DyHNet: Learning Dynamic Heterogeneous Network Representations

Hoang Nguyen
Ryerson University
Radin Hamidi Rad
Ryerson University
Fattane Zarrinkalam (fzarrink@uoguelph.ca)
University of Guelph
Ebrahim Bagheri
Ryerson University

Research Article

Keywords: Dynamic heterogeneous network, Network representation learning, Random walk

Posted Date: December 5th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2327811/v1

Additional Declarations: No competing interests reported.

DyHNet: Learning Dynamic Heterogeneous Network Representations

Hoang Nguyen[1], Radin Hamidi Rad[1], Fattane Zarrinkalam[2] and Ebrahim Bagheri[1]

[1] Toronto Metropolitan University, Toronto, Canada.
[2] University of Guelph, Guelph, Canada.

Contributing authors: hoang.cam.nguyen@ryerson.ca; radin@ryerson.ca; fzarrink@uoguelph.ca; bagheri@ryerson.ca;

Abstract

Many real-world networks, such as social networks, contain structural heterogeneity and experience temporal evolution. However, while there has been growing literature on network representation learning, only a few have addressed the need to learn representations for *dynamic heterogeneous* networks. The objective of our work in this paper is to introduce `DyHNet`, which learns representations for such networks and distinguishes itself from the state-of-the-art by systematically capturing (1) local node semantics, (2) global network semantics, and (3) longer-range temporal associations between network snapshots when learning network representations. Through experiments on four real-world datasets, we demonstrate that our proposed method is able to show consistently better and more robust performance compared to the state-of-the-art techniques.[*]

Keywords: Dynamic heterogeneous network, Network representation learning, Random walk

[*]The source code, data, and/or other artifacts have been made available at https://github.com/hoangntc/DyHNet

1 Introduction

The objective of network (graph)[1] representation learning is to encode the semantic and structural information of a network into low-dimensional space, which can then be used for downstream tasks such as link prediction, node classification, information retrieval and recommendation [1, 2]. Existing network representation learning methods can be classified based on whether they function over heterogeneous or homogeneous networks and whether they are limited to learning representations for static networks or are also able to capture the temporal nature of dynamic networks.

Most of existing network representation learning methods have mainly explored the development of representations for *static* networks [3]. In this context, earlier works offer intuitive approaches for adapting random walk [4, 5] and skip-gram [6] methods for learning network representations. These methods exploit first-order and higher-order proximity to characterize network structure [7]. Some methods approximate network structure through message passing schemes in order to capture the notion of node neighborhood [8–10]. More recently, several metapath-based neural network approaches have been developed that preserve the properties of heterogeneous networks such as the diversity of node and relation types [11–13]. Although these approaches have shown a great deal of success in some downstream tasks, they face two main limitations. First, given their heavy reliance on message-passing schemes, they seem to be very well equipped to capture local network features for nodes and edges but fail to preserve global network characteristics [14, 15]. Second, these methods have only been developed to operate over static networks and hence cannot be used for dynamic temporally evolving networks.

It is important to capture temporality in *dynamic* networks as real-world networks do naturally evolve and change over time and such evolution needs to be reflected in network representations. Dealing with dynamic networks has shown to be a challenging task as it requires the models to capture local and global temporal associations between nodes and edges of different network snapshots [16]. To date, most existing approaches for dynamic networks have been focused on addressing a specific task such as link prediction [17, 18] and recommendation [19]. These methods focus on learning features from the dynamic network that can optimize the task at hand, e.g., defining features between node pairs that can optimize link prediction. However, there have been more recent works that learn network representations for dynamic networks regardless of the downstream task which are limited to *homogeneous* networks. For instance, these methods use exponential distribution of time to weigh network edges [20] or use stacked graph neural networks (GNNs) or continuous modelling recurrent neural networks (RNNs) to fuse multiple network snapshots [21–23]. Such approaches are not able to handle heterogeneity as they assume the dynamic network to consist of only homogeneous nodes.

[1] We use the terms *graph* and *network* interchangeably in this paper.

To the best of our knowledge, there are only a few works that are able to explicitly learn network representations for dynamic heterogeneous networks [24–26]. These works attempt to reduce the complexity of dynamic heterogeneous networks through marginalizing either *temporality* or *heterogeneity*. For instance, `DHNE` [25] proposes that the temporal nature of a dynamic network can be captured by associating each node in a certain snapshot with the nodes associated with it in the previous or next snapshots. The limitation of this approach is that it is not able to capture longer-range temporal associations between network snapshots. In contrast, `DyHATR` [24] proposes to reduce a heterogeneous network into homogeneous sub-graphs whose nodes are then associated with each other through a hierarchical attention mechanism. While the attention mechanism allows for capturing longer-range dependencies between snapshots, the downside of such an approach is that by removing associations between heterogeneous node types, the model will not be able to learn explicit interaction dynamics between nodes of different types. In summary, we find that existing network representation learning methods that deal with dynamic networks may be limited by at least: (1) being designed specifically for homogeneous networks [27] or reduce a heterogeneous network into homogeneous subgraphs [28]; hence, potentially forgoing interactions between heterogeneous types; and/or (2) restricting temporal associations between dynamic network snapshots to those immediately proceeding or succeeding a timestamp; therefore, possibly missing longer-range temporal associations.

The objective of our work in this paper is to propose an approach, referred to as `DyHNet`, to learn representations for *dynamic heterogeneous networks* that would satisfy the following characteristics: **(a)** capture local and global network structural characteristics without being limited only to the local features of network nodes; **(b)** incorporate interactions between different node types without having to model individual node type relations in isolation; and **(c)** integrate longer-range temporal associations between network snapshots by moving beyond immediate neighboring snapshots.

The key contributions of our work are as follows: **(1)** We propose a neural architecture that captures local network semantics as well as global network semantics and effectively integrates them by considering network structure. **(2)** We utilize a meta-path neighborhood aggregation method to capture both network heterogeneity and higher-order proximity information. **(3)** We adopt an attention-based temporal encoding mechanism that captures longer-range graph evolution over several snapshots. **(4)** We show through our experiments on four real-world datasets that our proposed approach is able to show better performance compared to a range of state-of-the-art network representation learning methods.

2 Related work

Depending on whether they are limited to learning representations for static networks or are also able to capture the temporal nature of dynamic networks, we can divide related work into two categories: *static network representation learning* and *dynamic network representation learning.*

2.1 Static Network Representation Learning

Most existing network representation learning methods have mainly focused on representation learning for static graphs [3]. Some earlier studies proposed to utilize random walks to preserve the local and global graph structure. For example, DeepWalk [4] and Node2Vec [5] learn feature representation of nodes by applying the skip-gram model on the sequences of nodes generated by a random walk strategy. SDNE [29] and LINE [7] learn feature representation of nodes by optimizing an objective function that preserves first-order and second-order proximities of nodes within a graph. Further, due to the powerful ability of graph neural networks (GNNs) to learn feature representations, more recent studies have applied GNN-based methods for graph representation learning [8, 10]. For example, Graph Attention Network (GAT) [10] leverages masked self attention layers to learn the importance of nodes by considering the features of neighbors. Personalized Propagation of Neural Predictions (PPNP) [30] is derived by incorporating personalized PageRank into graph convolutional networks (GCNs).

While most network representation learning techniques are designed for homogeneous graphs and do not consider different types of nodes and relations, recently, a number of studies have aimed at designing techniques for heterogeneous graphs [31]. For example, DHNE [32] is a deep hyper network-based method that considers a non-linear tuple-wise similarity function in its embedding space while capturing both the local and global structures of a heterogeneous graph. Metapath2vec [33] uses a meta-path based random walk technique to build the heterogeneous neighborhood of a node and then applies a heterogeneous SkipGram model to embed the nodes. MetaGraph2Vec [34] generates heterogeneous node sequences by applying a meta-graph based random walk strategy and then the feature representation of nodes are learned by employing a heterogeneous skip-gram technique over the node sequences. HetGNN [35] samples a set of strongly correlated heterogeneous neighbors of each node by applying a random walk approach to group nodes based on their type and then applies Bi-LSTM on each group to aggregate feature information these groups of nodes. HAN [12] proposes a heterogeneous neural network based on hierarchical attention to learn the importance between a node and its metapaths and semantic-level attentions to learn the importance of different metapaths. Node embeddings are finally generated by aggregating features from metapath-based neighbors of each node.

Most network representation learning methods employ paths for capturing relationships among nodes and overlook the structural and semantic information in the subgraphs of a node. To solve this issue, M-HIN [36] proposes a heterogeneous graph embedding model that generates metagraphs and then apply the Hadamard function to describe the relationship between nodes and metagraphs. Recently, to take into account the properties of subgraphs, subgraph-based network representation learning has been proposed and has demonstrated advantages in many important downstream applications like link prediction and graph classification [15]. For example, SubGNN [15] is a subgraph-level GNN that propagates neural messages between the subgraph's components and randomly samples patches from the whole graph and aggregates their features to learn feature representations of subgraphs. To improve SubGNN by distinguishing nodes inside and outside the subgraph, GNN with LAbeling trickS for Subgraph (GLASS) [37] utilizes an expressive and scalable labeling trick to enhance GNNs for subgraph representation learning. Most recently, to address the scalability issue in the subgraph representation learning problem via GNNs, SUREL [38] reduces the redundancy of subgraph extraction and supports parallel processing by decoupling the graph structure into sets of walks and reusing the walks to form subgraphs.

2.2 Dynamic Network Representation Learning

The importance of capturing temporality in dynamic graphs has received increasing attention in the literature [16]. Most of these studies have been designed for homogenous graphs. For example, DynamicTraid [39] learns the embeddings of each node at different time steps by modeling the evolution of a dynamic graph based on the triadic closure process. DynGEM [40] proposes a deep autoencoder model to construct stable embeddings for dynamic graphs. It can handle growing dynamic graphs and incrementally learn embeddings of each snapshot of the graph from previous time steps. Dyngraph2vec [41] proposes an LSTM-based model to process the snapshots of dynamic graphs by representing the graph structure using an adjacency matrix. EvolveGCN [22] proposes a dynamic network representation learning method by combining the GCN and RNN. To capture the dynamicity of the graph, this method uses RNNs to evolve the GCN parameters. DySAT [21] proposes a self-attentional neural network architecture in which dynamic node representations are captured by applying self-attention over the structural neighborhood and historical snapshots.

The above-mentioned methods do not capture the heterogeneity of dynamic networks. Recently, a few studies have focused on designing techniques for dynamic heterogeneous network representation learning. For example, change2vec [26] represents a dynamic heterogeneous graph through static graph snapshots at different timestamps and then instead of processing the static graphs, it utilizes a *change model* to only train over their deltas. Further, it uses Metapath2vec [33] to model the heterogeneity of each static graph snapshot. As another method, DHNE [42] captures temporal characteristics of a dynamic graph by constructing comprehensive historical current graphs based on subgraphs of snapshots. Then, to learn node embedding, it proposes a dynamic heterogeneous skip-gram model over node sequences extracted based on a metapath-based random walk model. DHNE cannot incrementally update the node embeddings after each change and needs to be retrained which is time-consuming. To solve this issue, DyHNE [43] proposes a model with meta-path based proximity in which the change of structure and semantics are incorporated by metapath augmented adjacency matrices, and perturbation theory is used to efficiently learn the node embeddings. In contrast, DyHATR [24] uses a hierarchical attention mechanism to capture the heterogeneity of a graph and introduces a temporal attentive temporal RNN model to capture dynamic characteristics of the graph. Most recently, M-DHIN [44] utilizes the triadic evolution process to represent every snapshot of the dynamic heterogeneous graphs. This approach predicts the future of dynamic graphs via an LSTM-based deep autoencoder.

