ROLAND: Graph Learning Framework for Dynamic Graphs

Jiaxuan You
jiaxuan@cs.stanford.edu
Stanford University
California, USA

Tianyu Du
tianyudu@stanford.edu
Stanford University
California, USA

Jure Leskovec
jure@cs.stanford.edu
Stanford University
California, USA

ABSTRACT

Graph Neural Networks (GNNs) have been successfully applied to many real-world static graphs. However, the success of static graphs has not fully translated to dynamic graphs due to the limitations in model design, evaluation settings, and training strategies. Concretely, existing dynamic GNNs do not incorporate state-of-the-art designs from static GNNs, which limits their performance. Current evaluation settings for dynamic GNNs do not fully reflect the evolving nature of dynamic graphs. Finally, commonly used training methods for dynamic GNNs are not scalable. Here we propose ROLAND, an effective graph representation learning framework for real-world dynamic graphs. At its core, the ROLAND framework can help researchers easily repurpose any static GNN to dynamic graphs. Our insight is to view the node embeddings at different GNN layers as hierarchical node states and then recurrently update them over time. We then introduce a live-update evaluation setting for dynamic graphs that mimics real-world use cases, where GNNs are making predictions and being updated on a rolling basis. Finally, we propose a scalable and efficient training approach for dynamic GNNs via incremental training and meta-learning. We conduct experiments over eight different dynamic graph datasets on future link prediction tasks. Models built using the ROLAND framework achieve on average 62.7% relative mean reciprocal rank (MRR) improvement over state-of-the-art baselines under the standard evaluation settings on three datasets. We find state-of-the-art baselines experience out-of-memory errors for larger datasets, while ROLAND can easily scale to dynamic graphs with 56 million edges. After re-implementing these baselines using the ROLAND training strategy, ROLAND models still achieve on average 15.5% relative MRR improvement over the baselines.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Information systems → Information systems applications**.

KEYWORDS

Graph Neural Networks; Dynamic Graphs; Network Analysis

ACM Reference Format:
Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: Graph Learning Framework for Dynamic Graphs . In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539300

1 INTRODUCTION AND RELATED WORK

The problem of learning from dynamic networks arises in many application domains, such as fraud detection [13, 23], anti-money laundering [40], and recommender systems [44]. Graph Neural Networks (GNNs) are a general class of models that can perform various learning tasks on graphs. GNNs have gained tremendous success in learning from static graphs [1, 9, 42, 45, 49]. Although various GNNs have been proposed for dynamic graphs [2, 22, 24, 30, 31, 34, 36, 39, 47, 50] these approaches have limitations of *model design*, *evaluation settings* and *training strategies*. Overcoming these limitations is crucial for real-world dynamic graph applications.

Limitations of model design. Existing approaches fail to transfer successful static GNN designs/architectures to dynamic graph applications. Many existing works treat a GNN as a feature encoder and then build a sequence model on top of the GNN [31, 39, 47]. Other works take Recurrent Neural Networks (RNNs), then replace the linear layers in the RNN cells with graph convolution layers [25, 34, 50]. Although these approaches are intuitive, they do not incorporate state-of-the-art designs from static GNNs. For instance, the incorporation of skip-connections [6, 10, 20], batch normalization [6, 11, 46], edge embedding [7, 43] has been beneficial for GNN message passing, but has not been explored for dynamic GNNs. To avoid re-exploring these design choices for dynamic GNNs, instead of building dynamic GNNs from scratch, a better design principle would be to start from a mature static GNN design and adapt it for dynamic graphs.

Limitations of evaluation settings. When evaluating dynamic GNNs, existing literature usually ignores the evolving nature of data and models. Concretely, dynamic graph datasets are often deterministically split by time, *e.g.*, if ten months of data are available, the first eight months of data will be used as training, one month as validation, and the last month as the test set [30, 50]. Such protocol evaluates the model only on edges from the last month of the available dataset; therefore, it tends to overestimate the model performance given the presence of long-term pattern changes. Additionally, in most literature, a non-updated model is used for prediction within the time span at evaluation time, which means the model gets stale over time [24, 39, 50]. However, in real applications, users can update their models based on new data and then make future predictions.

Limitations of training strategies. The common training methods for dynamic GNNs can be improved in terms of scalability and efficiency [34, 35, 50]. First, most existing training methods for dynamic GNNs usually require keeping the entire or a large portion of the graph in GPU memory [34, 35, 50], since all or a large

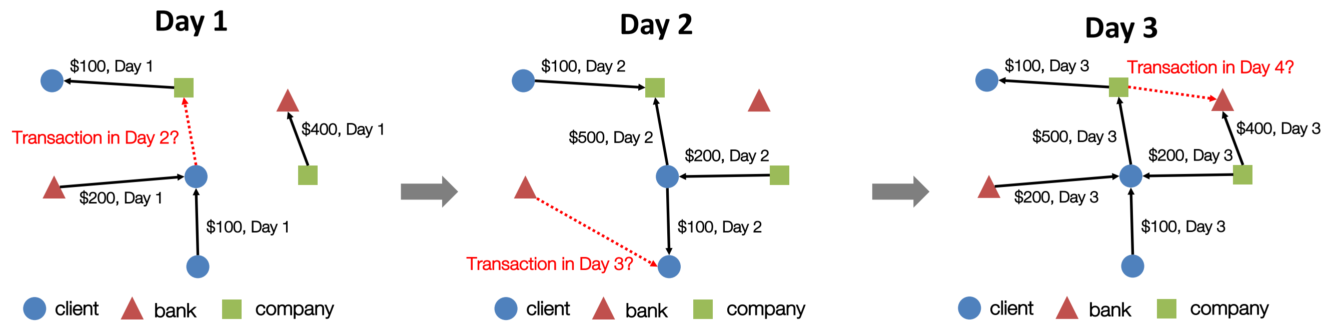

Figure 1: Example ROLAND use case for future link prediction on a dynamic transaction graph. We use information up to time t to predict potential edges at time $t + 1$.

portion of historical edges are used for message passing. Therefore, dynamic GNNs are often evaluated on small networks with only a few hundred nodes [5, 47] or small transaction graphs with fewer than 2 million edges [30, 35]. Moreover, there is little research on making dynamic GNNs generalize and quickly adapt to new data.

Present work. Here we propose ROLAND, an effective graph representation learning framework for real-world dynamic graphs. We focus on the snapshot-based representation of dynamic graphs where nodes and edges are arriving in batches (*e.g.*, daily, weekly). ROLAND framework can help researchers re-purpose any static GNN to a dynamic graph learning task; consequently, we can adapt state-of-the-art designs from static GNNs and significantly lower the barrier to learning from dynamic graphs.

