Bipartite Dynamic Representations for Abuse Detection

Andrew Z Wang[1], Rex Ying[1], Pan Li[2], Nikhil Rao[3], Karthik Subbian[3], Jure Leskovec[1]
[1]Department of Computer Science, Stanford University, Stanford, CA
[2]Department of Computer Science, Purdue University, West Lafayette, IN
[3]Amazon, Palo Alto, CA
anwang@cs.stanford.edu,rexying@stanford.edu,panli@purdue.edu,{nikhilsr,ksubbian}@amazon.com,jure@cs.stanford.edu

ABSTRACT

Abusive behavior in online retail websites and communities threatens the experience of regular community members. Such behavior often takes place within a complex, dynamic, and large-scale network of users interacting with items. Detecting abuse is challenging due to the scarcity of labeled abuse instances and complexity of combining temporal and network patterns while operating at a massive scale. Previous approaches to dynamic graph modeling either do not scale, do not effectively generalize from a few labeled instances, or compromise performance for scalability. Here we present BiDyn, a general method to detect abusive behavior in dynamic bipartite networks at scale, while generalizing from limited training labels. BiDyn develops an efficient hybrid RNN-GNN architecture trained via a novel stacked ensemble training scheme. We also propose a novel pre-training framework for dynamic graphs that helps to achieve superior performance at scale. Our approach outperforms recent large-scale dynamic graph baselines in an abuse classification task by up to **14%** AUROC while requiring **10x** less memory per training batch in both open and proprietary datasets.

CCS CONCEPTS

• **Information systems** → *Content ranking*.

KEYWORDS

Fraud Detection;Anomaly Detection;Graph Neural Networks

ACM Reference Format:
Andrew Z Wang[1], Rex Ying[1], Pan Li[2], Nikhil Rao[3], Karthik Subbian[3], Jure Leskovec[1]. 2021. Bipartite Dynamic Representations for Abuse Detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467141

1 INTRODUCTION

Detecting abuse is an important problem in online social communities and e-commerce websites. In online social communities (such as Reddit or Wikipedia), propagating misinformation, trolling, and using offensive language are considered as abuse [5, 19]. In e-commerce websites, abuse consists of fradulent activities such as artificially raising the search ranking of an item via fake reviews or purchases [19]. Such abusive behavior reduces user trust, engagement, and satisfaction.

The online activity of abusive users tends to have significantly different interaction patterns compared to genuine users. For example, in e-commerce websites, abusive customers' engagement with items (e.g., daily clicks or purchases) follows fluctuating patterns [2]. Similar patterns are observed in online forums, where abusive users target a small set of articles by posting harmful comments [5]. In the above scenarios, an accurate machine learning approach is required to model the structural and temporal dynamics of users.

There are three major challenges in detecting abuse in real-world applications. (1) Abuse in online communities or e-commerce websites manifests itself over time, giving rise to *dynamic* graph structure. Common abusive behaviors, such as continuously clicking on an item to boost its ranking, and offering such fraudulent services to several abusive items [19], involve both temporal and graph-structured aspects of the data. Thus, it is crucial to effectively combine both sources of information. (2) The real-world abuse detection problem in industry requires training and prediction on extremely large-scale graph datasets containing hundreds of millions of nodes and edges [27]. (3) It is prohibitively costly to obtain human annotated labels for abusive users to train abuse detection models, and this difficulty is compounded by the rarity of abusive behavior [1]. Hence, the model should be able to learn from a very small amount of annotated data.

There is a plethora of existing work on dynamic graph models [7, 13, 20, 22, 26]. However, the challenge is to combine the information at scale while maintaining model performance. For instance, dynamic graph modeling methods that model the full interrelationships between users and items incur great runtime costs, limiting them to small datasets [7]. Another line of work based on graph neural networks (GNNs) [1, 17, 26] is limited to shallow networks due to the exponential memory cost of increasing model depth with minibatch training. On the other hand, scalable graph modeling techniques based on random walk propagation [12] are able to efficiently handle large-scale graphs, but they do not capture dynamic edge feature information and therefore cannot be extended to dynamic graph settings.

In this paper, we address the above challenges and propose a novel dynamic graph model BiDyn focused on abuse detection.[1] We apply our work to a large e-commerce dataset as well as to public datasets. Our work has the following components:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '21, August 14–18, 2021, Virtual Event, Singapore
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467141

[1]Project website with data and code: http://snap.stanford.edu/bidyn

Joint modeling of time and graph information: We introduce an efficient neural architecture for dynamic graph data on massive graphs called **BiDyn** (Bipartite Dynamic Representations). BiDyn can handle both static and dynamic features via recurrent (RNN) and graph neural network (GNN) modules. We propose compact feature representations and minibatching practices to ensure low memory usage, and set aggregations to handle simultaneous events in a dynamic graph. We outperform the most recent dynamic graph baselines (TGAT and JODIE) by up to **14%** in AUROC on an abuse classification task in both open and proprietary datasets.

Scalable training scheme: We introduce a scalable alternating training scheme for our model that is applicable to general node classification on dynamic bipartite graphs (Figure 1). The training scheme evolves node embeddings by training on user and item nodes in alternating optimization rounds. Training on one class of nodes at a time allows for training of deep non-linear GNNs without the memory expense of end-to-end training. Our novel training scheme makes our model **10X** more memory-efficient than popular baselines, making training of deeper models feasible, thus leading to significant performance gains.

Self-supervised pretraining framework: We tackle the problem of label sparsity by leveraging inductive biases from domain knowledge. We propose a memory-efficient, self-supervised pretraining task that simultaneously combines sequence and graph autoencoder objectives, allowing the model to generalize from limited training labels. We demonstrate that this task provides an initial separation of abusive and normal nodes before the main training stage, providing a significant performance boost.

The rest of the paper is organized as follows. We summarize the related work in Section 2. We define our problem and notation in Section 3. We also provide a motivating analysis for our model, via observations from a subsampled e-commerce dataset in this section. Our model along with a scalable training mechanism and pretraining routines is described in Section 4. We provide extensive empirical evidence demonstrating the practical use of our method in the presence of a) large datasets and b) label sparsity when compared to several baselines on multiple datasets in Section 5. We explain practical deployment strategies in Section 6.

2 RELATED WORK

Dynamic graph modeling. General machine learning models have been introduced for dynamic graph data [7, 13, 17, 20, 22, 26]. However, due to the high number of events associated with each node compared to static graphs, dynamic graph models suffer from high memory requirements needed to maintain a wide receptive field in both time and graph. Existing models either do not scale to large graphs [7] or make concessions by not propagating gradients back in time [13, 20], or by using a very small receptive field in large graphs [22, 26]. As a result, existing models either cannot be applied to large-scale graphs, or are limited to short-term predictions [13, 20]. In contrast, BiDyn models node behavior considering a long time range and large number of interactions (edges), crucial for high performance in abuse detection.