These methods attempt to reduce the complexity of dynamic heterogeneous networks through marginalizing either temporality or heterogeneity. Nevertheless, there is still lack of work that would take advantage of substructures of the graph when learning representations. Our proposed approach, `DyHNet`, takes such an approach and attempts to generate graph embeddings which capture different topological substructures while still maintaining both the heterogeneous and temporal evolutionary patterns of the graph. Specifically, `DyHNet` captures node-level graph substructures by adopting a GNN approach. Further, `DyHNet` enriches the semantics of the node by obtaining and summarizing the representative information of its graph substructure, i.e., subgraphs. `DyHNet` also preserves heterogeneity through a meta-path based random walk approach and captures the temporal evolution of the network through an attention mechanism.

3 Preliminaries

We first introduce notational conventions and then formulate the dynamic heterogeneous network representation learning problem. Frequently used symbols are summarized in Table 1 for reference.

3.1 Notations

Definition 1 (Heterogeneous Information Network (HIN)) A HIN is defined as a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ denotes the set of edges between nodes. Each node $v_i \in \mathcal{V}$ and each edge $e_i \in \mathcal{E}$ are affiliated with a type mapping function $\psi : \mathcal{V} \rightarrow A$ and $\psi : \mathcal{E} \rightarrow R$, respectively. A and R represent the set of all possible node types and edge types with $|A| > 1$ and $|R| > 1$.

A HIN can be extended to support multiple time snapshots in order to represent a Dynamic HIN, defined as below.

Definition 2 (Dynamic Heterogeneous Information Network (Dynamic HIN)) We define a dynamic HIN as a series of graph snapshots, $G = \left\{\mathcal{G}^1, \mathcal{G}^2, ..., \mathcal{G}^T\right\}$ where T is the number of snapshots. A static HIN at time t is a HIN $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$. For two consecutive timestamps t and $t+1$, the conditions, $|\mathcal{V}^t| \neq |\mathcal{V}^{t+1}|$ and $|\mathcal{E}^t| \neq |\mathcal{E}^{t+1}|$, are satisfied when the nodes and the edges are updated, respectively.

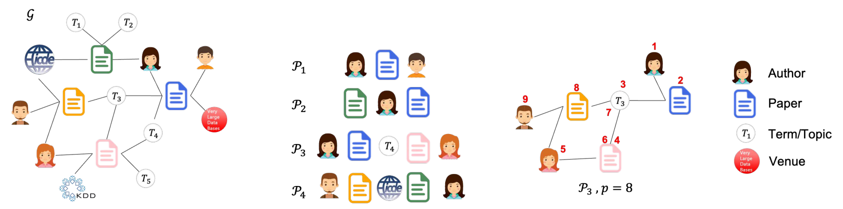
Table 1 Summary of Frequently Used Notations.

Symbol	Meaning
G	the dynamic heterogeneous information network
T	the number of timestamps (snapshots) in the network G
N	the number of nodes in the network G
$\mathcal{V}$	the set of nodes in G
$\mathcal{E}$	the set of edges in G
$\mathcal{G}^t$	the heterogeneous information network at timestamp t
$\mathcal{G}'$	the subgraph of network $\mathcal{G}$
M	the number of subgraphs $\mathcal{G}'$ in network $\mathcal{G}$
$\mathcal{LGE}$	the local graph encoder
$\mathcal{GGE}$	the global graph encoder
$\mathcal{SGE}$	the subgraph graph encoder
$\mathcal{SE}$	the structural encoder
$\mathcal{TE}$	the temporal encoder
$\mathcal{AS}$	the set of sampled anchor subgraphs
k	the number of sampled anchor subgraphs
m	the number of sampled structure subgraphs
l_i^t	the local representation for node v_i at timestamp t
h_i^t	the global representation for node v_i at timestamp t
s_i^t	the fused representation for node v_i at timestamp t
z_i^t	the final representation for node v_i at timestamp t

Most real-world information networks are dynamic and heterogeneous. They contain more than one type of nodes, edges and each node/edge is associated with a certain timestamp. The DBLP bibliographic network is a typical example of a Dynamic HIN. The network may contain different types of nodes such as papers, authors, keywords/terms and venues. Many different types of edges can be formed between these nodes. For example, the relationship "write" is denoted by an edge between the author and paper nodes, and the relationship between paper and venue nodes can be described through a "published in" edge type. Further, a graph such as one built based on the DBLP dataset is a dynamic network as the interactions between its nodes (e.g., published papers or publishing authors) evolve over time. Given the heterogeneity of such networks, semantically meaningful or permissible traversals over the network are often expressed through *metapaths*.

Definition 3 (Metapath) A metapath $\mathcal{P}$ is defined as a pattern of the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} A_3 \cdots A_{p-1} \xrightarrow{R_{p-1}} A_p$ which represents a composite relation $R = R_1 \cdot R_2 \cdots R_{p-1}$ between node types A_1 and A_p on a HIN. p is the length of metapath $\mathcal{P}$ and $\cdot$ denotes the relation composition operator.

Considering the DBLP dataset schema, shown in Figure 1, which consists of four node types, i.e., papers, authors, terms and venues, and three relations, i.e., papers-authors, papers-terms and papers-venues, it is possible to define different metapaths, as already suggested in earlier work such as [45, 46]. For example, the metapath "author-paper-term-paper-author" captures the relation between papers written by two authors who share similar topics.



(a) Heterogeneous graph. (b) Metapaths (c) Metapath-based subgraph.

Fig. 1 Illustration of definitions on the DBLP dataset.

Definition 4 (Subgraph) Given a network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a subgraph of $\mathcal{G}$, denoted by $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$, is a graph whose node set is in $\mathcal{V}$ ($\mathcal{V}' \subseteq \mathcal{V}$) and its edge set is a subset of $\mathcal{E}$ ($\mathcal{E}' \subseteq \mathcal{E}$).

A heterogeneous network can also be defined based on the set of subgraphs that it contains. There can be many different types of subgraphs depending on the way we construct them.

A *metapath-based subgraph* is a subgraph which is generated by a set of nodes connected by a metapath. This type of subgraph also shares the same semantics with its corresponding metapath [11, 45]. Figure 1c depicts an example of the metapath-based subgraph in the DBLP dataset. In this example, the subgraph is constructed by the metapath $\mathcal{P}_3$ with the length of 9.

4 Proposed Approach

Here, we first give an overview of our approach, namely `DyHNet`, and then provide more details about its components.

4.1 Overview

The overview of our proposed approach is shown in Figure 2. In our approach, we advocate that various characteristics of dynamic heterogeneous graphs need to be taken into consideration, namely (1) local semantics, (2) global semantics, and (3) temporal semantics, when learning representations. To this end, we capture local semantics through a *local graph encoder* ($\mathcal{LGE}$) in each snapshot. The global semantics are captured by a *global graph encoder* ($\mathcal{GGE}$), which consists of two successive stages. First, multiple structure subgraphs associated with each node in a single snapshot are sampled and encoded through a *metapath-aware subgraph encoder* ($\mathcal{SGE}$); then we learn a *structural encoder* ($\mathcal{SE}$) to project subgraph embeddings onto a common latent space which represents the global semantics of the node. Finally, local and global semantics are fused in order to provide a unified embedding for every node in each snapshot. Finally, we incorporate a *temporal encoder* ($\mathcal{TE}$) to capture the evolution of the graph over sequential snapshots.

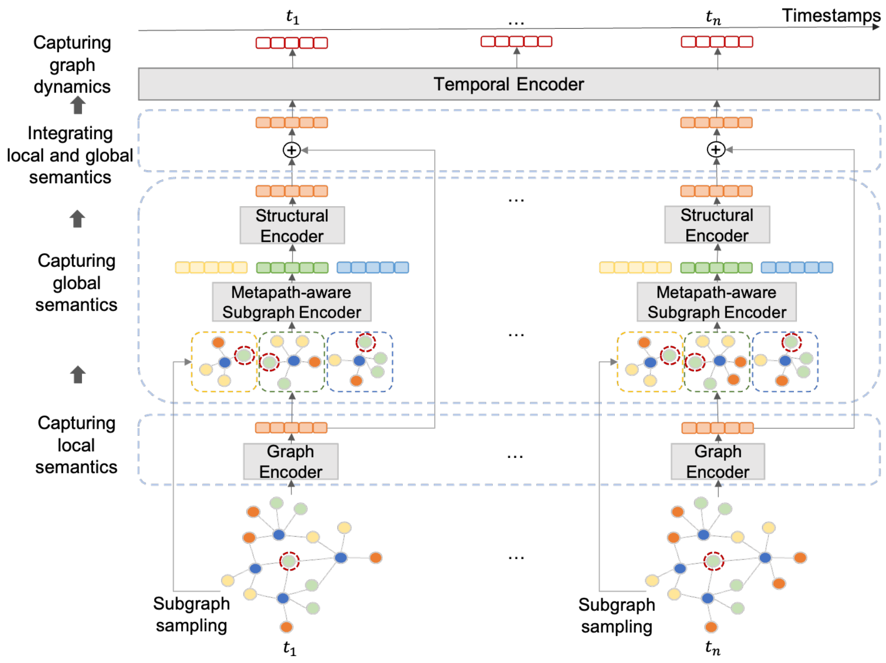

Timestamps
t_1
...
t_n
Capturing graph dynamics
Temporal Encoder
Integrating local and global semantics
Structural Encoder
Capturing global semantics
Metapath-aware Subgraph Encoder
Capturing local semantics
Graph Encoder
Subgraph sampling

Fig. 2 Overview of our proposed `DyHNet` approach.

4.2 DyHNet Components

In this section, we explain the main components of `DyHNet` in detail.

4.2.1 Capturing Local Semantics

The intuition behind our formulation of local semantics is based on the idea that nodes and their neighbors are more likely to be correlated with each other; therefore, for any node, its representation can be determined based on the representations of its neighbors [9]. Hence, at each timestamp, we define a local graph encoder $\mathcal{LGE}: \mathbb{R}^{N\times N} \rightarrow \mathbb{R}^{N\times d_1}$, which takes the structure of the HIN at each timestamp as input to learn the representation for each node v_i in that timestamp as follows:

$$e_i^k = AGGREGATE^{(k)}\left(\left\{l_j^{(k-1)} : v_j \in \mathcal{N}_{v_i}\right\}\right) \tag{1}$$

$$l_i^k = COMBINE^{(k)}\left(l_i^{(k-1)}, e_i^k\right) \tag{2}$$

where l_i^k is the feature vector of node v_i at the k^{th} iteration, $\mathcal{N}_{v_i}$ is its neighbor node set, $AGGREGATE^{(k)}(.)$ and $COMBINE^{(k)}(.)$ are arbitrary differentiable functions. Xu et al. [47] argue that the choice of these two functions is critical and suggest that $AGGREGATE^{(k)}(.)$ can be operationalized through element-wise max-pooling and $COMBINE^{(k)}(.)$ as concatenation followed by a linear mapping, which we adopt in our implementation. The node representations learned in this approach represent the aggregated combination of the representations of neighboring nodes around each node; and, hence capture the *local semantics* of nodes.

The local graph encoder $\mathcal{LGE}$ is adopted for learning the local representation of each node in every time snapshot $\mathcal{G}^t \in G$. We denote the encoded representation of node v_i in $\mathcal{G}^t$ as l_i^t.

4.2.2 Capturing Global Semantics

We further hypothesize that each node is not only characterized locally within its surrounding nodes but also by the structure of the subgraphs it belongs to. Intuitively, the subgraphs which share similar structural characteristics would have similar representations regardless of the distance between them within a large network. Therefore, if two nodes experience similar regional structures, their representations should be close to each other in latent space. For this purpose, at each timestamp t, we develop a global graph encoder $\mathcal{GGE}$: $\mathbb{R}^{N\times d_1} \rightarrow \mathbb{R}^{N\times d_2}$ which first focuses on the associations between the subgraphs to learn the subgraph representation and then uses these representations to encode the global information of the nodes. The global graph encoder ($\mathcal{GGE}$) consists of two components, metapath-aware subgraph encoder ($\mathcal{SGE}$) and structural encoder ($\mathcal{SE}$).