ROLAND model. Our insight is that any GNN for static graphs can be extended for dynamic graph use cases. We offer a new viewpoint for static GNNs, where the node embeddings at different GNN layers are viewed as *hierarchical node states*. To generalize a static GNN to a dynamic setting, we only need to define how to update these hierarchical nodes states based on newly observed nodes and edges. We explore a variety of update-modules, including moving average, Multi-layer Perceptron (MLP), and Gated Recurrent Unit (GRU) [3]. This way of building dynamic GNNs is simple and effective, and more importantly, this approach can keep the design of a given static GNN and use it on a dynamic graph.

ROLAND evaluation. We then introduce a live-update evaluation setting for dynamic graphs, where GNNs are used to make predictions on a rolling basis (*e.g.*, daily, weekly). When making each prediction, we make sure the model has only been trained using historical data so that there is no information leakage from the future to the past. We additionally allow the model to be fine-tuned using the new graph snapshot, mimicking real-world use cases, in which the model is constantly updated to fit evolving data.

ROLAND training. Finally, we propose a scalable and efficient training approach for dynamic GNNs. We propose an incremental training strategy that does a truncated version of back-propagation-through-time (BPTT) [41], which significantly saves GPU memory cost. Concretely, we only keep the incoming new graph snapshot and historical node states in GPU memory. Using this technique, we are able to train GNNs on a dynamic transaction network with 56 million edges, which is about 13 times larger than existing benchmarks in terms of edges per snapshot. Furthermore, we formulate prediction tasks over dynamic graphs as a *meta-learning problem*, where we treat making predictions in different periods (*e.g.*, every day) as different tasks arriving sequentially. We find a meta-model, which serves as a good initialization that is used to derive a specialized model for future unseen prediction tasks quickly.

Key results. We conduct experiments over eight different dynamic graph datasets, with up to 56 million edges and 733 graph snapshots. We first compare ROLAND to six baseline models under the evaluation setting in the existing literature on three datasets, ROLAND achieves 62.7% performance gain over the best baseline on average, demonstrating the effectiveness of ROLAND design framework. We then evaluate our models and baselines on the proposed live-update setting, which provides a more realistic evaluation setting for dynamic graphs. We find that existing dynamic GNN training methods based on BPTT fail to scale to large datasets. To get meaningful comparisons, we re-implement baselines using ROLAND training, which greatly reduces GPU memory cost. ROLAND still achieves on average 15.5% performance gain over the ROLAND version of baselines.

Contributions. We summarize ROLAND's innovation as follows.

(1) *Model design*: ROLAND describes how to repurpose a static GNN for dynamic settings effectively. Furthermore, ROLAND shows that successful static GNN designs lead to significant performance gain on dynamic prediction tasks.
(2) *Training*: ROLAND can scale to dynamic graphs with 56 million edges, which is at least 13 times larger than existing benchmarks. In addition, we innovatively formulate predicting over dynamic graphs as a meta-learning problem to achieve fast model adaptation.
(3) *Evaluation*: Compared to the common deterministic dataset split, ROLAND's live-update evaluation can reflect the evolving nature of data and model.

2 PRELIMINARIES

Graphs and dynamic graphs. A graph can be represented as $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the node set and $E \subseteq V \times V$ is the edge multi-set. Nodes can be paired with features $X = \{\mathbf{x}_v \mid v \in V\}$, and edges can also have features $F = \{\mathbf{f}_{uv} \mid (u, v) \in E\}$. For dynamic graphs, each node v further has a timestamp τ_v and edge e has a timestamp τ_e. We focus on the snapshot-based representation for dynamic graphs: a dynamic graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$ can be represented as a sequence of graph snapshots, where each snapshot is a static

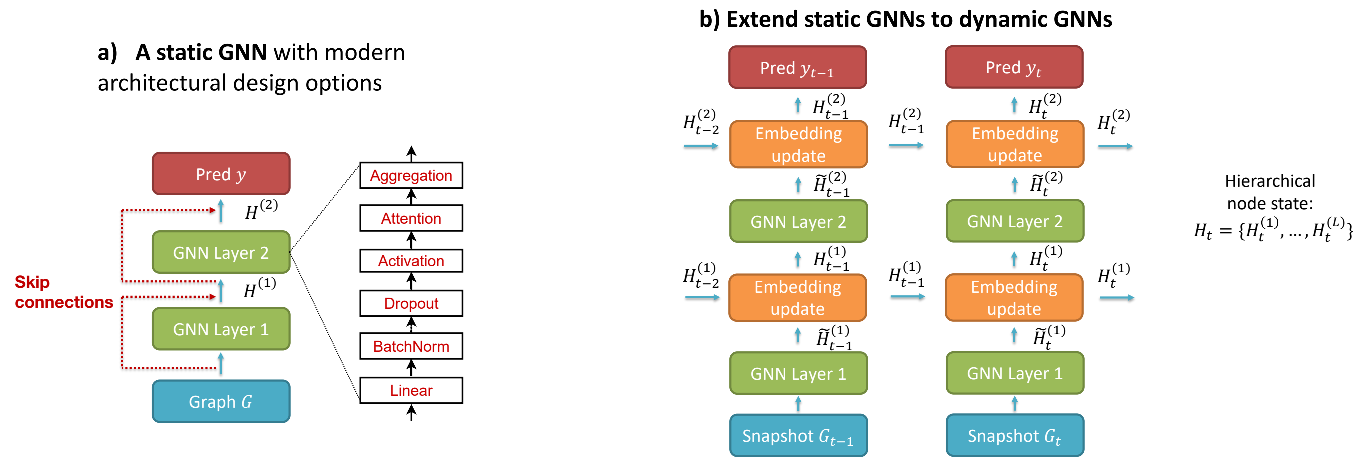

Figure 2: ROLAND model design principle. (a) Example static GNN with modern architecture designs, including intra-layer designs such as BatchNorm and attention, and inter-layer designs such as skip-connections. (b) ROLAND can easily extend any static GNN to dynamic graphs, by inserting embedding update modules that update hierarchical node states H_t over time.

graph $G_t = (V_t, E_t)$ with $V_t = \{v \in V \mid \tau_v = t\}$ and $E_t = \{e \in E \mid \tau_e = t\}$. By modeling over graph snapshots, ROLAND can naturally handle node addition/deletion since different graph snapshots could have different sets of nodes.