Scalable graph neural networks. A number of scalable static graph neural network methods have been proposed [12, 24, 25, 27]. Such methods decouple model depth from receptive field size, by either removing nonlinearities between model layers [24] or using random walks to propagate messages over long distances [12, 27]. These techniques reduce the runtime and memory usage of the models. However, removing nonlinearities also reduces the expressiveness of the model, potentially reducing performance for complex tasks. Furthermore, unlike previous works, BiDyn extends scalable static GNNs to the dynamic graph setting.

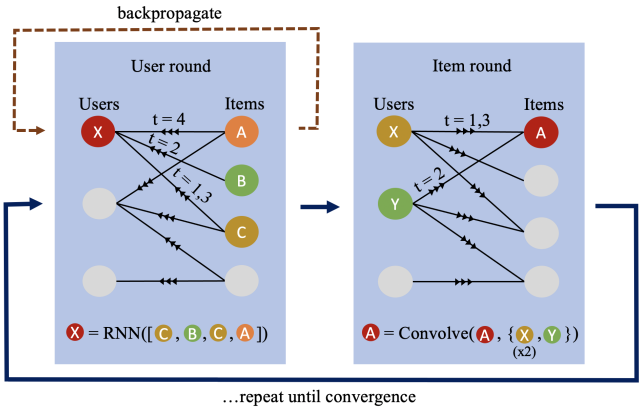


Figure 1: BiDyn iteratively learns embeddings through alternating item and user updates. In the item round, a graph convolution is applied to each item's neighbors. In the user round, a recurrent neural network is applied to each user's neighbors. This training scheme improves efficiency by 10x over popular baselines, and allows BiDyn to run on datasets with more than 100M edges.

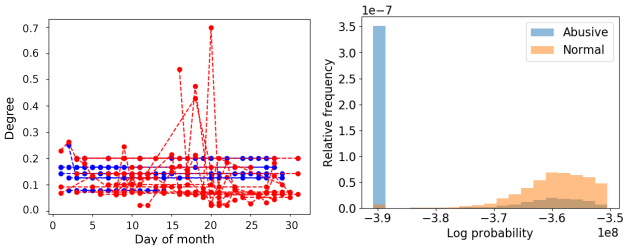
Anomaly detection. Unsupervised approaches have been introduced for detecting anomalous behavior both for static [2, 14, 15, 19] and dynamic [3, 28, 30] graphs. A general principle for anomaly detection is to look for nodes whose behavior has low probability compared to normal behavior [4]. Such methods circumvent the need to collect expensive ground truth labels, but are limited to detecting specific behaviors from prior knowledge. This strategy is fundamentally hard for detecting abuse, since a shrewd abuser will look to "blend in" with normal users [1], making unsupervised anomaly detection methods perform poorly. BiDyn demonstrates that combining scalable pretraining and supervised fine-tuning effectively incorporates inductive bias from this probabilistic approach while also making use of supervised training labels.

3 ABUSE DETECTION PROBLEM

We first introduce our problem setup and the model architecture. We further demonstrate an efficient training algorithm to scale to large-scale graph datasets. Finally, we use pretraining to improve performance in rare-label scenarios, crucial for abuse detection.

3.1 Problem Setup and Notation

Throughout, let $[x_1, \ldots, x_n]$ represent an ordered sequence and ; denote concatenation of vectors. We consider dynamic bipartite graphs $G = (V, E, f, g, h)$ with nodes representing users $A \subset V$ and items $B \subset V$ ($A \cup B = V, A \cap B = \emptyset$), and timestamped edges between them, of the form $(u \in V, v \in V, t \in \mathbb{R}) \in E$, where t is



(a) Temporal signals of abuse. (b) Graph signals of abuse.

Figure 2: (a) Degree of abusive and non-abusive users in the e-commerce network as a function of time. Abusive users (red) have more fluctuating activity levels than normal users (blue), which remain relatively constant. (b) Abusive nodes in the e-commerce network have systematically lower log-probability than normal nodes under a graph autoencoder model.

the timestamp, and $u \in A$ iff $v \in B$. Note that E may be a multiset if there are co-occurring events between two nodes. The graph is considered to be undirected ($(u, v, t) \in E$ iff $(v, u, t) \in E$). We also assume without loss of generality that all timestamps occur in a time interval $[0, T_{max}]$. Let $N(u) = \{(u', v, t) \in E \mid u' = u\}$ denote the set of temporal neighbors of node u, also referred to as u's events. Let $w(u, v)$ denote the "weight" of the static edge between u and v, defined by the number of events that occur between them, $w(u, v) = |\{(u', v', t) \in E | u = u', v = v'\}|$. Nodes and edges may contain arbitrary feature vectors; let $f(v) \in \mathbb{R}^d$ denote the features of a user or item node v, and $g(u, v, t) \in \mathbb{R}^{d_e}$ denote the features of edge (u, v, t). The goal is to predict the binary label $y_v \in \{0, 1\}$ of each user $v \in A$. We consider a transductive setup in which a subset of the user labels are visible during training, and the goal is to predict the labels of the remaining nodes in the graph.

3.2 Motivating Analysis

Due to the challenges outlined in the previous section for detecting abusive behavior, it is important we understand the ways in which it manifests itself in real-world datasets. We first perform an observational study of such behavior on an anonymized e-commerce network dataset. The analysis provides insights to building an effective model for abuse detection.

Dynamic graph information. Time series information is important in modeling abuse in the e-commerce domain. Figure 2(a) shows the degree (a proxy for amount of activity) on each day for both normal (blue) and abusive (red) nodes. We see that the abusive nodes fluctuate much more than normal nodes – this "bursty behavior" distinguishes normal and abusive nodes. Furthermore, we see that it's important to model the whole sequence of events, taking the event ordering into account: as shown in Appendix Table 6, an LSTM model that uses the event sequence to predict the node label outperforms a model that predicts the node label based on statistics about the distribution of purchase counts (i.e. ignoring time order, as in a purely graph-based approach).

Table 1 shows that abusive actors tend to interact with each other, suggesting that trustworthiness propagates between nearby

Graph statistic	Abusive	Normal
% abusive neighbors (users)	~ 1	~ 0.3
% abusive neighbors (items)	~ 1.75	~ 1.5
Degree centrality	$1.4 * 10^{-4}$	$2.1 * 10^{-5}$

Table 1: High level statistics of the e-commerce network. We see that there's homophily in the data: abusive nodes tend to associate more with other abusive nodes, and also see more activity.

nodes. Additionally, abusive nodes also tend to have high activity, with higher centrality in the network. Homophily and structural information about a node's local neighborhood are thus important features of abuse [1], which can be leveraged via transductive graph models.

Unsupervised modeling of user behavior. Both temporal and graph models of user behavior reveal differences between abusive and non-abusive nodes. In a toy experiment, here we use unsupervised models to provide evidence for the importance of graph structure and temporal information.

