DyHANE: dynamic heterogeneous attributed network embedding through experience node replay

Liliana Martirano, Dino Ienco, Roberto Interdonato, Andrea Tagarelli

Applied Network Science

RESEARCH

Open Access

DyHANE: dynamic heterogeneous attributed network embedding through experience node replay

Liliana Martirano[1*], Dino Ienco[2,4], Roberto Interdonato[3,4] and Andrea Tagarelli[1]

*Correspondence:
liliana.martirano@dimes.unical.it

[1] Department of DIMES, University of Calabria, 87036 Rende, CS, Italy
[2] INRAE, UMR TETIS, University of Montpellier, 34090 Montpellier, France
[3] CIRAD, UMR TETIS, University of Montpellier, 34090 Montpellier, France
[4] INRIA, University of Montpellier, 34090 Montpellier, France

Abstract

With real-world network systems typically comprising a large number of interactive components and inherently dynamic, Graph Continual Learning (GCL) has gained increasing popularity in recent years. Furthermore, most applications involve multiple entities and relationships with associated attributes, which has led to widely adopting Heterogeneous Information Networks (HINs) for capturing such rich structural and semantic meaning. In this context, we deal with the problem of learning multi-type node representations in a time evolving graph setting, harnessing the expressive power of Graph Neural Networks (GNNs). To this purpose, we propose a novel framework, named DyHANE—Dynamic Heterogeneous Attributed Network Embedding, which dynamically identifies a representative sample of multi-typed nodes as training set and updates the parameters of a GNN module, enabling the generation of up-to-date representations for all nodes in the network. We show the advantage of employing HINs on a data-incremental classification task. We compare the results obtained by DyHANE on a multi-step, incremental heterogeneous GAT model trained on a sample of changed and unchanged nodes, with the results obtained by either the same model trained from scratch or the same model trained solely on changed nodes. We demonstrate the effectiveness of the proposed approach in facing two major related challenges: (i) to avoid model re-train from scratch if only a subset of the network has been changed and (ii) to mitigate the risk of losing established patterns if the new nodes exhibit unseen properties. To the best of our knowledge, this is the first work that deals with the task of (deep) graph continual learning on HINs.

Keywords: Graph Continual Learning, Heterogeneous Information Networks, Incremental Graph Neural Networks

Introduction

Contemporary scenarios involve a wide variety of actors and relationships that evolve over time, making networks suitable models for analyzing data interdependence. In a dynamic discrete-time setting with changes in network structure and entities features, detecting and adapting to changes is crucial for an accurate model, so that it can generate effective representations for all nodes in the network. It is necessary to integrate new knowledge, by identifying new patterns in the data, while simultaneously refining

existing knowledge, by consolidating existing patterns. This approach helps avoiding the so-called *catastrophic forgetting problem*, i.e., the tendency of a neural network to lose proficiency in previously learned tasks when adapting to or acquiring new information (McCloskey and Cohen 1989).

As noted by Khoshraftar and An (2022), Graph Neural Networks (GNNs) offer refined graph representations and flexibility in handling attributes, but scalability issues limit their efficiency compared to shallow approaches. For this reason, there are surprisingly few GNN-based works on Graph Continual Learning (GCL), i.e., continual learning approaches capable of harnessing the expressive power of GNNs for the integration of new knowledge in graph representations without relying on the entire network. Even less explored is the study of *incremental* GNNs for HINs, i.e., GNNs trained incrementally to simultaneously detect new patterns in heterogeneous nodes and consolidate current ones. We specify that, although in the literature the terms 'incremental' and 'streaming' are often used interchangeably, in the following we will exclusively employ the former to emphasize the incremental, step-wise nature of the proposed approach.

Existing GNN-based works on heterogeneous networks need to store the history of nodes and apply an additional recurrent architecture or attention mechanism to update the node representations, as shown by Yang et al. (2020); Xue et al. (2020); Martirano et al. (2022). In contrast, multiple architectures for homogeneous graphs, i.e., graphs with only one type of node and relationship, have been proposed to overcome scalability issues arising from storing multiple embeddings for each node or retraining the network from scratch at each observation time (Ma et al. 2020; Zhou and Cao 2021; Wang et al. 2020; Perini et al. 2022).

Wang et al. (2020) outline three scenarios for handling evolving graph data by employing GNN-based methods: (i) *pre-trained GNNs*, employing a pre-trained GNN model for generating representations of unseen and changed nodes, although effectiveness might decrease if node patterns significantly differ from the pre-trained model; (ii) *retrained GNNs*, which train a new GNN module on the entire graph data at each timestamp, achieving higher performance, but incurring substantial time and space costs, particularly if only a subset of the network changes over time; (iii) *online GNNs*, learning node representations by training solely on new nodes using parameters from the previous timestamp, despite a risk of loss of knowledge and degraded representations of unchanged nodes if patterns in new nodes differ from the existing network. All these strategies have clear limitations in handling the complexities of large-scale incremental graph data characterized by shifting pattern distributions over time.

To strike a balance between learning new knowledge and preserving existing knowledge, several works employ a regularization technique, such as the Elastic Weight Consolidation (EWC) (Kirkpatrick et al. 2016), by introducing a regularization term in the loss function that penalizes large changes in important weights learned during previous timestamps. The regularization term effectively constrains the neural network from making drastic changes to the parameters that are crucial for the performance on unchanged nodes.

A recent research direction involves replay-based (or reharsal) methods storing representative history data or well-designed data representations, and incorporating these elements during the training of new tasks. Strategies like *experience replay* and *generative replay* based on the storing, respectively, of a representative subset of past data (Zhou and Cao 2021; Wang et al. 2020; Perini et al. 2022) or of synthetic samples resembling data (Wang et al. 2022) in a memory buffer, proved to be effective in preventing the loss of knowledge in homogeneous graphs, but rehearsal approaches are still unexplored in HINs.

To fill the above gap in the literature, we propose a novel framework, named DyHANE (Dynamic Heterogeneous Attributed Network Embedding), which is designed to generate at each new timestamp multi-typed up-to-date node representations for all nodes in the network, by incrementally updating GNN parameters training on subset of the network. To this aim, we identify a novel strategy to detect and integrate the set of nodes affected by changes in a HIN—new knowledge detection—and a novel strategy to update the heterogeneous memory buffer used for experience node replay—existing knowledge consolidation. We experimentally evaluate our framework on a multi-class node classification task on a closed-world setting, i.e., known and fixed number of classes, handling both changes in network topology and node features. First, we demonstrate the advantage of HIN modeling. Then, we assess the effectiveness of DyHANE in comparison to Retrained GNNs and Online GNNs in terms of achieved performance and training time. We notice the flexibility of the proposed framework, particularly our base model, namely a Graph Attention Network, is actually interchangeable with any other GNN model.

Plan of the paper

The remainder of this paper is structured as follows. "Related work" Section discusses recent works particularly related to our framework. "Preliminaries" Section introduces and formalize preliminary concepts on HINs and incremental GNNs. "Proposed framework" Section describes our proposed framework in detail, and also provides a discussion on computational complexity aspects. "Experimental evaluation" Section presents our experimental evaluation, which was carried out by referring to the task of classification of authors' primary field of study as a case in point. Finally, "Conclusion" Section contains concluding remarks and provides pointers for future research.

Related work

To narrow our focus, we concentrate on GNN-based continual learning models in a discrete-time setting, with events propagating in batches at defined intervals. Discrete dynamic GNNs are known to be faster and generally simpler models compared to the continuous models, since iterating over snapshots rather than edge-by-edge. With no identified direct competitor for DyHANE in GNN-based continual learning on HINs, we provide a brief overview of (i) GNN-based approaches for HINs incorporating the temporal dimension and (ii) incremental GNNs for homogeneous networks using a memory buffer for experience node replay.

GNN-based approaches for HINs with temporal dimension

Existing literature commonly treats the temporal component as an additional dimension, using extra recurrent architectures or attention mechanisms for updating node representations. HDGNN (Zhou et al. 2020) combines GNN and RNN architectures for scientific impact propagation learning. DYHAN (Yang et al. 2020) employs hierarchical attention for link prediction. DyHATR (Xue et al. 2020) uses a hierarchical attention model for learning static snapshots and a temporal attentive RNN for evolutionary patterns in link prediction. Co-MLHAN (Martirano et al. 2022) learns node representations with hierarchical attention and a collaborative contrastive cross-view mechanism in an unsupervised setting, modeling the temporal dimension across layers. The issue inherent in these methodologies lies in their static nature, since assuming knowledge of the entire network. DyHINE (Xie et al. 2021) constructs an online update model with a dynamic operator on top of a dynamic time-series embedding model. It employs hierarchical attention to aggregate neighbor features and temporal random walks to capture dynamic interactions. In contrast to our approach, it integrates all the neighbors of changed nodes and has the need to store the history of nodes, separating each embedding into past and changing embedding. More similar to our approach is LIME (Peng et al. 2022). After mapping the nodes into a shared cuboid space and employing dynamic meta-path guided random walks and Recursive Neural Networks for initial node embeddings, it applies incremental learning to update node representations using the dynamic Minimum Cost Maximum Flow algorithm. The update, however, is less controlled and does not directly depend on relevant changes in the neighborhood.

Differently from the proposed approach, any of these works (except for HDGNN) is concerned with encoding node dynamics, such as changes in node features, while our dataset has both fixed and dynamic attributes (cf. "Data" Section). Moreover, our framework can be extended without efforts to handle removals.

Incremental GNNs approaches for homogeneous networks based on experience node replay

In the following we present the models designed for homogeneous graphs more related to our approach, with which we share the goal of employing incremental GNNs to support classification accuracy in the presence of changes in class distribution, based on a curated, smaller subset of the network. ER-GNN (Zhou and Cao 2021) retains the most representative nodes of different classes in a buffer according to different selection strategies and replay them at following timestamps. The proposed strategies based on the mean of feature, coverage maximization, and influence maximization are not immediately applicable to tasks other than classification. Conversely, similarly to our approach, ContinualGNN (Wang et al. 2020) prioritizes nodes more likely to be located at the class boundary based on their impact on the gradient, and stores those with attributes significantly distinct from their neighbors. Our approach extends this notion to diversity in attributes and local structure and to multiple node and edge types. Random-Based Rehearsal (RBR) and Priority-Based Rehearsal (PBR) (Perini et al. 2022) introduce incremental GNN models with experience replay, yielding a uniform sample of the training graph or prioritizing data points based on the model prediction error, respectively, as

strategies for sample selection. In contrast to our approach, they assume a constant number of changes over time, and they do not handle changes over the node set.

Despite prioritizing the most representative nodes for updating the experience memory buffer, DyHANE is not an active learning approach. It involves the continual adaptation of machine learning models to evolving graph-structured data over time, distinguishing it from graph active learning, which specifically focuses on selecting data points for labeling.

Preliminaries

In this section, we first provide the essential background underlying our proposed framework, then we formally define the problem addressed in this work.

Background

Here we recall the concepts of dynamic Heterogeneous Information Network (HIN), meta-paths as composite relations on HINs, and the evolution of HINs in discrete time.

