Group Project Report

ROLAND: Graph Learning Framework for Dynamic Graphs [1]

Group 16

QI Shihao	24052653G
YANG hao	24123933G
CHEN Yize	24133368G

1 Introduction

1.1 Research Problem

Graph Neural Networks (GNN) are becoming a powerful tool for modeling complex relationships in data from a variety of domains, including social networks, biological systems, and recommendation engines. However, traditional GNN models often face limitations when applied to dynamic graph problems and large-scale datasets. The ROLAND framework solved these challenges and propelled the advancement of graph neural networks. It can accomplish the following two tasks:
1. Embedding a static GNN model to enable it to handle dynamic graph problems.
2. Large-scale training data can be used to train models.

1.2 Motivation

The ROLAND framework represents a significant advancement in the field of GNN. It aims to enhance the versatility and extensibility of GNN through two main features.

First, it supports the embedding of static GNN models to efficiently handle dynamic graph problems. Dynamic graphs, characterized by their time-evolving nature, pose unique challenges that are difficult for static models to address. ROLAND's innovative approach enables these models to adapt to changes in graph structure and node characteristics, thus improving their applicability in real-world scenarios with evolving data.

Second, ROLAND facilitates the efficient training of models using large-scale data. Dealing with large amounts of data, traditional GNNs often run into scalability issues, leading to increased computational demands and performance degradation. ROLAND addresses these challenges by incorporating mechanisms to optimize the training process, enabling the processing of large amounts of data without compromising speed or accuracy.

Besides, the ROLAND framework can live-update evaluation settings, which can evaluate the performance of the model more accurately.

1.3 Challenges

While Graph Neural Networks (GNNs) have achieved remarkable success in a variety of applications, their adaptation to dynamic graph environments still faces some significant challenges. These challenges highlight the limitations of current approaches and emphasize the need for innovative solutions such as the ROLAND framework.

One of the main challenges in adapting static GNN models to dynamic maps is the limitations of the model design itself. Existing methods often struggle to incorporate advanced techniques such as edge embedding into the conversion process from static to dynamic GNNs. This limitation requires the development of dynamic GNNs from scratch, which is resource-intensive and time-consuming, and hinders the progress and efficiency of dynamic map processing research.

Another key issue is that the current evaluation setup is not sufficient to accurately assess model performance. Many studies rely on using only the last batch of data as validation data, which can lead to overestimation of model validity. This approach fails to account for the continuous and evolving nature of dynamic plots, in which data change over time. As a result, the evaluation metrics may not truly reflect the model's ability to handle dynamic environments, leading to misleading conclusions about its performance.

Training strategies for GNNs also face significant challenges, especially when dealing with large-scale data. The traditional approach involves loading most of the training data into GPU memory, which is only applicable to smaller datasets. This strategy is impractical for large-scale data, as it exceeds memory limits and hinders the training process. Therefore, there is an urgent need for training strategies that can accommodate large amounts of data without compromising computational efficiency or model accuracy.

2 Process

2.1 Dataset

In this report, we used two public datasets for our analysis. The first one is called Autonomous Systems (AS-733). It shows how the Internet is organized into smaller groups called Autonomous Systems (AS), where each group exchanges data with its neighbors. This network of connections is built from BGP (Border Gateway Protocol) logs. The data comes from the University of Oregon Route Views Project and includes 733 daily records over 785 days, from November 8, 1997, to January 2, 2000.[2]