At each timestamp, graph $\mathcal{G}$ can be represented as a set of subgraphs, i.e., $\mathcal{G} = \{(\mathcal{G}'_i)\}$. Each node $v_i \in \mathcal{V}$ belongs to a set of subgraphs. In order to learn global semantics for nodes, the metapath-aware subgraph encoder is first utilized to obtain the representations of all the subgraphs $\mathcal{G}'$, then the structural encoder leverages these representations to generate node embedding v_i.

Metapath-aware Subgraph Encoder. We propose to capture network subgraph representations based on a metapath-aware subgraph encoder $\mathcal{SGE}$: $\mathbb{R}^{N\times d_1} \rightarrow \mathbb{R}^{M\times d_2}$, which maps a subgraph into the latent space through a three-phase graph traversal method. More precisely, in the first phase, namely *anchor subgraph selection*, a set of metapath-based subgraphs, referred to as anchor subgraphs, are randomly sampled using a metapath-guided random walker [48]. In the second phase, i.e., the *anchor subgraph encoder*, messages from these anchor subgraphs are encoded by a K-hop neighborhood model applied to the sequence of nodes generated in the first phase. In the last phase, i.e., *subgraph encoder*, messages from the anchor subgraphs are propagated to all other subgraphs to produce the subgraph representations. We present the details of each of these three phases in the following.

Phase 1. Anchor Subgraph Selection

Let $\mathcal{AN}^t = \{AN_1^t, AN_2^t, \ldots, AN_{\mathrm{k}}^t\}$ be a set of k seed nodes in a graph snapshot $\mathcal{G}^t$. At each timestamp t, we define $W_i^t = \left\{v_{AN_i^t}^1, v_{AN_i^t}^2, \ldots, v_{AN_i^t}^p\right\}$ as a meta-path guided random walk of length p starting from node $AN_i^t \in \mathcal{AN}^t$, $AN_{\mathrm{i}}^t \equiv v_{AN_i^t}^1$. The strategy for the walker is as follows:

$$p\left(v^{i+1} \mid v_a^i, \mathcal{P}\right) = \begin{cases} \frac{1}{|N_{a+1}(v_a^i)|} & \left(v^{i+1}, v_a^i\right) \in \mathcal{E}, \psi\left(v^{i+1}\right) = a+1 \\ 0 & \left(v^{i+1}, v_a^i\right) \in \mathcal{E}, \psi\left(v^{i+1}\right) \neq a+1 \\ 0 & \left(v^{i+1}, v_a^i\right) \notin \mathcal{E} \end{cases} \tag{3}$$

where v_a^i denotes the i^{th} node in a given meta-path $\mathcal{P}$ with node type a.

The walk W_i^t initialized from AN_i^t then forms a anchor subgraph AS_i^t. We denote a set of anchor subgraphs corresponding to the set of starting nodes $\mathcal{AN}^t$ as $\mathcal{AS}^t = \{AS_1^t, AS_2^t, \ldots, AS_{\mathrm{C}}^t\}$. Hence, the size of two sets, $\mathcal{AN}^t$ and $\mathcal{AS}^t$, are equal.

Phase 2. Anchor Subgraph Encoder

The feature vector a_i^t of the anchor subgraph $AS_i^t \in \mathcal{AS}^t$ at timestamp t can be obtained by a K-hop Neighborhood Aggregation mechanism (KNG):

$$a_i^t = \mathrm{KNG}(l_{v^1_{AN_i^t}}, l_{v^2_{AN_i^t}}, \ldots, l_{v^p_{AN_i^t}}) \tag{4}$$

where $l_{v^j_{AN_i^t}}$ is the node representation of the j^{th} node in the walk W_i^t which forms the anchor subgraph AS_i^t. The representation of each node in the walk is computed by Equations 1 and 2. The KNG mechanism allows aggregating the messages sequentially from a node to its neighbors following the semantics of the meta-path random walk and finally aggregates the messages from all the neighboring nodes to be used as the anchor subgraph representation. KNG can be any predefined functions or learnable neural network, which has the ability to aggregate incremental signals from previously visited nodes [49] such as an attention mechanism [49] or a Long Short-Term Memory network (LSTM) [15]. In our work, we learn KNG by adopting a bidirectional long short-term memory recurrent network and use the SUM operator to aggregate incoming messages.

Phase 3. Subgraph Encoder

At each graph snapshot, given the graph $\mathcal{G}^t = \{(\mathcal{G'}_i^t)\}$, the subgraph encoder aims to vectorize each subgraph $\mathcal{G'}_i^t$. Once we have the anchor subgraphs and their feature vector (messages), we adopt a message passing model which computes and gathers the structural similarity between the subgraph $\mathcal{G'}_i^t$ and the anchor subgraphs to produce the final subgraph representation. Formally, subgraph representation g_i^t of the subgraph $\mathcal{G'}_i^t \in \mathcal{G}^t$ can be calculated as:

$$g'^{t}_{i} = \gamma\left(\left\{sim(\mathcal{G}'^{t}_{i}, AS^{t}_{j}) \cdot a^{t}_{j}, \quad \forall AS^{t}_{j} \in \mathcal{AS}^{t}\right\}\right) \tag{5}$$

$$g^{t}_{i} = \mathcal{SGE}(\mathcal{G}'^{t}_{i}) = \sigma(\mathbf{W}g'^{t}_{i} + b) \tag{6}$$

where γ is a function that gathers messages in regard to the i^{th} subgraph, sim is a similarity function, and σ is a non-linear activation function. We adopt the dynamic time warping (DTW) function calculated based on the connection characteristics between the two subgraphs for sim and SUM for γ as suggested in [15]. It is worth noting that our model has a separate set of anchor subgraphs for performing message passing at each snapshot $\mathcal{G}^t \in G$ while the parameters of subgraph encoder $\mathcal{SGE}$ are shared across all snapshots.
Structural Encoder. Based on the learnt subgraph representations, we now integrate the information from the subgraph representations to learn node representations that capture global semantics. Formally, at each graph snapshot, given node v_i and the m subgraphs $\{(\mathcal{G}'^{t}_{j})\}_{j=1}^{m}$ that it appears in, we adopt a structural encoder, $\mathcal{SE} : \mathbb{R}^{m\times d_2} \to \mathbb{R}^{d_2}$, to encode all subgraph representations into a unified representation to capture the global structural characteristics of the node. Here, we adopt an attention mechanism to learn the importance of each subgraph among all subgraphs and then, the output is computed by taking the weighted average of the subgraphs' representations. This way, the latent representation contains the summaries of topology of all subgraphs. This process can be summarized as follows:

$$\alpha^{t}_{ij} = f(\mathcal{G}'^{t}_{j}, c) = o^T \tanh(W g^{t}_{j} + Uc) \tag{7}$$

where tanh(.) is the tanh activation function, $o \in \mathbb{R}^{d_2}$ is the weight vector, $c \in \mathbb{R}^{d_2}$ is the learnable context vector, and W, $U \in \mathbb{R}^{d_2\times d_2}$ are the learnable weight matrices. As such, the final subgraph representation based on global semantics is computed as follows:

$$h^{t}_{i} = \mathcal{SE}(\{(\mathcal{G}'^{t}_{j})\}_{j=1}^{m}) = \sum_{j=1}^{m} \alpha^{t}_{ij} g^{t}_{j} \tag{8}$$

The global information of the node v_i in $\mathcal{G}^t$ is produced by $\mathcal{GGE} = \mathcal{SGE} \circ \mathcal{SE}$ and it is denoted as h^t_i.

4.2.3 Integrating Local and Global Semantics

In order to obtain a latent representation s^t_i for node v_i with dimension d at timestamp t that can capture both local and global semantics of the given node, we aggregate the representations trained by the local graph encoder ($\mathcal{LGE}$) and the global graph encoder ($\mathcal{GGE}$) as follows:

$$s^{t}_{i} = AGGREGATE(l^{t}_{i}, h^{t}_{i}) \tag{9}$$

where l^t_i is the representation obtained by the local graph encoder $\mathcal{LGE}$, h^t_i is the summary representation of all subgraphs of the given node produced by $\mathcal{GGE}$; and $AGGREGATE(.)$ denotes the aggregation operation. In our proposed model, we adopt concatenation for $AGGREGATE(.)$.

4.2.4 Capturing Graph Dynamics

In order to be able to capture network evolution over different temporal snapshots, over both closer (shorter-range) and more distant (longer-range) snapshots, we employ a multi-head attention mechanism to systematically fuse the representations of all graph snapshots over time. Furthermore, we incorporate the chronological order of the snapshots by applying a Transformer-like positional encoder [50] before feeding the input to the temporal encoder.

At timestamp t, the temporal encoder, $\mathcal{TE} : \mathbb{R}^{T\times N\times d} \to \mathbb{R}^{N\times d'}$, produces the position vector $p_i^t \in \mathbb{R}^d$ as follows:

$$\mathcal{TE}(t) = p_i^t = \begin{cases} sin(w_k \cdot t) & \text{if } i = 2k \\ cos(w_k \cdot t) & \text{if } i = 2k+1 \end{cases} \tag{10}$$

where $w_k = \frac{1}{10000^{2k/d}}$ as indicated by Vaswani et al. [50].

The temporal encoder at the t^{th} graph snapshot takes a sequence of node representations at the all graph snapshots up to t and each representation is updated as:

$$\hat{s}_i^{t'} = s_i^{t'} + p_i^{t'} \tag{11}$$

where $1 \le t' \le t$.

Given a node v_i and the input sequence $\left\{\hat{s}_i^1, \hat{s}_i^2, ..., \hat{s}_i^T\right\}$ across T snapshots produced by Equation 11, we denote the packed representation of input with dimensionality d by matrix $\hat{S}_i \in \mathbb{R}^{T\times d}$ and output with dimensionality d' of node v_i by $Z_i \in \mathbb{R}^{T\times d'}$. The input is first transformed into queries $Q_i = \hat{S}_i W_q$, keys $K_i = \hat{S}_i W_k$ and values $V_i = \hat{S}_i W_v$ by projection matrices $W_q \in \mathbb{R}^{d\times d'}$, $W_k \in \mathbb{R}^{d\times d'}$, and $W_v \in \mathbb{R}^{d\times d'}$. Then, the temporal-level attention model is defined as:

$$\hat{Z}_i = softmax(\frac{Q_i K_i^{\intercal}}{\sqrt{d_2}} + \mathbf{M}).V_i \tag{12}$$

where $\mathbf{M} \in \mathbb{R}^{T\times T}$ is a mask matrix so that the output only attends to historical representations up to the t^{th} graph snapshot.

$$\mathbf{M} = \begin{cases} 0 & \text{if } i \le j \\ -\infty & \text{otherwise} \end{cases} \tag{13}$$

Given the multi-head attention, the final representation Z_i of node v_i with k' heads is computed as:

$$Z_i = \|_{j=1}^{k'} \hat{Z}_i^j \tag{14}$$

4.3 Optimization

For training the overall model, we formulate the training procedure as supervised learning. The details on how we obtain the labels for training is described in Sections 5.4 and 5.5. To optimize our model parameters, we leverage two loss functions, namely Binary Cross Entropy ($\mathcal{L}_{BCE}$) and Kullback-Leibler divergence ($\mathcal{L}_{KL}$). The loss $\mathcal{L}$ in our work is computed as follows:

$$\mathcal{L} = \mathcal{L}_{BCE} + \mathcal{L}_{KL} \tag{15}$$

The cross entropy loss for supervised learning is:

$$\mathcal{L}_{BCE} = -\sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij} log(\sigma(q'_{ij})) + (1 - y_{ij}) log(1 - \sigma(q'_{ij})) \tag{16}$$

where L is the number of labels, σ is the sigmoid function and $q' \in \mathbb{R}^{N \times L}$ is the distribution of the linear transformation of Z.

