Autoencoder-based Node Embedding Regeneration and Prediction in Dynamic Graphs

Mohamed Darghouthi
Department of Informatics and Mathematics
University of Passau
Passau, Germany
dargho02@ads.uni-passau.de

Hakim Ghazzai
CEMSE, King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
hakim.ghazzai@kaust.edu.sa

Lokman Sboui
Systems Engineering Department
École de Technologie Supérieure (ÉTS),
University of Québec, Montréal, Canada
lokman.sboui@etsmtl.ca

Yehia Massoud
CEMSE, King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
yehia.massoud@kaust.edu.sa

***Abstract*—In this paper, we introduce a neural network model capable of generating embeddings for graph nodes using just their initial features and without any prior knowledge about their neighbors. To this end, we start with generating embeddings from a model trained on the entire graph knowledge (node features, node neighbors, and random walks). We then build a deep autoencoder model that learns to map node features to embeddings similar to those generated by the model trained on the entire graph knowledge. By using a tuned loss function that acts as a strong regularization for the autoencoder, our model is designed to generate embeddings with minimum error to those generated from the first model. It learns how to generate embeddings that reflect the positioning of the nodes in the graph even though it only uses nodes' initial features. We evaluate the model on the amazon and Alibaba graph datasets by computing the mean squared error, cosine similarity, and mean average percentage error between the embeddings generated by our model and the ones we used as reference. We also evaluate the ability to reconstruct the initial graph by repredicting the neighbors of each node using cosine similarity between nodes generated embedding and its neighbors' reference embeddings. Results have shown an efficiency higher than 80% in graph generation and neighborhood detection.**

***Index Terms*—Representation learning, graph node embeddings, Autoencoders, link prediction**

I. Introduction

Machine and deep learning models have more than proved their effectiveness in learning patterns from data with Euclidean underlying structures and generating data analysis insights. However, most of the real world applications generate non-Euclidean data. As practical examples, we can cite the social interactions in social networks [1], molecular graphs used for modeling chemical structures in drug discovery [2], cellular networks, [3] or recommendation systems graphs [4] to name a few. In these applications, data is usually modeled as graphs with vertices connected using weighted or non-weighted edges. Unlike Euclidean data, it is hard to extract features and patterns from graphs to design for typical machine learning models. Hence, to cope with this issue, many deterministic/heuristic techniques have been proposed to manipulate information represented by graphs [5], [6]. However, limited performance was achieved due to the manual feature engineering which is not only time consuming but also domain-specific.

Nowadays, research attention is drawn towards novel techniques that autonomously learn the non-Euclidean underlying structure of data, precisely Graph Neural Networks (GNNs) [7], [8]. GNNs are capable of learning patterns and extracting better intermediate representations. These models can autonomously find low dimensional embeddings for nodes and/or edges that efficiently represent the graph of interest and encode much of the information about these nodes with regards to the graph as a whole. These fine-tuned embeddings can be then exploited by other machine learning algorithms for different tasks such as node classification, link prediction, and optimization [9]–[14]. However, GNNs are not quite accurate when generating embeddings especially for newly introduced nodes in dynamic graphs since information about appearing nodes as well as their connections in the graph are not known yet [9].

This paper tackles the problem of representation learning for new nodes in dynamic graphs, nodes for which we only have knowledge about their initial features. The goal is to generate an embedding for this new node without knowing its exact connections in the graph. As a first step, a GNN model is trained on the entire graph knowledge including node features, node neighbors, and multiple random walks from the graph, to generate embeddings for the different existing

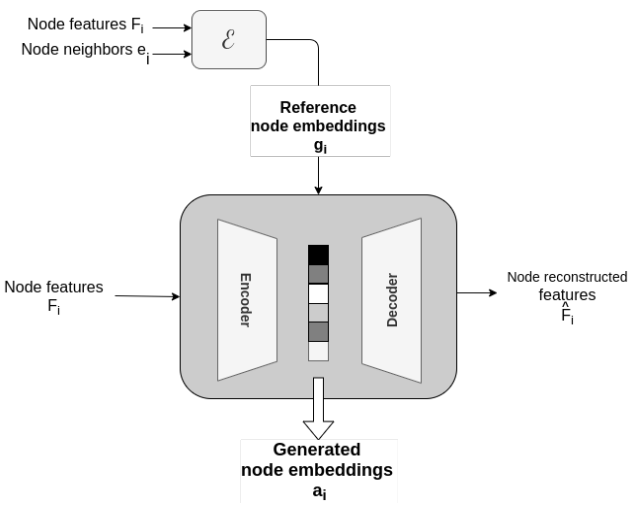

Fig. 1: The proposed mechanism to generate new node embedding in dynamic graphs.

