Lifelong Learning in Evolving Graphs with Limited Labeled Data and Unseen Class Detection

Lukas Galke, Iacopo Vagliano, Benedikt Franke, Tobias Zielke, Ansgar Scherp

***Abstract*—Large-scale graph data in the real-world are often dynamic rather than static. The data are changing with new nodes, edges, and even classes appearing over time, such as in citation networks and research-and-development collaboration networks. Graph neural networks (GNNs) have emerged as the standard method for numerous tasks on graph-structured data. In this work, we employ a two-step procedure to explore how GNNs can be incrementally adapted to new unseen graph data. First, we analyze the verge between transductive and inductive learning on standard benchmark datasets. After inductive pre-training, we add unlabeled data to the graph and show that the models are stable. Then, we explore the case of continually adding more and more labeled data, while considering cases, where not all past instances are annotated with class labels. Furthermore, we introduce new classes while the graph evolves and explore methods that automatically detect instances from previously unseen classes. In order to deal with evolving graphs in a principled way, we propose a lifelong learning framework for graph data along with an evaluation protocol. In this framework, we evaluate representative GNN architectures. We observe that implicit knowledge within model parameters becomes more important when explicit knowledge, i.e., data from past tasks, is limited. We find that in open-world node classification, the data from surprisingly few past tasks are sufficient to reach the performance reached by remembering data from all past tasks. In the challenging task of unseen class detection, we find that using a weighted cross-entropy loss is important for stability.**

***Index Terms*—Lifelong Learning, Open-World Learning, Node Classification, Unseen Class Detection, Evolving Graphs, Graph Neural Networks, Learning with Limited Labeled Data.**

I. Introduction

A recent study on shortcut learning [1] has brought up a compelling argument that evaluating learning systems on standard benchmarks is not sufficient to assess their quality. Rather, it is necessary to evaluate their performance in more challenging testing conditions, such as real-world scenarios. In such scenarios, the assumption of independent and identically distributed data is immediately violated. This aspect is even more crucial when dealing with graph data. Apart from raw conditional distribution, $p(y|\boldsymbol{x})$, also the graph structure evolves over time. Even worse, the set of classes itself, Y, might change over time with the emergence of entirely new classes. Thus, building learning systems for graphs that can work in real-world applications is particularly challenging. At the same time, having well-performing learning systems for graphs is extremely valuable because the graph representation is versatile and needed in many applications [2].

Graph neural networks [3] (GNNs) have emerged as state-of-the-art methods in numerous tasks on graph-structured data such as vertex classification [4]–[7], graph classification [8], link prediction [9], and unsupervised vertex representation learning [10].

An intriguing property of GNNs is that they are capable of inductive learning [5]. An inductive model for graph data only depends on the vertex features and the graph structure given by its edges. In many cases, such as [5], [11], [12], this is a major advantage over models that rely on a static vertex embedding [13], which would need to be retrained [14] as soon as new data arrives. This is known as the transductive learning scenario [2]. In contrast, inductively trained GNNs can be applied to new data – or even a different graph – without any retraining because of their property of only requiring vertex features and edges.

However, being able to apply the same model to unseen data also comes with challenges that have not been sufficiently addressed in the literature so far. Let us assume that we have a vertex classification model and new data streams over time, i. e., new edges and vertices arrive; then even *new classes* may arise. This raises several research questions: Do we need to retrain the model, and when do we need to retrain it? How much past data should be preserved for retraining? Is it helpful to preserve implicit knowledge within the model parameters or should we retrain from scratch? How much new labeled data is needed for stable training? Lastly, and most importantly, how can we automatically detect in an unsupervised manner if a new class has arrived in the dataset.

To answer these questions, we frame the problem as an instance of lifelong machine learning [15]–[17]. In lifelong learning, as illustrated in Figure 1, the learner has to perform a sequence of tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t$, and may use knowledge $\mathcal{K}$ gained in previous tasks to perform task $\mathcal{T}_t$. In our case, each task consists of classifying vertices given an attributed graph. Knowledge $\mathcal{K}$ may be stored explicitly (the training data of past tasks) or implicitly within the model parameters. A particular challenge of lifelong learning in the context of graph data is that vertices cannot be processed independently because models typically take connected vertices into account. We also consider the challenge that the set of classes in task $\mathcal{T}_t$ differs from classes in previous tasks, which is known as the *open-world classification* [17] problem.

We address these challenges by introducing a new incremental training method that retrains the model for each task.

L. Galke (0000-0001-6124-1092) was with Kiel University and ZBW – Leibniz Information Centre for Economics, Kiel, Germany while conducting the research.