The distribution between the true label and the predicted probabilities to estimated by the KL divergence:

$$\mathcal{L}_{KL}(p \parallel q) = \sum_{i} p(i) log \frac{p(i)}{q(i)} \tag{17}$$

where p is the probability distribution of the target label and $q = softmax(Z)$ is the distribution of output. BCE loss is used to explicitly capture whether the model is able to correctly predict positive labels, while the goodness of the model can be reflected implicitly by measuring the correlation of the cross-label distribution with KL divergence. Therefore, by jointly learning two loss functions, the model is trained to ensure the similar embedding features among the nodes sharing the same characteristics.

4.4 `DyHNet` Algorithm

In this section, we will outline the training process of `DyHNet` and discuss its computational complexity.

The Algorithm. The pseudocode for `DyHNet` is shown in Algorithm 1. Before the training stage, the metapath-aware subgraph encoder ($\mathcal{SGE}$) performs the first phase for anchor node (Line 3) and anchor subgraph (Line 4) sampling, respectively. At the beginning of the training stage, the local graph encoder ($\mathcal{LGE}$) computes the local semantic node embeddings at each timestamp in Line 9. After this, the metapath-aware subgraph encoder ($\mathcal{SGE}$) continues performing the next two phases, anchor subgraph encoder (Line 11) and subgraph encoder (Line 13-14) to compute the representations for the structure subgraphs. Once the representaions are obtained, the global semantic node embeddings is calculated in Line 15 by the structural encoder ($\mathcal{SE}$). The final embeddings containing both local and global semantics are computed in Line

Algorithm 1: The DyHNet algorithm.

```
Input:
  • A dynamic heterogeneous graph $G$ represented by a sequence of graph snapshots $G = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$.
  • Parameters: Random walk length $p$, Number of anchor subgraphs $k$, Number of sampled structure subgraphs $m$, Number of attention heads $k'$.
Output: Node embeddings $Z = \{z^1, z^2, \ldots, z^T\}$
1  begin
2      for each timestamp t from 1 to T do
3          $\mathcal{AN}^t \leftarrow$ Sample a set of $k$ seed nodes in $\mathcal{G}^t$;
4          $\mathcal{AS}^t \leftarrow$ Construct a set of $k$ anchor subgraphs in $\mathcal{G}^t$ following
             random walker strategy in Equation 3;
5      end
6      for each iteration do
7          for each timestamp t from 1 to T do
               /* capture local semantics */
8              foreach node $v_i$ in $\mathcal{G}^t$ do
9                  $l_i^t \leftarrow$ Obtain the local semantic node embeddings by Eq.
                     1 and 2;
10             end
               /* capture global semantics */
11             $\{a_1^t, a_2^t, \ldots, a_k^t\} \leftarrow$ Generate embedding for each anchor
                 subgraph in $\mathcal{AS}^t$ by Equation 4;
12             foreach node $v_i$ in $\mathcal{G}^t$ do
13                 $\{\mathcal{G}'^t_j\}_{j=1}^m \leftarrow$ Randomly sample $m$ struture subgraphs of
                     $\mathcal{G}^t$ which $v_i$ belongs to;
14                 $\{g_j^t\}_{j=1}^m \leftarrow$ Compute structure subgraph representations
                     by Eq. 5 and 6;
15                 $h_i^t \leftarrow$ Obtain the global semantic node embedding by
                     Eq. 7 and 8;
16             end
               /* integrate local and global semantics */
17             $S^t \leftarrow$ Obtain the fused semantic node embeddings by Eq. 9;
18         end
           /* capture graph dynamics */
19         $Z \leftarrow$ Obtain the node embeddings by Equation 14;
20         Optimize the loss function $\mathcal{L}$ in Equation 15;
21     end
22 end
```

17. After the node embeddings in all consecutive snapshots are obtained, we leverage a temporal multiple head self-attention mechanism to compute the final node embeddings (Line 19). The training stage will optimize the loss

function in Line 20 until the maximum number of iterations is reached or the early stopping criteria is satisfied.

Complexity Analysis. The complexity of `DyHNet` can be computed based on its three main components .

Complexity of $\mathcal{LGE}$. This component adopts a GNN which is based on message passing mechanisms to extract the local node representations. Therefore, the computational complexity of one layer of this encoder is $\mathcal{O}(N\overline{deg})$, where N is the total number of nodes and $\overline{deg}$ is the average degree of the node in the input graph.

Complexity of $\mathcal{GGE}$. This is a composite mapping of two separate encoders, the subgraph encoder ($\mathcal{SGE}$) and the structural encoder ($\mathcal{SE}$). At each snapshot, the computation for each node embedding is the summarization of the representations of the sampled subgraphs performed by $\mathcal{SE}$. Further, $\mathcal{SGE}$ obtains the representation of each structure subgraph by summarizing the similarity between the structure of it and all the anchor subgraphs.

The $\mathcal{SGE}$ involves three phases, however, the first phase which is for the anchor subgraph selection can be computed before the training step. Therefore, the computation of this component during training is mainly because of the last two phases, namely the anchor subgraph encoder and the subgraph encoder. The anchor subgraph encoder is constructed based on an LSTM, hence, its complexity is $\mathcal{O}(Np)$ where p is the length of the metapath random walks [51]. The subgraph encoder requires the similarity computation between the structural information of the subgraph and the anchor subgraphs (Equation 5). The complexity of this computation is proportional to the number of the structure subgraphs $|\{\mathcal{G}'\}|$ and the sampled anchor subgraphs k, which is $\mathcal{O}(|\{\mathcal{G}'\}|k)$. Further, to compute structural similarity, we adopt the DTW function as mentioned in Equation 5, whose complexity is proportional to the size of the two structure and anchor subgraphs. The anchor subgraphs are generated via fixed-length random walks as explained in Equation 3. The size of the subgraphs as well as the anchor subgraphs are relatively small compared to the total number of node in the graph, so they are constants. Besides, the similarity computation can be computed before the training phase and the operator can be parallelized. In summary, the total complexity of the $\mathcal{SGE}$ is $\mathcal{O}(Np+|\{\mathcal{G}'\}|k)$ in the worst case when we have to compute the representations for all the structure subgraphs.

The $\mathcal{SE}$, for each node in the graph, computes the node embeddings based on the weighted average of the embeddings of the respective structure subgraphs. We employ an additive attention mechanism to calculate the attention weights. The computation for each node representation only requires the element-wise product between a global query representation and the representation of the sampled structure subgraphs it belongs to. It then uses a linear transformation to scale the attention scores. In this way, the computational complexity of this component is linear to the total number of nodes and the maximum number of structure subgraphs m in Equation 8 used to summarize the global information for the node representation, which is $\mathcal{O}(Nm)$. We note that the computation of the representation for all nodes in the network can be parallelized.

Complexity of $\mathcal{TE}$. This encoder is built upon the masked multi-head self-attention operation over the historical graph snapshots which is parallelizable by different timestamps [52]. From Equation 12, the representation for each head is computed by the dot-product between the input representation pairs at each point of time up to the current timestamp; hence, the complexity is quadratic to the number of timestamps T in the graph which is *constant* and can be safely ignored in the complexity analysis. In summary, the complexity of the temporal self-attention mechanism can be expressed as $\mathcal{O}(Nk'T^2)$ where k' is the number of attention heads.

We also explore the affects of the values k, m and k' on the performance of our model in Section 5.7 and the results show that it is not necessary to set those hyper-parameters to a high value for a reasonable performance. Therefore, those can be considered as constants independent of N.

Summary. We conclude that the computation of `DyHNet` is linear to the number of nodes in the graph.

5 Experiments

In this section, we present our experiments in terms of dataset, setup and performance compared to the state-of-the-art to demonstrate the efficacy of `DyHNet` for dynamic heterogeneous network representation learning. More specifically, the experiments aim to address the following research questions:

- **RQ1**: How does `DyHNet` perform in the link prediction task to predict the future evolution of a heterogenous information network?
- **RQ2**: How does `DyHNet` perform in the node classification task to evaluate if `DyHNet` is able to generate proper dynamic node embeddings?
- **RQ3**: What is the impact of the major components of `DyHNet`, i.e., $\mathcal{LGE}$ (capturing Local Semantics), $\mathcal{GGE}$ (capturing Global Semantics) and $\mathcal{TE}$ (capturing Graph Dynamics) on its overall performance?
- **RQ4**: Is the performance of `DyHNet` sensitive to hyper-parameter setting?

5.1 Datasets

To evaluate the performance of `DyHNet`, we consider two tasks: (1) node classification; and, (2) link prediction. In each task, we adopt two widely used dynamic heterogeneous network datasets from various domains.

For the node classification task, we require datasets that include node labels. The two datasets that are widely used in the node classification task [53] and consist of appropriate node labels include Yelp and DBLP datasets: **(1) Yelp** is a social media dataset provided by Yelp Challenge 2022. Similar to [54], we extract information related to restaurants of three sub-categories: *American (New) Food*, *Fast Food* and *Sushi Bars*. After extracting, the dataset contains 2,693 business of 968 American Food, 714 Fast Food and 1,011 Sushi

Table 2 Statistics of datasets used in our experiments. Meta-paths are defined based on node types in the second column.

Dataset	# nodes	# edges	Time span	# labels	meta-paths
IMDB	# movie: 4,178 # director: 2,079 # actor: 5,253 # genre: 3	# movie-director: 4,178 # movie-actor: 12,530 # movie-genre: 4,178	1916 - 2016	3	movie-director-movie, movie-actor-movie director-movie-actor-movie-director actor-movie-director-movie-actor actor-movie-genre-movie-actor director-movie-actor-movie-director director-movie-genre-movie-director
AMiner	# author: 1,840 # paper: 10,674 # term: 2000 # venue: 21	# paper-author: 26,405 # paper-term: 70,809 # paper-venue: 10,674	1972 - 2016	5	author-paper-author paper-term-paper author-paper-term-paper-author author-paper-venue-paper-author
Yelp	# business: 2,693 # user: 2,011 # review: 30,593 # location: 15 # star: 5	# business-review: 30,593 # business-location: 14,196 # review-user: 30,593 # review-star: 30,593	2015 - 2021	3	business-location-business user-review-star-review-user user-review-business-review-user business-review-user-review-business business-review-star-review-business
DBLP	# author: 14,475 # paper: 14,376 # term: 8,920 # venue: 20	# paper-author: 41,794 # paper-term: 114,624 # paper-venue: 14,376	1969 - 2020	4	author-paper-author paper-term-paper author-paper-term-paper-author author-paper-venue-paper-author

Bars, $2,011$ users, $30,0593$ reviews, 15 locations and 5 stars. The time span is from 2015 to 2021. **(2) DBLP** We use the dataset published by [55, 56] that contains $14,475$ authors, $14,476$ papers, $8,920$ terms and 20 venues. There are $4,057$ authors labelled with their main research area and they are categorized into four areas: *Database* ($1,197$ authors), *Data Mining* (745 authors), *Machine Learning* ($1,109$ authors) and *Information Retrieval* ($1,006$ authors). To obtain the temporal information, we crawl the published year of the paper in DBLP and Google Scholar websites. The time span is from 1969 to 2020.