nodes. Afterward, using the acquired knowledge, we design an autoencoder model that treats each node separately and in isolation to generate representative embeddings that are similar to those generated by the GNN model. The model is fed by the new node's features. During training, a regularization term to the loss function is added to compel it to minimize the Mean Squared Error (MSE) between embeddings generated by the autoencoder and those generated by the GNN model which we use as reference.

We test and evaluate our proposed autoencoder for new node embedding generation on two different graph datasets: the Amazon and Alibaba social graphs. We empirically determine how similar our generated embeddings are to those we used as references. For both datasets at the test setting, our embeddings solely generated from node features yield small MSE error values (~ 0.1) to those we used as references. We also show the efficiency of the model in encoding the original graph structure by trying to predict potential neighbors of the new node using the generated embeddings.

II. Proposed Autoencoder for New Node Embedding Generation

This section presents the proposed autoencoder for generating embeddings of newly introduced nodes in a graph.

A. Problem Statement

Given a graph $\mathcal{G}$, with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$, we can use a GNN model $\mathscr{E}$ to generate embeddings for nodes using both the nodes features $\mathcal{F}$ and the edges $\mathcal{E}$. Considering the assumption that the graph $\mathcal{G}$ is dynamic meaning that new nodes can be introduced and existing nodes can disappear, we aim to design a model capable of generating embeddings for new nodes without actually relying on (knowing) its connections. Indeed, in a dynamic graph, new nodes are initially isolated from other nodes in the graph. Hence, a typical GNN which aggregates neighborhood embeddings to generate a new one is not able to generate an accurate representation of new still-isolated nodes.

To this end, we propose to use an autoencoder model $\mathscr{A}$ that uses only node features to find an embedding for new nodes. We feed the autoencoder with node features and train it to generate embeddings similar to those that would have been generated by a model trained on entire graph knowledge (reference embeddings). During training, we use the reference embeddings to regularize our model as illustrated in Fig. 1. This phase is basically a transfer learning phase in which the knowledge acquired by $\mathscr{E}$ regarding the initial graph structure is transferred to our model $\mathscr{E}$. This transferred knowledge will be used to regularize how our autoencoder generates embeddings for new nodes' features. During inference (testing phase), the encoder part from our autoencoder, denoted by $\mathscr{A}$, will only use node features to generate a low dimensional representation of these nodes. The ultimate objective is to use the generated embeddings to understand the positioning of the new nodes given its features in the graph. The implementation of the proposed autoencoder model for new node embedding generation is critical for many real world applications. For instance, in recommendation systems, social networks, and IoT networks which are usually modeled by dynamic graphs, we regularly have to deal with new users or devices that need to be accurately placed to integrate the network [15]. For example, a new Amazon customer with no associations and no history should have an embedding that reflects his/her set of features and what future services could be generated from that.

B. Original Embedding Generation

In this study, we use a GNN model $\mathscr{E}$ as a reference. To this end, we employ the GNN, named General Attributed Multiplex HeTerogeneous Network Embedding (GATNE) [16] to generate embedding for a static graph. The GATNE is a state-of-the-art GNN model used for node representation in graphs that are attributed, multiplex (multiple edge types) and heterogeneous (multiple node types). For each node v_i in the graph, this model defines the embedding of a node as the sum of three components: the first two are base and attribute embeddings, which are essentially transformations of the initial features of the node, while the third and most important component is an aggregation of the neighborhood information of the node of interest with the self-attention mechanism.