- **Abusive nodes have unpredictable event sequences**: We use an RNN autoencoder to reconstruct a given item's purchase counts for each day in the dataset. The reconstruction error (MSE) is significantly higher for abusive items, and predicting the abuse label solely based on the reconstruction error achieves 0.6 AUROC. Hence, the temporal behavior of nodes themselves provides important information about potential abusive behaviors.
- **Abusive nodes have lower likelihood under generative model**: We use a graph variational autoencoder [10] to reconstruct the (static) graph, ignoring all temporal information. Given the static adjacency matrix M ($M_{ij} = 1$ if $\exists t : (i, j, t) \in E$ else 0), the model learns an embedding vector $z_i \in \mathbb{R}^d$ for each node such that the probability of each user node $u \in A$'s connections is given by:

$$P(u) := \prod_{j \in B} p(M_{ij} | z_i, z_j), \text{ with } p(M_{ij} = 1 | z_i, z_j) = \sigma(z_i^T z_j), \quad (1)$$

  where σ is the logistic sigmoid function. We observe that the probability assigned by the generative model is significantly lower for abusive nodes than regular nodes ($p < 0.001$), as shown in Figure 2(b). Hence, a likelihood model of all nodes in the graph provides a valuable way to distinguish anomalous nodes.

3.3 Design Choices

The observations in Section 3.2 provide insights into important components of an abuse detection model, which we take as design choices for our model. The model should (1) combine modules which consider the temporal and the graph nature of the interaction data; (2) use a pretraining objective to model whether nodes are anomalous in their time or graph behavior, so that anomalous nodes can be automatically identified by the model even with limited training data. Next we develop a scalable model that satisfies these conditions.

4 PROPOSED METHOD: BIDYN (BIPARTITE DYNAMIC REPRESENTATIONS)

BiDyn can be viewed as a stacked ensemble learning method [23] for transductive learning on dynamic graphs, achieving high performance by layering many copies of a core architecture and training them efficiently. The core of the model is a streamlined architecture combining recurrent and graph neural network components to model time and graph information. Importantly, we introduce a scalable training scheme for this model that enables training of the core model on large real-world graphs while maintaining a wide receptive field. Finally, we introduce a pretraining framework within this training scheme that allows incorporation of prior domain knowledge to tackle label sparsity. We develop an efficient dynamic graph modeling task that is particularly suited to detecting abuse and other rare events. The architecture, training method and pretraining framework combine to form a scalable, powerful method for detecting abuse in production-scale dynamic networks.

4.1 Core Model Architecture

BiDyn consists of a Recurrent Neural Network (RNN) phase, followed by a Graph Neural Network (GNN) phase. We process the time and graph information through these separate components in order to allow each component to receive tailored representations of each information source. In particular, in the RNN phase, we aggregate co-occurring events to obtain a sequence of event counts over time, resulting in a highly memory-efficient node feature representation that can also capture bursty behavior. In the GNN phase, we aggregate neighbors of a node regardless of their interaction times, allowing the model to capture homophily effects and enabling memory-efficient batching by subsampling from a node's set of neighbors at every layer.

RNN phase. In the RNN phase, an RNN is applied to each node; a separate RNN is used for each node type. In the bipartite graph setting, we use RNN_A to denote the RNN component for users and RNN_B for items. Each RNN receives as input an encoding of the sequence of neighbors $N(u)$ of the given node u, in time order. In particular, for a given node u, it receives a sequence

$$[f(v_1); g(u, v_1, t_1); \phi(t_1), \ldots, f(v_k); g(u, v_k, t_k); \phi(t_k)] \quad (2)$$

for each $(u, v_i, t_i) \in N(u)$, where $t_1 \leq t_2 \leq \ldots \leq t_k$. Here $\phi(t_i) : \mathbb{R} \to \mathbb{R}^{D_{\text{time}}}$ is a sinusoidal encoding of the timestamp [21], defined per entry by

$$\phi_{2j}(t_i) = \sin(\frac{100}{T_{max}} t_i / 10000^{2i/D_{\text{time}}}),$$
$$\phi_{2j+1}(t_i) = \cos(\frac{100}{T_{max}} t_i / 10000^{2i/D_{\text{time}}}), \quad (3)$$

where D_{time} is the dimension of the encoding. The last hidden state of the RNN is concatenated with u's features $f(u)$ to produce an RNN output $h_u^0 := (RNN_A(\text{u}); \text{f(u)})$ for users $u \in A$ and $(RNN_B(\text{u}); \text{f(u)})$ for items $u \in B$. We use a 2-layer LSTM [9] as the RNN architecture due to its best performance in handling long-range dependencies.

GNN phase. In the GNN phase, the per-node outputs from the RNN phase are propagated throughout the *static* network to generate a prediction for each node. To compute the prediction for node u, we first compute its embedding by applying multiple GNN layers to the RNN outputs h_u^0, which are used as the initial node features to the GNN. To get from the embedding h_u^l at GNN layer l to the embedding h_u^{l+1} at layer $l+1$, we apply a modification of SAGE convolution [8] that incorporates edge weights, setting

$$m_{v,u}^l = \text{ReLU}(W_{msg}^l(h_v^l; w(u, v)))$$
$$h_u^{l+1} = \text{ReLU}\left(W^l \left[h_u^l; \sum_{(u,v,t) \in N(u)} m_{v,u}^l\right]\right). \quad (4)$$

The final embeddings h_u^L are then fed through a logistic regression classifier to produce the predicted abuse probability and predicted label $\hat{y}_u$. To enable the model to fit in GPU memory, we use minibatching with random neighbor sampling in the GNN phase: at each layer, instead of considering the full set $N(u)$ of neighbors in the aggregation function, we subsample up to D elements from $N(u)$ uniformly at random.

Extension to coarse-grained timestamps. In many settings, events are timestamped on a coarse-grained scale, such as to the nearest day. Subsequently, many events have identical timestamps. For example, multiple users interact with an item on a single day, and the exact sequence in which these users interacted with the item is not known. BiDyn makes sure that the order in which simultaneous events are fed to the RNN does not affect the final model output.

We propose an extension to our RNN to make our model invariant to reordering of simultaneous events. For a given node u, the RNN will receive a sequence $[\text{Agg}(S_0), \ldots, \text{Agg}(S_{T_{max}})]$ of length T_{max} as input, where Agg is a permutation-invariant function and $S_t = \{v | (u', v, t') \in E, t' = t, u' = u\}$ is the (multi)set of u's neighbors at time t. We use the aggregation function $\text{Agg}(S_t) = (\frac{1}{|S_t|} \sum_{v \in S_t} (f(v); g(u, v, t)); |S_t|)$, concatenating the mean node and edge features with the number of events on each day, since we find that the mean generalizes better than more complex approaches such as DeepSets [29], while we found that the number of events is a useful feature for detecting burstiness. Note that the choice of Agg is the degree-scaling used in principal neighborhood aggregation [6] to better distinguish neighborhoods with similar features. We adopt this aggregation scheme when testing the core architecture on large datasets (e-commerce) in the experiments (RNN-GNN), which proves to be effective and memory efficient.