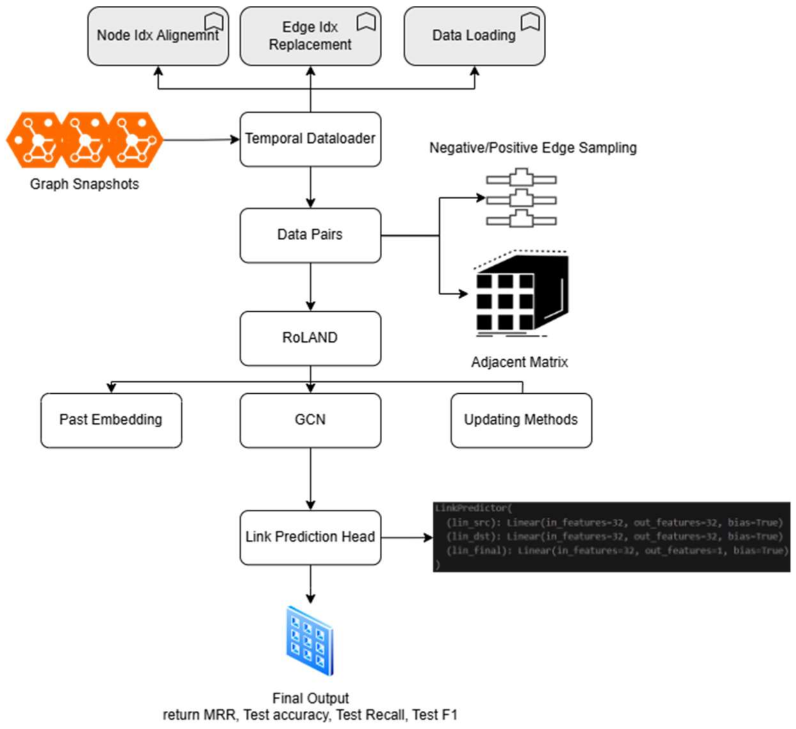
The second dataset is the Bitcoin Alpha trust network (Bitcoin-Alpha). It maps out who trusts whom among people trading Bitcoin on the Bitcoin Alpha platform. Since Bitcoin users are anonymous, it's important to track reputations to avoid dealing with untrustworthy users. In this network, users rate each other from -10 (meaning total distrust) to +10 (meaning total trust). This dataset is the first of its kind to show trust relationships in a detailed way.[3]

2.2 Model

The model generally aligned with the paper, in the shape of using $H_{t-1}^{(l)}$ to update the current embedding. However, building a dynamic moved graph isn't easy, although RoLAND already provides clear guidance on model updating, an entirely architecture still require a lot of auxiliary function to assist node loading.

$$H_t^{(l)} = MLP(CONCAT(H_{t-1}^{(l)}, \tilde{H}_t^l))$$

Below is attached with an entire graph map of model structure, from data loading till the last model output result.

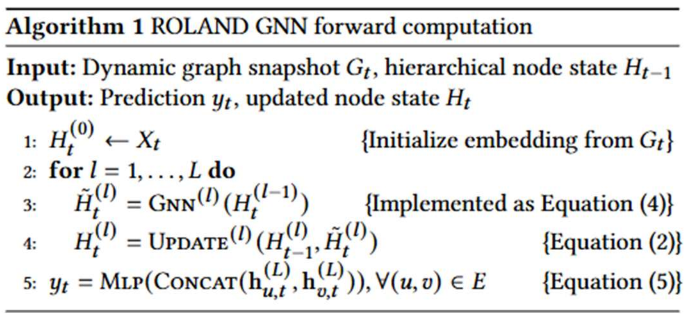


```
LinkPredictor(
  (lin_src): Linear(in_features=32, out_features=32, bias=True)
  (lin_dst): Linear(in_features=32, out_features=32, bias=True)
  (lin_final): Linear(in_features=32, out_features=1, bias=True)
)
```

Model Structure

The model reproduction is strictly followed by 3 updating methods between former $H_{t-1}^{(l)}$ and current $H_t^{(l)}$ embedding.

$Moving\ Average$	$H_t^{(l)} = k_{t,v} H_{t-1,v}^{(l)} + (1 - k_{t,v})\tilde{H}_t^l$ $where\ k_{t,v} = \frac{\sum_{\tau=1}^{t-1} \lvert E_\tau \rvert}{\sum_{\tau=1}^{t-1} \lvert E_\tau \rvert + \lvert E_t \rvert} \in [0,1]$
MLP	$H_t^{(l)} = MLP(CONCAT(H_{t-1}^{(l)}, \tilde{H}_t^l))$
GRU	$H_t^{(l)} = GRU(H_{t-1}^{(l)}, \tilde{H}_t^l)$



```
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)}
```

Figure 1 – RoLAND Updating Method

Truncate for Alignment – However, during the model, there is no guarantee that between each snapshot the size of the nodes will be the same; thus, during snapshot updating, we use a simple method to align different moment embeddings:

1. Initialize an all zero torch Tensor in shape of $H_t^{(l)}$
2. Based on shape of $H_t^{(l)}$ to truncate past embedding size, and padding the empty space with 0, if there is necessary.