I. Vagliano (0000-0002-3066-9464) is with Amsterdam University – University Medical Centre, the Netherlands.

A. Scherp (0000-0002-2653-9245), B. Franke, and T. Zielke are with Ulm University, Germany.

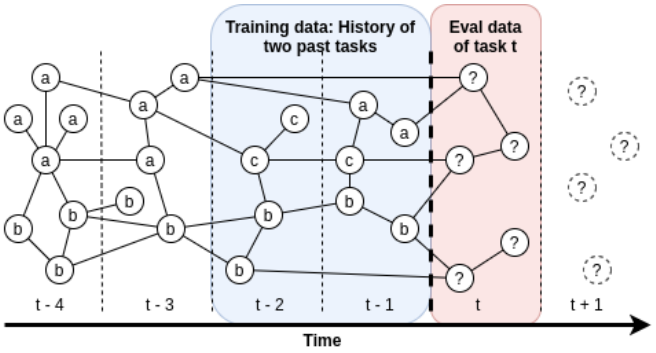

Fig. 1. Lifelong Open-World Node Classification. At each time t the learner has to classify new vertices of task $\mathcal{T}_t$ (red). The learner may use knowledge from previous tasks to adapt to the current task, eventually cut off by a history size (blue). The current task might come with previously unseen classes, e. g., see class "c" that emerged only at task $t-2$ and was subsequently added to the class set. After evaluating each task $\mathcal{T}_t$, we continue with task $\mathcal{T}_{t+1}$.

In our experiments, we thoroughly evaluate representative and scalable GNN architectures and a graph-agnostic multi-layer perceptron. We use a *history size* that limits the amount of past data (called here: *explicit knowledge*) available for training and compare *limited history-size* retraining against unlimited *full-history* retraining. Furthermore, we compare reusing model parameters from previous tasks (*warm restart*) against retraining from scratch (*cold restarts*) to analyze the influence of *implicit knowledge*.

Apart from this lifelong learning scenario, we consider two more scenarios that are motivated by challenges of real-world graph data. In the one case, we limit the number of training vertices that come with labels. The motivation for this experiment is that gathering ground truth labels is expensive in real-world applications. In the other case, the models should actively detect instances of unseen classes, while classifying other instances into known classes. *Crisp* detection of unseen classes is a particularly challenging problem because it needs to be conducted unsupervised, e. g., by analyzing the distribution of the logits. In other methods [18], the classes are balanced. In real-world graphs, however, the class distribution is skewed and balancing it by under- or oversampling would be difficult in naturally evolving graphs because vertices are not independent from each other. Thus, we propose to weight the binary-cross entropy loss function such that the resulting logit distribution is more amenable for unsupervised unseen class detection.

We close our experiments with an ablation study to confirm that incremental training is necessary to apply GNNs on evolving graphs as often found in the real-world.

To facilitate our analyses, we provide three new datasets for lifelong learning; one co-authorship and two citation graph datasets with different degrees of changes in the class set. On these datasets, we have experimented with 48 different incremental training configurations, namely 6 architectures $\times$ 4 history sizes $\times$ cold restarts and warm restarts, which we evaluate on three new datasets. Then we use the best performing base model and the most challenging dataset to investigate limited labeled data with 8 different label rates, and unseen class detection with 4 history sizes.

In summary, the research contributions made by our paper are the following results:

- Incremental training is a simple yet effective technique to tackle lifelong learning on graphs. Data from only few past tasks are sufficient to maintain a high level of accuracy that matches the accuracy of the same model retrained using all past data.
- Reusing parameters requires less data. When labeled data is limited, parameter reuse becomes even more important.
- Unsupervised, crisp detection of unseen classes requires training with weighted cross-entropy. Using less data from the past in combination with warm restarts is beneficial for unseen class detection.

These insights have direct consequences for using GNNs in practical applications. It allows to decide how much historical data should be kept to maintain a certain performance versus having memory available in the GPUs. This is an important criterion that influences which GNN methods are applicable [12], [19]. We publicly provide the datasets as well as the evaluation framework to extend our experiments and accelerate research in lifelong learning on graphs.

This paper extends our earlier experiments (IJCNN 2021 conference [20] and ICLR 2019 workshop paper [21]) by two new variants of the problem: learning with limited labeled data and unseen class detection.

In Section II, we discuss the related works and we provide a problem statement in Section III. We explain our proposed training procedure, new measure to determine an optimal history size for graphs, and the employed GNN methods in Section IV. Our datasets are described in Section V, while the results of our experiments are reported in Section VI. We discuss the results in Section VIII, before concluding.

II. Related Work

We discuss the works in lifelong machine learning and especially lifelong learning on graphs. Subsequently, we discuss methods for evolving graphs, out-of-distribution detection, as well as methods regarding history sizes.

a) Lifelong learning: Similarly to us, Wang et al. [22] decompose lifelong (machine) learning into the subproblems of rejecting unknown instances, classifying accepted instances, and reducing the cost of learning. In 2013, Ruvolo and Eaton [23] introduced a lifelong learning algorithm with convergence guarantees that employs multi-task learning such that later tasks can improve earlier tasks. In 2016, Fei et al. [16] analyzed SVMs in a lifelong learning setting and introduce cumulative learning. Cumulative learning relates to our approach, as we consider that some data is shared among the tasks. However, we further investigate *how much* past data is necessary to retain accuracy compared to a fully-cumulative approach. Lopez-Paz & Ranzato [24] have introduced a gradient episodic memory framework for the image domain, where examples can be processed independently, and tackle the catastrophic forgetting problem, i. e., the loss of previously learned information when new information is learned [25]. For an overview of lifelong learning, we refer to a recent textbook [17].