4.2 Scalable Training

While our core architecture is an effective dynamic network model, as we demonstrate in the experiments, it suffers from high memory usage when performing minibatch training on large datasets[1], since memory usage has complexity $O(D^L)$ with increasing number of GNN layers L, where D is the maximum number of neighbors subsampled at each layer during minibatching. The dynamic graph setting further compounds this memory problem. First, since temporal edges often represent interactions such as edits, clicks or purchases, the fanout D is often very large (order of hundreds). Second, prepending an RNN module to the GNN further exacerbates the memory cost of backpropagation, in practice limiting L to be 1 or 2 in order for the model to fit within a GPU minibatch. These difficulties combine to make training the core model in a standard end-to-end fashion impractical for large-scale, real-world networks.

[1]Existing alternatives to minibatching, such as SIGN [18], cannot be applied due to the dynamic RNN component.

Algorithm 1 Alternating training of users and items

$X \in \mathbb{R}^{|V| \times d}$ is node embedding matrix with rows x_u ($\forall u \in V$)
$x_u \leftarrow 0, \forall u \in V$
while stopping condition not reached **do**
  for all $u \in A$ **do** ▷ User round
    Clear gradients
    Let $k = |N(u)|$, $\{(u, v_i, t_i)\} = N(u)$ where $t_1 \leq \ldots \leq t_k$
    $x_u \leftarrow \text{RNN}_1([f(v_1); g(u, v_1, t_1); \phi(t_1); x_{v_1}, \ldots,$
      $f(v_k); g(u, v_k, t_k); \phi(t_k); x_{v_k}])$
    $L \leftarrow \text{CrossEntropy}(\text{LogisticRegression}(x_u), y_u)$
    Backpropagate L
  end for
  for all $u \in B$ **do** ▷ Item round
    $x_u \leftarrow \text{Convolve}(u, N(u), X)$
  end for
end while
return X

We tackle this difficulty by introducing an alternative training scheme that iteratively updates node embeddings by stacking multiple copies of the core architecture, circumventing end-to-end training. We (1) extend the core architecture to a deep model that alternates between the RNN and graph convolution layers, and (2) train this model one layer at a time. Although training is no-longer end-to-end, it makes training of a much deeper model possible, which in practice results in significant performance improvement.

Figure 1 shows the flow of information through the model. Throughout training, BiDyn maintains a table of current embeddings of all nodes, $x_u \in \mathbb{R}^d$, and alternates between updating user and item embeddings, as shown in Algorithm 1.

User round. In the user round (left component of Figure 1), the RNN phase is applied to generate updated user embeddings, which are saved back to the table and trained on the prediction task to update the RNN and logistic regression weights. With each event fed into the RNN, we concatenate the embedding of the associated item. Hence, the updated embedding of user $u \in A$ is computed as

$$x_u = \text{RNN}_1([f(v_1); g(u, v_1, t_1); \phi(t_1); x_{v_1}, \ldots, f(v_k); g(u, v_k, t_k); \phi(t_k); x_{v_k}]), \quad (5)$$

a function of the sequence of node and edge features, time encoding and node embedding for each event. The user embeddings are fed into a logistic regression layer to compute the final abuse prediction for each user. The RNN is trained by backpropagating from the predicted abuse scores (dotted line in Figure 1). We apply the RNN in the user round since users were observed to be the source of bursty behavior (Figure 2(a)).

Item round. In the item round (right component of Figure 1), a graph convolution (i.e. a layer from the GNN phase) is applied to generate updated item embeddings by propagating the neighboring user embeddings. Hence, the updated embedding of item $u \in B$ is computed as

$$x_u = \text{Convolve}(u, N(u), X) \quad (6)$$

for some graph convolution layer Convolve.

In summary, each round improves either the user or item embeddings using the updated embeddings from the previous round.

	GNN	BiDyn	APPNP-I
Memory usage per batch	$O(D^L)$	$O(D)$	$O(1)$
Receptive field radius	L	L	L
Number of nonlinearities	$O(L)$	$O(L)$	0

Table 2: The scalable training scheme of BiDyn is memory-efficient, with memory cost for each training batch comparable to that of a scalable static GNN, APPNP-I. Simultaneously, it is capable of a wide receptive field and deep network, enabling high performance. Here D is the maximum number of neighbors subsampled at each layer during mini-batching, and L is the number of layers in the model.

This mutually recursive relationship ensures that the embeddings continually improve over time. Finally, the process terminates when a stopping condition for training is reached; in our experiments, we train for a fixed number of training rounds, then evaluate the model with lowest validation loss.

Graph convolution without learnable parameters. Since there is no gradient flow between the user and item round, we opt to simplify the GNN convolution layer in the item round to a layer without learnable parameters, while the user round uses a learnable RNN to characterize user preferences through aggregation of items over time. Fixed graph convolutions have been shown to be effective in recent simplified graph neural network methods [12, 24]. In the spirit of these methods, we divide the separate roles of traditional graph convolution operations into separate model components, using a fixed convolution operation to widen the receptive field of the model, and successive applications of the RNN component to increase the neural network depth. Hence, we define

$$\text{Convolve}(u, N(u), X) = \alpha x_u + (1 - \alpha) \sum_{v \in N(u)} x_v, \quad (7)$$

a sum aggregation with running average update. The use of sum aggregation can be viewed as a multi-dimensional version of the update step of the HITS algorithm for identifying trustworthy nodes in a network [11]. We demonstrate in the experiments that sum aggregation performs better than more complex alternatives, such as autoencoders, while also being much more efficient. We also compare against mean aggregation, which performs better in some cases, in the experiments. The hyperparameter $\alpha \in [0, 1]$ controls the rate of diffusion of information throughout the network; thus, a high α will more heavily prioritize nodes closer to u, better preserving the local neighborhood. We also design a convolution operation in the setting with coarse-grained timestamps:

$$\text{Convolve}(u, N(u), X) = \alpha x_u + (1 - \alpha) \frac{1}{|N(u)|} \sum_{t=0}^{T_{\max}} \frac{1}{|S_t|} \sum_{v \in S_t} x_v \quad (8)$$

with the α and S_t as defined from before. We choose this convolution empirically, with ablations in the experiments. Overall, only the RNN component of BiDyn contains learnable parameters, hence backpropagation only occurs during the user round.