By using such a method, the model can flexibly fit into various kinds of embedding shapes regardless of mismatch issues. After testing, we found that this operation won't affect the model's behavior (comparison result in session 3).

GNN Add-drop Strategy – There is another problem to focus on: according to the inherent features of GNNs and their inner propagation schemes, deep-layer GNNs can lead to overfitting and over-propagation if the model isn't refreshed between different snapshots. To address this issue, the paper presents a meta GNN that will be reloaded and refreshed each time before a new snapshot starts:

$$\boldsymbol{GNN}^{(\boldsymbol{meta})} = (1-\alpha)\boldsymbol{GNN}^{(\boldsymbol{meta})} + \alpha\boldsymbol{GNN}^{(\boldsymbol{meta})}$$

However, after investigating the provided source code of RoLAND, the team discovered that such a meta GNN can be quickly refreshed using the torch.xavier_normal_ function. It is believed that the source code also includes this option to reload the GNN.

Our Model

Considering that RoLAND is proposed as an architecture, the team has tested various models with RoLAND, including **GraphSAGE**, **GCN**, and **GAT**. However, as mentioned before, the default 2-layer GCN can already provide sufficient feature collection. Moreover, models like GAT concatenate neighbor node structures and features, which can easily lead to gradient explosion. This problem was first observed with GraphSAGE, leading to severe gradient explosion and overfitting on the training dataset.

RoLAND(GraphSage) on AS-733

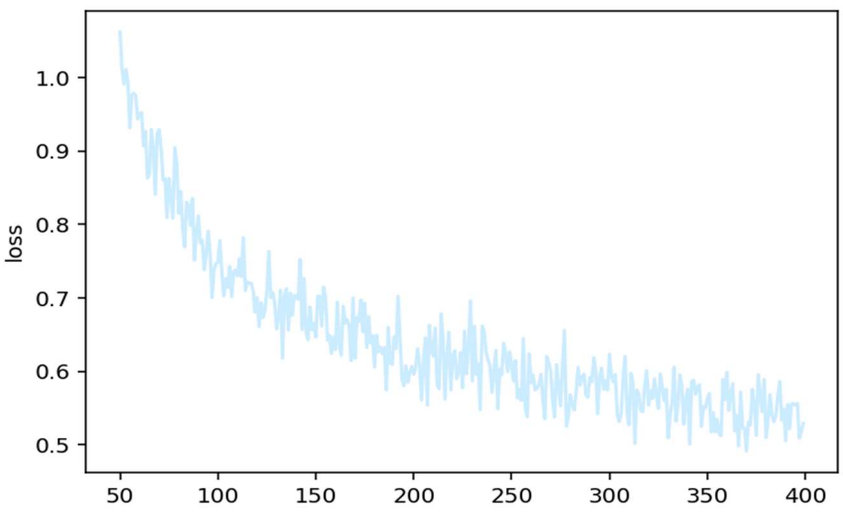


Figure 2 RoLAND + normalized GraphSage

Thus, despite the team providing lots of experiments on inserting models to seek improvements, none of them were effective. Inspired by the recurrent memory structure of RoLAND, the team tried to build an **inner connection structure**.

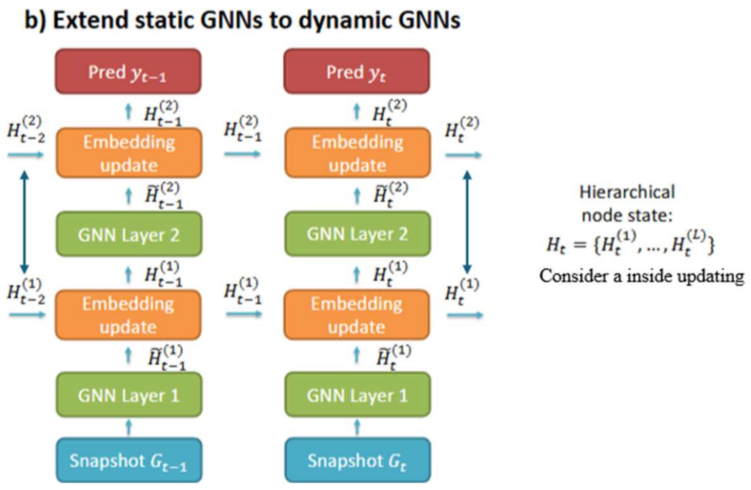


Figure 3 - Naïve Inside Embedding Connection

However, after testing, the team observed that too frequent internal combinations could lead to gradient explosion. Therefore, the team gradually increased the interval between epochs to update the intra-GNN. Ultimately, the team found that updating the GRU every 40 epochs could lead to the best performance; however, this improvement did not have a significant impact on performance compared to the original RoLAND model.