Dynamic heterogeneous information networks (HINs)

Dynamic HINs or interchangeably *dynamic heterogeneous attributed graphs* are networks with multiple node and/or edge types and external information associated with nodes available as a set of attributes. Formally, we define a dynamic HIN at a generic timestamp t as $G^t = \langle \mathcal{V}^t, \mathcal{E}^t, A, R, \phi, \varphi, \boldsymbol{\mathcal{X}}^t \rangle$, where $\mathcal{V}^t$ and $\mathcal{E}^t$ are the sets of nodes and edges at timestamp t, A and R are the (fixed) sets of node and relation types, with $|A| + |R| > 2$, $\phi : \mathcal{V}^t \to A$ and $\varphi : \mathcal{E}^t \to R$ are the node- and edge-type mapping functions, and $\boldsymbol{\mathcal{X}}^t$ is the set of matrices storing node attributes at time t. We specify that, since different node types could be associated with different types of content, the attribute vectors for different node types could be of different lengths, resulting in $\boldsymbol{\mathcal{X}}^t = \{\boldsymbol{\mathcal{X}}^t_a\}, \forall a \in A$.

Meta-paths in dynamic HINs

A *meta-path* type—for short *meta-path*—Sin a HIN is a composite relation used to model high-order proximity in the form $\sigma(a_1, a_{x+1}) = a_1 \xrightarrow{r_1} a_2 \xrightarrow{r_2} \cdots \xrightarrow{r_x} a_{x+1}$ between two node types, a_1 and a_{x+1}, which are expected to share some information. Since many high-order relations can be established between node types, we denote as σ_m the m-meta-path type in the set $\mathcal{M}$ of all selected meta-path types. The most informative meta-paths are usually few and of short length, and can be either identified through a hand-crafted sample by domain experts or computed through new approaches of meta-path reduction (Wei et al. 2018), interesting meta-paths mining (Shi and Weninger 2014) and meta-path discovery (Wan et al. 2020) even in large-scale Heterogeneous Information Networks. The length of a meta-path is the number of nodes held in the corresponding composite relation and is denoted as $len(\sigma_m)$

A *meta-path instance* of the meta-path type σ_m is a sequence of size $len(\sigma_m)$ of connected nodes matching the node and edge types in the meta-path, able to make two distant nodes in the network (the *terminal* nodes) reachable. In the following, we

denote as p a pair of nodes connected by at least one meta-path instance. A *meta-path-based graph* is a graph comprised of all nodes connected via meta-path instances of a determined meta-path σ_m. Given a target node type $\overline{a} \in A$, i.e., the node type targeted for a task at hand, and a meta-path σ_m with terminal nodes of the same type $\overline{a}$, the resulting meta-path-based graph at time t is a homogeneous weighted graph with one node type $\overline{a}$ and one relation type σ_m, obtained from all meta-path instances of σ_m by removing the intermediate nodes and establishing a direct link between the pair p weighted on the number of meta-path instances connecting p. The *meta-path-based neighbors* under meta-path σ_m of a node $v_i \in \mathcal{V}^t$ is the set of nodes ${N_{i,\sigma_m}}^t$ connected at time t to v_i via at least one meta-path instance of σ_m.

Evolution of HINs in discrete time

We define the discrete network evolution over time as a set of events at each timestamp t, corresponding to the set of changed edges ${\mathcal{E}_c}^t = \bigcup_{i,j} e_{ij}$, with $e_{ij} = \langle i, x_i, j, x_j, r, x_{ij}, s, c \rangle$. Each event is an edge between nodes of indices i and j of type $\varphi(e) = r$, with $\{v_i, v_j\} \in \mathcal{V}^t$; x_i and x_j denote the attribute vectors of nodes v_i and v_j resp., while x_{ij} denotes the attribute vector associated with the edge. In our formulation, $s = 1$ (resp. $s = 0$) denotes the addition (resp. removal) of e_{ij} of type r; we point out that we assume $s = 1$ in all the events, but all the proposed algorithms are designed to work equally if removals occur; c denotes the *category* of the event: $c = 1$ (resp. $c = 0$) corresponds to an event with *strong* (resp. *weak*) impact on its neighbors, as explained in "Identification of influenced nodes" Section. To handle the attribute updates of isolated nodes, i.e., nodes not involved in any change on network topology, we map the corresponding event as a self loop, with $s = 1$ and $c = 1$. We specify that each event contributes to changes in the network topology $\Delta G^t = G^t - G^{t-1}$, with ΔG^t comprising new nodes ($\mathcal{V}^t - \mathcal{V}^{t-1}$) and edges ($\mathcal{E}^t - \mathcal{E}^{t-1}$), and/or changes in the attribute matrices $\Delta \mathcal{X}^t = \mathcal{X}^t - \mathcal{X}^{t-1}$.

Problem setting

Our proposed framework is designed to deal with *incremental graph neural network models on HINs*. In the following, we provide the key elements underlying such models contextualized in our problem setting.

We are given a set of T timestamps $\{t_1, t_2, \ldots, t_T\}$, each of which is associated with a set of events ${\mathcal{E}_c}^t$ determining a change in $\Delta G^t = G^t - G^{t-1}$ and/or $\Delta \mathcal{X}^t = \mathcal{X}^t - \mathcal{X}^{t-1}$. An incremental GNN gradually learns $\{\theta^{t_1}, \theta^{t_2}, \ldots, \theta^{t_T}\}$, where θ^t are the GNN parameters at timestamp t to generate effective representations ${z_i}^t, \forall v_i \in \mathcal{V}^t$, using only a sample of nodes in G^t as training set.

The training set $\mathcal{T}$ comprises all the unseen and changed nodes plus a curated subset of unchanged nodes. By denoting with $\mathcal{I}^t$ the *influenced node set* at time t, i.e., the (minimum) subset of nodes affected by changes, and with $\mathcal{B}^{t-1}$ the *experience memory buffer* updated at time $t-1$ and replayed at time t, the training set at time t is built as $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$, where $\mathcal{I}^t \supseteq \Delta \mathcal{V}^t$, $\mathcal{B}^{t-1} \subseteq \mathcal{V}^{t-1}$ and $\mathcal{I}^t \bigcup \mathcal{B}^{t-1} \subseteq \mathcal{V}^t$. Note that both $\mathcal{I}^t$ and $\mathcal{B}^{t-1}$ comprise nodes of different types.

Proposed framework

Algorithm 1 DyHANE incremental GNN.

Require: Set of events ${\mathcal{E}_c}^t$ at current timestamp, experience node buffer $\mathcal{B}^{t-1}$, GNN parametrized by θ^{t-1}, number of epochs num_epochs.
Ensure: GNN parametrized by θ^t learned at current timestamp, updated experience buffer $\mathcal{B}^t$.
Obtain influenced node set $\mathcal{I}^t$ from ${\mathcal{E}_c}^t$ via Algorithm 2.
if $t > 0$ **then**
    Load the experience buffer $\mathcal{B}^{t-1}$ and the GNN model initialed with θ^{t-1}.
else
    Load G^0 and initialize the GNN parameters at random.
end if
Build training set $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$
Train - test - validation splitting as in [18], or load the test set for fair comparison (cf. Sec. 5).
$i = 0$
while i $< num\_epochs+1$ **do**
    Calculate loss function $\mathcal{L}$ (e.g., cross-entropy for node classification)
    Update GNN parameters using SGD
    $i = i + 1$
end while
Update the experience buffer $\mathcal{B}^t$ from $\mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ via Algorithm 3
return θ^t

Our proposed DyHANE is an incremental GNN model that consists of a sequence of steps reiterated at each new timestamp *t*, as summarized in Algorithm 1 and depicted in Fig. 1:

1. Detection of the influenced node set $\mathcal{I}^t$, as presented in Algorithm 2.
2. Resume of the experience memory buffer $\mathcal{B}^{t-1}$ computed at previous timestamp and GNN parameters θ^{t-1} learned at previous timestamp.
3. Update of the GNN module, learning θ^t able to generate effective representations for all nodes $v_i \in \mathcal{V}^t$.
4. Update of the memory buffer $\mathcal{B}^t$ from $\mathcal{I}^t \bigcup \mathcal{B}^{t-1}$, as presented in Algorithm 3.

Note that (1) and (2) can be executed in parallel, while (4) can be performed at any time while waiting for the next increment. In the following, we describe the two proposed algorithms for the detection of influenced nodes and the update of the experience buffer, respectively.

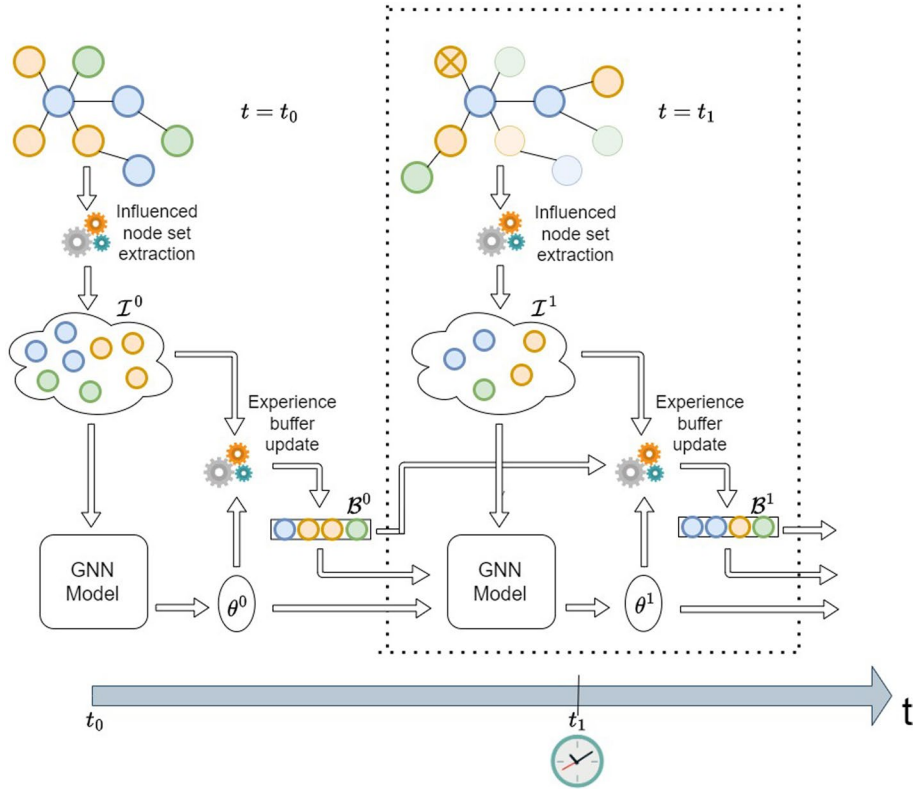


Fig. 1 Workflow of DyHANE, with the dashed rectangle highlighting a given incremental step $t = t_1$, involving the detection of the influenced node set at current time $\mathcal{I}^1$ (cf. Algorithm 2) and the resume of the experience buffer $\mathcal{B}^0$ and GNN initialization parameters θ^0 updated at previous timestamp. The model outputs the updated parameters of the neural network model θ^1, which should be able to generate effective representations for both changed and unchanged nodes in $\mathcal{V}^1$. Finally, the memory buffer $\mathcal{B}^1$ is updated for the next interval (cf. Algorithm 3)

Please also note that our chosen GNN model is a Graph Attention Network (GAT) (Brody et al. 2021), which is widely recognized as a highly effective GNN for various downstream tasks; however, it should be emphasized that DyHANE is not constrained to a particular GNN architecture or to a unique implementation of the proposed strategies.