For the link prediction task and in order to show that the performance of our proposed approach is consistent across a range of datasets, we adopt two additional datasets that are often used for the link prediction task [18, 57], namely the IMDB and AMiner datasets as follows: **(1) IMDB** is an online database about movies and television programs, including information such as cast, production crew, and plot summaries. We use a subset of IMDB containing $4,178$ movies, $2,081$ directors, and $5,257$ actors. Movies are labelled as one of three classes (*Action* ($1,094$), *Comedy* ($1,562$), and *Drama* ($1,522$)) based on their genre information. In this dataset, each structure subgraph is formed by a single movie node and its connected nodes (i.e., directors, actors and genres). **(2) AMiner** is a bibliographic dataset in Computer Science. We only keep the historical records of the authors who have papers published in the top-5 most popular database venues: *KDD*, *VLDB*, *CIKM*, *ICDE* and *SIGMOD*. This subset contains $10,674$ papers, $1,840$ authors, $2,000$ terms, and 21 venues. The number of papers published in KDD, VLDB, CIKM, ICDE and SIGMOD is $1,269$, $1,212$, 970, 949 and 943 respectively. The time span is from 1972 to 2016. The structure subgraphs in this dataset are formed similar to the DBLP dataset.

We note that while IMDB and AMiner datasets cannot be used for the node classification task as they do not have node labels; however, both Yelp and DBLP dataset can be used for link prediction. However, due to space limitation, we report the results for link prediction on Yelp and DBLP on our online Github repository. The statistics of the real-world datasets used in our experiments are summarized in Table 2.

5.2 Baselines

We compare our proposed method with state-of-the-art dynamic *homogeneous* network representation and dynamic *heterogeneous* network representation learning methods.

The selected dynamic *homogeneous* network representation learning methods include: **(1)** `Dyngraph2vec` (with two variants: `DynAE`, `DynAERNN`) [23]. DynAE utilizes the fully connected layers and DynAERNN uses the recurrent layers (LSTM [51]) to reconstruct the current adjacent matrices from historical adjacent matrices through an encoder-decoder architecture. **(2)** `DynGEM` [40]. This method is the extension of SDNE [29] for dynamic graphs. The model leverages deep auto-encoders for incrementally learning dynamic network embeddings. **(3)** `DySAT` [21]. The method uses the scaled dot-product form of an attention layer to capture the dynamics of the sequences of networks. **(4)** `VGRNN` [58]. This approach integrates a variational auto-encoder at each timestamp which is later fed into the recurrent layer to take into account the structural evolution of the dynamic network. **(5)** `EvolveGCN` [22]. This technique is a dynamic variant of the GCN method, which is composed of RNN layers to model the evolution of GCN parameters at each timestamp. **(6)** `CTGCN-C` [59]. This method first employs GCN in k-cores to extract the node embedding at each timestamp, then utilizes an RNN to model the temporal dependency of node representations between different timestamps.

The set of baselines for dynamic *heterogeneous* network representation learning includes: **(7)** `DHNE` [25]. This method performs metapath-based random walk between historical snapshots and the current snapshot and benefits from a dynamic heterogeneous skip-gram model to capture representations of nodes. **(8)** `DyHATR` [24]. The technique uses hierarchical attention to learn heterogeneous information and incorporates RNNs with temporal attention to capture evolutionary patterns between different snapshots.

5.3 Experimental Setup

For each baseline, we adopt the implementation provided by the `CTGCN` library [59], and retain the default hyper-parameters as well as the model architecture in this implementation, which is equivalent to the recommended hyper-parameters by the authors of each method. More specifically, we set the learning rate to 10^{-3}, the walk length to 20, the number of negative samples to 20. We train GNNs for 100 epochs and apply early stopping with a patience of 30.

In our proposed method (`DyHNet`), we set the length of random walk to 15, number of layers of Subgraph Encoder to 2, the hidden dimension of local graph encoder $\mathcal{LGE}$ to 128, the hidden dimension of global graph encoder $\mathcal{GGE}$ to 128, and the number of attention heads in temporal encoder $\mathcal{TE}$ is set to

4. The learning rate is tuned by grid search within the range $[10^{-3}, 10^{-2}]$. We will also report hyper-parameter sensitivity analysis for our proposed approach later in the experiments. For fair comparison, the dimensions of the final representation is set to 128 for all methods. For optimization, we use the Adam optimizer [60]. The training, validation and test sets are the same for all methods.

5.4 RQ1. The Link Prediction Task

Given a series of graphs, the link prediction task aims at using the representation in the current time step t to predict the existence of an edge in the next time step $(t + 1)$. We conduct experiments on the IMDB and AMiner datasets to compare the performance of different methods on the link prediction task, which predicts director-genre and actor-genre links in the IMDB dataset and author-venue link in the AMiner dataset. It is worth noting that each author (or director/actor) can connect to one or many venue (or genre) nodes. Further, we assume that the latent space can be inferred from the label representations. Therefore, in both datasets, the labels for each node are the labels of all structure subgraphs it belongs to. In the IMDB dataset, each movie is considered to construct a structure subgraph, and the label of each subgraph is the genre of the movie. In the AMiner dataset, each paper and its connected nodes form a structure subgraph and each subgraph is labelled by the venue that it appears in. As a result, the training procedure for link prediction task is considered as a multi-label classification problem.

Once we generate the representation for all the nodes, we use the Hadamard operator on the representation of the node pairs to compute edge feature vectors. We train a Logistic Regression (LR) classifier to distinguish between positive and negative edges. We use all nodes in the last snapshot for testing, the remaining is split into 80%/20% for training and validation. We employ three popular metrics, *Micro F1* (F1), *Micro Recall* and *Micro Precision* as the evaluation metrics for this task as recommended in [61, 62].

The results of our experiments on the IMDB and AMiner datasets are reported in Tables 3 and 4, respectively, for top-k predicted neighbor nodes. We report the top-k when $k = 1, 2$. We make several observations based on the results: **(a)** our proposed `DyHNet` method has been able to consistently show better performance compared to all baselines and on both datasets (for $K = 1$ and 2). The performance improvements are 13% and 15% on F1-score for the IMDB and AMiner datasets, respectively, over the best performing baseline method. **(b)** Our proposed approach shows stable performance over two different datasets; however, the baseline methods are not comparatively as robust. For instance, on the IMDB dataset, `CTGCN-C` is the runner-up baseline at both $k = 1$ and $k = 2$, while this method is not among the top-3 best baselines on the AMiner dataset. In contrast, `DyHATR` is the runner-up for $k = 1$ on the AMiner dataset but not competitive on the IMDB dataset. Based on the results, we found that compared to the state-of-the-art baselines on both datasets, our proposed method (i.e., `DyHNet`) is the most robust method. **(c)**

Table 3 Results on the IMDB dataset for link prediction task. *Bold* is best performance and *underline* is runner-up.

	@k=1			@k=2		
Method	**F1**	**Recall**	**Precision**	**F1**	**Recall**	**Precision**
DynAE	0.3705	0.3344	0.4154	0.5141	0.6709	0.4167
DynAERNN	0.3647	0.3291	0.4089	0.5100	0.6656	0.4134
DynGEM	0.4367	0.3941	0.4896	0.5494	0.7170	0.4453
DySAT	0.3717	0.3354	0.4167	0.5052	0.6593	0.4095
VGRNN	0.3949	0.3564	0.4427	0.5084	0.6635	0.4121
EvolveGCN	0.3717	0.3354	0.4167	0.5133	0.6698	0.4160
CTGCN-C	<u>0.4448</u>	<u>0.4015</u>	<u>0.4987</u>	<u>0.5590</u>	<u>0.7296</u>	<u>0.4531</u>
DHNE	0.3577	0.3229	0.4010	0.5116	0.6677	0.4147
DyHATR	0.3438	0.3103	0.3854	0.4956	0.6468	0.4017
DyHNet	**0.5029**	**0.4539**	**0.5638**	**0.5831**	**0.7610**	**0.4727**
Δ% over best	+13.06%	+13.05%	+13.05%	+4.31%	+4.30%	+4.33%

Table 4 Results on the AMiner dataset for link prediction task. *Bold* is best performance and *underline* is runner-up.

	@k=1			@k=2		
Method	**F1**	**Recall**	**Precision**	**F1**	**Recall**	**Precision**
DynAE	0.1848	0.1568	0.2250	0.2708	0.3241	0.2325
DynAERNN	0.1972	0.1673	0.2401	0.2708	0.3241	0.2325
DynGEM	0.2609	0.2213	0.3176	0.3786	<u>0.4532</u>	0.3251
DySAT	0.2624	0.2227	0.3195	0.3368	0.4032	0.2892
VGRNN	0.2686	0.2279	0.3270	0.3060	0.3663	0.2628
EvolveGCN	0.2034	0.1726	0.2476	0.2829	0.3386	0.2429
CTGCN-C	0.2360	0.2003	0.2873	0.3247	0.3887	0.2788
DHNE	0.2407	0.2042	0.2930	0.3434	0.4111	0.2949
DyHATR	<u>0.2866</u>	<u>0.2431</u>	<u>0.3491</u>	<u>0.3788</u>	0.4531	<u>0.3254</u>
DyHNet	**0.3307**	**0.2806**	**0.4026**	**0.4260**	**0.5099**	**0.3658**
Δ% over best	+15.39%	+15.43%	+15.33%	+12.46%	+12.51%	+12.41%

The baselines that are specifically designed to address *dynamic heterogeneous* networks (gray rows) do not have competitive performance not only to `DyHNet` but also when compared to baselines that do not take node heterogeneity into account on the IMDB dataset. On the AMiner dataset, `DyHATR` is the runner-up method but its performance is essentially the same as `DynGEM` (which is a method for homogeneous networks) at $k = 2$. This is an important observation as it shows that considering node heterogeneity does not necessarily lead to improved downstream performance since interactions between heterogeneous node types need to be carefully taken into account, which may otherwise lead to incorrect or missing associations between node types, which can impact performance. For instance, `DyHATR`, unlike `DyHNet`, reduces the heterogeneous graph into homogeneous subgraphs. The lower performance of `DyHATR` could be attributed to such a reduction that can lead to missing associations between different heterogeneous node types, which is clearly addressed in our `DyHNet` method.

5.5 RQ2. The Node Classification Task

The node classification task aims to predict the label corresponding to each node. We conduct experiments on the Yelp and DBLP datasets to compare the performance of different models on the node classification task. Since there

Table 5 Results on the Yelp and DBLP datasets for node classification. *Bold* is best performance and *underline* is runner-up.