Graph Neural Networks. The goal of a GNN is to learn node embeddings based on an iterative aggregation of messages from the local network neighborhood. We use embedding matrix $H^{(L)} = \{\mathbf{h}_v^{(L)}\}_{v \in V}$ to denote the embedding for all the nodes after applying an L-layer GNN. The l-th layer of a GNN, $H^{(l)} = \textsc{Gnn}^{(l)}(H^{(l-1)})$, can be written as:

$$\begin{aligned} \mathbf{m}_{u \to v}^{(l)} &= \textsc{Msg}^{(l)}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}) \\ \mathbf{h}_v^{(l)} &= \textsc{Agg}^{(l)}\left(\{\mathbf{m}_{u \to v}^{(l)} \mid u \in \mathcal{N}(v)\}, \mathbf{h}_v^{(l-1)}\right) \end{aligned} \tag{1}$$

where $\mathbf{h}_v^{(l)}$ is the node embedding for $v \in V$ after passing through l layers , $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $\mathbf{m}_v^{(l)}$ is the message embedding, and $\mathcal{N}(v)$ is the direct neighbors of v. Different GNNs can have have various definitions of message-passing functions $\textsc{Msg}^{(l)}(\cdot)$ and aggregation functions $\textsc{Agg}^{(l)}(\cdot)$. We refer to GNNs defined in Equation (1) as static GNNs, since they are designed to learn from static graphs and do not capture the dynamic information of the graph.

3 PROPOSED ROLAND FRAMEWORK

3.1 From Static GNNs to Dynamic GNNs

Here we propose the ROLAND framework that generalizes static GNNs to a dynamic setting, illustrated in Figure 2. We summarize the forward computation of the ROLAND model in Algorithm 1.

Revisit static GNNs. A static GNN often consists of a stack of L GNN layers, where each layer follows the general formulation in Equation (1). We denote the node embedding matrix after the l-th GNN layer as $H^{(l)} = \{\mathbf{h}_v^{(l)}\}_{v \in V}$. The node embeddings computed in all layers characterize a given node in a hierarchical way; concretely, $H^{(l)}$ summarizes the information from the neighboring nodes that are l hops away from a given node. We use $H = \{H^{(1)}, ..., H^{(L)}\}$ to represent the embeddings in all GNN layers.

Algorithm 1 ROLAND GNN forward computation

Input: Dynamic graph snapshot G_t, hierarchical node state H_{t-1}
Output: Prediction y_t, updated node state H_t

```
1: $H_t^{(0)} \leftarrow X_t$                                   {Initialize embedding from $G_t$}
2: for $l = 1, \ldots, L$ do
3:     $\tilde{H}_t^{(l)} = \textsc{Gnn}^{(l)}(H_t^{(l-1)})$            {Implemented as Equation (4)}
4:     $H_t^{(l)} = \textsc{Update}^{(l)}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)})$                    {Equation (2)}
5: $y_t = \textsc{Mlp}(\textsc{Concat}(\mathbf{h}_{u,t}^{(L)}, \mathbf{h}_{v,t}^{(L)})), \forall (u,v) \in E$        {Equation (5)}
```

From static embeddings to dynamic states. We propose to extend the semantics of H from static embeddings to dynamic node states. Building on previous discussions, we view H_t as the *hierarchical node state* at time t, where each $H^{(l)}$ captures multi-hop node neighbor information. Suppose the input graph G has dynamically changed, then the computed embedding H will also change. Therefore, we argue that the key to generalizing any static GNN to a dynamic graph relies on how to update the hierarchical node states H over time.

Differences with prior works. In the common approach where a sequence model is built on top of a GNN, only the top-level node state $H^{(L)}$ is being kept and updated [31, 39, 47]. All the lower-level node states $H_t^{(1)}, ..., H_t^{(l)}$ are always recomputed from scratch based on the new graph snapshot G_t. Here we propose to keep and update the entire hierarchical node state $H_t^{(1)}, ..., H_t^{(l)}$ at all levels.

Hierarchically update modules in dynamic GNNs. In ROLAND, we propose an *update-module* that updates node embeddings hierarchically and dynamically. The update-module can be inserted to any static GNN. The new level l node state $H_t^{(l)}$ depends on both lower layer node state $\tilde{H}_t^{(l)}$ and historical node state $H_{t-1}^{(l)}$.

$$H_t^{(l)} = \textsc{Update}^{(l)}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)}), \quad \tilde{H}_t^{(l)} = \textsc{Gnn}^{(l)}(H_t^{(l-1)}) \tag{2}$$

We examine three simple yet effective embedding update methods. (1) *Moving Average*: embedding of node v at time t is updated by

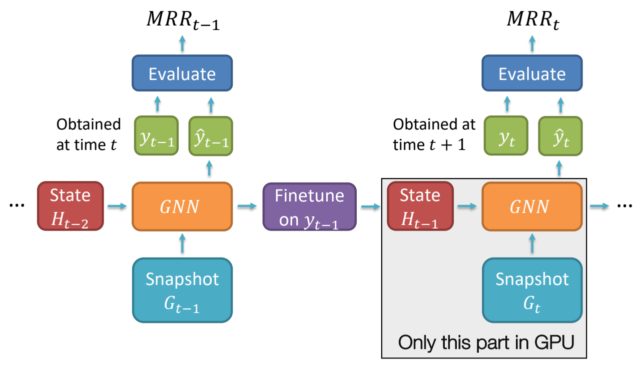

Figure 3: ROLAND live-update evaluation. ROLAND fine-tunes GNN with y_{t-1} and updates node embeddings H_{t-1} for the next prediction task. After obtaining labels y_t, ROLAND evaluates GNN's predictive performance based on historical state H_{t-1} and current snapshot G_t.

$H_{t,v}^{(l)} = \kappa_{t,v} H_{t-1,v}^{(l)} + (1-\kappa_{t,v})\tilde{H}_{t,v}^{(l)}$. Moving average can naturally capture dynamics of embedding and is free from trainable parameters. We define the moving average weight $\kappa_{t,v}$ as

$$\kappa_{t,v} = \frac{\sum_{\tau=1}^{t-1} |E_\tau|}{\sum_{\tau=1}^{t-1} |E_\tau| + |E_t|} \in [0,1] \tag{3}$$

(2) *MLP*: node embeddings are updated by a 2-layer MLP, $H_t^{(l)} = \textsc{Mlp}(\textsc{Concat}(H_{t-1}^{(l)}, \tilde{H}_t^{(l-1)}))$. (3) *GRU*: node embeddings are updated by a GRU cell [3], $H_t^{(l)} = \textsc{Gru}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)})$, where $H_{t-1}^{(l)}$ is the hidden state and $\tilde{H}_t^{(l)}$ is the input for the GRU cell.

