9] Tian Guo, Tao Lin, and Nino Antulov-Fantulin. 2019. Exploring Interpretable LSTM Neural Networks over Multi-Variable Data. *arXiv preprint 1905.12034* (2019).
[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.
[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[12] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph Structure Learning for Robust Graph Neural Networks. *arXiv preprint arXiv:2005.10203* (2020).
[13] Komal Kapoor, Mingxuan Sun, Jaideep Srivastava, and Tao Ye. 2014. A hazard based approach to user return time prediction. In *KDD*. ACM, 1719–1728.
[14] Jaya Kawale, Aditya Pal, and Jaideep Srivastava. 2009. Churn prediction in MMORPGs: A social influence based approach. In *ICCSE*, Vol. 4. IEEE, 423–428.
[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[16] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[17] Rohan Kumar, Mohit Kumar, Neil Shah, and Christos Faloutsos. 2018. Did We Get It Right? Predicting Query Performance in E-commerce Search. *arXiv preprint arXiv:1808.00239* (2018).
[18] Hemank Lamba and Neil Shah. 2019. Modeling Dwell Time Engagement on Visual Multimedia. In *KDD*. 1104–1113.
[19] Zhiyuan Lin, Tim Althoff, and Jure Leskovec. 2018. I'll Be Back: On the Multiple Lives of Users of a Mobile Activity Tracking Application. In *WWW*. 1501–1511.
[20] Yozen Liu, Xiaolin Shi, Lucas Pierce, and Xiang Ren. 2019. Characterizing and Forecasting User Engagement with In-app Action Graph: A Case Study of Snapchat. In *KDD*.
[21] Caroline Lo, Dan Frankowski, and Jure Leskovec. 2016. Understanding behaviors that lead to purchasing: A case study of pinterest. In *KDD*. ACM, 531–540.
[22] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *KDD*.
[23] Yao Ma, Suhang Wang, Chara C Aggarwal, Dawei Yin, and Jiliang Tang. 2019. Multi-dimensional graph convolutional networks. In *SDM*.
[24] Panagiotis Papapetrou and George Roussos. 2014. Social context discovery from temporal app use patterns. In *Ubicomp*. 397–402.
[25] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. 2019. Explainability Methods for Graph Convolutional Neural Networks. In *CVPR*. 10772–10781.
[26] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. 2017. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971* (2017).
[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*. ACM, 1135–1144.
[28] Gregor PJ Schmitz, Chris Aldrich, and Francois S Gouws. 1999. ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *TNN* 10, 6 (1999), 1392–1401.
[29] Neil Shah. 2017. Flock: Combating astroturfing on livestreaming platforms. In *WWW*. 1083–1091.
[30] Neil Shah, Hemank Lamba, Alex Beutel, and Christos Faloutsos. 2017. The many faces of link fraud. In *ICDM*. IEEE, 1069–1074.
[31] Kai Shu, Limeng Cui, Suhang Wang, Dongwon Lee, and Huan Liu. 2019. defend: Explainable fake news detection. In *KDD*.
[32] Mani R Subramani and Balaji Rajagopalan. 2003. Knowledge-sharing and influence in online social networks via viral marketing. *CACM* (2003).
[33] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. 2020. Transferring Robustness for Graph Neural Network Against Poisoning Attacks. In *WSDM*.
[34] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Joint Modeling of Local and Global Temporal Dynamics for Multivariate Time Series Forecasting with Missing Values. (2020).
[35] William Trouleau, Azin Ashkan, Weicong Ding, and Brian Eriksson. 2016. Just one more: Modeling binge watching behavior. In *KDD*. ACM, 1215–1224.
[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.
[37] Yanbo Xu, Siddharth Biswal, Shriprasad R Deshpande, Kevin O Maher, and Jimeng Sun. 2018. Raim: Recurrent attentive and intensive model of multimodal patient monitoring data. In *KDD*. ACM, 2565–2573.
[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.
[39] Jiang Yang, Xiao Wei, Mark S Ackerman, and Lada A Adamic. 2010. Activity lifespan: An analysis of user survival patterns in online knowledge sharing communities. In *ICWSM*.
[40] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI*.
[41] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNN Explainer: A Tool for Post-hoc Explanation of Graph Neural Networks. In *NeurIPS*.
[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.
[43] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. 2016. DeepRED–Rule extraction from deep neural networks. In *ICDS*. Springer, 457–473.