Memory usage. As shown in Table 2, the stacked ensemble training scheme enables the model to take advantage of many of the performance benefits of a deep model without the memory expense

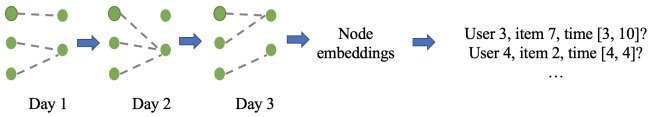


Figure 3: The "random access query" pretraining task learns static node representations that can be used to make dynamic predictions about whether an event occurs between a user and item in a given time range.

of end-to-end training. The memory usage of BiDyn approaches that of the inference-time variant of APPNP (denoted by APPNP-I) [12], a scalable GNN model that trains a classifier on individual nodes and uses random walk propagation at inference time only. Each round of backpropagation for BiDyn only requires querying a node's direct neighbors, only uses linear memory $O(D)$, and mini-batches can easily fit on a GPU, even with very large number of neighbors D[1]. At the same time, the receptive field of the model is not bounded by memory constraints, since each user or item round increases the receptive field radius by one. Furthermore, each user round adds at least one additional nonlinearity for all embeddings that pass through it, so this stacking approach naturally receives some of the performance benefits of a deeper model.

4.3 Pretraining Framework

We further propose a scheme for pretraining in this alternating training framework, in order to benefit from unsupervised objectives. To pretrain on an auxiliary objective, simply train on this auxiliary objective during the user round for a number of epochs, then switch to the main objective. This method effectively creates a prior on both the embeddings and the network weights.

Due to the label sparsity common to abuse tasks, we propose an unsupervised pretraining task that can serve as a prior. We develop a pretraining task that draws from the anomaly detection approach to abuse detection, which leverages the phenomenon that abusive nodes will have lower probability than normal nodes under normal nodes (Observation 1) [4]. Hence, we propose a probabilistic model of user behavior in the dynamic graph setting under which abusive users will be distinct from normal users, thereby providing an initial separation of nodes for classification. Since user activity on online platforms often includes both time series and graph information, we propose a simple pretraining task that generalizes probabilistic time series and graph models. We assume a probabilistic model of user behavior in which each node u has an associated latent vector z_u, and the distribution of events (dynamic edges) between two nodes $u \in A$ and $v \in B$ is a general point process that depends only on z_u and z_v:

$$\log P(G|Z) = \sum_{u\in A}\sum_{v\in A}\Big(\sum_{t:(u,v,t)\in N(u)} \log\lambda(z_u, z_v, t) + \int_0^{T_{\max}} (1-\lambda(z_u, z_v, t))dt \Big) \tag{9}$$

[1]Note that storage requirements for the graph G and the embedding matrix X are not included in the memory requirements for any of the methods in Table 2, since they they do not need to be stored on GPU during backpropagation.

where Z denotes the matrix of all latent vectors z_u, and

$$\lambda(z_u, z_v, t) := \lim_{\Delta t\to 0}\frac{1}{\Delta t}P(\exists t' \in [t, t+\Delta t] : (u, v, t') \in E | z_u, z_v, t) \tag{10}$$

gives the intensity of an event occurring between nodes u and v at time t, given their latent vectors and the observed history of events between the two nodes until time t.

We propose the following "random access query" pretraining task which models dynamics on several levels of resolution. Our task is to learn an embedding z_u for each node in the graph. The task is to predict, for a user-item pair ($u \in A, v \in B$) and time interval $[t_1, t_2]$, and given u's history of events until time t_1, whether an edge appears between (u, v) in the desired time interval. We mask out all edges associated with u occurring at any time $t \geq t_1$. We use a multi-layer perceptron (MLP) to estimate the probability of the edge appearing within this time interval:

$$P(\exists(u, v, t) \in E, t \in [t_1, t_2] | z_u, z_v, t_1) = \sigma(\text{MLP}(z_u, z_v, \phi(t_1), \phi(t_2))) \tag{11}$$

In principle, modeling the behavior within every time interval allows us to estimate λ (by taking $t_2 \to t_1$), hence the task fully models the time series between two nodes, while being extremely lightweight in terms of memory consumption (requiring no back-propagation through time). Furthermore, the random access objective allows for simultaneous modeling of different time scales: a model trained on this objective can perform both time series and graph autoencoder tasks, predicting events at a specific time when $t_1 \approx t_2$ and predicting static links when $t_1 = T_{\min}$ and $t_2 = T_{\max}$.

The objective, given an edge $(u, v, t) \in E$ and arbitrary time interval $[t_1, t_2]$, is as follows:

$$\begin{aligned} L(u, v, t_1, t_2) = {} & M(z_u, z_v, f(u), f(v), \phi(t_1), \phi(t_2)) \\ & + M(z_u, z_{v'}, f(u), f(v'), \phi(t_1), \phi(t_2)) \end{aligned} \tag{12}$$

Here M is the cross entropy loss of the MLP, where the label is whether an edge truly appears between the given user and item in the given time interval. v' is a randomly chosen (perturbed) item node (hence the second term represents negative examples). We encode t_1 and t_2 using the sinusoidal encoding ϕ used previously. We optimize this objective for uniformly randomly selected time intervals and perturbed item nodes.

5 EXPERIMENTS

We perform many comparisons to demonstrate the efficacy of each component of our model. We demonstrate that BiDyn effectively predicts abuse on several domains, thus the approach is general.

(1) BiDyn's dynamic graph architecture shows favorable performance in the transductive prediction setting compared to both ablations and alternative dynamic graph methods.
(2) Alternating training improves performance across datasets compared to end-to-end training, while using much less GPU memory, enabling higher throughput for production-scale use cases.
(3) Our pretraining framework and task improve performance compared to alternatives.

Experimental setup. We consider a transductive framework in which the entire dynamic graph is visible during training, but only a subset of the user labels are visible. The goal is to predict the labels

of the remaining users in the graph. 5% of the labels are visible during training; the remainder are split equally between validation and test. We train all models on all datasets for 20 epochs and take the test performance on the model with lowest validation loss.

Datasets. We perform experiments on the following proprietary (**e-commerce**) and open dynamic graph datasets (**Wikipedia**,**Reddit**).

- **e-commerce**. We used anonymized and subsampled data from an e-commerce website. The data is subsampled in a manner so as to be non-reflective of actual production traffic. The data consists of 500K users (1K positive), 6M items and 113M interactions aggregated over an arbitrary 2 month window. Labels indicate users marked for abusive behavior.
- **Wikipedia**. The Wikipedia dataset consists of users linked to the pages they edit. Positive users indicate those who were banned in the data collection period. There are 8227 users (187 positive), 1000 pages and 157474 interactions.
- **Reddit**. This dataset consists of users linked to the communities they participate in. Positive users indicate those who were banned in the data collection period. There are 10000 users (333 positive), 1000 communities and 672447 interactions.

Baselines. We compare against alternative dynamic graph models, as well as models that only use graph or time information.