Identification of influenced nodes

To identify nodes affected by changes, focusing on the entire network is unnecessary, especially when the node neighborhood is stable (Du et al. 2018). Likewise, relying solely on changed nodes is inadequate, since the new patterns can effect existing nodes due to interdependence in network data. Given this premise, we propose a dedicated procedure (Algorithm 2) to mine nodes affected by changes during network evolution.

Algorithm 2 Influenced node set detection.

Require: Set of events ${\mathcal{E}_c}^t$ at current timestamp, set of edges $\mathcal{E}^{t-1}$, set of attribute matrices $\mathcal{X}^{\mathbf{t-1}}$ and set of meta-path-based adjacency matrices of target node type $\overline{a} \bigcup_{\sigma_m} \mathcal{A}\text{adj}_{\sigma_m}^{t-1}$ at previous timestamp.
Ensure: Influenced node set $\mathcal{I}^t$.
Initialize $\mathcal{I}^t = \emptyset$
for each $e_{ij} \in {\mathcal{E}_c}^t$ **do**
  Update $\mathcal{X}^t_{\phi(v_i)}$ and $\mathcal{X}^t_{\phi(v_j)}$
  $\mathcal{I}^t = \mathcal{I}^t \bigcup \{i, j\}$
  Initialize event category as $c = 1$ $(strong)$
  Generate new meta-paths instances mps for each type σ_m crossing e_{ij} and connecting nodes of target type $\overline{a}$
  if $mps = \emptyset$ **then**
    $c = 0$ $(weak)$
  else
    for each pair of terminal nodes $p \in \text{mps}(i.e., \forall\ \sigma_m$ crossing $e_{ij})$ **do**
      $ok = false$
      for each meta-path-based adjacency matrix $\mathcal{A}\text{adj}_{\sigma_m}^{t-1}$ **do**
        if $\mathcal{A}\text{adj}_{\sigma_m}^{t-1}[p[0]][p[1]] == 0$ **then**
          $ok = true$
          break
        end if
      end for
      if not ok **then**
        break $(c = 1$ $(strong))$
      end if
    end for
    $c = 0$ $(weak)$
  end if
  if c == 1 **then**
    Update $\mathcal{I}^t = \mathcal{I}^t \bigcup {N_i}^t \bigcup {N_j}^t$
    for each $x \in \{i, j\}$ **do**
      if $\phi(x) = \overline{a}$ **then**
        Update $\mathcal{I}^t = \mathcal{I}^t \bigcup N^t_{x,\sigma_m} \forall \sigma_m$
      end if
    end for
  end if
end for
Remove duplicates: $\mathcal{I}^t = \text{set}(\mathcal{I}^t)$
return $\mathcal{I}^t$

To identify the minimum set of influenced nodes at current timestamp $\mathcal{I}^t$, we categorize new events in the graph as *strong* or *weak* based on their impact on the network topology. The intuition is that weak events do not generate new knowledge, while strong events need to be propagated further. Inspired by recent work on graph representation learning for dynamic homogeneous networks (Trivedi et al. 2019) and acknowledging the effectiveness of meta-path based models in capturing heterogeneous information in large networks (Dong et al. 2017; Shang et al. 2016), we classify the edge corresponding to an event as *weak* if it is not crossed by any meta-path instance connecting target node types, or if generated meta-path instances already exist; otherwise it is said *strong*. For all events, we add to the set of influenced nodes

the incident nodes. For strong events, we add to $\mathcal{I}^t$ also the incident nodes' heterogeneous one-hop neighborhood; if any of the incident nodes is of target type $\overline{a}$, the affected nodes will include also their meta-path-based neighborhood. Since a node can contribute to multiple events, we ensure not to include duplicates in the influenced node set. Note that to handle attribute changes of isolated nodes, i.e., nodes not involved in any change on network topology, we map the corresponding event as a particular self-loop and add the node and its neighborhood to $\mathcal{I}^t$.

Because only incremental meta-paths on changed nodes are computed, and categorization is done based on look-ups of sparse matrices, the proposed algorithm has been shown to be effective in practice, compared to training on the entire network. We elaborate on this aspect in "Experimental evaluation" Section.

Update of the memory buffer

Algorithm 3 Update of the experience memory buffer.

Require: The model at current timestamp t — including training set $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$, ground truth y, fixed buffer size $|\mathcal{B}|$, number of nodes to be used to compute type importance $top{-}k$.
Ensure: Updated experience buffer $\mathcal{B}^t$.
1: Initialize the Explainer (e.g., the CaptumExplainer).
2: Initialize a vector $\mathbf{s}^t \in R^{1\times|A|}$ to store node type importance s.t. it sums to 1.
3: Compute the importance score (impact on the gradient) $\delta_i \forall (not\ masked) v_i \in \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ via the Explainer.
4: **for each** node type $a \in A$ **do**:
5: Compute the relative importance of features of $top{-}k$ nodes of type a (sum) and store the obtained score in $\mathbf{s}^t[a]$.
6: Compute the number $top{-}k_a$ of nodes of type a to be stored in $\mathcal{B}^t$ weighted on $\mathbf{s}^t[a]$.
7: Select the $top{-}k_a$ nodes of type a from $\mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ with highest impact score δ_i as sum of the importance of their features, and store them in $\mathcal{B}^t$.
8: **end for**
9: **return** $\mathcal{B}^t$

To update the memory buffer $\mathcal{B}^t$, we detect the most relevant nodes for classification using a Captum-based explainer, able to identify crucial subgraph structures and node features influencing GNN predictions. Employing the Integrated Gradients algorithm (Sundararajan et al. 2017) for multi-instance explanations, we assess each input feature's contribution to the model output and identify nodes with major impact on gradient (Algorithm 3).

Integrated Gradients is a principled approach that satisfies several desirable properties for feature attribution, including completeness—the sum of the attributions exactly accounts for the difference in the model's output between the actual input and the baseline—and sensitivity—small variations in the input features lead to proportionally small changes in the attributions. Specifically, the Integrated Gradients algorithm comprises several steps:

- *Baseline selection:* Choose a baseline or reference point as a point in the input space with known or neutral feature values.

- *Path construction:* Define a path from the baseline to the input data point, being a straight line in the input space.
- *Gradient calculation:* Compute the gradient of the model's output with respect to the input features at multiple points along the constructed path, by taking the partial derivative of the model's output with respect to each input feature.
- *Integration:* Integrate the computed gradients along the path, using the trapezoidal rule or a more sophisticated method. The result is a set of values, each representing the accumulated effect of a specific feature along the path.
- *Attribution calculation:* Multiply the integrated gradients by the difference between the input and baseline at each point along the path and sum these values, yielding the attribution of each feature to the model's output.
- *Scaling:* Scale the attributions by the difference between the baseline and the actual input, helping ensure that the attributions are meaningful and consistent across different inputs.

In the process, scores are assigned to each feature of each node. We determine the node-level score by summing along the feature dimension, i.e., by aggregating the contributions of all its features. Subsequently, we compute the cumulative contributions of the *top-k* nodes for each node type, with k being a given constant. This computation results in the determination of the percentage of nodes for each type $a \in A$ to be stored in the experience memory buffer. $\mathcal{B}^t$ will store the *top-*k_a nodes for each node type a, i.e., the nodes of type a with the most substantial impact on the gradient. We point out that the most frequent type in the memory buffer does not necessarily correspond to the target type and can change at each new computation.

Note that the update of the experience memory buffer to be replayed at next timestamp can be performed at any time while waiting for the new batch of events. Different buffer sizes according to different *top-k* thresholds were tested in our experimental evaluation (cf. "Experimental evaluation" Section).

Computational complexity aspects

In this section, we discuss the computational complexity of DyHANE. In our analysis, we assume network evolution at discrete time, i.e., a set of *deferred* timestamps $\{t_1, t_2, \ldots, t_T\}$ corresponding to as many sets of events determining changes in network topology and/or in node attributes (cf. "Preliminaries" Section).

We assume sparse graphs (both the initial network G^0 and each network increment ΔG^t), and dense content-features of nodes. We also make the reasonable assumption that each increment is significantly smaller with respect to the network size, i.e., $|\mathcal{E}_c^t| \ll |\mathcal{E}^t|$ and $|\mathcal{I}^t| \ll |\mathcal{V}^t|$. We recall that the size of node embeddings is equal to d, with $d \ll |\mathcal{V}^t|$, and that the training set is $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$, with $|\mathcal{B}^{t-1}| \ll |\mathcal{I}^t|$.

In the following, we delve into the time computational aspects of the proposed approach, then we analyze the memory space required between two consecutive updates and for storing intermediate representations.

Time complexity

The time complexity analysis of DyHANE can be divided according to the three main steps of Algorithm 1, namely the detection of the influenced node set (Algorithm 2), the update of the GNN module, and the update of the memory buffer.

Detection of the influenced node set Algorithm 2 requires, for each event (edge) $e \in \mathcal{E}_c^t$, the generation of all instances for each meta-path type crossing the edge and connecting nodes of target type, being the most expensive operation. Given an edge of type r, generating all instances for a certain meta-path type σ_m crossing r costs $\prod_{\hat{r} \in R_{\sigma_m - r}} |\mathcal{E}_{\hat{r}}|$, with $R_{\sigma_m - r}$ denoting all the edge types except for (the first) r crossed by the k-meta-path type. The number of products is thus equal to the length of the meta-path type minus 2. Since these computations can be carried out independently, and hence can be parallelized, the time cost is dominated by the longest meta-path type, i.e., $\mathcal{O}(\max(len(\sigma_1), len(\sigma_2), \ldots, len(\sigma_{\mathcal{M}}))$. To summarize, the time cost of the detection of the influenced node set is $\mathcal{O}(|\mathcal{E}_c^t| \times |\mathcal{E}_{\tilde{r}}^{t-1}|^{\text{len}(\sigma_{\tilde{m}})-2})$, with $\tilde{r} = \max_{r \in R}(|\mathcal{E}_r^{t-1}|)$ and $\sigma_{\tilde{m}}$ being the meta-path type of maximum length. For each identified meta-path instance, a lookup at the corresponding meta-path adjacency matrix is required to check if the pair of terminal nodes already exists. We assume the worst case in terms of number of computations, i.e., all possible meta-path instances generated and all node pairs checked; nonetheless, in practical scenarios, the most informative meta-paths are usually few and of short length (cf. "Preliminaries" Section).