Besides, the approach of increasing update intervals may hinder effective message passing, crucial for GNNs. In consideration of the propagation equation,

$$H^{l+1} = \sigma(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^{l}W)$$

which emphasizes the importance of frequent updates to capture local node features. By extending intervals, the model risks losing critical neighborhood information, leading to suboptimal learning and performance stagnation, ultimately undermining the GNN's ability to adaptively learn from graph structures.

2.3 Experimental settings

Output: Predict future links in dynamic graphs. At each time *t*, the model uses the information before *t+1* to predict edges in snapshot *t+1*. Then, for each node, randomly sample 1000 negative edges emitting from it, and calculate the rank of edges among all negative edges.

train-test split methods:

The authors used two different types of methods to split training and test data. The first approach, which is commonly used in other studies, has limitations and can overestimate the performance of the model. The second method is the dynamic evaluation method proposed by the author.

1. Fixed-split: Only all sides in the last 10% of the snapshots are used to evaluate the performance of the model.
2. Live-update: Evaluate model performance on all available snapshots by randomly selecting 10% of edges at each time snapshot *t*.

Baselines:

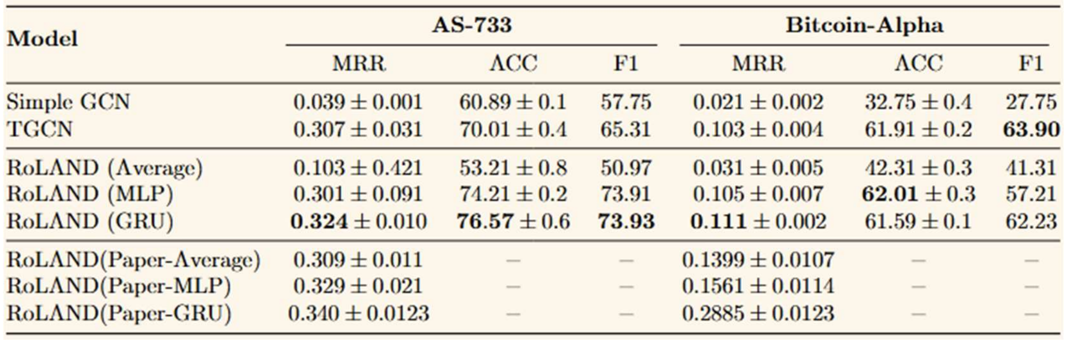
The authors compared their model to six dynamic GNN models, but here the team selected 2 models, which are GCN and T-GCN.

Other settings:

1. 128 hidden dimensions for node states, GNN layers with skip-connection, sum aggregation, and batch-normalization.
2. at most 100 epochs for each live-update in each time step.
3. hyperparameters:
    1) the number of preprocessing, messaging, and post-processing layers in the model.
    2) learning rate.
    3) use single or bidirectional messages.
    4) α level for meta-learning.

3 Results

We used the live-update method proposed by ROLAND authors to segment the training set and the test set, conducted experiments on AS-733 and Bitcoin-Alpha data sets, and calculated the mean reciprocal rank (MRR), accuracy (ACC) and F1 score (F1). Details are shown in Table 1.