- **Dynamic graph models**. We compare against TGAT [26], JODIE [13], DyRep [20] and TGN [17], four recently-proposed dynamic graph models. TGAT uses a GNN with temporal attention layers to attend to a node's history. JODIE predicts trajectories of node embeddings over time as a form of self-supervision. DyRep models dynamics jointly on small and large time scales. TGN makes use of memory modules and graph-based operators. We adapt them to the transductive setting by training their dynamic node embeddings to predict the abuse labels (details in Appendix). We also compare against the core model architecture, RNN-GNN, with standard end-to-end training, using a 2-layer GNN (the maximum that can fit in GPU memory for the e-commerce dataset).
- **GNN models**. We compare against a static GNN to demonstrate that incorporating time information can improve performance in the transductive setting.
- **Recurrent models**. We compare against an RNN model with time encoding to demonstrate the value of a semisupervised approach (using information from neighboring nodes).

See the Appendix for implementation details and extended experimental results.

5.1 Model Comparison

Table 3 shows the performance of BiDyn in AUROC compared to the baselines. We compare different graph convolution variants: BiDyn (sum) represents the convolution in equation (6), BiDyn (mean) uses equation (7) but takes the mean $\frac{1}{|N(u)|}\sum_{v\in N(u)} x_v$ rather than the sum $\sum_{v\in N(u)} x_v$ over embeddings, and BiDyn (coarse) is the convolution in equation (8).

We evaluate the e-commerce data in both the regime where 5% of labels are seen in training ("e-com (5%)") and 50% of labels are seen in training ("e-com (50%)") (the 5% regime is used for the remainder of the experiments).

	Method	e-com (5%)	e-com (50%)	Wikipedia	Reddit
Ablation	GNN	+0.0	+0.0	69.6 ± 0.2	53.7 ± 1.8
	RNN	+0.0 ± 0.4	+1.8 ± 0.2	78.0 ± 2.3	51.3 ± 4.7
	RNN-GNN	-2.2 ± 0.5	+1.9 ± 0.5	70.0 ± 0.2	53.8 ± 2.2
Baselines	TGAT	-4.0 ± 0.1	-2.5 ± 0.6	73.6 ± 4.7	51.5 ± 2.9
	TGN	OOM	OOM	49.0 ± 0.6	**67.0 ± 0.6**
	DyRep	OOM	OOM	52.5 ± 0.2	61.4 ± 0.8
	JODIE	OOM	OOM	53.0 ± 0.5	61.2 ± 0.4
Ours	BiDyn (coarse)	+1.2 ± 0.5	**+3.9 ± 0.1**	—	—
	BiDyn (mean)	—	—	80.5 ± 2.3	56.0 ± 3.3
	BiDyn (sum)	—	—	86.5 ± 1.6	46.8 ± 3.9
	+ pretraining	**+4.5 ± 0.5**	+1.6 ± 0.2	**87.5 ± 0.6**	50.5 ± 2.6

Table 3: Model comparison (AUROC; absolute gain for e-commerce) classifying whether users in the network are abusive. For e-commerce, we report model performance relative to a GNN, while for public datasets we report absolute numbers. We observe that the scalable training scheme of BiDyn gives performance benefits over end-to-end training of an RNN-GNN architecture, as well as baseline dynamic graph models. Pretraining leads to further improvements in the regime with limited training labels. Many baselines do not scale, running out of memory (OOM) on e-commerce. '–' denotes entries that do not apply, e.g. BiDyn (coarse) only applies to datasets with coarse-grained timestamps.

Dynamic graph models. We see that BiDyn (even with standard end-to-end training), and indeed the pure RNN model, outperforms TGAT in the transductive setting. Hence, we confirm the importance of modeling the entire time series of events, which TGAT's attention layers do not sufficiently capture. By prepending an RNN making use of a compact event representation, we ensure that BiDyn models the entire time series while being light on memory usage.

Furthermore, alternating training leads to competitive or even improved performance compared to end-to-end training, despite using a GNN layer without learnable parameters. We attribute this improvement to the increased model depth made possible by a stacked model. While JODIE navigates the performance-scalability tradeoff by disconnecting gradient flow through time, BiDyn disconnects gradient flow across the layers of the stacked model, which is a more effective compromise in the transductive setting.

Finally, the performance of TGN, DyRep and JODIE are varied, with the models failing to learn the Wikipedia task (achieving near random performance) but achieving higher performance than BiDyn on the Reddit task. We hypothesize that BiDyn is more successful on Wikipedia due to its success in domains with greater network homophily: the Pearson correlation between the labels of pairs of users who share an item is 3% on Wikipedia, and only 0.5% on Reddit (different with $p < 0.001$). Despite their advantages on the Reddit domain, these models are unable to scale to the larger e-commerce dataset, running out of memory (OOM) when trained on GPU and taking over 300 hours per epoch when trained on CPU, hence cannot be used in a web-scale abuse detection pipeline. Appendix B also demonstrates that BiDyn performs comparably to these baselines on another dynamic graph dataset, MOOC [13].

Graph models. BiDyn outperforms a static GNN (the GNN phase of the Core Model Architecture), hence demonstrating the value of incorporating time information for abuse detection.

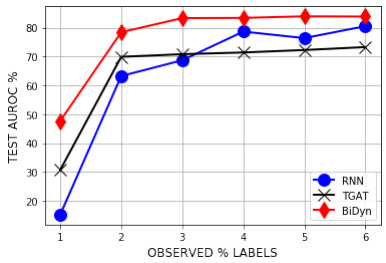


Figure 4: Comparison between BiDyn and baselines (RNN, TGAT) on AUROC by varying the amount of observed node labels on Wikipedia. Even when we observe less then 5% of node labels, BiDyn achieves high AUROC compared to baselines, hence it is robust to rare training labels.

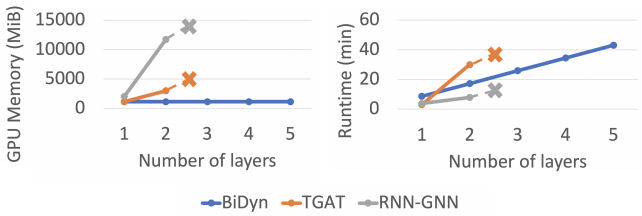


Figure 5: GPU memory usage (left) and runtime (right) for BiDyn, RNN-GNN and TGAT. TGAT and RNN-GNN run out of memory for 3 or more layers (denoted by X), while memory usage of BiDyn remains constant with increasing depth. Furthermore, BiDyn achieves comparable or lower runtime than the baselines.

Recurrent models. BiDyn outperforms the RNN model with time encoding (the RNN phase of the Core Model Architecture), hence leveraging information from nearby nodes is advantageous for dealing with label sparsity.

Pretraining. Pretraining leads to further performance improvement, indicating that incorporating domain knowledge via the random access query objective can help to deal with label sparsity. We select the best-performing variant of BiDyn on each dataset (either coarse, mean or sum) for pretraining.

5.2 Robustness to Sparse Training Data

Figure 4 compares the effectiveness of BiDyn compared to TGAT and RNN at different amounts of supervision, varying the percentage of labeled nodes seen in training. We chose these baselines as they were the two best competitors to BiDyn on the Wikipedia dataset (Table 3). BiDyn outperforms the baselines when the number of training samples seen is very low (< 5%). Moreover, when the number of samples is low, the methods that rely on graph information (BiDyn, TGAT) outperform RNN. For our abuse detection use case, performing well with limited supervised data is crucial.