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 $d_{\text{in}}\cdot d_{\text{out}}$ (see Eqn. 1), and that for the tensor-based GCN layer is $K\cdot(\frac{d_{\text{in}}}{K}\cdot\frac{d_{\text{out}}}{K})=\frac{d_{\text{in}}\cdot d_{\text{out}}}{K}$ (see Eqn. 2, assume they are equally divided into each category of user action features). Therefore, tensor-based GCN reduces network complexity by $(1-\frac{1}{K})d_{\text{in}}\cdot d_{\text{out}}$ number of parameters. Similarly, the standard LSTM layer has $4(d_{\text{in}}\cdot d_{\text{out}}+d_{\text{out}}^2+d_{\text{out}})$ trainable parameters (corresponding to the input transition, hidden state transition, and the bias); while the tensor-based LSTM layer only maintains $4(\frac{d_{\text{in}}\cdot d_{\text{out}}}{K}+\frac{d_{\text{out}}^2}{K}+d_{\text{out}})$ number of parameters (for $\mathcal{U}_*$, $\mathcal{U}_*^{\mathbf{h}}$ and $\mathbf{b}_*$ in Eqn. 7). As a result, the total number of parameters is reduced by $4(1-\frac{1}{K})(d_{\text{in}}+d_{\text{out}})d_{\text{out}}$ 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 $N^2\cdot d_{\text{in}}+N\cdot d_{\text{in}}\cdot d_{\text{out}}$ and $N^2\cdot\frac{d_{\text{in}}}{K}\cdot K+N\cdot\frac{d_{\text{in}}}{K}\cdot\frac{d_{\text{out}}}{K}\cdot K=N^2\cdot d_{\text{in}}+N\cdot\frac{d_{\text{in}}\cdot d_{\text{out}}}{K}$, respectively. The reduction is then $N(1-\frac{1}{K})d_{\text{in}}\cdot d_{\text{out}}=O(d_{\text{in}}\cdot d_{\text{out}})$. For an LSTM layer (Eqn. 7), it takes $4(d_{\text{in}}\cdot d_{\text{out}}+d_{\text{out}}^2)+3d_{\text{out}}$ multiplications to update its hidden and gate, while the tensor-based LSTM layer takes only $4(\frac{d_{\text{in}}}{K}\cdot\frac{d_{\text{out}}}{K}\cdot K+\frac{d_{\text{in}}^2}{K^2}\cdot K)+3d_{\text{out}}=4(\frac{d_{\text{in}}\cdot d_{\text{out}}}{K}+\frac{d_{\text{out}}^2}{K})+3d_{\text{out}}$ multiplications. Thus, the reduction of computational complexity by the tensor-based LSTM is $O((d_{\text{in}}+d_{\text{out}})d_{\text{out}})$. □

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 aligned 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$_{ts}$ and FATE, to demonstrate the improvement by using tensor-based designs for FATE over a non tensor-based model FATE$_{ts}$ Both training and testing (inference) run times are reported in 3. We can see that training FATE takes significantly less time then FATE$_{ts}$ 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$_{ts}$.

		Region 1		Region 2	
		Train	**Test**	**Train**	**Test**
Task 1	**FATE$_{ts}$**	216.96	148.20	132.25	90.42
	FATE	**181.35**	**119.40**	**117.70**	**73.43**
Task 2	**FATE$_{ts}$**	207.23	151.48	137.50	89.61
	FATE	**172.00**	**115.10**	**110.92**	**69.05**

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.

	Region 1	Region 2
Time period	09/16/2019 - 10/27/2019	
Avg. # users	153006	108452
Avg. node degree	51.58	36.95
# node features	13	
# edge features	3	

Table 5: Selected features for user actions on Snapchat.

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

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 0.5 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:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{S}|}\sum_{u\in\mathcal{S}}(e^u - \hat{e}^u)^2},$$
$$\text{MAPE} = \frac{1}{|\mathcal{S}|}\sum_{u\in\mathcal{S}}\frac{|e^u - \hat{e}^u|}{\hat{e}^u},$$
$$\text{MAE} = \frac{1}{|\mathcal{S}|}\sum_{u\in\mathcal{S}}|e^u - \hat{e}^u|, \tag{17}$$

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