Update of the GNN module The time complexity of a GNN is typically determined by the feature transformation step, the neighborhood aggregation (message passing) step, and architectural aspects of the neural network such as the number of layers and, in a GAT, the number of attention heads. We denote with $\mathcal{E}_{\mathcal{T}^t}$ the set of edges of the subgraph induced by the nodes in $\mathcal{T}^t$. Assuming each node's attention mechanism requires computation time linear in the number of neighboring nodes, the time complexity of message passing for each node can be expressed as $\mathcal{O}(max_{deg} \times d)$, where max_{deg} is the maximum node degree and d is the dimension of the input feature vector for each node. Considering all nodes, it results in $\mathcal{O}(|\mathcal{T}^t| \times max_{deg} \times d)$. In the heterogeneous case, having the same attention mechanism for each relation r, the time complexity is dominated by the most abundant relationship $r^* = \max_{r \in R}(|\mathcal{E}_{\mathcal{T}_r^t}|)$. The computational complexity with a single attention head in a general GAT is $\mathcal{O}(|V_r|d^2 + |E_r|d)$ (Brody et al. 2021), where V_r is the set of nodes connected through the edges in E_r. The first term concerns the feature transformation step of GATv2, while the second term corresponds to the cost of calculating a general attention function, which can be parallelized. In the case of Q attention heads, both the first and the second terms are multiplied by a factor of Q, where the different heads can still be parallelized. The time cost of our heterogeneous incremental GAT module is hence $\mathcal{O}(|\mathcal{T}_{r^*}| \times d^2 + |\mathcal{E}_{\mathcal{T}_{r^*}}| \times d)$. We note that for the same GNN architecture, the number of training iterations required for convergence usually decreases when training on a smaller subset of the graph, since the model learns from fewer examples.

Update of the memory buffer In our setting, the detection of the most relevant nodes for classification to be stored as experience replay is accomplished by the Captum-

based explainer on the GNN model. The time cost of the explainer accounts for the model evaluation and the Integrated Gradients calculation. The model evaluation time is linear with respect to the size of the input graph, i.e., $\mathcal{O}(|\mathcal{T}^t|)$. The time cost of the Integrated Gradients calculation also depends on the size of the input graph scaled on the number of steps in the integration process ($s = 50$ by default), i.e., $\mathcal{O}(|\mathcal{T}^t| \times s)$. This is hence the dominating term of the time cost of the explainer.

Once obtained the output score for each node, we sort the output scores independently for each node type, which is $\mathcal{O}(|\mathcal{T}^t_{\tilde{a}}| \times \log |\mathcal{T}^t_{\tilde{a}}|)$, with $\mathcal{T}^t_{\tilde{a}}$ denoting the nodes of the most abundant type $\tilde{a} \in A$. This operation is hence dominated by the type with more nodes. The computation of the relative importance of the top-k nodes for a node type (with k shared by all node types and timestamps) is $\mathcal{O}(k)$, since it is the sum of the top-k scores for that type, and is negligible given $k \ll |\mathcal{T}^t_{\tilde{a}}|, \forall a \in A$. Analogously, selecting the top-k_a nodes for each node type on the sorted scores has a time complexity of $\mathcal{O}(k_a)$, which is negligible given $k_a \ll |\mathcal{T}^t_{\tilde{a}}| \forall a \in A$. The total time cost of the update of the memory buffer is $|\mathcal{T}^t| \times s + |\mathcal{T}^t_{\tilde{a}}| \times \log |\mathcal{T}^t_{\tilde{a}}|$

By summing the above contributions, the overall time cost of DyHANE is $\mathcal{O}((|\mathcal{E}^t_c| \times |\mathcal{E}^{t-1}_{\tilde{r}}|^{\text{len}(\sigma_{\tilde{m}})-2}) + (|\mathcal{T}_{r^*}| \times d^2 + |\mathcal{E}_{\mathcal{T}_{r^*}}| \times d)) + (|\mathcal{T}^t| + |\mathcal{T}^t| \times s + |\mathcal{T}^t_{\tilde{a}}| \times \log |\mathcal{T}^t_{\tilde{a}}|)$, where, we recall that, $\mathcal{E}^t_c$ is the set of events (changed edges) at current time, $\mathcal{E}^{t-1}_{\tilde{r}}$ is the set of edges of G^{t-1} of type $\tilde{r}$, with $\tilde{r}$ being $\max_{r \in R}(|\mathcal{E}^{t-1}_r|)$, $\sigma_{\tilde{m}}$ is the meta-path type of maximum length, $\mathcal{T}_{r^*}$ is the set of nodes of training set connected via the edges in $\mathcal{E}_{r*}$, with r^* being $\max_{r \in R}(|\mathcal{E}_{\mathcal{T}^t_r}|)$, $\mathcal{T}^t$ is the training set at current time, and $\mathcal{T}^t_{\tilde{a}}$ is the subset of nodes of current training set of type $\tilde{a} \in A$, with $\tilde{a}$ being $\max_{a \in A}(|\mathcal{T}^t_a|)$.

It can be noticed that the total time cost of the proposed approach, at each timestamp, is dominated by the detection of nodes affected by changes (Algorithm 2) and the training of the GNN module. The memory buffer update, beside being a computationally lighter operation, can be performed downstream of classification at any time while waiting for the next increment, exploiting knowledge of the network architecture and stored parameters for subsequent model initialization.

Space complexity

As concerns the space complexity, the memory requirement is mainly given by the storage of the node attributes ($|\mathcal{V}^t| \times d$), the edges $\mathcal{E}^t$, the learned parameters θ^t—including the weights associated with the attention mechanism and any other learnable parameters in the model—and the meta-path adjacency matrices $\mathcal{A}\,\text{adj}^{t-1}_{\sigma_m}$ for all meta-path types $\sigma_m \in \mathcal{M}$ connecting target node types ($|\mathcal{V}_{\bar{a}}| \times |\mathcal{V}_{\bar{a}}| \times |\mathcal{M}|$), which are stored for the next increment. A fixed storage space is devoted to the indices of the nodes included in the memory buffer.

Algorithms 2 and 3, as well as the GAT, require additional space to store intermediate representations. Algorithm 2 needs to store, for each event (edge), all meta-path instances crossing that edge. During training and inference, GAT stores intermediate representations of nodes and attention weights, whose space complexity depends on factors such as the size of the graph and the number of layers in the model (2 in our case). Algorithm 3 needs to store the GAT model as input of the Captum explainer, and subsequently the output score for each node, and the top-k indices and scores, resulting

in $\mathcal{O}(|\mathcal{T}| + \sum_{a \in A} k_a)$, where the first term is the cost for storing the output scores and the second term is the cost for storing the top-k indices and scores for each node type, assuming each index and score is stored using a constant amount of memory.

Experimental evaluation

In this section, we describe the experimental evaluation of our framework. Our main goal is to assess the effectiveness of DyHANE with respect to the Retrained and Online approaches, showing that the proposed heterogeneous incremental GNN model strikes a satisfying balance between the performance achieved by Retrained GNNs, in terms of accuracy in classifying changed and unchanged nodes, and by Online GNNs, in terms of efficient incorporation of new knowledge.

"Data" Section introduces the data, "Advantage of HINs" Section discusses the advantages offered by HIN models, "Competing methods" Section presents the competing methods, "Experimental settings" Section discusses the experimental settings, and "Results" Section describes the main experimental results.

Data

Our heterogeneous dataset is a collaboration-citation-affiliation network comprising 3 different node types, i.e., Author (A), Paper (P) and Institution (I), and 3×2 relation types, i.e., "A writes P" (A–P), "P cites P" (P–P), "A is affiliated with I" (A–I), and their counterparts, as shown in Fig. 2. We selected A as target node type, towards which we built 3 different meta-paths: co-autorship (A–P–A), citation (A–P–P–A) and affiliation (A–I–A), i.e., we are interested in pairs of authors who wrote the same paper, or one cited the other, or have the same affiliation. Table 1 shows the dataset statistics in terms of number of nodes, number of edges and number of meta-path instances, with focus on the last increment. Table 2 provides more insights about centrality,

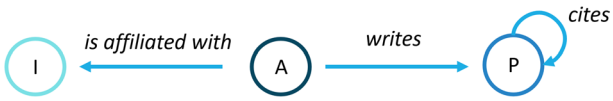


Fig. 2 Network schema of the proposed collaboration—citation—affiliation HIN

Table 1 Dataset statistics, in terms of no. of nodes, edges and meta-path instances, with focus on the last increment (2022)

		2019–2022	2019–2021	2022
# Nodes	Authors	15397	13116	2281
	Papers	10174	8642	1532
	Institutions	1831	1612	219
# Edges	A-P	18768	16020	2748
	A-I	4893	4181	712
	P-P	192	113	79
# Meta-paths	A–P–A (w)	65687	57736	7951
	A–I–A (w)	25567	18507	7060
	A–P–P–A (w)	587	302	285

Table 2 Centrality, connectivity, path-based and mesoscopic measures of our dataset computed on different subgraphs. Specifically, G_AP and G_AI are bipartite graphs (with A and P, A and I resp., as node types). G_A is the inferred homogeneous directed unweighted graph, corresponding to the unique meta-path-based graph comprising all meta-paths. All other graphs are variant of G_A: *w* denotes the addition of edge weights, *attr* denotes the addition of numeric attributes for node type A, and *und* denotes the removal of edge direction. The * denotes that the calculation is performed w.r.t. the node of target type A, either in the bipartite or homogeneous graph

Measure	Value	Graph
Average in-degree from P *	1.22	G_AP
Average in-degree from I *	0.26	G_AI
Average in-degree from A*	3.61	G_A
Average weighted in-degree from A*	4.17	G_A_w
Degree assortativity	0.97	G_A_w
Attribute assortativity (label)	0.39	G_A_attr
Attribute assortativity (n_works)	0.09	G_A_attr
Attribute assortativity (n_cit)	0.15	G_A_attr
Attribute assortativity (impact_factor)	0.18	G_A_attr
Attribute assortativity (h_index)	0.16	G_A_attr
Attribute assortativity (i10_index)	0.18	G_A_attr
Transitivity	0.96	G_A
Clustering coefficient	0.65	G_A
Density	0.000563	G_A
Average path length largest CC	13.01 (897 nodes)	G_A_und
Diameter largest CC	33 (897 nodes)	G_A_und
# Strongly connected components	2355	G_A
# Weakly connected components	2309	G_A
# Communities	2769 - 2327	G_A - G_A_und
Modularity	0.93 - 0.97	G_A - G_A_und

connectivity, path-based and mesoscopic measures. We underline that since there are no specific libraries for computing the statistics of multi-types of nodes and relationships, we derived from the heterogeneous graph several subgraphs (bipartite, weighted, directed and undirected). For each measure we report the specific subgraph on which it is computed.

We built our dataset from OpenAlex (Priem et al. 2022) by a combination of paper attributes, including the language ('en'), the type ('journal article' or 'proceedings-article' or 'book'), publication year >= 2019 and 'Computer science' as *concept* with score >= 0.5, from which we derived the connected entities. We used the year as our time interval, and thus examine 4 time intervals, or rather 3 increments, from 2019 to 2022.