GNN architecture design in ROLAND. In ROLAND, we extend the GNN layer definition in Equation (1) to incorporate successful GNN designs in static GNNs. Specifically, we have

$$\begin{aligned} \mathbf{m}_{u\to v}^{(l)} &= \mathbf{W}^{(l)} \textsc{Concat}\big(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{f}_{uv}\big), \\ \mathbf{h}_v^{(l)} &= \textsc{Agg}^{(l)}\big(\{\mathbf{m}_{u\to v}^{(l)} \mid u \in \mathcal{N}(v)\}\big) + \mathbf{h}_v^{(l-1)} \end{aligned} \tag{4}$$

Here, we consider edge features $\mathbf{f}_{uv}$ in message computation $\textsc{Msg}^{(l)}(\cdot)$, since dynamic graphs usually have timestamps as edge features; additionally, we include $\mathbf{h}_v^{(l-1)}$ in the message passing computation, so that bidirectional information flow can be modeled; we explore different aggregation functions Agg including summation Sum, maximum Max, and average Mean; finally, we add skip-connections [10, 20] when stacking GNN layers by adding $\mathbf{h}_v^{(l-1)}$ after message aggregation. We show in Section 4.4 that these designs lead to performance boost. Based on the computed node embeddings, ROLAND predicts the probability of a future edge from node u to v via an MLP prediction head

$$y_t = \textsc{Mlp}(\textsc{Concat}(\mathbf{h}_{u,t}^{(L)}, \mathbf{h}_{v,t}^{(L)})), \forall (u,v) \in V \times V \tag{5}$$

3.2 Live-update Evaluation

Standard fixed split evaluation setting. The evaluation procedure used in prior works constructs training and testing sets by splitting available graph snapshots sequentially. For example, the first 90% of snapshots for training and cross-validation, then models are evaluated using all edges from the last 10% of snapshots [30]. However, the data distribution of dynamic graphs is constantly evolving in the real world. For example, the number of purchase transactions made in the holiday season is higher than usual. Therefore, evaluating models solely based on edges from the last 10% of snapshots can provide misleading results.

Live-update evaluation setting. To utilize all snapshots to evaluate the model, we propose a live-update evaluation procedure (Figure 3 and Algorithm 2), which consists of the following key steps: (1) Fine-tune GNN model using newly observed data (2) evaluate predictions using new data, (3) making predictions using historical information, Specifically, ROLAND first collects link prediction labels y_{t-1} at time t (predict future links), which are split into $y_{t-1}^{(train)}$ and $y_{t-1}^{(val)}$. Then, ROLAND fine-tunes Gnn on training labels $y_{t-1}^{(train)}$, while performance $\text{MRR}_{t-1}^{(val)}$ on validation labels $y_{t-1}^{(val)}$ are used as the early stopping criterion. We use Mean reciprocal rank (MRR) instead of ROC AUC, since negative labels significantly outnumber positive labels in a link prediction task, and MMR has been adopted in prior work as well [30]. Finally, after obtaining edge labels y_t at time $t+1$, we report ROLAND's performance MRR_t based on historical state H_{t-1} and current snapshot G_t. This live-update evaluation is free from information leakage, since no future information is used during both training and evaluation. We report the average MRR_t over all prediction steps as the final model performance.

Algorithm 2 ROLAND live-update evaluation

Input: Dynamic graph $\mathcal{G} = \{G_1, \dots, G_T\}$, link prediction labels $y_1, \dots, y_T$, number of snapshots T, $\textsc{Gnn}(\cdot)$ defined in Algorithm 1
Output: Performance MRR, model Gnn

```
1:  Initialize hierarchical node state $H_0$
2:  for $t = 2, \dots, T$ do
3:      Collect link prediction labels $y_{t-1} = y_{t-1}^{(train)} \cup y_{t-1}^{(val)}, y_t$
4:      while $\text{MRR}_{t-1}^{(val)}$ is increasing do
5:          $H_{t-1}, \hat{y}_{t-1} \leftarrow \textsc{Gnn}(G_{t-1}, H_{t-2}), \hat{y}_{t-1} = \hat{y}_{t-1}^{(train)} \cup \hat{y}_{t-1}^{(val)}$
6:          Update Gnn via backprop based on $\hat{y}_{t-1}^{(train)}, y_{t-1}^{(train)}$
7:          $\text{MRR}_{t-1}^{(val)} \leftarrow \textsc{Evaluate}(\hat{y}_{t-1}^{(val)}, y_{t-1}^{(val)})$
8:      $H_t, \hat{y}_t \leftarrow \textsc{Gnn}(G_t, H_{t-1})$
9:      $\text{MRR}_t \leftarrow \textsc{Evaluate}(\hat{y}_t, y_t)$
10: $\text{MRR} = \sum_{t=2}^{T} \text{MRR}_t/(T-1)$
```

3.3 Training Strategies

Scalability: Incremental training. We propose a scalable and efficient training approach for dynamic GNNs that borrows the idea of the truncated version of back-propagation-through-time, which has been widely used for training RNNs [41]. Concretely, we only keep the model Gnn, the incoming new graph snapshot G_t, and historical node states H_{t-1} into GPU memory. Since the historical node states H_{t-1} has already encoded information up to time $t-1$, instead of training Gnn to predict y_t from the whole sequence $\{G_1, \dots, G_{t-1}\}$, the model only needs to learn from $\{H_{t-1}$, and the newly observed graph snapshot $G_t\}$. With this incremental training strategy, ROLAND's memory complexity is agnostic to the

Table 1: Summary of dataset statistics.