5.3 Memory and Time Comparison

Figure 5 shows BiDyn uses 10x less memory than the base RNN-GNN model (which already uses memory-saving feature representations) on the e-commerce dataset. These memory savings allow it to fit comfortably on a GPU instance and hence be efficiently trained in a production setting. Note in particular that it does not use extra memory per "layer" (training round), while TGAT and RNN-GNN increase rapidly per layer. Its low memory usage allows for larger receptive fields and less subsampling during minibatching, accounting for its higher performance. Note that TGAT uses less memory than RNN-GNN, but does not perform as well in our transductive problem setting, which prioritizes modeling the whole time series associated with each node. Furthermore, BiDyn achieves comparable or lower runtime than the baselines. When comparing memory and runtime, all models are trained with the same batch size of 64. A 128-dimensional feature is used for user nodes, and no features are used for item nodes. Though RNN-GNN (barely) fits on GPU in 2 layers, the small batch size of 64 (which was used to provide an common ground for comparison) is too small to enable high throughput in a production setting, and the memory footprint is too large to accommodate higher-dimensional node features.

5.4 Architecture Ablation

Table 4 compares different choices for the convolution operation. In "predict item labels", the model must predict the binary label of the item for a separate but related item classification task. In "autoencoder", the layer takes in an RNN sequence of its neighbors' embeddings as input, outputs a bottleneck embedding and uses that embedding as the first hidden state of a decoder RNN whose task is to output the same sequence of neighbor embeddings, with minimum mean-squared error. The coarse-grained convolution operation of equation (7) performs best on the e-commerce dataset. The autoencoder objective performs worst, which could be explained by the high MSE of the learned model, which suggests the task may be too difficult to learn usable representations from.

5.5 Pretraining Validation

Table 5 compares the random access query pretraining task with a standard (static) graph autoencoder objective [10] and finds that the random access query task leads to superior performance on the abuse detection task. Thus, it is important to jointly model the temporal and graph structure to provide useful embeddings for the abuse detection task. Figure 6 confirms that the pretraining task provides a useful prior by creating an initial separation of nodes: we see a separation between abusive and normal nodes after pretraining, which is further accentuated after regular training as the model fine-tunes the embeddings for the abuse detection task.

6 DEPLOYMENT STRATEGIES

BiDyn can be deployed to detect abuse on e-commerce websites, social media, etc. The training method is scalable, and existing GPU infrastructure can be easily used to train and deploy these models into production. Since periodic retraining of graph models as the graph data evolves is essential in practice, efficient GPU training and light memory usage can make it easier to retrain more often for best performance. All training happens offline, and once the abusive nodes are identified, they can be passed to other downstream systems that act on this information, such as banning the users in case of Wikipedia or Reddit.

	e-com
Autoencoder	-13.4
Predict item labels	-3.8
BiDyn (sum)	-7.1
BiDyn (mean)	-3.3
BiDyn (coarse)	**0.0**

	Wikipedia
Autoencoder	79.8 ± 2.8
BiDyn (mean)	80.5 ± 2.3
BiDyn (sum)	**86.5 ± 1.6**

Table 4: Comparison of different objectives and aggregation schemes in the item step (relative change in AUROC). Coarse-grained convolution performs best on e-commerce, while sum aggregation performs best on Wikipedia.

	e-commerce	Wikipedia
No pretraining	0.0	86.5 ± 1.6
Graph autoencoder	-0.5 ± 2.2	86.7 ± 1.9
Random access	**+3.3 ± 0.5**	**87.5 ± 0.6**

Table 5: Comparison of performance on the abuse task after pretraining on different self-supervised tasks. The random access query pretraining task boosts performance on the abuse task compared to no pretraining, while the graph autoencoder task does not improve performance, hence self-supervision on both the temporal and graph structure provides an important advantage on the abuse detection task.

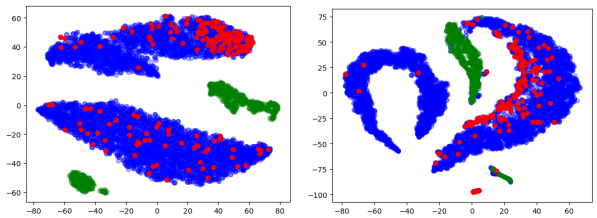


Figure 6: Pretraining provides initial separation between abusive and normal node embeddings on Wikipedia (left), which is further enhanced through fine-tuning on the abuse detection task (right). Here, blue nodes are normal users, red nodes are abusive users and green nodes are items. Visualized with TSNE.

One can also envision a human-in-the-loop scenario, where BiDyn determines a set of users to be banned. These users can be sent to a human auditor who makes the final decision. This process will help improve the yield of human auditors, who can rely on the ML model to send them relevant users to audit, rather than (potentially) random users. Augmenting machine learning models with a human to take the final labeling decision has been explored before [16].

7 CONCLUSION

We have presented BiDyn, a scalable dynamic graph model, training scheme and pretraining framework for detecting abuse on production-scale graphs. BiDyn efficiently combines time and structural information to detect bad actors. Our model achieves state of the art results on both public and proprietary datasets.

Acknowledgements. We thank Eddie Huang for insightful discussions, and the reviewers for their valuable feedback.

REFERENCES

[1] Bijaya Adhikari, Liangyue Li, Nikhil Rao, and Karthik Subbian. 2021. Finding Needles in Heterogeneous Haystacks. In *IAAI*.
[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting Anomalies in Weighted Graphs. In *PAKDD*.
[3] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. MIDAS: Microcluster-Based Detector of Anomalies in Edge Streams. In *AAAI*.
[4] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* (2019).
[5] Justin Cheng, Cristian Danescu-Niculescu-Mizil, and Jure Leskovec. 2015. Antisocial Behavior in Online Discussion Communities. *CoRR* (2015).
[6] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Lio, and Petar Velickovic. 2020. Principal Neighbourhood Aggregation for Graph Nets. *arXiv preprint arXiv:2004.05718* (2020).
[7] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2016. Deep Coevolutionary Network: Embedding User and Item Features for Recommendation. *arXiv preprint arXiv:1609.03675* (2016).
[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.
[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* (1997).
[10] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).
[11] Jon M Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *JACM* (1999).
[12] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Combining Neural Networks with Personalized PageRank for Classification on Graphs. In *ICLR*.
[13] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *KDD*. ACM.
[14] Kingsly Leung and Christopher Leckie. 2005. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*. 333–342.
[15] Caleb C Noble and Diane J Cook. 2003. Graph-Based Anomaly Detection. In *KDD*.
[16] Nikhil Rao, Joseph Harrison, Tyler Karrels, Robert Nowak, and Timothy T Rogers. 2010. Using Machines to Improve Human Saliency Detection. In *ASILOMAR*. IEEE.
[17] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.
[18] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *ICML*.
[19] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnemann, Disha Makhija, Mohit Kumar, and Christos Faloutsos. 2016. Edgecentric: Anomaly Detection in Edge-Attributed Networks. In *ICDMW*.
[20] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.
[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. *NeurIPS* (2017).
[22] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Casual Anonymous Walk. In *ICLR*.
[23] David H Wolpert. 1992. Stacked Generalization. *Neural Networks* (1992).
[24] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.
[25] Liwei Wu, Hsiang-Fu Yu, Nikhil Rao, James Sharpnack, and Cho-Jui Hsieh. 2020. Graph DNA: Deep Neighborhood Aware Graph Encoding for Collaborative Filtering. In *AISTATS*.
[26] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive Representation Learning on Temporal Graphs. In *ICLR*.
[27] 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*.
[28] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and Accurate Anomaly Detection in Dynamic Graphs with a Two-Pronged Approach. In *KDD*.
[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *NeurIPS*.
[30] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. 2019. AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN. *IJCAI*.