Although we chose GATNE as reference, our proposed solution can be employed with any GNN model or embedder. Our choice is justified by the ability of GATNE in handling graphs with many characteristics including featured heterogeneous nodes with multi-edges and multiple relations, i.e., the closest to real-world scenarios such as social media.

C. Architecture of the Autoencoder

We propose to employ the autoencorder as the neural network to learn and imitate the original embeddings of the graph. The autoencoder is an unsupervised learning technique that has

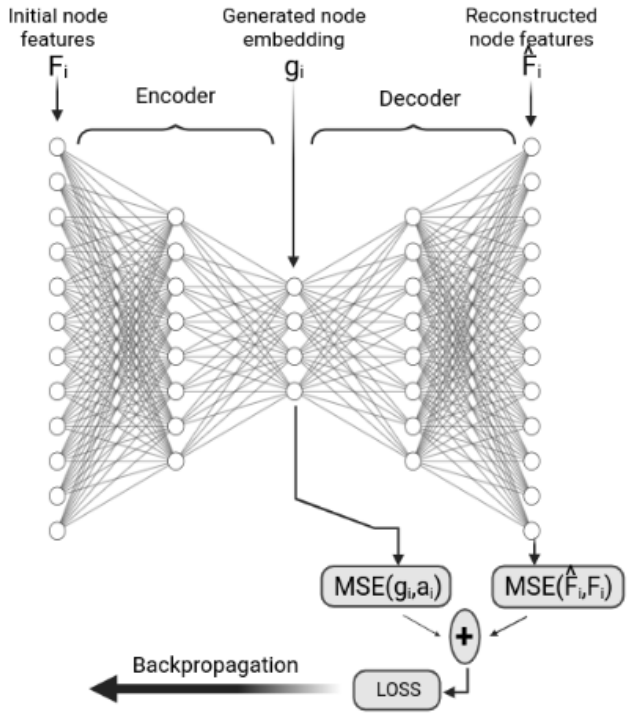

Fig. 2: Architecture of the tuned Autoencoder model.

been used for many applications such as dimensionality reduction, image compression, and even image generation. However, representation learning remains the main application for this technique. With two neural networks, the autoencoder acts in two steps: the first one, called the encoder, encodes the input data into usually a lower-dimensional representation. The latter will hold a compressed knowledge about the node features and their positioning. The second neural network, called the decoder, uses that the low-dimensional representation to try reconstructing the initial node features with the least amount of error between the original and reconstructed features.

In our tuned version of the autoencoder, we do not only seek the reconstructed features to be similar and close to the original features but also we aim to generate embeddings that are similar to the ones generated by the GNN model. However, since the two models have different architectures, and because one model, i.e., $\mathscr{E}$, is trained on the entire graph while the other one, $\mathscr{A}$, will be trained on node features only, their generated embeddings will not necessarily be exactly the same. Thus, to ensure that the autoencoder model generates embeddings a_i similar to the reference embeddings g_i for a given node v_i, we tune the model's loss function so that instead of just minimizing the error between the input features and reconstructed features, at each training step, we also minimize the MSE between the embedding generated by the autoencoder a_i and the one generated by the GNN g_i model as shown in Fig. 2. Consequently, the embeddings solely generated from features reflect how a new node with a set of initial features F_i would be positioned in the graph.

TABLE I: Data sources and statistics

Dataset \ Parameters	Number of training nodes	Number of test nodes	Node's feature size
Amazon graph	9000	1066	1156
Alibaba graph	350	150	142

TABLE II: Tuned hyper-parameters for encoder and decoder

Hyperparameter	Number of hidden layers	Number of units per hidden layer	Dropout rate	Learning rate
Encoder	3	256; 128; 50	0.2	10^{-4}
Decoder	3	50; 128; 256	0.2	10^{-4}

III. Results & Discussion

A. Training and Building the Autoencoder

To train and evaluate our model, we use two graph datasets provided by Alibaba and Amazon. As specified in Table I, the two datasets have different sizes. More details about the datasets can be found in [17], [18]. Amazon and Alibaba are each popular in different regions of the world meaning the underlying structure and the interactions in both graphs might be inherently different; an important factor to make sure that the model is applicable to more than one graph. For evaluation, we test our models performance on new nodes by actually hiding some of the graphs' nodes at training. We then compare the generated embeddings for these never-before-seen nodes to those generated by the GNN model. We apply a grid search on some hyperparameters which we specify in Table II.