Datasets	Metrics	Train %	DynAE	DynAERNN	DynGEM	DySAT	EvolveGCN	CTGCN-C	DHNE	DyHATR	DyHNet	Δ% over the best
Yelp	AUC	20%	0.5319	0.5040	0.5756	0.4947	0.6273	<u>0.6368</u>	0.5001	0.6037	**0.6965**	+9.38%
		40%	0.5498	0.5109	0.6040	0.5076	0.6245	<u>0.6690</u>	0.4986	0.6201	**0.7021**	+4.94%
		60%	0.5503	0.5089	0.6043	0.5043	0.6304	<u>0.6697</u>	0.4890	0.6142	**0.6963**	+3.97%
		80%	0.5643	0.5205	0.6097	0.4970	0.6311	<u>0.6673</u>	0.4898	0.6225	**0.7147**	+7.11%
	Macro F1	20%	0.1741	0.2845	0.3455	0.2589	0.1741	<u>0.4414</u>	0.1759	0.3983	**0.4871**	+10.37%
		40%	0.1791	0.2215	0.4088	0.2672	0.1791	<u>0.4854</u>	0.1791	0.4056	**0.5035**	+3.74%
		60%	0.1771	0.1771	0.3996	0.2783	0.1771	<u>0.4671</u>	0.1771	0.4043	**0.4917**	+5.25%
		80%	0.1817	0.2878	0.4054	0.2657	0.1738	<u>0.4579</u>	0.1817	0.4194	**0.5123**	+11.87%
	Micro F1	20%	0.3534	0.3687	0.3956	0.3442	0.3534	<u>0.4351</u>	0.3544	0.4082	**0.4882**	+12.21%
		40%	0.3673	0.3704	0.4224	0.3587	0.3673	<u>0.4774</u>	0.3673	0.4088	**0.5090**	+6.61%
		60%	0.3618	0.3618	0.4128	0.3673	0.3618	<u>0.4592</u>	0.3618	0.4091	**0.4944**	+7.68%
		80%	0.3748	0.3748	0.4119	0.3636	0.3525	<u>0.4527</u>	0.3748	0.4193	**0.5121**	+13.11%
DBLP	AUC	20%	0.5828	0.5880	0.6263	0.5030	0.5056	<u>0.7152</u>	0.4895	0.6301	**0.7304**	+2.14%
		40%	0.5860	0.5919	0.6347	0.5078	0.5002	<u>0.7387</u>	0.4924	0.6393	**0.7565**	+2.42%
		60%	0.5875	0.5942	0.6375	0.5130	0.5193	<u>0.7481</u>	0.5072	0.6410	**0.7797**	+4.22%
		80%	0.5799	0.5906	0.6294	0.4996	0.5044	<u>0.7529</u>	0.5067	0.6366	**0.8098**	+7.55%
	Macro F1	20%	0.2348	0.2348	0.2966	0.1849	0.1362	<u>0.4341</u>	0.1023	0.2736	**0.4488**	+3.39%
		40%	0.2547	0.2661	0.3482	0.1925	0.1357	<u>0.4693</u>	0.1009	0.2913	**0.5107**	+8.82%
		60%	0.2499	0.2499	0.3400	0.1874	0.1348	<u>0.4780</u>	0.1704	0.2734	**0.5362**	+12.16%
		80%	0.2322	0.2322	0.3214	0.1793	0.1427	<u>0.4895</u>	0.1252	0.2528	**0.5785**	+18.16%
	Micro F1	20%	0.3012	0.3012	0.3218	0.2907	0.2849	<u>0.4469</u>	0.2571	0.3348	**0.4632**	+3.65%
		40%	0.3194	0.3175	0.3614	0.2965	0.2895	<u>0.4842</u>	0.2527	0.3561	**0.5212**	+7.63%
		60%	0.3165	0.3165	0.3510	0.2931	0.2888	<u>0.4951</u>	0.2802	0.3393	**0.5468**	+10.45%
		80%	0.2945	0.2945	0.3321	0.2878	0.3149	<u>0.5031</u>	0.2270	0.3161	**0.5916**	+17.60%

is exactly one class corresponding to each labelled node, the learning objective is a single-label classification problem. Similar to the settings for the DBLP dataset in Section 5.4, the structure subgraphs in the DBLP dataset are also divided by the published academic papers. Meanwhile, in Yelp, we construct each review and its connected nodes (i.e., business, location, user and rating star) as a structure subgraph.

We feed the embedding vectors of labelled nodes (i.e., businesses in Yelp and authors in DBLP) generated by each learning model to a linear regression classifier with varying training (training/validation) proportions (i.e., 20% (10%/10%), 40% (30%/10%), 60% (50%/10%) and 80% (70%/10%)), the remaining is used for testing. Again, the train/validation/test splits are also the same across embedding methods. We use *Area Under The Curve* (AUC), *Macro F1* and *Micro F1* as the evaluation metrics. Because of the memory size limitations that we faced with the `VGRNN` model, we were not able to report the results of this method for the node classification task.

The results of our experiments on Yelp and DBLP datasets are reported in Table 5, based on which we make the following observations: **(a)** our proposed method (i.e., `DyHNet`) consistently outperforms all the baselines. For the train-test split of 80%-20%, the Micro F1 scores archived by our model increase by 13% and 17% compared to the runner-up model (i.e., `CTGCN-C`) on the Yelp and DBLP datasets respectively. **(b)** Our model is more robust in the presence of class imbalance. The significant performance gap between Macro F1 and Micro F1 of some models (e.g., `EvolveGCN` and `DHNE` on the Yelp dataset) shows that such methods tend to favor the majority class when dealing with imbalanced dataset, which is an undesirable outcome. **(c)** The performance between `CTGCN-C` and `DyHATR`, which are the best baseline among the homogeneous and heterogeneous network representation learning methods respectively, are comparable. From this observation, we hypothesize that in the node classification task, the network structure seems to be more important than the historical information. `DyHATR` is more capable to preserve the local connective proximity than the global structural information. The reason is it makes use of the attention mechanism to aggregate information only from the neighboring nodes and overlooks the subgraph structure. Meanwhile, `CTGCN-C` decomposes the graph into subgraphs in order to retain the global structure. Our proposed approach is able to achieve better performance which demonstrates the effectiveness of `DyHNet` in preserving graph structural information.

5.6 RQ3. Ablation Study

In this section, we analyze the impact of the three main components of our proposed method (`DyHNet`), namely the local graph encoder ($\mathcal{LGE}$), the global graph encoder ($\mathcal{GGE}$) and the temporal encoder ($\mathcal{TE}$). To do so, we compare `DyHNet` to its six variants: (1) `DyHNet-NoL` (which removes $\mathcal{LGE}$), (2) `DyHNet-NoG` (which removes $\mathcal{GGE}$), (3) `DyHNet-NoT` (which removes $\mathcal{TE}$), (4) `DyHNet-NoLG` (which removes $\mathcal{LGE}$ and $\mathcal{GGE}$), (5) `DyHNet-NoGT` (which removes $\mathcal{GGE}$ and $\mathcal{TE}$) and (6) `DyHNet-NoLT` (which removes $\mathcal{LGE}$ and $\mathcal{TE}$).

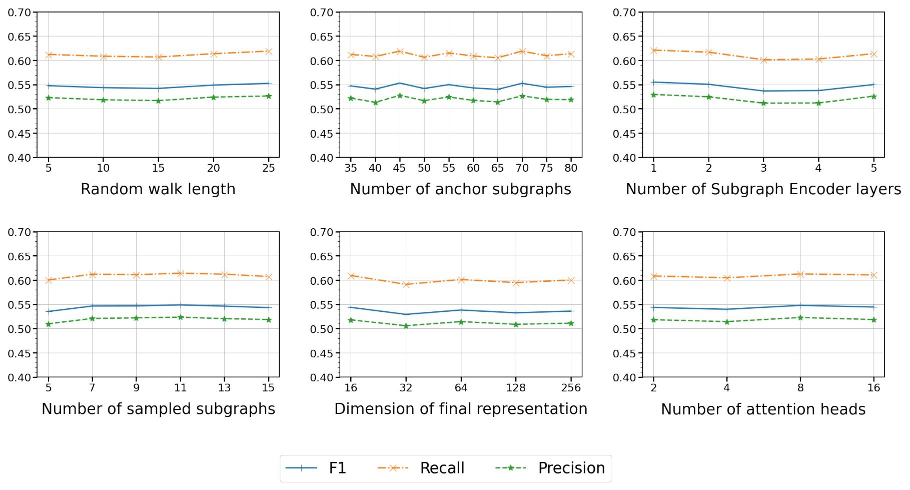
Table 6 Ablation on the IMDB dataset for link prediction.

	@k=1			@k=2		
Method	**F1**	**Recall**	**Precision**	**F1**	**Recall**	**Precision**
DyHNet	**0.5029**	**0.4539**	**0.5638**	**0.5831**	**0.7610**	**0.4727**
DyHNet-NoL	0.4878	0.4403	0.5469	0.5783	0.7547	0.4688
DyHNet-NoG	0.3728	0.3365	0.4180	0.5213	0.6803	0.4225
DyHNet-NoT	0.4321	0.3899	0.4844	0.5574	0.7275	0.4518
DyHNet-NoLG	0.3659	0.3302	0.4102	0.5165	0.6740	0.4186
DyHNet-NoGT	0.3705	0.3344	0.4154	0.5084	0.6635	0.4121
DyHNet-NoLT	0.4379	0.3952	0.4909	0.5614	0.7327	0.4551

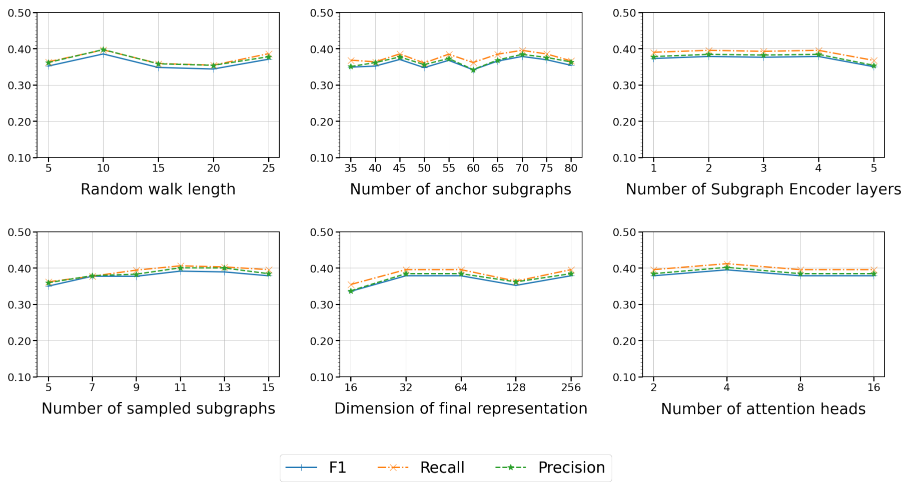
The performances of all models over the four datasets are shown in Tables 6 and 7. For the IMDB and AMiner datasets, the metric is computed by taking the average of Micro F1 score at $k = 1$ and $k = 2$. For the other datasets, the metric is computed on the dataset in which the training/testing ratio is 80%/20%.

Tables 6 and 7 report the performance of the proposed method and its variants when removing different components for the link prediction task. As seen in Tables 6 and 7, the proposed `DyHNet` method, in the presence of the local, global, and temporal encoders performs the best. Results achieved for the IMDB dataset, presented in Table 6, suggest that among the different components, the local graph encoder ($\mathcal{LGE}$) has the least impact as the performance drops slightly when this encoder is removed from the model. On the other hand, the global graph encoder ($\mathcal{GGE}$) has a significantly stronger impact on the performance. On all datasets, the performance dropped dramatically when the global graph encoder ($\mathcal{GGE}$) is removed from the model. This trend can also be observed when the other encoders are removed at the same time as the Global Encoder. For instance, comparing the `DyHNet-NoL` variation where only the Local Encoder is removed with (`DyHNet-NoLG`) where the Global Encoder is also removed, one can observe a significant drop in performance showing the strong impact of the global encoder. We also observe that the temporal encoder also has a noticeable impact on the overall performance of `DyHNet` and the degree o f this impact is significantly larger than that of the Local Encoder.

In the context of the AMiner dataset, we make similar observations to that of the IMDB dataset where the global graph encoder ($\mathcal{GGE}$) still has the highest impact, but it appears that the local graph encoder ($\mathcal{LGE}$) component contributes more noticeably to the link prediction task on the AMiner dataset in comparison to the IMDB dataset. This can be explained as other researchers have already reported the performance of graph encoders can vary depending on the characteristics of each dataset [22, 44]. Table 8 reports the performance of different variants of our method for node classification. Similar to Tables 6 and 7, the global graph encoder ($\mathcal{GGE}$) has the most impact on the overall performance. Following the Global Encoder as the most effective component on both Yelp and DBLP datasets, temporal encoder ($\mathcal{TE}$) is the second most effective component that can drastically change performance when removed. We note that the least effective component is the local graph encoder ($\mathcal{LGE}$) as



(a) IMDB



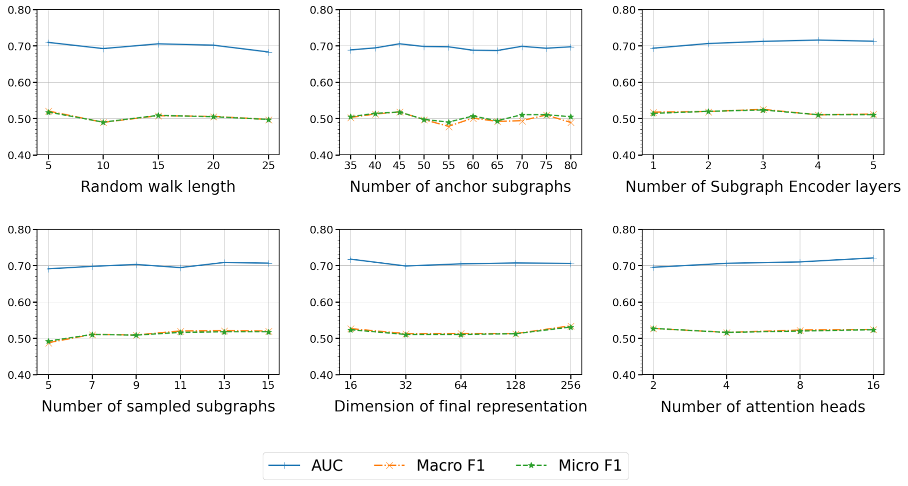
(b) AMiner

Fig. 3 Sensitivity for link prediction on IMDB and AMiner.