	# Edges	# Nodes	Range	Snapshot Frequency	# Snapshots
BSI-ZK	56,194,191	1,744,561	Jan 01, 2008 - Dec 30, 2008	daily	257
AS-733	11,965,533	7,716	Nov 8, 1997 - Jan 2, 2000	daily	733
Reddit-Title	571,927	54,075	Dec 31, 2013 - Apr 30, 2017	weekly	178
Reddit-Body	286,561	35,776	Dec 31, 2013 - Apr 30, 2017	weekly	178
BSI-SVT	190,133	89,564	Jan 27, 2008 - Dec 30, 2008	weekly	49
UCI-Message	59,835	1,899	Apr 15, 2004 - Oct 26, 2004	weekly	29
Bitcoin-OTC	35,592	5,881	Nov 8, 2010 - Jan 24, 2016	weekly	279
Bitcoin-Alpha	24,186	3,783	Nov 7, 2010 - Jan 21, 2016	weekly	274

Algorithm 3 ROLAND training algorithm

Input: Graph snapshot G_t, link prediction label y_t, hierarchical node state H_{t-1}, smoothing factor α, meta-model $\textsc{Gnn}^{(meta)}$

Output: Model $\textsc{Gnn}$, updated meta-model $\textsc{Gnn}^{(meta)}$

```
1: $\textsc{Gnn} \leftarrow \textsc{Gnn}^{(meta)}$
2: Move $\textsc{Gnn}, G_t, H_{t-1}$ to GPU
3: while $\text{MRR}_t^{(val)}$ is increasing do
4:     $H_t, \hat{y}_t \leftarrow \textsc{Gnn}(G_t, H_{t-1})$, $\hat{y}_t = \hat{y}_t^{(train)} \cup \hat{y}_t^{(val)}$
5:     Update $\textsc{Gnn}$ via backprop based on $\hat{y}_t^{(train)}, y_t^{(train)}$
6:     $\text{MRR}_t^{(val)} \leftarrow \textsc{Evaluate}(\hat{y}_t^{(val)}, y_t^{(val)})$
7: Remove $\textsc{Gnn}, G_t, H_{t-1}$ from GPU
8: $\textsc{Gnn}^{(meta)} \leftarrow (1-\alpha)\textsc{Gnn}^{(meta)} + \alpha\textsc{Gnn}$
```

number of graph snapshots, and we can train GNNs on dynamic graphs with 56 million edges and 733 graph snapshots.

Fast adaptation: Meta-training. We formulate prediction tasks over dynamic graphs as a meta-learning problem, where we treat making predictions in different periods (*e.g.*, every day) as different tasks. The standard approach of updating $\textsc{Gnn}$ is simply fine-tuning the model using new information. However, always keeping the prior model for the next future prediction task is not necessarily optimal. For example, for dynamic graphs with weekly patterns, the optimal model on Friday may be drastically different from the optimal model on Saturday; therefore, simply fine-tuning the previous model could lead to inferior models. Instead, our proposal is to find a *meta-model* $\text{GNN}^{(meta)}$, which serves as a good initialization that is used to derive a specialized model for future unseen prediction tasks quickly. In principle, any meta-learning algorithm can be used to update $\text{GNN}^{(meta)}$; we follow the Reptile algorithm [28] which is simple and effective. As is shown in Algorithm 3, at each time t, we first initialize the GNN model using $\text{GNN}^{(meta)}$ and then use back-propagation with early stopping to fine-tune the model for the next prediction task. Then, we updates the meta-model $\text{GNN}^{(meta)}$ by computing a moving average of the trained models $(1-\alpha)\textsc{Gnn}^{(meta)} + \alpha\textsc{Gnn}$, where $\alpha \in [0, 1]$ is the smoothing factor.

4 EXPERIMENTS

4.1 Experimental Setup

Datasets. We perform experiments using eight different datasets. **(1)** BSI-ZK and **(2)** BSI-SVT consist of financial transactions among companies using two different systems of the Bank of Slovenia, and each node has five categorical features; each edge has transaction amount as the feature. **(3)** Bitcoin-OTC and Bitcoin-Alpha contain who-trusts-whom networks of people who trade on the OTC and Alpha platforms [16, 17]. **(4)** The Autonomous systems AS-733 dataset of traffic flows among routers comprising the Internet [19]. **(5)** UCI-Message dataset consists of private messages sent on an online social network system among students [29]. **(6)** Reddit-Title and **(7)** Reddit-Body are networks of hyperlinks in titles and bodies of Reddit posts, respectively. Each hyperlink represents a directed edge between two subreddits [15]. Table 1 provides summary statistics of the above-mentioned datasets.

Task. We evaluate the ROLAND framework over the future link prediction task. At each time t, the model utilizes information accumulated up to time t to predict edges in snapshot $t+1$. We use mean reciprocal rank (MRR) to evaluate candidate models. For each node u with positive edge (u, v) at $t+1$, we randomly sample 1000[1] negative edges emitting from u and identify the rank of edge (u, v)'s prediction score among all other negative edges. MRR score is the mean of reciprocal ranks over all nodes u. We consider two different train-test split methods in this paper. (1) *Fixed-split* evaluates models using all edges from the last 10% of snapshots. However, the transaction pattern is constantly evolving in the real world; evaluating models solely based on edges from the last few weeks provides misleading results. (2) *Live-update* evaluates model performance over all the available snapshots. We randomly choose 10% of edges in each snapshot to determine the early-stopping condition.

Baselines. We compare our ROLAND models to 6 state-of-the-art dynamic GNNs. **(1)** EvolveGCN-H and **(2)** EvolveGCN-O utilizes an RNN to dynamically update weights of internal GNNs, which allows the GNN model to change during the test time [30]. **(3)** T-GCN internalizes a GNN into the GRU cell by replacing linear transformations in GRU with graph convolution operators [50]. **(4)** GCRN-GRU and **(5)** GCRN-LSTM generalize TGCN by capturing temporal information using either GRU or LSTM. Instead of GCNs, GCRN uses ChebNet [4] for spatial information. Besides, GCRN uses separate GNNs to compute different gates of RNNs. Hence GCRN models have much more parameters compared with other models [34]. **(6)** GCRN-Baseline GCRN baseline firstly constructs node features using a Chebyshev spectral graph convolution layer to capture spatial information; afterward, node features are fed into an LSTM cell for temporal information [34].

[1]We only sampled 100 negative edges in the BSI-ZK dataset due to memory constraints.

Table 2: Results in the standard fixed split setting. Performance of the baselines are reported in the EvolveGCN paper [30]. Our experiments ensure the same snapshot frequency, set of test snapshots and method to compute MRRs, etc., for fair comparisons. We run each experiment with 3 random seeds and report the average performance together with the standard error.