B. Model Evaluation

We evaluate our model by applying it to the datasets described earlier. The objective is to determine the similarity between the generated embeddings and those generated by a GNN model trained on entire Graph knowledge. We also aim to measure the efficiency of the generated embeddings at predicting initial node neighbors and hence, reconstruct part of the graph.

1) Model Loss and Mean Squared Error (MSE): The loss function used for training the autoencoder is not a typical loss function of regular autoencoders, which would have been the MSE$(F_i, \widehat{F_i})$, where F_i and $\widehat{F_i}$ are the initial and reconstructed features of the node of interest v_i, respectively. Instead, we use a loss function with a regularization term to force the encoder part in the model to generate accurate representative embeddings a_i that are similar to the reference embeddings g_i. The loss function of a node v_i is expressed as follows:

$$l_i = MSE(F_i, \widehat{F_i}) + MSE(g_i, a_i). \quad (1)$$

After each training step, we compute the average l_i over the entire training set and entire testing set.

Fig. 3 illustrates the loss evolution of the proposed model on Amazon and Alibaba datasets. As the number of epochs increases, the loss keeps decreasing to reach a plateau at 0.012 on the training set and 0.012 on the testing set for the

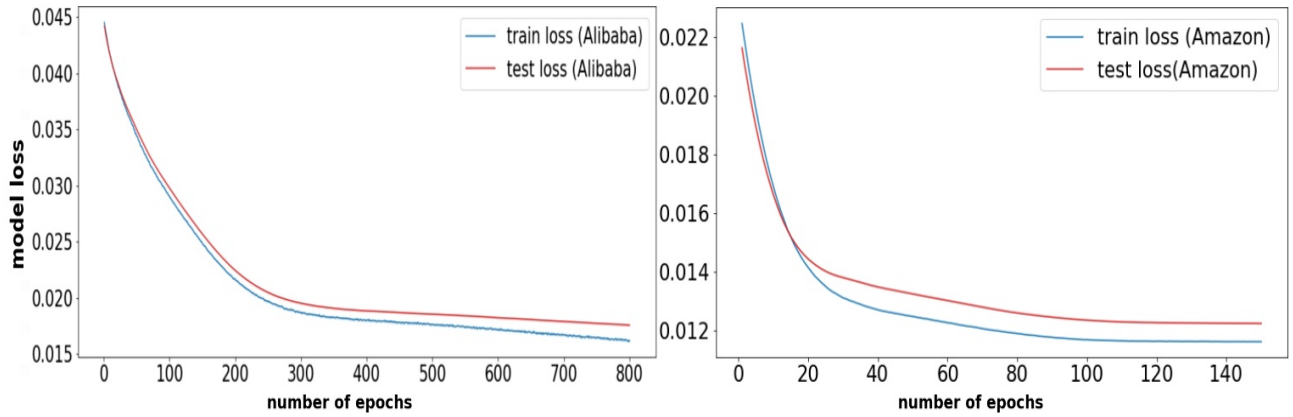

Fig. 3: Loss evolution of the proposed model $\mathscr{A}$ during training for both Alibaba (left) and Amazon (right) graphs.