Each node type is associated with categorical, numeric, and/or textual attributes. Author attributes include full name, impact factor, h-index and i10-index, number of published papers, and number of received citations. Institution attributes include name, country, and type (e.g., education, company, or nonprofit). For papers, in addition to information used as a filter for scraping, the title, the location (e.g., the journal or book title), the number of citations per year, the full abstract, and the weighted list of relevant concepts are provided. For attribute encoding, we employed one-hot

encoding for categorical attributes, feature scaling for numeric attributes and SentenceBERT (Reimers and Gurevych 2019) (with all-MiniLM-L6-v2 as pretrained model) for text attributes; since the max sequence length is set to 256, we split long texts (such as abstracts) into chunks, encoded individual chunks and computed their mean. All node types have their attribute vectors of different sizes. Note that our dataset contains a mixture of static and dynamic attributes; for instance, the number of published works or the number of citations may vary each year.

We validated the proposed framework on a multi-class classification task, where the class attribute corresponds to the authors' first *concept* (i.e., main area of expertise), and the labels are 8 subfields of 'Computer science (CS)': 'CS.Economics and Business' (0),'CS.Engineering' (1), 'CS.Mathematics' (2), 'CS.Natural/Earth science and Medicine' (3), 'CS.Philosophy and Art' (4), 'CS.Physics' (5), 'CS.Political science' (6) and 'CS.Psychology, CS.Sociology and History' (7). The derivation of the classes exploited the OpenAlex 6-level hierarchy of concepts, which is a modified version of the tree structure proposed by Shen et al. (2018), and the 'wikidata' attribute, which is the link to the associated Wikipedia page. The distribution of classes for each of the 4 years is shown in Fig. 3.

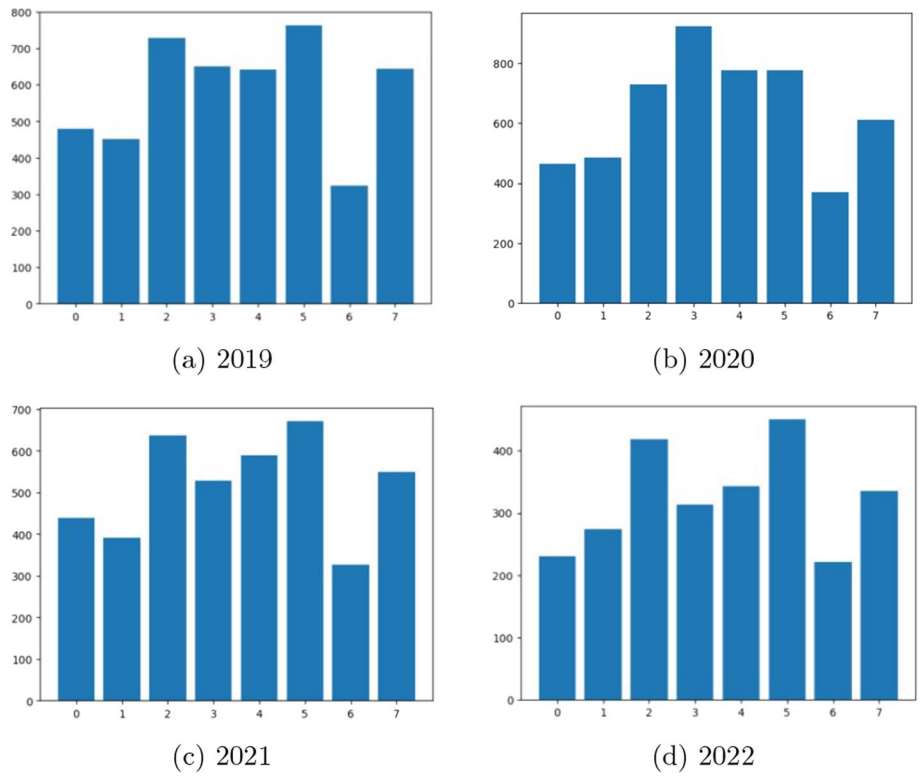


Fig. 3 Distribution of classes for each of the 4 years. The label mapping is as follows: 0: 'Economics and Business', 1: 'Engineering', 2: 'Mathematics', 3: 'Natural/Earth Science and Medicine', 4: 'Philosophy and Art', 5: 'Physics', 6: 'Political science', 7: 'Psychology, Sociology and History'

Table 3 Comparison of different models trained on the entire dataset spanning the years from 2019 to 2022 according to multiple evaluation metrics. The first column designates both the architecture and dataset employed. The best results are highlighted in bold, the second best are underlined

Model	F1-micro	F1-macro	F1-weighted	ROC-AUC
MLP on only A-type features	0.145 ± 0.000	0.036 ± 0.000	0.042 ± 0.000	0.515 ± 0.000
GAT on homogeneous net with A-type nodes	0.339 ± 0.001	0.332 ± 0.001	0.337 ± 0.001	$\underline{0.701} \pm \underline{0.001}$
GAT on heterogeneous net with A-type nodes	$\underline{0.560} \pm \underline{0.003}$	$\underline{0.553} \pm \underline{0.002}$	$\underline{0.558} \pm \underline{0.002}$	0.698 ± 0.002
GAT on heterogeneous net with A-, P- and I-type nodes	$\mathbf{0.742 \pm 0.006}$	$\mathbf{0.735 \pm 0.006}$	$\mathbf{0.742 \pm 0.006}$	$\mathbf{0.956 \pm 0.002}$

Advantage of HINs

Network models, particularly heterogeneous networks, offer a holistic approach that exploits the interconnectedness of academic entities and their features, thus enabling more robust and informative predictions. With the aim of classifying authors' primary fields of study, we investigated different models by progressively enriching the feature space available for classification, thereby enhancing predictive accuracy. For each model, we conducted five independent runs for 500 epochs each on the entire dataset spanning the years 2019–2022, with training, validation and test sets comprising the same authors (70%, 15% and 15% split, respectively). Table 3 reports the mean and standard deviation values for micro F1-score, macro F1-score, weighted F1-score, and ROC-AUC.

Initially, we examined the results of classification based solely on author attributes, excluding any structural information derived from their relationships. We trained a simple MLP model with one hidden layer using the Adam optimization algorithm (Kingma and Ba 2017) with full batch size and learning rate set to 0.01. Our findings revealed that author attributes alone were inadequate for effective classification. We observed a marginal enhancement over a dummy model—representing completely random classification aligned with class distribution—particularly in the metrics accommodating class imbalance.

Subsequently, we incorporated structural information by modeling the data using a two-layer GAT over a homogeneous network, with Adam optimization algorithm and learning rate equal to 0.01. This network featured a single node type (Author) and a single relationship type, wherein a link exists between two authors if they are co-authors, affiliated with the same institution, or one cited the other. Table 3 shows a substantial increase in performance, justifying the adoption of a graph structure-based model.

Nevertheless, the proposed model still overlooks the various semantics inherent in relationships, each contributing distinctively to the classification task. Consequently, we explored a second network model—a first HIN model, following the definition $|A| + |R| > 2$—maintaining authors as unique node type but distinguishing between the 3 relationship types. We kept the same optimization function and hyperparameters as the homogeneous model, while making the architecture explicitly discerning the contribution of different types of relationships. This enrichment led to a further performance improvement, although it neglected information embedded in other node types,

specifically Papers and Institutions. The best results are achieved by a HIN model—an enriched HIN model, following the definition $|A| > 2$ & $|R| > 2$ which featured 3 node types, 3 edge types, and 3 meta-path types, as outlined in the "Data" Section. The optimization algorithm and hyperparameters are the same as the previous model.

The semantic richness resulting from modeling multiple relationships and more specifically multiple node types, along with their specific information content, proved to be beneficial for the given task. As a consequence, we shall focus our study on graph continual learning on HINs using the latest and richest proposed model of heterogeneous GAT as an upper bound of our incremental model.

Competing methods

As we previously discussed (cf. "Introduction" and "Related work" Sections), direct competitors for DyHANE in GNN-based continual learning on HINs are missing or unable to accommodate dynamic attributes.

Having discussed the benefits offered by network models with multiple node and edge types, we compared our proposed incremental model with two baselines, the Retrained and Online models, on our multi-class classification task to assess the effectiveness and efficiency of the proposed approach. More specifically, we trained DyHANE at each timestamp on $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$, i.e., on a training set comprising the detected nodes affected by changes and the nodes stored in the experience buffer, and compared it with the following methods:

- *Retrained*GNN, having the same architecture as DyHANE and $\mathcal{T}^t = \mathcal{V}^t$, i.e., trained on all nodes existing in the network at the given timestamp;
- *Online*GNN, having the same architecture as DyHANE and $\mathcal{T}^t = \mathcal{V}^t - \mathcal{V}^{t-1}$, i.e., trained on changed nodes only.

The two baselines employ the same GNN architecture as the proposed approach and reflect an upper bound in performance and execution time, respectively.

Experimental settings

We conducted our experiments on a Docker VM with 1x3090 GPU, 128GB of RAM, and 64 dedicated cores, on a 2x56-core Intel(R) Xeon(R) Gold 6258R CPU, with 256GB RAM and two NVIDIA GeForce RTX3090s, and OS Ubuntu Linux 22.04 LTS.

Although our model can handle different attribute vector sizes for different node types, we performed dimensionality reduction via Principal Component Analysis (Pearson 1901) on the attribute vectors and set all node dimensions to 128. As previously mentioned, we used a GATv2 (Brody et al. 2021) architecture as our GNN model for all experiments. Specifically, we implemented a two-layer GAT with hidden channels dimension set to 64 and out channel dimension set to 8 as the number of classes in our dataset (cf. "Data" Section). We employed a weighted cross entropy loss function, which addresses the class imbalance by assigning higher weights to underrepresented classes during the training procedure. We trained the model using the Adam optimization algorithm with full batch size, for 500 epochs for 5 independent runs, and we

set the optimal hyperparameters for the learning process via grid search algorithm in the range of {0.05, 0.01, 0.005, 0.001} for the learning rate, {0.005, 0.001, 0.0005, 0.0001} for the weight decay and {0.2, 0.3, 0.4, 0.5, 0.6, 0.7} for the dropout rate in combination with different configurations of the memory buffer. We carried out a grid search over {256, 512, 768, 1024, 2048} for the buffer size $|\mathbf{B^t}|$ and tested two different buffer compositions, storing the experience nodes uniformly w.r.t. their type or according to the importance of their type. Regarding the latter case, for the update of the memory buffer $\mathcal{B}^t$, we employed *top-k* = 64 nodes, for each type, to compute the type importance and select the number of nodes of that type to be stored as experience (cf. "Update of the memory buffer" Section). We found the best configuration as that corresponding to 0.01 as learning rate, 0.0001 as weight decay, 0.3 as dropout, and buffer size equal to 768 with different rates of nodes for each type summing to 1 (0.125 A, 0.77 P, 0.105 I).