Model	**AS-733**			**Bitcoin-Alpha**		
	MRR	ACC	F1	MRR	ACC	F1
Simple GCN	0.039 ± 0.001	60.89 ± 0.1	57.75	0.021 ± 0.002	32.75 ± 0.4	27.75
TGCN	0.307 ± 0.031	70.01 ± 0.4	65.31	0.103 ± 0.004	61.91 ± 0.2	**63.90**
RoLAND (Average)	0.103 ± 0.421	53.21 ± 0.8	50.97	0.031 ± 0.005	42.31 ± 0.3	41.31
RoLAND (MLP)	0.301 ± 0.091	74.21 ± 0.2	73.91	0.105 ± 0.007	$\mathbf{62.01} \pm 0.3$	57.21
RoLAND (GRU)	$\mathbf{0.324} \pm 0.010$	$\mathbf{76.57} \pm 0.6$	**73.93**	$\mathbf{0.111} \pm 0.002$	61.59 ± 0.1	62.23
RoLAND(Paper-Average)	0.309 ± 0.011	–	–	0.1399 ± 0.0107	–	–
RoLAND(Paper-MLP)	0.329 ± 0.021	–	–	0.1561 ± 0.0114	–	–
RoLAND(Paper-GRU)	0.340 ± 0.0123	–	–	0.2885 ± 0.0123	–	–

Table 1: Five different model results and the results of the original paper

Our experimental results show that on both datasets, the best MRR can be obtained when the GRU is embedded into the ROLAND framework. However, compared with the original ROLAND paper, the MRR of the average-updating method has decreased significantly. The reason we will discuss later.

4 Discussions

In the field of dynamic graph neural networks, there are many other approaches to solving the problems ROLAND faces.

Training strategies: FU et al. used a new framework consisting of a load balancing scheduler on two-level GPU and operator overload pipeline to increase the training efficiency of dynamic graphs.[4] Zhou et al. used a new distributed architecture to train a time-based GNN model, improving the efficiency of training the model on large-scale datasets.[5]

Model design: Zhu et al. introduced a sliding time window to calculate the gradient between each time step t, and then aggregated the gradient of the window and the current snapshot t to update the parameters.[6] Wang et al. used a GAN model to generate fake graph information, and then replayed these fake samples using a GNN model to avoid losing historical information.[7]

However, there is something observed by team

1. Average updating method downgrade a lot compares that with RoLAND paper.
2. Models are easy to encounter with gradient exploration and overfitting.

For the first question, the team suspects that it's caused by different $k_{t,v}$ updating methods. After 2 round experiments a roughly conclusion is, the result of $k_{t,v}$ is better at 0.5. The low performance is caused by edge size difference in dataset and imbalanced edge distribution.

For second problem, as mentioned above, it's caused by over-deep GCN layer, and it can be solved by applying normalization and dropout to solve.

5 Conclusions

In this report, the team reproduced the RoLAND paper and analyzed its concepts. RoLAND presents a robust framework for dynamic graph representation learning, enabling the effective adaptation of static GNNs to dynamic environments. Its innovative live-update pipeline and scalable training methods significantly enhance performance. The team reproduced the code and compared the performance with that in the paper. At the same time, the team also observed several interesting cases regarding model behavior and found that normalization is useful for maintaining model stability and robustness [8].

References

[1] You, J., Du, T., & Leskovec, J. (2022, August). ROLAND: graph learning framework for dynamic graphs. In Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining (pp. 2358-2366).
[2] Leskovec, J., Kleinberg, J., & Faloutsos, C. (2005, August). 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 (pp. 177-187).
[3] Kumar, S., Spezzano, F., Subrahmanian, V. S., & Faloutsos, C. (2016, December). Edge weight prediction in weighted signed networks. In 2016 IEEE 16th international conference on data mining (ICDM) (pp. 221-230). IEEE.
[4] Fu, K., Chen, Q., Yang, Y., Shi, J., Li, C., & Guo, M. (2023, November). BLAD: Adaptive Load Balanced Scheduling and Operator Overlap Pipeline For Accelerating The Dynamic GNN Training. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-13).
[5] Zhou, H., Zheng, D., Song, X., Karypis, G., & Prasanna, V. (2023, November). Disttgl: Distributed memory-based temporal graph neural network training. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-12).
[6] Zhu, Y., Cong, F., Zhang, D., Gong, W., Lin, Q., Feng, W., ... & Tang, J. (2023, August). Wingnn: Dynamic graph neural networks with random gradient aggregation window. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 3650-3662).
[7] Wang, J., Zhu, W., Song, G., & Wang, L. (2022, August). Streaming graph neural networks with generative replay. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 1878-1888).
[8] Y. Luo, L. Shi, and X.-M. Wu, "Classic GNNs are Strong Baselines: Reassessing GNNs for Node Classification," 2024, doi: 10.48550/arxiv.2406.08993