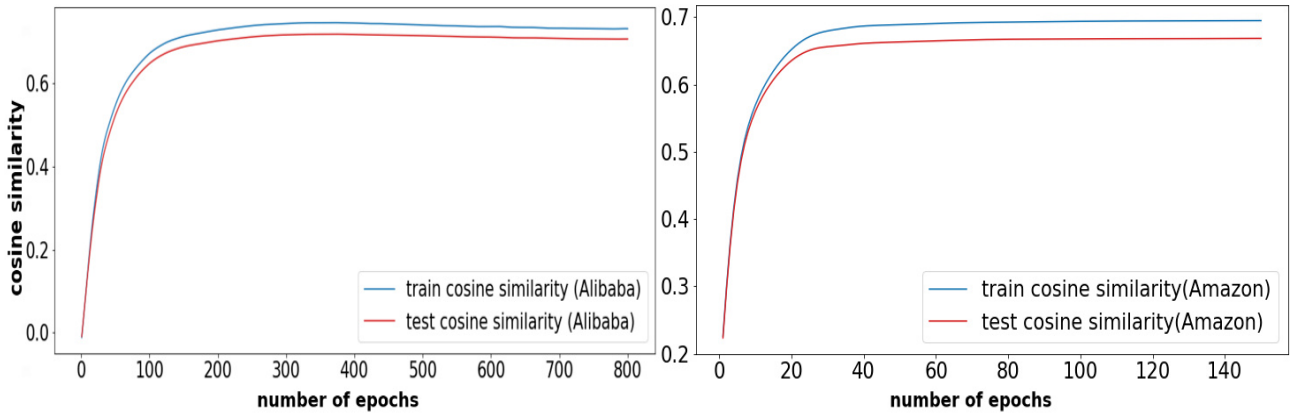

Fig. 4: Average cosine similarity evolution of the proposed autoencoder model $\mathscr{A}$ during training for both the Alibaba (left) and Amazon (right) graphs.

Amazon dataset after 80 epochs of learning from the data. The same behavior is noticed for the Alibaba dataset for which the loss on the training set converges at a value of 0.015 while the loss on the test set is 0.0169, but for this graph the model took more epochs to converge ($\approx$ 500 epochs). Having more initial features for the Amazon nodes (1156) allows the model to reach convergence faster than it took for the Alibaba graph with only 142 initial features. Nevertheless, for both cases, the proposed model is able to encode node features into accurate node embeddings similar and close to the reference embeddings in terms of MSE.

2) Cosine Similarity: A second intuitive measure to evaluate the embedding generation efficiency is the cosine similarity between these generated embeddings a_i and the ones used as references g_i. This similarity measure ranges from -1 to +1 with values closer to the poles 1 and -1 indicating correlation and opposite correlation respectively and values close to 0 indicating decorrelation. This similarity function is expressed as:

$$cos(\boldsymbol{g}_i, \boldsymbol{a}_i) = \frac{\boldsymbol{g}_i \cdot \boldsymbol{a}_i}{||\boldsymbol{g}_i|| \cdot ||\boldsymbol{a}_i||}. \quad (2)$$

For each node v_i, we compute the average cosine similarity over the entire training and testing datasets after each epoch. As illustrated in Fig. 4, as the training goes on, the proposed model learns to generate embeddings that are closer and closer to those generated by the original model trained on full graph knowledge. The cosine similarity between g_i and a_i keeps increasing while having positive values. These positive increasing values mean not only that g_i and a_i are positively correlated but also that they are getting more similar. The average cosine similarity finally reaches convergence for training at a value of 0.7 for the Amazon graph and convergence at testing of 0.66 for the same graph. As for the Alibaba data, the model also converges after 200 epochs to reach a value of 0.73 on the training data and a value of 0.706 on the test data.

3) Mean Absolute Percentage Error (MAPE): We aim to compute a metric that measures the relative error especially for large dataset such as the Amazon graph and thus, we choose to use the MAPE which evaluates the percentage of relative error for each node.

Fig. 5 shows the average MAPE values on the training and testing sets. It is shown that the model starts generating node

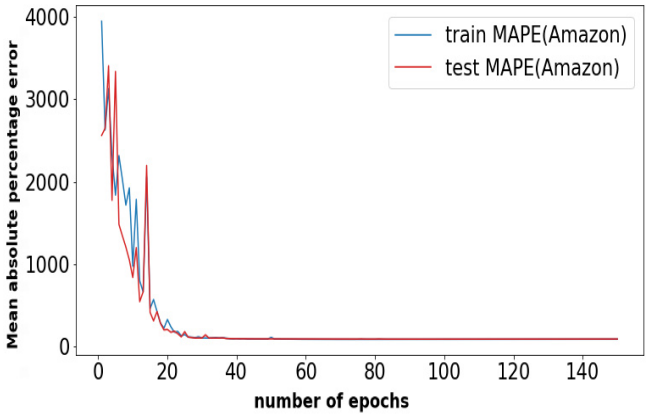

Fig. 5: MAPE evolution during training for the Amazon graph.