Results

We organize the presentation of our experimental results into four parts. We first compare our best model (cf. "Experimental settings" Section)—hereinafter referred to as DyHANE$_{\mathcal{B}768}$—with the two baselines in two different scenarios, followed by a sensitivity analysis of the (hyper)parameters of DyHANE$_{\mathcal{B}768}$. We then delve into different configurations of our framework, focusing on different sizes and compositions of the memory buffer and performing an ablation study of the memory buffer. Finally, we discuss general remarks on our framework variants.

Comparison with baselines

For a fair and meaningful comparison, we compare the results obtained by our best model and the baselines in two different scenarios, corresponding to two different test sets equal for all models:

- 'C': test set consisting only of changed nodes, corresponding to the Online model's test set;
- 'C+U': test set consisting of a set of both changed and unchanged nodes, selecting a subset of nodes that were neither identified by our algorithms as nodes affected by changes (Algorithm 2) nor as experience to be stored in the buffer (Algorithm 3).

In the first scenario, we assess the effectiveness of our method to learn new patterns; in the second scenario, we assess the capacity of our method to retain unchanged patterns while integrating new ones.

It should be noted that, in both scenarios, the changed nodes in the test set are new nodes of the last increment, e.g., they are 2022 nodes not existing in 2019-2021; the unchanged nodes are nodes existing before the last increment and not modified based on events of 2022. Furthermore, we examined the behavior of incremental models over multiple increments, i.e., 2019–2020, 2020–2021, and 2021–2022, to investigate performance degradation under sequential (continuous) application of the models. We show the results for different scenarios and number of increments under multiple evaluation metrics in Table 4, specifying the total execution time for each model. The total execution time corresponds solely to training time for baselines, while for our model it

Table 4 Comparison with baselines using the same test set; scenario 'C' refers to the test set comprising only changed nodes, scenario 'C+U' refers to the test set comprising both changed and unchanged nodes. Incremental models are tested after 1 and 3 network increment(s). Time is expressed in seconds; for DyHANE$_{\mathcal{B}768}$, it consists of both the detection of nodes affected by changes and training time; for the baselines it corresponds solely to training time. The best results are highlighted in bold, the second best are underlined

Scen.	#Incr.	Model	F1-micro	F1-macro	F1-weighted	ROC-AUC	Time
'C'	–	*Retrained*GNN	**0.742 ± 0.006**	**0.735 ± 0.006**	**0.742 ± 0.006**	**0.956 ± 0.002**	159.9
	1	*Online*GNN	0.431 ± 0.001	0.422 ± 0.001	0.429 ± 0.001	0.692 ± 0.001	**65.5**
	3	*Online*GNN	0.398 ± 0.001	0.386 ± 0.001	0.397 ± 0.001	0.651 ± 0.001	65.6
	1	DyHANE$_{\mathcal{B}768}$	<u>0.664</u> ± <u>0.001</u>	<u>0.659</u> ± <u>0.001</u>	<u>0.664</u> ± <u>0.001</u>	<u>0.932</u> ± <u>0.001</u>	<u>78.9</u>
	3	DyHANE$_{\mathcal{B}768}$	0.661 ± 0.001	0.655 ± 0.001	0.660 ± 0.002	0.918 ± 0.001	80.0
'C+U'	–	*Retrained*GNN	**0.712 ± 0.006**	**0.709 ± 0.006**	**0.712 ± 0.006**	**0.953 ± 0.003**	159.9
	1	*Online*GNN	0.285 ± 0.004	0.277 ± 0.003	0.281 ± 0.003	0.645 ± 0.002	**65.6**
	3	*Online*GNN	0.259 ± 0.004	0.252 ± 0.003	0.256 ± 0.003	0.592 ± 0.003	**65.6**
	1	DyHANE$_{\mathcal{B}768}$	<u>0.676</u> ± <u>0.001</u>	<u>0.671</u> ± <u>0.001</u>	<u>0.676</u> ± <u>0.001</u>	<u>0.940</u> ± <u>0.001</u>	<u>78.9</u>
	3	DyHANE$_{\mathcal{B}768}$	0.674 ± 0.001	0.669 ± 0.001	0.674 ±0.001	0.937 ± 0.001	79.4

comprises both the detection of nodes affected by changes and training time. Time (in seconds) is the average over 5 independent runs of the model on a single increment; in the case of multiple increments, it is an average over all increments.

Comparing the overall performance of each model in the two different scenarios, we note that, as evidenced in Table 4, only DyHANE$_{\mathcal{B}768}$ improves performance on the 'C+U' scenario, i.e., when testing on a sample comprising both changed and unchanged nodes; the *Online* model fails to make correct predictions for unchanged nodes, while apparently the *Retrained* model suffers from the reduction of its training set.

To gain a comprehensive understanding of models' performances, we consider multiple evaluation metrics, including F1 measures and Area Under the Receiver Operating Characteristic Curve (ROC-AUC), which provides a summary measure of the model's ability to rank instances. A high ROC-AUC suggests good discrimination ability, while a lower F1-score is related to class imbalance; this applies especially to the macro-F1 score which treats all classes equally. Table 4, especially F1-macro exhibiting the lowest values for all models in each scenario, can be explained based on the imbalance of our dataset, which features unbalanced increments and is not constrained to a constant number of changes over time.

Our framework always achieves worse performance w.r.t. the *Retrained* model, due to the smaller size of the training set and the higher class imbalance at each increment. The weakened difference between the ROC-AUC values, w.r.t. the difference in F1-measures, confirms this hypothesis. Simultaneously, we observe to be more than twice faster in terms of execution time, considering that we include the calculation of the influenced node set in the elapsed time value.

On the contrary, DyHANE significantly outperform the Online GNN model in comparable time. We spot that it achieves the lowest performance on both scenarios, probably due to the smaller amount of data. Specifically, it dramatically underperforms when

tested on unchanged nodes, demonstrating the inadequacy of only initializing the GNN parameters to retain the knowledge of the past network.

We specify that the generation of the influenced node set $\mathcal{I}^t$ at each new timestamp requires a maximum of 8.0 seconds. As previously pointed out (cf. "Identification of influenced nodes" and "Computational complexity aspects" Sections), the identification of nodes affected by change relies on the calculation of incremental meta-paths. Meta-path processing occurs in parallel, and the longest meta-path type in our dataset is A–P–P–A with length four, thus requiring the Cartesian product between two (sub)sets of edges. More specifically, if the event to be processed is an edge of type A–P, we intersect it with all of type P–P and then with all of type A–P; if the event to be processed is of type P–P, we intersect it twice with all of type A–P. We should also consider that authors typically work in groups and not alone, so we are unlikely to reach the worst case of checking all pairs of edges. The resuming time of the memory buffer $\mathcal{B}^{t-1}$ and the parameters θ^{t-1} learned at the previous timestamp is negligible. Moreover, it is noteworthy that the computation of $\mathcal{B}^{t-1}$ with the Explainer (cf. "Update of the memory buffer" and "Computational complexity aspects" Sections) consistently concludes within a maximum time frame of 9.7 s.

We note that the execution time of our model in the case of multiple increments increases slightly. Recalling that the execution time reported in Table 4 is the average over the three increments and excludes the update of the memory buffer (which is computed while waiting for the next increment), the difference is due to processing several times, i.e., in multiple increments, some of the edges, and not just once as in the single-increment case. From the perspective of expressiveness, we notice only a slight degradation of the values in the case of multiple increments, which demonstrates the effectiveness of the proposed approach, especially in updating $\mathcal{B}^t$ from $\mathcal{B}^{t-1} \bigcup \mathcal{I}^t$, with $|\mathcal{B}^{t-1} \bigcup \mathcal{I}^t| \ll |G^t|$. *Online*GNN, conversely to our model, is significantly affected by multiple increments.

Finally, we observe remarkable stability of DyHANE$_{\mathcal{B}768}$ with respect to the baselines, as reflected by the low standard deviation in comparison with competitors.

Our proposed framework thus emerges as a promising trade-off between expressiveness and computational efficiency, positioning itself as a potential candidate for applications with limited or fixed computational budget. Achieving comparable performance w.r.t. our evaluation metrics on both scenarios, i.e., on both testing sets, we can assess that using the Explainer to keep the most important nodes with the experience reply strategy is effective in retain previous knowledge and that the proposed strategy to detect changes leveraging the semantics of meta-paths helps in integrate knew knowledge.

Analysis of the DyHANE variants

We investigated multiple variants of our model, differing in the size of the memory buffer $|B^t|$, to gain deeper insights into its impact on our node classification task. More precisely, we conducted an ablation study by removing the memory buffer (equivalently, buffer size equal to 0), and then floated the size in the set $\{256, 512, 768, 1024, 2048\}$. The specific size for each model can be easily identified in the subscript appended to its name. For each buffer size, we experimented two different compositions: number of

nodes for each type based on type importance (cf. Algorithm 3) and same number of nodes for each type. In order to distinguish models with the same memory buffer size but different compositions, we marked with *f* the models using a uniform composition for different types, where *f* stands for *fixed-size of node types* as opposed to dynamic computation of type importance. To summarize:

- DyHANE$_{\mathcal{B}0}$, with $\mathcal{T}^t = \mathcal{I}^t$, i.e., trained only on the set of nodes directly or indirectly affected by changes
- DyHANE$_{\mathcal{B}256}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 256$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 256 with floating composition w.r.t. different node types
- DyHANE$_{\mathcal{B}256f}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 256$ and $|A_{\mathcal{B}}| = |P_{\mathcal{B}}| = |I_{\mathcal{B}}|$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 256 with uniform composition w.r.t. different node types
- DyHANE$_{\mathcal{B}512}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 512$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 512 with floating composition w.r.t. different node types
- DyHANE$_{\mathcal{B}512f}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 512$ and $|A_{\mathcal{B}}| = |P_{\mathcal{B}}| = |I_{\mathcal{B}}|$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 512 with uniform composition w.r.t. different node types
- DyHANE$_{\mathcal{B}768}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 768$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 768 with floating composition w.r.t. different node types
- DyHANE$_{\mathcal{B}768f}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 768$ and $|A_{\mathcal{B}}| = |P_{\mathcal{B}}| = |I_{\mathcal{B}}|$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 768 with uniform composition w.r.t. different node types
- DyHANE$_{\mathcal{B}1024}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 1024$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 1024 with floating composition w.r.t. different node types
- DyHANE$_{\mathcal{B}1024f}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 1024$ and $|A_{\mathcal{B}}| = |P_{\mathcal{B}}| = |I_{\mathcal{B}}|$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 1024 with uniform composition w.r.t. different node types
- DyHANE$_{\mathcal{B}2048}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 2048$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 2048 with floating composition w.r.t. different node types
- DyHANE$_{\mathcal{B}2048f}$, with $\mathcal{T}^t = \mathcal{I}^t \bigcup \mathcal{B}^{t-1}$ and $|\mathcal{B}^{t-1}| = 2048$ and $|A_{\mathcal{B}}| = |P_{\mathcal{B}}| = |I_{\mathcal{B}}|$, i.e., trained on the set of nodes affected by changes and experience nodes stored in a buffer of cardinality equal to 2048 with uniform composition w.r.t. different node types