A ABUSIVE BEHAVIOR ANALYSIS

Model	Rel. AUROC
LSTM	0.0
Transformer	-5.2
Distribution	-9.5

Table 6: The LSTM model outperforms a Transformer model, and a model using handcrafted features about the distribution of event counts, at predicting whether a user is abusive.

Table 6 shows that an LSTM model trained to predict a node's abuse label outperforms a Transformer model, as well as a model that predict the label based on the mean and standard deviation of the number of events on each day in the training data, on the ecommerce dataset. All models receive as input a sequence of the number of edges that occurred on that day for which the given node was an endpoint, and the objective is to classify the node's binary abuse label. Thus, considering the full time series, particularly the time ordering of events, is crucial to modeling abuse.

B EXTENDED PERFORMANCE COMPARISON

Method	MOOC
RNN	63.1 ± 0.7
TGAT	70.0 ± 0.5
TGN	68.0 ± 0.4
BiDyn (mean)	63.5 ± 0.8
BiDyn (sum)	69.7 ± 1.0
+ pretraining	70.7 ± 1.5

Table 7: Model comparison on MOOC dataset (AUROC). BiDyn achieves performance comparable to that of more resource-heavy dynamic graph baselines, TGAT and TGN, and makes further gains through the self-supervised pretraining objective.

We further demonstrate the effectiveness of BiDyn as a general model for semi-supervised node classification on dynamic bipartite graphs. We evaluate its performance in binary node classification on the MOOC dataset [13], which consists of students in an online course linked to the videos and other course activities participate in. Positive users indicate those who dropped out of the course. There are 7047 users (2463 positive), 98 items and 411749 interactions. We see that BiDyn achieves performance comparable to that achieved by more resource-heavy dynamic graph baselines, TGAT and TGN. Furthermore, the pretraining task leads BiDyn to make further performance gains.

C IMPLEMENTATION DETAILS

Here we describe the hyperparameters and other implementation details of the models tested in the experiments. We manually tuned the hyperparameters for each model. Due to the proprietary nature of the e-commerce dataset, we describe settings on the open Wikipedia and Reddit datasets. We plan to make our code publicly available.

C.1 General

We test all methods for 10 trials and report mean and standard deviation AUROC. For RNN, RNN-GNN and BiDyn, we use a time encoding of dimension $D_{\text{time}} = 32$.

C.2 GNN

The hyperparameters to the GNN are listed in Table 8. The GNN is the GNN phase defined in CoreModel Architecture (using a logistic regression on the last hidden state to predict the abuse label). We observe a slight performance advantage from using separate weight matrices W^l and W^l_{msg} when computing embeddings for user versus item nodes.

Parameter	Value
Hidden dimension	128
Learning rate	10^{-3}
Neighbors sampled per layer	60
Batch size	1000
Epochs	30
Layers	2

Table 8: Hyperparameters for the GNN model.

C.3 RNN

The hyperparameters to the RNN are listed in Table 9. The RNN is the RNN phase defined in Core Model Architecture (using a logistic regression on the last hidden state to predict the abuse label).

Parameter	Value
Hidden dimension	64
Learning rate	10^{-3}
Batch size	256
Epochs	20
Layers	2
Dropout	0.5

Table 9: Hyperparameters for the RNN model.

C.4 RNN-GNN

When testing on e-commerce, we do not use node or edge features in the RNN phase (so the RNN is fed a sequence of the number of events on each timestamp), in order to fit the model in GPU memory; node features are used in the GNN phase. For the other datasets, we use the full RNN (i.e. equation (2), not the coarse-grained version).

The hyperparameters to the RNN-GNN are listed in Table 10.

C.5 Baselines

TGAT. We adopt the TGAT architecture, but adapt the training objective to our transductive task. We compute the embedding of each user node on its last time step and use a logistic regression head on this embedding to predict the binary abuse label, trained via cross-entropy loss. We use the official implementation provided in https://github.com/StatsDLMathsRecomSys/Inductive-

Parameter	Value
Hidden dimension	32
Learning rate	10^{-3}
Batch size	1000
Epochs	20
RNN layers	2
GNN layers	2
Dropout	0

Table 10: Hyperparameters for the RNN-GNN model.

representation-learning-on-temporal-graphs. The hyperparameters to TGAT are listed in Table 11.

Parameter	Value
Layers	2
Neighbors sampled per layer	60
Attention heads	2
Learning rate	10^{-4}
Node dimension	100
Time dimension	100
Batch size	256
Epochs	20
Dropout	0.1

Table 11: Hyperparameters for the TGAT model.

TGN, JODIE, DyRep. We use the implementation for TGN, JODIE and DyRep provided in https://github.com/twitter-research/tgn. We first pretrain all models with their respective self-supervised event prediction tasks. Then, we fine-tune the models on a transductive prediction task. In this task, we append each model's respective decoder head to the temporal node embedding of each user, which predicts the binary abuse label. Finally, at evaluation time, we predict the abuse label of each user by applying the decoder head to the last temporal embedding of that user. We use the default parameters for each model.

C.6 BiDyn

Hyperparameters. The hyperparameters to BiDyn are listed in Table 12. While the model can support much higher number of neighbors sampled per layer, we choose to uniformly randomly subsample 200 neighbors for each node during preprocessing due to diminishing performance gains with higher number of neighbors.

Parameter	Value
Neighbors sampled per layer	200
Learning rate	10^{-3}
Embedding dimension	64
Batch size	256
Epochs	20
Dropout	0.5

Table 12: Hyperparameters for the BiDyn model.

C.7 Pretraining

Link prediction. We use the non-probabilistic graph autoencoder model [10] to learn node embeddings. To create a training batch, we sample a random set of user nodes and a random set of item nodes, and predict all the pairwise edge relationships between them, where the ground truth label is whether than edge actually appears between that user and item. The model parameters are the same as in Table 12.