	Bitcoin-OTC	Bitcoin-Alpha	UCI-Message
GCN [14]	0.0025	0.0031	0.1141
DynGEM [12]	0.0921	0.1287	0.1055
dyngraph2vecAE [8]	0.0916	0.1478	0.0540
dyngraph2vecAERNN [8]	**0.1268**	**0.1945**	0.0713
EvolveGCN-H [30]	0.0690	0.1104	0.0899
EvolveGCN-O [30]	0.0968	0.1185	**0.1379**
ROLAND Moving Average	0.0468 ± 0.0022	0.1399 ± 0.0107	0.0649 ± 0.0049
ROLAND MLP	0.0778 ± 0.0024	0.1561 ± 0.0114	0.0875 ± 0.0110
ROLAND GRU	**0.2203 ± 0.0167**	**0.2885 ± 0.0123**	**0.2289 ± 0.0618**
Improvement over best baseline	73.74%	43.33%	65.99%

ROLAND architecture. ROLAND is designed to re-purpose any static GNN into a dynamic one. Thus the hyper-parameter space of the ROLAND model is similar to the hyper-parameter space of the underlying static GNN. We use 128 hidden dimensions for node states, GNN layers with skip-connection, sum aggregation, and batch-normalization. We allow for at most 100 epochs for each live-update in each time step before early stopping. In addition, for each type of update-module and dataset, we search the hyper-parameters over (1) numbers of pre-processing, message-passing, and post-processing layers (1 layer to 5 layers); (2) learning rate for live-update (0.001 to 0.01); (3) whether to use single or bidirectional messages; and (4) the α level for meta-learning to determine the best configuration. We report the test set MRR when the best validation MRR is achieved. We use NVIDIA RTX 8000 GPU in the experiments. We implement ROLAND[1] with the GraphGym library [46].

4.2 Results in the Standard Evaluation Settings

We firstly compare our models with baselines under the standard fixed split setting. In these experiments, we follow the experimental setting in the EvolveGCN paper [30]; specifically, we use the same snapshot frequency, set of test snapshots, and the method to compute MRR to ensure fair comparisons. Table 2 shows that ROLAND with GRU update-module consistently outperforms baseline models and achieves on average 62.69% performance gain. This agrees with the fact that the GRU update-module is more expressive than simple moving average or MLP update-modules. The significant performance gain demonstrates that ROLAND can effectively transfer successful static GNN designs to dynamic graph tasks. In Section 4.4, we provide comprehensive ablation studies to explain the success of ROLAND further.

4.3 Results in the Live-update Settings

Baselines with BPTT training vs. ROLAND. By default, the baseline models are trained with back-propagation-through-time (BPTT), which requires storing all the historical node embeddings in GPU memory. This training strategy cannot scale to large graphs. The top part of Table 3 summarizes the experiments on training the baseline models using BPTT. Our experiment results indicate that the default BPTT training fails to scale to large datasets, such as BSI-ZK and AS-733, and fails to work with large models such as GCRNs. For datasets with a moderate number of edges and snapshots like BSI-SVT, BPTT still leads to OOM for large models like GCRNs and TGCN. Even though this training pipeline works on small datasets, the best model trained from this procedure still underperforms compared to baseline models trained using ROLAND incremental training by almost 10%.

Baselines with ROLAND training vs. ROLAND. To quantitatively compare our models with baselines, we re-implemented baseline models and adapted them to be trained using our ROLAND training. The middle and bottom parts of Table 3 contrast the performance of our models and baselines on all eight datasets. All baseline models are successfully trained using our ROLAND training, demonstrating the flexibility of ROLAND Moreover, ROLAND with GRU update-module outperforms the best baseline model by 15.48% on average. Except for the AS-733 dataset, our model consistently outperforms baseline models trained using the ROLAND pipeline. The performance gain of using our models ranges from 2.40% on BSI-ZK to 44.22% on the Reddit-Body dataset. The performance gain is the largest on datasets with rich edge features such as Reddit hyperlink graphs, on which ROLAND brings 38.21% and 44.22% performance gains, respectively. These results demonstrate that ROLAND can utilize edge features effectively. We include more analysis of the results in the Appendix.

4.4 Ablation Study

Effectiveness of ROLAND architecture design. As described in Section 2.2, we introduce several successful GNN designs in static GNNs to ROLAND, so that they can work with dynamic graphs. Here, we conduct experiments to show that the ROLAND architecture design is effective in boosting performance.

Concretely, we examine the ROLAND framework on the BSI-SVT dataset under the proposed live-update setting. The setting is similar to Table 3 in the main paper. We found that Batch Normalization, skip-connection, and max aggregation are desirable for GNN architectural design, leading to a significant performance boost. For example, the introduction of skip-connection leads to more than 20% performance gain.

[1]The source code of ROLAND is available at https://github.com/snap-stanford/roland

Table 3: Results in the live-update settings. We run experiments (except for BSI-ZK) with 3 random seeds to report the average and standard deviation of MRRs. We attempted each experiment five times before concluding the out-of-memory (OOM) error. The top part of the results consists of baseline models trained using BPTT, the middle and bottom portions summarize the performance of baselines, and our models using ROLAND incremental training.

	BSI-ZK	AS-733	Reddit-Title	Reddit-Body	BSI-SVT	UCI-Message	Bitcoin-OTC	Bitcoin-Alpha
				Baseline Models with standard training				
EvolveGCN-H	N/A, OOM	N/A, OOM	N/A, OOM	**0.148 ± 0.013**	**0.031 ± 0.016**	0.061 ± 0.040	0.067 ± 0.035	0.079 ± 0.032
EvolveGCN-O	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.015 ± 0.006	0.071 ± 0.009	0.085 ± 0.022	0.071 ± 0.025
GCRN-GRU	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.080 ± 0.012	N/A, OOM	N/A, OOM
GCRN-LSTM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	**0.083 ± 0.001**	N/A, OOM	N/A, OOM
GCRN-Baseline	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.069 ± 0.004	**0.152 ± 0.011**	**0.141 ± 0.005**
TGCN	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.054 ± 0.024	0.128 ± 0.049	0.088 ± 0.038
				Baseline Models with ROLAND Training				
EvolveGCN-H	N/A, OOM	0.251 ± 0.079	0.165 ± 0.026	0.102 ± 0.010	0.032 ± 0.008	0.057 ± 0.012	0.076 ± 0.022	0.054 ± 0.015
EvolveGCN-O	0.396	0.163 ± 0.002	0.047 ± 0.004	0.033 ± 0.001	0.018 ± 0.003	0.066 ± 0.012	0.032 ± 0.004	0.034 ± 0.002
GCRN-GRU	N/A, OOM	**0.344 ± 0.001**	0.338 ± 0.006	0.217 ± 0.004	0.050 ± 0.004	0.089 ± 0.004	0.173 ± 0.003	0.140 ± 0.004
GCRN-LSTM	N/A, OOM	0.341 ± 0.001	0.344 ± 0.005	0.216 ± 0.000	0.051 ± 0.002	0.091 ± 0.010	0.174 ± 0.004	**0.146 ± 0.005**
GCRN-Baseline	0.754	0.336 ± 0.002	0.351 ± 0.001	0.218 ± 0.002	0.054 ± 0.002	**0.095 ± 0.008**	**0.183 ± 0.002**	0.145 ± 0.003
TGCN	**0.831**	0.343 ± 0.002	**0.391 ± 0.004**	**0.251 ± 0.001**	**0.157 ± 0.004**	0.080 ± 0.015	0.083 ± 0.011	0.069 ± 0.013
				ROLAND results				
Moving Average	0.819	0.309 ± 0.011	0.362 ± 0.007	0.289 ± 0.038	0.177 ± 0.006	0.075 ± 0.006	0.120 ± 0.002	0.0962 ± 0.010
MLP-Update	0.834	0.329 ± 0.021	0.395 ± 0.006	0.291 ± 0.008	**0.217 ± 0.003**	0.103 ± 0.010	0.154 ± 0.010	0.148 ± 0.012
GRU-Update	**0.851**	**0.340** ± 0.001	**0.425** ± 0.015	**0.362** ± 0.002	0.205 ± 0.014	**0.112** ± 0.008	**0.194** ± 0.004	**0.157** ± 0.007
Improvement over the best baseline	2.40%	-1.16%	8.70%	44.22%	38.21%	17.89%	6.01%	7.53%