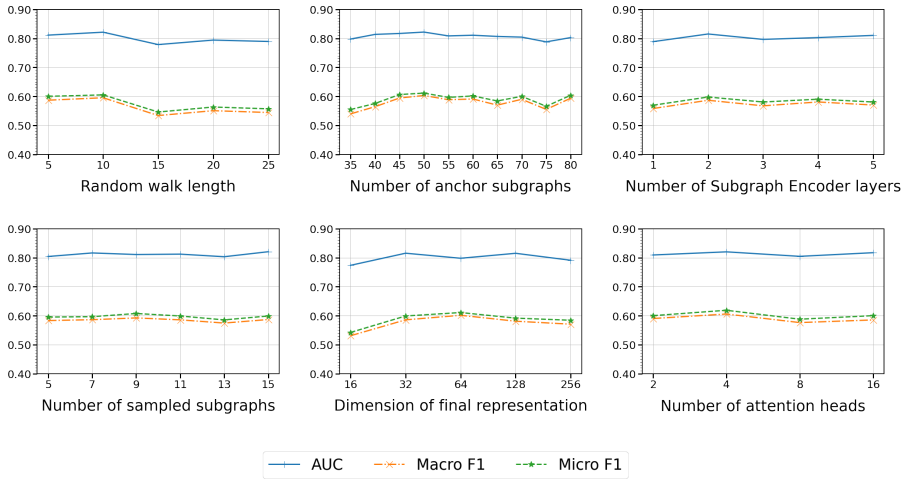
was also observed in the link prediction task. Again, the variation of `DyHNet` that includes all three components yields the best result for node classification.

5.7 RQ4. Sensitivity Analysis

We further investigate the sensitivity of the proposed approach to its hyper-parameters. The main hyper-parameters of `DyHNet` are the random walk length (p), the number of anchor subgraphs (k), the number of sampled subgraphs



(a) Yelp



(b) DBLP

Fig. 4 Sensitivity for node classification on Yelp and DBLP.

(m) in Equation 8, the number of global subgraph encoder layers, the number of attention heads (k') in $\mathcal{TE}$ and the dimension (d') of the final representation Z. For the link prediction task, we tested and reported the result for the IMDB and AMiner datasets. For node classification, we report the evaluation metrics with training ratio = 80% on the Yelp and DBLP datasets. Figures 3 and 4 report the impact of hyper-parameters on the link prediction and node classification tasks respectively.

Table 7 Ablation on the AMiner dataset for link prediction.

	@k=1			@k=2		
Method	**F1**	**Recall**	**Precision**	**F1**	**Recall**	**Precision**
DyHNet	**0.3307**	**0.2806**	**0.4026**	**0.4260**	**0.5099**	**0.3658**
DyHNet-NoL	0.2298	0.1950	0.2798	0.3654	0.4374	0.3138
DyHNet-NoG	0.1848	0.1568	0.2250	0.3654	0.4374	0.3138
DyHNet-NoT	0.3106	0.2635	0.3781	0.3786	0.4532	0.3251
DyHNet-NoLG	0.2360	0.2003	0.2873	0.3346	0.4005	0.2873
DyHNet-NoGT	0.1957	0.1660	0.2382	0.3731	0.4466	0.3204
DyHNet-NoLT	0.2158	0.1831	0.2628	0.3632	0.4348	0.3119

Table 8 Ablation on Yelp and DBLP for node classification.

		Yelp			DBLP		
method	**Train %**	**AUC**	**Macro F1**	**Micro F1**	**AUC**	**Macro F1**	**Micro F1**
DyHNet	20%	**0.6965**	**0.4871**	**0.4882**	**0.7304**	**0.4488**	**0.4632**
	40%	**0.7021**	**0.5035**	**0.5090**	**0.7872**	**0.5498**	**0.5627**
	60%	**0.6963**	**0.4917**	**0.4944**	**0.7943**	**0.5601**	**0.5732**
	80%	**0.7147**	**0.5123**	**0.5121**	**0.8098**	**0.5785**	**0.5916**
DyHNet-NoL	20%	0.6965	0.4789	0.4817	0.7192	0.4544	0.4632
	40%	0.7017	0.4834	0.4898	0.7843	0.5343	0.5461
	60%	0.7082	0.4806	0.4879	0.7974	0.5586	0.5670
	80%	0.7044	0.5043	0.5083	0.8079	0.5746	0.5904
DyHNet-NoG	20%	0.6054	0.1742	0.3536	0.6233	0.2947	0.3184
	40%	0.6145	0.1791	0.3673	0.6182	0.2356	0.3063
	60%	0.6017	0.1817	0.3748	0.6200	0.2363	0.3075
	80%	0.6241	0.4074	0.4082	0.6204	0.2363	0.3075
DyHNet-NoT	20%	0.6203	0.3881	0.4028	0.6582	0.3961	0.4007
	40%	0.6253	0.3814	0.4125	0.6841	0.4236	0.4304
	60%	0.6227	0.3912	0.4082	0.7073	0.4399	0.4526
	80%	0.6052	0.3960	0.4119	0.7345	0.4904	0.4982
DyHNet-NoLG	20%	0.5000	0.1742	0.3536	0.6231	0.3000	0.3144
	40%	0.5043	0.1817	0.3748	0.6180	0.2363	0.3075
	60%	0.5032	0.1817	0.3748	0.6192	0.2363	0.3075
	80%	0.5000	0.1817	0.3748	0.6186	0.2363	0.3075
DyHNet-NoGT	20%	0.5983	0.1742	0.3536	0.6010	0.1152	0.2994
	40%	0.5939	0.3826	0.3933	0.5775	0.1714	0.3235
	60%	0.5908	0.3979	0.4045	0.6072	0.3132	0.3260
	80%	0.5911	0.4055	0.4119	0.6122	0.3250	0.3383
DyHNet-NoLT	20%	0.6143	0.3726	0.4079	0.6635	0.3937	0.3988
	40%	0.6075	0.3689	0.3952	0.7089	0.4435	0.4588
	60%	0.6063	0.3721	0.3989	0.7145	0.4663	0.4723
	80%	0.6073	0.3812	0.4007	0.7346	0.4820	0.4883

As seen in Figure 3, the proposed model shows robustness against different hyper-parameters of the model. While changing any of the six hyper-parameters will only slightly change the performance of the overall model and the model performance remains stable regardless of the changes to the hyper-parameters indicating its robustness. Similarly and as shown in Figure 4, for the node classification task, the proposed method still shows robustness in light of changes to different model hyper-parameters again indicating that the proposed model is quite stable. We conclude the proposed model is stable to the choice of hyper-parameters regardless of the downstream task, i.e., node classification and link prediction.

We do note that the model can be configured to show its best performance by finding the best hyper-parameters, e.g., based on a held-out validation set. For example, in Figure 4 (a), for the Yelp dataset, performance peaks when the dimension of the final representation being is set to 256, or in Figure 4 (b), for the DBLP dataset, the configuration that uses four attention heads shows the best performance. However, increased performances are not significant.

6 Concluding Remarks

This objective of our work paper has been on learning neural representations for *dynamic heterogeneous networks.* We proposed `DyHNet`, which systematically integrates local node semantics, global network semantics, as well as longer-range temporal associations between network snapshots. Based on experiments on two distinct tasks, four different datasets and in comparison with several strong state-of-the-art baselines, we have shown that our proposed `DyHNet` method has been able to show consistently better and more robust performance compared to state-of-the-art methods.

We would like to point to some potential limitations of this work which we aim to address as future work:

1. **Online representation learning for streaming networks.** Our model requires that all the nodes in the network be present during training in order to learn their embeddings. There is a limited number of existing work that perform representation learning on network streams [63]. As future work, we intend to extend `DyHNet` to provide support for *online/continual learning*;
2. **Novel subgraph sampling strategy.** Our method relies on the semantics of the anchor subgraphs sampled by the metapath-based approach and the structure subgraphs selected randomly. Prior research recommends to utilize a range of subgraph sampling strategies to fully extract richer information [64], which we will further in our future work.

Declarations

- Funding: The research leading to these results received funding from NSERC
- Conflict of interest: The authors have no competing interests to declare that are relevant to the content of this article.
- Ethics approval: Not applicable
- Consent to participate: Not applicable
- Consent for publication: Not applicable
- Availability of data and materials: The data and other artifacts have been made available at https://github.com/hoangntc/DyHNet
- Code availability: The source code has been made available at https://github.com/hoangntc/DyHNet.
- Authors' contributions: Hoang Nguyen and Radin Hamidi Rad implemented the method and ran experiments. Fattane Zarrinkalam and Ebrahim Bagheri wrote and revised the manuscript.

References

[1] Lv, Q., Ding, M., Liu, Q., Chen, Y., Feng, W., He, S., Zhou, C., Jiang, J., Dong, Y., Tang, J.: Are we really making much progress?: Revisiting, benchmarking and refining heterogeneous graph neural networks. In: Zhu, F., Ooi, B.C., Miao, C. (eds.) KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021, pp. 1150–1160 (2021). https://doi.org/10.1145/3447548.3467350. https://doi.org/10.1145/3447548.3467350

[2] Doan, K.D., Manchanda, S., Mahapatra, S., Reddy, C.K.: Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In: Diaz, F., Shah, C., Suel, T., Castells, P., Jones, R., Sakai, T. (eds.) SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021, pp. 665–674 (2021). https://doi.org/10.1145/3404835.3462960. https://doi.org/10.1145/3404835.3462960

[3] Hamilton, W.L.: Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning **14**(3), 1–159 (2020)

[4] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Macskassy, S.A., Perlich, C., Leskovec, J., Wang, W., Ghani, R. (eds.) The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014, pp. 701–710 (2014). https://doi.org/10.1145/2623330.2623732. https://doi.org/10.1145/2623330.2623732

[5] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 855–864 (2016). https://doi.org/10.1145/2939672.2939754. https://doi.org/10.1145/2939672.2939754

[6] Cen, Y., Zou, X., Zhang, J., Yang, H., Zhou, J., Tang, J.: Representation learning for attributed multiplex heterogeneous network. In: Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., Karypis, G. (eds.) Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pp. 1358–1368 (2019). https://doi.org/10.1145/3292500.3330964. https://doi.org/10.1145/3292500.3330964

[7] Xu, Y.: An empirical study of locally updated large-scale information network embedding (LINE). PhD thesis, University of California, Los Angeles, USA (2017). http://www.escholarship.org/uc/item/2mp915ck

[8] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017). https://openreview.net/forum?id=SJU4ayYgl

[9] Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 1024–1034 (2017)

[10] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. CoRR **abs/1710.10903** (2017) 1710.10903

[11] Fu, X., Zhang, J., Meng, Z., King, I.: MAGNN: metapath aggregated graph neural network for heterogeneous graph embedding. In: Huang, Y., King, I., Liu, T., van Steen, M. (eds.) WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, pp. 2331–2341 (2020). https://doi.org/10.1145/3366423.3380297. https://doi.org/10.1145/3366423.3380297