Table 5 Training set size and composition for baselines and variants of DyHANE, coupled with their mean execution time on the last network increment. The cardinalities of the training sets and running times are in ascending order. The training set discriminates between the different node types, (A)uthors, (P)apers and (I)nstitutions, with $|T| = |A| + |P| + |I|$. The training set of our models comprises both the nodes from $I \cup \beta$. Given the same buffer size, models marked with f have a different internal composition but same final training set size as their counterparts. The time is in seconds and refers to the best hyperparameters configuration of each model. For our models, it includes the detection of nodes affected by changes and the training time. Baselines are at the extremes

Model	|T|	|A|	|P|	|I|	Time (sec)
*Online*GNN	4611	2584	1563	464	65.6
DyHANE$_{\mathcal{B}0}$	5751	3409	1817	525	73.0
DyHANE$_{\mathcal{B}256}$	6007	3441	2014	552	74.4
DyHANE$_{\mathcal{B}256f}$	6007	3495	1902	610	74.4
DyHANE$_{\mathcal{B}512}$	6263	3483	2194	586	75.7
DyHANE$_{\mathcal{B}512f}$	6263	3580	1987	695	75.7
DyHANE$_{\mathcal{B}768}$	6519	3505	2408	606	78.9
DyHANE$_{\mathcal{B}768f}$	6519	3665	2073	781	78.9
DyHANE$_{\mathcal{B}1024}$	6775	3537	2605	633	86.3
DyHANE$_{\mathcal{B}1024f}$	6775	3751	2158	666	86.3
DyHANE$_{\mathcal{B}2048}$	7799	3665	3393	741	95.2
DyHANE$_{\mathcal{B}2048f}$	7799	4022	2499	1208	95.2
*Retrained*GNN	27402	15397	10174	1831	159.6

Table 6 Comparison of multiple DyHANE models in the 'C+U' scenario, with baselines at the extremities. A model is identified by the number shown as subscript in its name which refers to the size of the memory buffer, and the suffix *f* refers to a possible fixed composition of the buffer, with equal number of nodes for each type. Time is expressed in seconds and comprises both the detection of nodes affected by changes and training time

Model	F1-micro	F1-macro	F1-weighted	ROC-AUC	Time
*Online*GNN	0.285 ± 0.004	0.277 ± 0.003	0.281 ± 0.003	0.645 ± 0.002	65.6
DyHANE$_{\mathcal{B}0}$	0.561 ± 0.002	0.554 ± 0.001	0.559 ± 0.002	0.817 ± 0.002	73.0
DyHANE$_{\mathcal{B}256f}$	0.652 ± 0.002	0.644 ± 0.001	0.652 ± 0.001	0.880 ± 0.001	74.4
DyHANE$_{\mathcal{B}256}$	0.663 ± 0.002	0.656 ± 0.002	0.662 ± 0.002	0.891 ± 0.001	74.4
DyHANE$_{\mathcal{B}512f}$	0.665 ± 0.001	0.659 ± 0.001	0.666 ± 0.001	0.896 ± 0.001	75.7
DyHANE$_{\mathcal{B}512}$	0.671 ± 0.002	0.667 ± 0.001	0.671 ± 0.001	0.903 ± 0.001	75.7
DyHANE$_{\mathcal{B}768f}$	0.672 ± 0.002	0.668 ± 0.001	0.672 ± 0.002	0.919 ± 0.001	78.9
DyHANE$_{\mathcal{B}768}$	0.676 ± 0.001	0.671 ± 0.001	0.676 ± 0.001	0.940 ± 0.001	78.9
DyHANE$_{\mathcal{B}1024f}$	0.676 ± 0.001	0.671 ± 0.001	0.677 ± 0.001	0.941 ± 0.002	86.3
DyHANE$_{\mathcal{B}1024}$	0.678 ± 0.001	0.672 ± 0.002	0.678 ± 0.002	0.942 ± 0.001	86.3
DyHANE$_{\mathcal{B}2048f}$	0.680 ± 0.001	0.674 ± 0.001	0.681 ± 0.001	0.944 ± 0.002	95.2
DyHANE$_{\mathcal{B}2048}$	0.683 ± 0.002	0.677 ± 0.002	0.683 ± 0.002	0.949 ± 0.001	95.2
*Retrained*GNN	0.712 ± 0.006	0.709 ± 0.006	0.712 ± 0.006	0.956 ± 0.002	159.9

Table 5 provides for each model the total training set cardinality (which is the same for two models with equal memory buffer size) and detail for each node type, coupled with its execution time. The values refer to the last network increment (2022). We supplement the equivalent information for baselines to strengthen the comparison,

observing that all our models lie between the two. The table emerges sorted in ascending order of training set size and execution time.

Table 5 shows that the node type with the most impact on classifying authors' primary field of study is P, although it is not the target node type. This is due to the higher information content of Paper attributes, including abstracts. Node type P is followed by A and I, respectively, in lower ratio.

To gain deeper insights into the impact of heterogeneous nodes experience replay on our node classification task, we conducted a comprehensive assessment involving all the aforementioned configurations. The evaluation of the multiple variants of DyHANE, each under its best configuration, is referred to Table 6.

Table 6 shows a surprising stability in all the results of our models, as indicated by the low standard deviation. A low standard deviation typically denotes that the performance of the classifier is consistent across different classes.

We first performed an ablation study, removing the memory buffer from the training set, which thereby contains only the nodes affected by changes identified by Algorithm 2. We spotted that the model with buffer size equal to 0 records the lowest values among our models, but still considerably higher than the *Online* model. We can thus assess the effectiveness of the proposed approach in identifying the subset of nodes significantly related to changed nodes.

We then explored different sizes of the memory buffer. We observed that reducing the number of experience nodes leads to a degradation in the classification task while speeding up the training process, prompting to consider a trade-off. We spotted a satisfactory balance in correspondence to size 768. By increasing the buffer cardinality to 1024, the difference in scores is not adequate to justify the observed delay (we recall that a maximum of 8.0 s is required for the identification of nodes affected by changes); by reducing the buffer cardinality to 512, the saved time is not adequate to justify remarkable performance degradation for all evaluation metrics. We emphasize that although there are few nodes in the buffer, their identification based on Algorithm 3 has proven to be effective in selecting meaningful samples.

We noticed that the Explainer is successful also in calculating the importance of the contribution of different node types. More specifically, the comparison of pairs of models with the same buffer cardinality but different composition, i.e., models with and without f, shows that prioritizing one node type over another results in improved performance.

Sensitivity analysis

We performed sensitivity analysis of our main (hyper)parameters to ensure the robustness and reliability of the models, assessing the impact of variations in input parameters on model outputs and providing insights into the relative importance of different variables.

We first explored 100 DyHANE configurations varying the size of the memory buffer (for short, B_size) and two significant GAT's hyperparameters: dropout and learning rate, with weight decay (also known as L2 regularization) set to 0.0001 in all the experiments (cf. "Experimental settings" Section). The F1-weighted score for each triplet is shown in the scatter plot in Fig. 4.

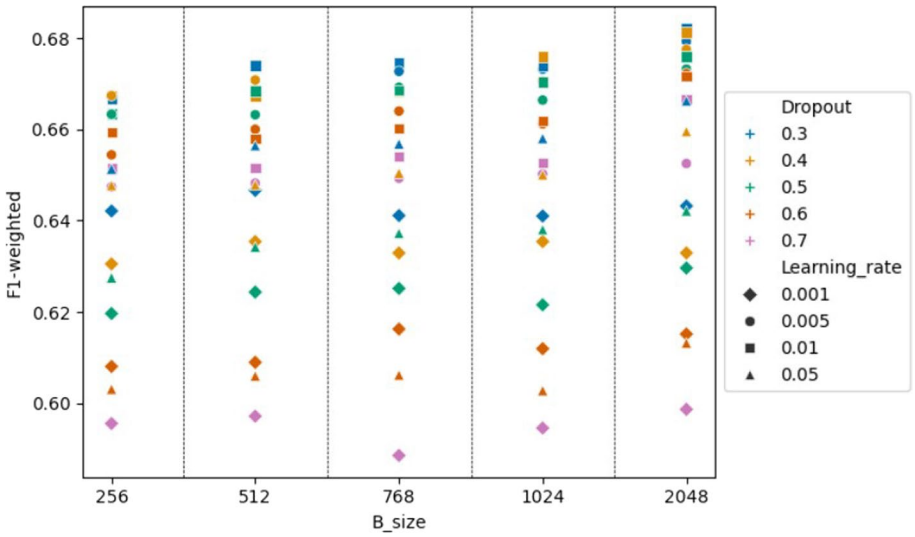


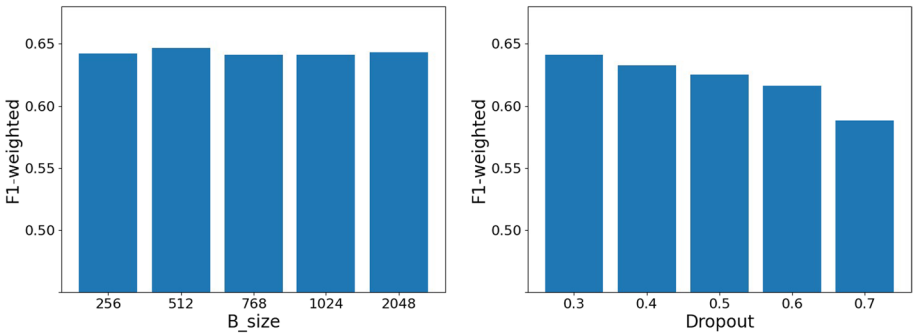
Fig. 4 Scatter plot of 100 DyHANE configurations. Floating the size of the memory buffer, we show the F1-weighted score based on dropout and learning rates. Different colors and symbols denote different dropout values and learning rates, respectively. The weight decay is set to 0.0001 in all the experiments. For visualization reasons, we removed from the plot the values with dropout 0.7 and learning rate 0.05, whose performance is below the plot limit

The combination of the highest learning rate and the highest dropout values (0.05 and 0.7, resp.) resulted in extremely poor performances, thus we preferred to remove them from the plot in order to improve readability. More generally, higher dropouts (0.7, 0.6 and even 0.5) and learning rates at the extremes (the highest value 0.05 and the lowest 0.001) lead to poor performance. For each buffer size, the highest scores correspond to lower values of dropout (0.3 or 0.4) and learning rate in the range [0.005, 0.01], hinting at the opportunity to limit regularization and balance convergence speed and stability during training. We recall that our training set at each increment is a subset of the entire network and may exhibit new patterns even completely different from previous ones. We therefore need low regularization values (both dropout and weight decay) for learning new patterns, but very low values result in the risk of overfitting, reason why we do not explore values below 0.3 for dropout and 0.0001 for weight decay.

We elaborate further on the behavior of single hyperparameters. Our analysis was performed by varying the value of a single parameter at time while maintaining the others to our best configuration (cf. "Experimental settings" Section). This involved our main (hyper)parameters, i.e., the size of the memory buffer and three significant GAT hyperparameters: dropout, learning rate and weight decay. Figure 5 shows the F1-weighted score for each combination.