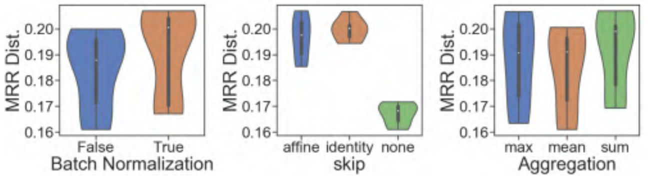

Figure 4: Effectiveness of ROLAND architectural design. We run experiments with different GNN models and analyze the effect of differ design dimensions. We show the MRR distribution of all the models under different design options. Results show that batch normalization, skip-connection and max aggregation are desirable for GNN architectural design.

Effectiveness of meta-learning. We examine the effectiveness of meta-learning. Specifically, for each of three embedding update methods and each dataset, we run 10 experiments with $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$ to study the effect of α on performance[1]. Here, $\alpha = 1$ corresponds to the baseline that directly uses the previous model to initialize the training of Gnn. Table 4 reports the performance gains in MRR from meta-learning. Enabling meta-learning leads to various levels of performance gain depending on the dataset and model; for different embedding update designs, the average performance gain ranges from 2.84% to 13.19%. Besides the performance gain in MRRs, we find incorporating meta-learning, in general, improves the stability of performance, as suggested by a lower standard deviation of MRRs. We include the complete results in Appendix.

[1]For BSI-ZK dataset, we run 6 experiments for each update method with $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$.

Effectiveness of model retraining. We investigate the effectiveness of model retraining in Figure 5. As is shown in Figure 5(top), the transaction pattern, measured by the number of transactions, significantly varies over time; in the meantime, ROLAND can automatically retrain itself to fit the data (Algorithm 2). Thanks to the meta-training, ROLAND only requires periodic retraining for a few epochs, and in most snapshots, as is shown in Figure 5(middle). We further compare our strategy with a baseline that stops retraining after training the first 25% of snapshots. We found that this baseline performs significantly worse than retraining for all the snapshots.

5 ADDITIONAL RELATED WORK

Dynamic GNNs. While many works combine GNNs with recurrent models (*e.g.*, GRU cells), we want to emphasize that our ROLAND framework is quite different. Most existing works take a recurrent model (RNN, Transformer) as the backbone architecture, then (1) replace either the feature encoder [31] with a GNN, or (2) replace the linear layers in the RNN cells with GNN layers [25, 34, 50]. The first line of approaches ignores the evolution of lower-level node embeddings, while the second approach only utilizes a given GNN layer rather than building upon an established and successful static GNN architecture. EvolveGCN [30] proposes to recurrently updates the GNN weights; in contrast, ROLAND recurrently updates hierarchical node embeddings. While the EvolveGCN approach is memory efficient, the explicit historical information is lost (e.g., past transaction information). Our experiments show that EvolveGCN

Table 4: Ablation study on the effectiveness of meta-learning. Except for the BSI-ZK dataset, the results are averaged over 3 random seeds. "Gain" refers to the MRR improvement from the best meta-learning setting over the non-meta-learning setting.

	BSI-ZK		AS-733		Reddit-Title		Reddit-Body		BSI-SVT		UCI-Message		Bitcoin-OTC		Bitcoin-Alpha		Average
Model	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	
Moving Average	0.5	0.99%	0.4	4.94%	0.7	2.68%	0.5	8.52%	0.5	3.27%	0.7	23.73%	0.9	7.45%	0.6	6.94%	7.33%
MLP-Update	0.5	4.04%	0.9	18.05%	0.7	1.09%	0.4	2.51%	0.4	11.40%	0.2	27.73%	0.3	33.76%	0.6	6.98%	13.19%
GRU-Update	0.7	0.56%	0.5	5.15%	0.1	2.97%	0.5	8.18%	0.8	2.10%	0.5	0.17%	0.9	2.82%	0.8	0.77%	2.84%