TABLE III: Accuracy of generated embeddings in Neighborhood prediction on both datasets

% / dataset	Neighborhood prediction percentage
Amazon graph	80.12%
Alibaba subgraph	76.33%

embeddings that are far from the reference with error values over 3000 for the first epochs meaning that for many nodes the reference and generated embeddings did not even have the same order of magnitude (component wise). However, as the model gets closer to convergence, the MAPE drops under 100, meaning that they got more and more similar and they now range at the same order of magnitude with small error values.

4) Link Prediction Accuracy: The main goal of our solution is to handle newly introduced nodes in dynamic graphs. Once we generate an embedding for those nodes, we can perform a link prediction computation to find out potential neighbors for these nodes by measuring the cosine similarity between the embeddings of new nodes and existing nodes. One way to make sure that the embeddings generated by our encoder are accurate is to try reconstructing the graph using these embeddings, meaning for each node we try to guess whether a link exists between that node and each of its neighbors in the original graph. Unlike other graph learning models, our model does not use the links and connections in the graph to learn embeddings. It treats each node separately and in complete isolation and thus being able to reconstruct the original neighborhood of nodes or at least a good part of it.

In Table III, we present our findings and the average percentage of the neighborhood we were able to re-predict their connections using their embeddings only for each of the used datasets. Recall that in our case, the embeddings are generated using the nodes' features only. For the Amazon graph, we achieve an efficiency of 80% while with the Alibaba dataset, we are able to determine around 76% of the possible neighborhood. The latter result might be explained by the small number of nodes used in training the Alibaba dataset.

IV. Conclusion

In this paper, we have addressed the problem of node representation in graphs, specifically the case of newly introduced nodes in a dynamic graph. We built and trained an autoencoder architecture capable of mapping node features to representative node embeddings reflecting how that node relates to the entire graph. The autoencoder is regularized by a prior GNN model which is trained on full graph knowledge. Using just the node features, this architecture showed its ability in generating representative embeddings for new graph nodes. We achieved a high level of representativeness by computing how much of each nodes original neighbors can be repredicted using the generated embedding and how similar is that generated embedding to the one generated by a model that learned on full graph knowledge.

References

[1] Y. Qiao, X. Luo, C. Li, H. Tian, and J. Ma, "Heterogeneous graph-based joint representation learning for users and pois in location-based social network," *Information Processing Management*, vol. 57, no. 2, 2020.

[2] Y. Liu, H. Yuan, L. Cai, and S. Ji, "Deep learning of high-order interactions for protein interface prediction," *CoRR*, vol. abs/2007.09334, 2020.

[3] A. Omran et al., "Joint relay selection and load balancing using d2d communications for 5g hetnet mec," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019.

[4] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *CoRR*, vol. abs/1806.01973, 2018.

[5] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," *CoRR*, vol. abs/1101.3291, 2011.

[6] S. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. 40, 2010.

[7] H. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques and applications," *CoRR*, vol. abs/1709.07604, 2017.

[8] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018.

[9] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[11] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web* (A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, eds.), (Cham), pp. 593–607, Springer International Publishing, 2018.

[12] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016.

[13] A. Khanfor, A. Nammouchi, H. Ghazzai, Y. Yang, M. R. Haider, and Y. Massoud, "Graph neural networks-based clustering for social internet of things," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1056–1059, 2020.

[14] A. Hamrouni, H. Ghazzai, T. Alelyani, and Y. Massoud, "Low-complexity recruitment for collaborative mobile crowdsourcing using graph neural networks," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 813–829, 2022.

[15] A. Bader, H. Ghazzai, A. Kadri, and M.-S. Alouini, "Front-end intelligence for large-scale application-oriented internet-of-things," *IEEE Access*, vol. 4, pp. 3257–3272, 2016.

[16] Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang, "Representation learning for attributed multiplex heterogeneous network," vol. abs/1905.01669, 2019.

[17] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *Proc. of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

[18] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proceedings of the International Conference on World Wide Web*, WWW '16, p. 507–517, 2016.