[12] Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L. (eds.) The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019, pp. 2022–2032 (2019). https://doi.org/10.1145/3308558.3313562. https://doi.org/10.1145/3308558.3313562

[13] Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph transformer networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 11960–11970 (2019)

[14] Sun, Q., Li, J., Peng, H., Wu, J., Ning, Y., Yu, P.S., He, L.: SUGAR: subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In: Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L. (eds.) WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021, pp. 2081–2091 (2021). https://doi.org/10.1145/3442381.3449822. https://doi.org/10.1145/3442381.3449822

[15] Alsentzer, E., Finlayson, S.G., Li, M.M., Zitnik, M.: Subgraph neural networks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual

Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual (2020)

[16] Kazemi, S.M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P.: Representation learning for dynamic graphs: A survey. J. Mach. Learn. Res. **21**(70), 1–73 (2020)

[17] Kong, C., Li, H., Zhang, L., Zhu, H., Liu, T.: Link prediction on dynamic heterogeneous information networks. In: International Conference on Computational Data and Social Networks, pp. 339–350 (2019). Springer

[18] Milani Fard, A., Bagheri, E., Wang, K.: Relationship prediction in dynamic heterogeneous information networks. In: European Conference on Information Retrieval, pp. 19–34 (2019). Springer

[19] Jiang, Z., Gao, Z., Lan, J., Yang, H., Lu, Y., Liu, X.: Task-oriented genetic activation for large-scale complex heterogeneous graph embedding. In: Huang, Y., King, I., Liu, T., van Steen, M. (eds.) WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, pp. 1581–1591 (2020). https://doi.org/10.1145/3366423.3380230. https://doi.org/10.1145/3366423.3380230

[20] Qu, L., Zhu, H., Duan, Q., Shi, Y.: Continuous-time link prediction via temporal dependent graph neural network. In: Huang, Y., King, I., Liu, T., van Steen, M. (eds.) WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, pp. 3026–3032 (2020). https://doi.org/10.1145/3366423.3380073. https://doi.org/10.1145/3366423.3380073

[21] Sankar, A., Wu, Y., Gou, L., Zhang, W., Yang, H.: Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In: Caverlee, J., Hu, X.B., Lalmas, M., Wang, W. (eds.) WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020, pp. 519–527 (2020). https://doi.org/10.1145/3336191.3371845. https://doi.org/10.1145/3336191.3371845

[22] Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T.B., Leiserson, C.E.: Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pp. 5363–5370 (2020). https://aaai.org/ojs/index.php/AAAI/article/view/5984

[23] Goyal, P., Chhetri, S.R., Canedo, A.: dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowl. Based

Syst. **187** (2020). https://doi.org/10.1016/j.knosys.2019.06.024

[24] Xue, H., Yang, L., Jiang, W., Wei, Y., Hu, Y., Lin, Y.: Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal RNN. In: Hutter, F., Kersting, K., Lijffijt, J., Valera, I. (eds.) Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12457, pp. 282–298 (2020). https://doi.org/10.1007/978-3-030-67658-2_17. https://doi.org/10.1007/978-3-030-67658-2_17

[25] Yin, Y., Ji, L., Zhang, J., Pei, Y.: DHNE: network representation learning method for dynamic heterogeneous networks. IEEE Access **7**, 134782–134792 (2019). https://doi.org/10.1109/ACCESS.2019.2942221

[26] Bian, R., Koh, Y.S., Dobbie, G., Divoli, A.: Network embedding and change modeling in dynamic heterogeneous networks. In: Piwowarski, B., Chevalier, M., Gaussier, É., Maarek, Y., Nie, J., Scholer, F. (eds.) Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019, pp. 861–864 (2019). https://doi.org/10.1145/3331184.3331273. https://doi.org/10.1145/3331184.3331273

[27] Wu, J., He, J., Xu, J.: Demo-net: Degree-specific graph neural networks for node and graph classification. In: Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., Karypis, G. (eds.) Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pp. 406–415 (2019). https://doi.org/10.1145/3292500.3330950. https://doi.org/10.1145/3292500.3330950

[28] Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., Tang, J.: Graph random neural networks for semi-supervised learning on graphs. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual (2020)

[29] Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 1225–1234 (2016). https://doi.org/10.1145/2939672.2939753. https://doi.org/10.1145/2939672.2939753

[30] Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate:

Graph neural networks meet personalized pagerank. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019). https://openreview.net/forum?id=H1gL-2A9Ym

[31] Shi, C., Wang, X., Yu, P.S.: Heterogeneous Graph Representation Learning and Applications. Artificial Intelligence: Foundations, Theory, and Algorithms, (2022). https://doi.org/10.1007/978-981-16-6166-2. https://doi.org/10.1007/978-981-16-6166-2

[32] Tu, K., Cui, P., Wang, X., Wang, F., Zhu, W.: Structural deep embedding for hyper-networks. CoRR **abs/1711.10146** (2017) 1711.10146

[33] Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017, pp. 135–144 (2017). https://doi.org/10.1145/3097983.3098036. https://doi.org/10.1145/3097983.3098036

[34] Zhang, D., Yin, J., Zhu, X., Zhang, C.: Metagraph2vec: Complex semantic path augmented heterogeneous network embedding. CoRR **abs/1803.02533** (2018) 1803.02533

[35] Zhang, C., Song, D., Huang, C., Swami, A., Chawla, N.V.: Heterogeneous graph neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 793–803 (2019)

[36] Fang, Y., Zhao, X., Huang, P., Xiao, W., de Rijke, M.: M-HIN: complex embeddings for heterogeneous information networks via metagraphs. In: Piwowarski, B., Chevalier, M., Gaussier, É., Maarek, Y., Nie, J., Scholer, F. (eds.) Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019, pp. 913–916 (2019). https://doi.org/10.1145/3331184.3331281. https://doi.org/10.1145/3331184.3331281

[37] Wang, X., Zhang, M.: GLASS: GNN with labeling tricks for subgraph representation learning. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022 (2022). https://openreview.net/forum?id=XLxhEjKNbXj

[38] Yin, H., Zhang, M., Wang, Y., Wang, J., Li, P.: Algorithm and system co-design for efficient subgraph-based graph representation learning. CoRR **abs/2202.13538** (2022) 2202.13538

[39] Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 571–578 (2018). https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16572

[40] Goyal, P., Kamra, N., He, X., Liu, Y.: Dyngem: Deep embedding method for dynamic graphs. CoRR **abs/1805.11273** (2018) 1805.11273

[41] Goyal, P., Chhetri, S.R., Canedo, A.: dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems **187**, 104816 (2020)

[42] Yin, Y., Ji, L., Zhang, J., Pei, Y.: DHNE: network representation learning method for dynamic heterogeneous networks. IEEE Access **7**, 134782–134792 (2019). https://doi.org/10.1109/ACCESS.2019.2942221

[43] Wang, X., Lu, Y., Shi, C., Wang, R., Cui, P., Mou, S.: Dynamic heterogeneous information network embedding with meta-path based proximity. IEEE Trans. Knowl. Data Eng. **34**(3), 1117–1132 (2022). https://doi.org/10.1109/TKDE.2020.2993870

[44] Fang, Y., Zhao, X., Huang, P., Xiao, W., de Rijke, M.: Scalable representation learning for dynamic heterogeneous information networks via metagraphs. ACM Trans. Inf. Syst. **40**(4), 64–16427 (2022). https://doi.org/10.1145/3485189

[45] Zhang, Z., Huang, J., Tan, Q.: SR-HGAT: symmetric relations based heterogeneous graph attention network. IEEE Access **8**, 165631–165645 (2020). https://doi.org/10.1109/ACCESS.2020.3022664

[46] Fang, Y., Yang, Y., Zhang, W., Lin, X., Cao, X.: Effective and efficient community search over large heterogeneous information networks. Proc. VLDB Endow. **13**(6), 854–867 (2020). https://doi.org/10.14778/3380750.3380756

[47] Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019). https://openreview.net/forum?id=ryGs6iA5Km

[48] Zhang, W., Fang, Y., Liu, Z., Wu, M., Zhang, X.: mg2vec: Learning

relationship-preserving heterogeneous graph representations via metagraph embedding. IEEE Transactions on Knowledge and Data Engineering **34**(3), 1317–1329 (2022). https://doi.org/10.1109/TKDE.2020.2992500

[49] Wu, M., Pan, S., Du, L., Tsang, I., Zhu, X., Du, B.: Long-short distance aggregation networks for positive unlabeled graph learning. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM 2019), pp. 2157–2160 (2019)

[50] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)

[51] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (1997). https://doi.org/10.1162/neco.1997.9.8.1735

[52] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. CoRR **abs/1706.03762** (2017) 1706.03762

[53] Zhao, J., Wang, X., Shi, C., Hu, B., Song, G., Ye, Y.: Heterogeneous graph structure learning for graph neural networks. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pp. 4697–4705 (2021). https://ojs.aaai.org/index.php/AAAI/article/view/16600

[54] Huang, Z., Mamoulis, N.: Heterogeneous information network embedding for meta path based proximity. CoRR **abs/1701.05291** (2017) 1701.05291

[55] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: Li, Y., Liu, B., Sarawagi, S. (eds.) Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008, pp. 990–998 (2008). https://doi.org/10.1145/1401890.1402008. https://doi.org/10.1145/1401890.1402008

[56] Ji, M., Sun, Y., Danilevsky, M., Han, J., Gao, J.: Graph regularized transductive classification on heterogeneous information networks. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.)

Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6321, pp. 570–586 (2010). https://doi.org/10.1007/978-3-642-15880-3_42. https://doi.org/10.1007/978-3-642-15880-3_42

[57] Fan, H., Zhang, F., Wei, Y., Li, Z., Zou, C., Gao, Y., Dai, Q.: Heterogeneous hypergraph variational autoencoder for link prediction. IEEE Trans. Pattern Anal. Mach. Intell. **44**(8), 4125–4138 (2022). https://doi.org/10.1109/TPAMI.2021.3059313

[58] Hajiramezanali, E., Hasanzadeh, A., Narayanan, K.R., Duffield, N., Zhou, M., Qian, X.: Variational graph recurrent neural networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 10700–10710 (2019)

[59] Liu, J., Xu, C., Yin, C., Wu, W., Song, Y.: K-core based temporal graph convolutional network for dynamic graphs. IEEE Transactions on Knowledge and Data Engineering, 1–1 (2020). https://doi.org/10.1109/TKDE.2020.3033829

[60] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015). http://arxiv.org/abs/1412.6980

[61] Fang, Y., Zhao, X., Huang, P., Xiao, W., de Rijke, M.: Scalable representation learning for dynamic heterogeneous information networks via metagraphs. ACM Trans. Inf. Syst. **40**(4), 64–16427 (2022). https://doi.org/10.1145/3485189

[62] Leone, M., Huber, S., Arora, A., García-Durán, A., West, R.: A critical reevaluation of neural methods for entity alignment. Proc. VLDB Endow. **15**(8), 1712–1725 (2022)

[63] Liu, X., Hsieh, P., Duffield, N., Chen, R., Xie, M., Wen, X.: Real-time streaming graph embedding through local actions. In: Amer-Yahia, S., Mahdian, M., Goel, A., Houben, G., Lerman, K., McAuley, J.J., Baeza-Yates, R., Zia, L. (eds.) Companion of The 2019 World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019, pp. 285–293 (2019). https://doi.org/10.1145/3308560.3316585. https://doi.org/10.1145/3308560.3316585

[64] Wang, J., Chen, P., Ma, B., Zhou, J., Ruan, Z., Chen, G., Xuan, Q.: Sampling subgraph network with application to graph classification. IEEE Trans. Netw. Sci. Eng. **8**(4), 3478–3490 (2021). https://doi.org/10.1109/TNSE.2021.3115104