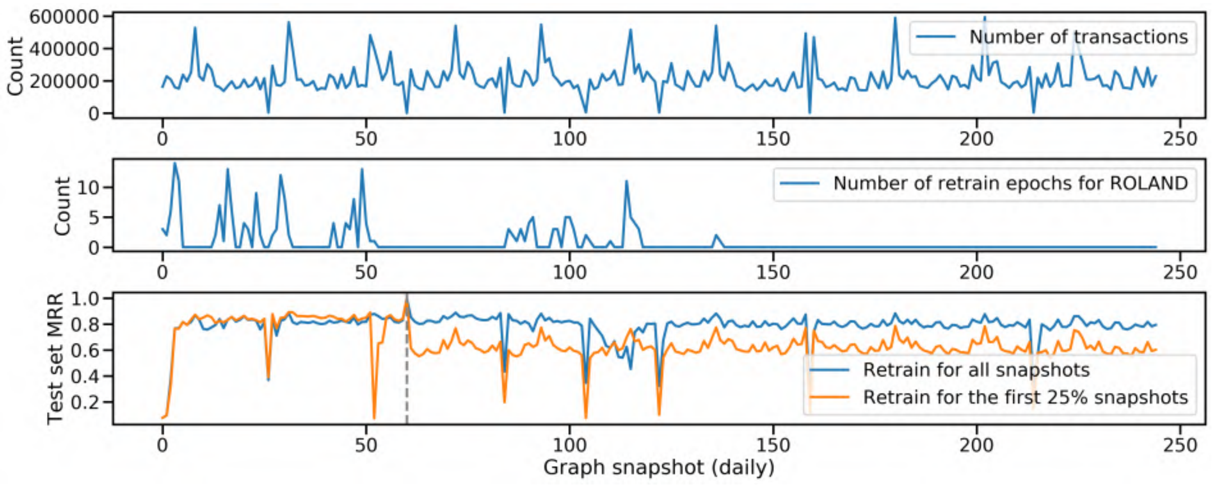

Figure 5: Effectiveness of ROLAND model retraining on BSI-ZK dataset. Top: number of transactions in each daily graph snapshot. Middle: number of training epochs before early stopping. Bottom: ROLAND's performance on the test set. We compare the option of retraining for all snapshots (blue curve), with a baseline that stopping retraining after training the first 25% snapshots (organ curve).

delivers undesirable performance when the number of graph snapshots is large. Moreover, few existing dynamic GNN works have explored state-of-the-art static GNN designs, such as the inclusion of edge features, batch normalization, skip-connections, etc. By utilizing mature static GNN designs, we show that ROLAND can significantly outperform existing dynamic GNNs.

Scalable training. Most existing approaches do not consider scalable training since the current benchmark dynamic graphs are small; thus, the entire graph can often fit into the GPU memory. Additionally, some dynamic GNN architectures can hardly scale; for example, when using Transformer as the base sequence model requires keeping the entire historical graph in the GPU to compute the attention over time [33]. Some existing approaches have used heuristics to scale training. For example, they would limit the number of historical graph snapshots used to predict the current snapshot [50]. In contrast, our proposed incremental training keeps the entire historical information in the node hidden states while only choosing to back-propagate within the given snapshot.

Non-snapshot dynamic graph representation. Instead of using snapshot-based representation, one can also represent a dynamic graph as a single graph where nodes and edges have time stamps [18, 26, 32]. As is described in Section 2, we can easily convert that representation to graph snapshots by grouping all the nodes/edges within a given period into a graph snapshot, and then apply our ROLAND framework.

Other learning methods for dynamic graphs. Besides GNNs, researchers have explored other learning methods for dynamic graphs, such as matrix factorization based methods [21], random walk based models [27, 48], point process based approaches [37, 38, 51]. We focus on designing GNN-based methods in this paper due to their performance and inductive learning capabilities.

ACKNOWLEDGEMENTS

We thank Rok Sosic, Daniel Borrajo, Vamsi Potluru, Naren Chittar, Pranjal Patil, Hao Yang, Yuhang Wu, Das Mahashweta for their helpful discussions on the paper. We also gratefully acknowledge the support of Stanford Data Science Initiative and JPMorgan Chase.

6 CONCLUSION

We propose ROLAND, an effective graph representation learning system for building, training, and evaluating dynamic GNNs. ROLAND helps researchers re-purpose any static GNN for a dynamic graph while keeping effective designs from static GNNs. ROLAND comes with a live-update pipeline that mimics real-world usage by allowing the model to be dynamically updated while being evaluated. Our experiments show that dynamic GNNs built using ROLAND framework successfully scale to large datasets and outperform existing state-of-the-art models. We hope ROLAND can enable more large-scale real-world applications over dynamic graphs.

REFERENCES

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] J. Chen, X. Xu, Y. Wu, and H. Zheng. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. In *arXiv preprint arXiv:1812.04206*, 2018.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[5] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[6] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

[8] P. Goyal, S. R. Chhetri, and A. Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. In *Knowledge-Based Systems*, 2020.

[9] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[12] N. Kamra, P. Goyal, X. He, and Y. Liu. Dyngem: deep embedding method for dynamic graphs. In *IJCAI International Workshop on Representation Learning for Graphs (ReLiG)*, 2017.

[13] A. Khazane, J. Rider, M. Serpe, A. Gogoglou, K. Hines, C. B. Bruss, and R. Serpe. Deeptrax: Embedding graphs of financial transactions. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019.

[14] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

[15] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[16] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 333–341. ACM, 2018.

[17] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 221–230. IEEE, 2016.

[18] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[19] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[20] G. Li, M. Müller, G. Qian, I. C. D. Perez, A. Abualshour, A. K. Thabet, and B. Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[21] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017.

[22] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan. Predicting path failure in time-evolving graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1279–1289, 2019.

[23] X. Li, J. Saúde, P. Reddy, and M. Veloso. Classifying and understanding financial data using graph neural network. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[24] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *arXiv preprint arXiv:1707.01926*, 2017.

[25] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR)*, 2018.

[26] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[27] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Continuous-time dynamic network embeddings. In *The Web Conference (WWW)*, 2018.

[28] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[29] P. Panzarasa, T. Opsahl, and K. M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

[30] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[31] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, et al. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. In *Information Sciences*, volume 521, pages 277–290. Elsevier, 2020.

[32] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[33] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2020.

[34] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, 2018.

[35] S. Sharma and R. Sharma. Forecasting transactional amount in bitcoin network using temporal gnn approach. In *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2020.

[36] A. Taheri, K. Gimpel, and T. Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion Proceedings of The 2019 World Wide Web Conference*, 2019.

[37] R. Trivedi, H. Dai, Y. Wang, and L. Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *International Conference on Machine Learning (ICML)*, 2017.

[38] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*, 2018.

[39] X. Wang, Y. Ma, Y. Wang, W. Jin, X. Wang, J. Tang, C. Jia, and J. Yu. Traffic flow prediction via spatial temporal graph neural network. In *The Web Conference (WWW)*, 2020.

[40] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. In *arXiv preprint arXiv:1908.02591*, 2019.

[41] R. J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[42] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec. Identity-aware graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[43] J. You, X. Ma, D. Ding, M. Kochenderfer, and J. Leskovec. Handling missing data with graph representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[44] J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, and J. Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The Web Conference (WWW)*, 2019.

[45] J. You, R. Ying, and J. Leskovec. Position-aware graph neural networks. *International Conference on Machine Learning (ICML)*, 2019.

[46] J. You, R. Ying, and J. Leskovec. Design space for graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[47] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[48] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

[49] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[50] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li. T-gcn: A temporal graph convolutional network for traffic prediction. In *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[51] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.