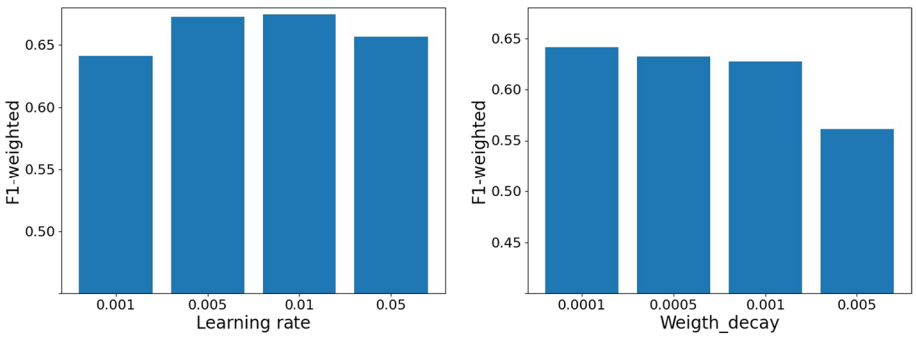
When varying the number of experience nodes (Fig. 5a), we observe small fluctuations on F1-weighted values. Indeed, low values of weight decay (specifically, 0.0001) and dropout (0.3 and 0.4), and learning rates in the range [0.005, 0.01] exhibited the best performance for all our models.

By varying the dropout (Fig. 5b), we note that performance degrades as the fraction of node or edge features set to zero increases during each training iteration, i.e., as the dropout increases. This is not always true in GAT-based models, so that the *Retrained*



(a) Learning rate: 0.01, dropout: 0.3, weight decay: 0.0001.

(b) B_size: 768, learning rate: 0.01, weight decay: 0.0001.



(c) B_size: 768, dropout: 0.3, weigth decay: 0.0001.

(d) B_size: 768, dropout: 0.3, learning rate: 0.01.

Fig. 5 Sensitivity analysis of our main (hyper)parameters w.r.t. DyHANE best configuration (cf. "Experimental settings" Section). The x-axis indicates the investigated parameter, while the y-axis indicates the corresponding F1-weighted value. The other parameters are fixed for each barchart

model uses a dropout of 0.6. This might be due to the reduced size of the training set, since higher dropout values with fewer examples may introduce excessive regularization, hindering the model's ability to effectively capture the complex relational structures underlying the network.

Concerning the learning rate (Fig. 5c), we identified the optimal values in the range [0.005, 0.01], striking a balance between convergence speed and stability during training.

As regards the weight decay (Fig. 5d), we note that performance degrades as large weights are more heavily penalized during training, i.e., as the weight decay increases. We found the optimal value at 0.0001, a relative low value which enables the model to learn complex patterns from the data without being overly constrained by regularization. Further decreasing this value may increase the risk of overfitting, since we are dealing with a reduced dataset corresponding to an increment.

Remarks on regularization strategies Note that the number of influenced nodes, varying at each timestamp, is usually significantly larger than the size of the experience buffer, which is instead of fixed size. To cope with the overfitting problem caused by the small number of replayed nodes, a commonly used strategy is to add an extra regularization term to the loss function to guarantee that the distance between the current and the

historical model parameters will not deviate further; more specifically, different importance is given to different parameters to keep small the changes of GNN parameters that are important to the past network while the others can be updated more drastically. We found that the Elastic Weight Consolidation (EWC) regularization-based method (Kirkpatrick et al. 2016) improve performance in all models so, for the sake of model comparison, we avoid to integrate any additional regularization term in our current formulation.

Conclusion

Incremental GNNs and HINs represent a widely unexplored field of research. In this regard, we presented DyHANE, a GNN-based incremental framework capable of handling multiple types of nodes and relationships in a dynamic scenario. DyHANE adapts to changes in network topology and node attributes efficiently, by updating GNN parameters and training on a sample of the network. DyHANE is comprised of two main modules, the one for identifying a reduced set of nodes affected by changes and the other for identifying a reduced set of nodes to be used as experience node replay. On a multi-class classification task, we demonstrated the advantages of modeling offered by HINs and showed the ability of our framework to achieve good performance on both changed and unchanged nodes, w.r.t. the GAT retrained on the entire network, in a comparable time to online GAT that suffers of performance degradation on unchanged nodes.

As further developments, we are interested in investigating and comparing the results obtained in various dynamic scenarios, including a stable distribution of classes, skew, abrupt shift, and concept drift. We plan to test DyHANE on new datasets, such as predicting the primary interest(s) of users in a social network, and on new tasks, like predicting the number of papers'/authors' citations, or co-authorship relationships. The flexibility of our pipeline will also enable further optimization of the proposed algorithms for detecting nodes affected by changes and experience nodes on heterogeneous graphs of growing size.

Author contributions
All authors contributed to the conceptualization, methodology, validation, and writing of the manuscript. LM also collected the data, developed the software, and analyzed the results.

Funding
LM was funded by the PON FSE-FESR Ricerca e Innovazione 2014-2020 (PON R &I), Azione I.1 "Dottorati Innovativi con caratterizzazione industriale", Avviso n. 1233, July 30, 2020. AT was partially funded by PNRR M4C2 1.3 S9 "Future Artificial Intelligence Research (FAIR)" (CUP H23C22000860006) and by PRIN "AWESOME: Analysis framework for WEb3 SOcial MEdia" (CUP H53D23003550006). This work was supported by the French National Centre for Space Studies (CNES), as part of the FRESA (Fouille de texte pour la REcherche de consensus sur la Sécurité Alimentaire) project, APR TOSCA 2022.

Data availability
Python code for the proposed methods, as well as the network datasets, is publicly available at https://github.com/lilymart/DyHANE.

Declarations

Ethics approval and consent to participate
No ethical concerns apply.

Consent for publication
The authors provide their consent for publication.

Competing interests
The authors declare that they have no competing interest and no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter discussed in this manuscript.

Received: 18 December 2023 Accepted: 2 June 2024
Published online: 05 July 2024

References

Brody S, Alon U, Yahav E (2021) How attentive are graph attention networks? CoRR. arXiv:2105.14491
Chen J, Ma T, Xiao C (2018) Fastgcn: fast learning with graph convolutional networks via importance sampling. CoRR arXiv:1801.10247
Dong Y, Chawla NV, Swami A (2017) etapath2vec: scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 135–144
Du L, Wang Y, Song G, Lu Z, Wang J (2018) Dynamic network embedding: an extended approach for skip-gram based network embedding. In: IJCAI, vol. 2018, pp 2086–2092
Khoshraftar S, An A (2022) A survey on graph representation learning methods. CoRR. https://doi.org/10.48550/arXiv.2204.01855
Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. CoRR. arXiv:1412.6980
Kirkpatrick J, Pascanu R, Rabinowitz NC, Veness J, Desjardins G, Rusu AA, Milan K, Quan J, Ramalho T, Grabska-Barwinska A, Hassabis D, Clopath C, Kumaran D, Hadsell R (2016) Overcoming catastrophic forgetting in neural networks. CoRR arXiv:1612.00796
Ma Y, Guo Z, Ren Z, Tang J, Yin D (2020) Streaming raph neural networks. In: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, pp 719–728
Martirano L, Zangari L, Tagarelli A (2022) Co-mlhan: contrastive learning for multilayer heterogeneous attributed networks. Appl Netw Sci 7(1):65. https://doi.org/10.1007/S41109-022-00504-9
McCloskey M, Cohen NJ (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, Academic Press, vol. 24, pp 109–165. https://doi.org/10.1016/S0079-7421(08)60536-8
Pearson K (1901) Liii. On lines and planes of closest fit to systems of points in space. Lond Edinb Dublin Philos Mag J Sci 2(11):559–572
Peng H, Yang R, Wang Z, Li J, He L, Yu PS, Zomaya AY, Ranjan R (2022) Lime: low-cost and incremental learning for dynamic heterogeneous information networks. IEEE Trans Comput 71(3):628–642. https://doi.org/10.1109/TC.2021.3057082
Perini M, Ramponi G, Carbone P, Kalavri V (2022) Learning on streaming graphs with experience replay. In: Proceedings of the 37th ACM/SIGAPP symposium on applied computing, pp 470–478
Priem J, Piwowar HA, Orr R (2022) Openalex: a fully-open index of scholarly works, authors, venues, institutions, and concepts. CoRR. https://doi.org/10.48550/ARXIV.2205.01833
Reimers N, Gurevych I (2019) Sentence-bert: sentence embeddings using siamese bert-networks. CoRR. arXiv:1908.10084
Shang J, Qu M, Liu J, Kaplan LM, Han J, Peng J (2016) Meta-path guided embedding for similarity search in large-scale heterogeneous information networks. CoRR arXiv:1610.09769
Shen Z, Ma H, Wang K (2018) web-scale system for scientific knowledge exploration. CoRR. arXiv:1805.12216
Shi B, Weninger, T (2014)ining interesting meta-paths from complex heterogeneous information networks. In: 2014 IEEE international conference on data mining workshop, IEEE, pp 488–495
Sundararajan M, Taly A, Yan Q (2017) Axiomatic attribution for deep networks. CoRR. arXiv:1703.01365
Trivedi R, Farajtabar M, Biswal P, Zha H (2019) Dyrep: learning representations over dynamic graphs. In: International conference on learning representations
Wan G, Du B, Pan S, Haffari G(2020) einforcement learning based meta-path discovery in large-scale heterogeneous information networks. In: Proceedings of the Aaai conference on artificial intelligence, vol. 34, pp 6094–6101
Wang J, Song G, Wu Y, Wang L (2020) Streaming graph neural networks via continual learning. CoRR arXiv:2009.10951
Wang J, Zhu W, Song G, Wang L (2022) Streaming graph neural networks with generative replay. In: Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining, pp 1878–1888
Wei X, Liu Z, Sun L, Yu, PS (2018) Unsupervised meta-path reduction on heterogeneous information networks. arXiv preprint arXiv:1810.12503
Xie Y, Ou Z, Chen L, Liu Y, Xu K, Yang C, Zheng Z (2021) Learning and updating node embedding on dynamic heterogeneous information network. In: Proceedings of the 14th ACM international conference on web search and data mining, pp 184–192
Xue H, Yang L, Jiang W, Wei Y, Hu Y, Lin Y (2020) Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal RNN. CoRR arXiv:2004.01024
Yang L, Xiao Z, Jiang W, Wei Y, Hu Y, Wang H (2020) Dynamic heterogeneous graph embedding using hierarchical attentions. In: Advances in information retrieval: 42nd European conference on IR research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42, Springer, pp 425–432
Zhou F, Cao C (2021) Overcoming catastrophic forgetting in graph neural networks with experience replay. In: Proceedings of the AAAI conference on artificial intelligence, vol. 35, pp 4714–4722
Zhou F, Xu X, Li C, Trajcevski G, Zhong T, Zhang K (2020) A heterogeneous dynamical graph neural networks approach to quantify scientific impact. CoRR arXiv:2003.12042

Publisher's Note