b) Lifelong Learning on Graphs: Related works on lifelong learning *on graphs* are very limited. Concurrent with our research, there is one recent work by Wang et al. [26] that also use GNNs on lifelong learning problems. The authors focus on catastrophic forgetting and explore a method to cast vertex classification as a graph classification task by transforming each vertex into a feature graph. Thus, vertices become independent such that they can follow the streaming setup from [24].

c) Graph Neural Networks: Dwivedi et al. [27] distinguish between isotropic and anisotropic GNN architectures. In isotropic GNNs, all edges are treated equally. Apart from graph convolutional networks [4], examples of isotropic GNNs are GraphSAGE-mean [5], DiffPool [8], and GIN [28]. In anisotropic GNNs, the weights for edges are computed dynamically. Instances of anisotropic GNNs include graph attention networks [6], GatedGCN [29], and MoNet [30]. There are further approaches, which have been specifically proposed to scale GNNs to large graphs. These approaches fall into two categories: sampling [5], [12], [31], [32], and separating neighborhood aggregation from the neural network [19], [33], [34]. From each of these four categories (anisotropic versus isotropic GNNs, and preprocessing versus sampling), we select one representative for our experiments (see Section IV-E).

d) Evolving Graphs: Different methods for *evolving graphs* have been proposed. Those include dynamic embedding methods [35], [36], autoencoder-based methods [37], [38], GNNs for graphs with fixed vertex set [39]–[45], and inductive GNN methods that can deal with previously unseen vertices [11], [46]. These methods focus on the case with dynamic outputs. That means a vertex can be in class a at time t and in class b in time $t+1$. However, in our case the output variable is static but the graph as a whole is evolving.

e) Unseen Class Detection: Numerous approaches have been proposed for out-of-distribution detection. A key challenge is that the softmax activation, often used as final layer for classification, leads to highly confident mispredictions even when the input data is far away from the training distribution. To address this, Liang et al. [47] resort to temperature scaling, while Lee et al. [48] propose to use Mahalanoubis distance. Both combined with dedicated preprocessing of the input. Macedo et al. [49], [50] replace the softmax activation by an entropy-aware IsoMax activation. Other approaches rely on explicit outlier data that can be used for training [51], [52]. However, we are particularly interested in methods that emit a crisp decision whether the classification of an instance should be rejected. In that regard, there are several approaches for detecting new classes with classic machine learning methods [16], [53], [54]. Wu et al. [55] have used variational graph autoencoders for uncertain vertex representation learning. They generate multiple versions of features and test the certainty of a vertex belonging to a known class. In Deep Open Classification (DOC) [18], the authors propose the final softmax activation of a neural network by elementwise sigmoid activation. Then, they derive a threshold for unseen class detection. Their experiments on datasets with balanced classes indicate that DOC is preferable over OpenMax [54] and cbsSVM [16]. Thus, we employ a DOC-inspired module on-top of graph neural networks for unseen class detection.

f) History sizes in Data Streams: Regarding finding the optimal *history size in data streams*, Fish and Caceres [56] treated the window size as a hyperparameter and proposed an optimization algorithm which requires multiple runs of the model. This is a rather costly procedure and the study does not yield insights on how much predictive power can be preserved when selecting a near-optimal but much smaller, and thus more efficient, window size. Other works, e. g., [57], indicate that smaller history sizes might be beneficial in some scenarios. However, there is no systematic study of the influence of history sizes in lifelong learning on graphs.

g) Summary: To summarize, lifelong learning on graphs is, so far, an unexplored topic. In particular, none of the discussed works analyzes the problem of *open-world classification* in graph data and how much past training data is necessary – or how few is enough – to maintain good predictive power.

III. Problem Formulation

We define our problem of open-world classification of graph vertices as a form of lifelong learning [15], [17], [58].

Definition 1 (Lifelong Learning [17])**.** At any time t, the learner has performed a sequence of t learning tasks, $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t$ and has accumulated the knowledge $\mathcal{K}$ learned in these past tasks. At time $t+1$, it is faced with a new learning task $\mathcal{T}_{t+1}$. The learner is able to make use of past knowledge to help perform the new learning task $\mathcal{T}_{t+1}$.

We cast this definition into a lifelong graph learning problem by considering each task $\mathcal{T}_t := (\mathcal{G}_t, \boldsymbol{X}^{(t)}, \boldsymbol{y}^{(t)})$ to be a vertex classification task with graph $\mathcal{G}_t = (V_t, E_t)$, corresponding vertex features $\boldsymbol{X}^{(t)} \in \mathbb{R}^{|V_t| \times D}$, and vertex labels $\boldsymbol{y}^{(t)} \in \mathbb{N}^{|V_t|}$. We denote the set of all classes at time t as $\mathbb{Y}_t$. To ensure that past knowledge is helpful to perform $\mathcal{T}_t$, we impose $\mathcal{G}_{t-1} \cap \mathcal{G}_t \neq \emptyset$. We assume that the features and labels of the vertices do not change: $\boldsymbol{X}_u^{(t-1)} = \boldsymbol{X}_u^{(t)}, \boldsymbol{y}_u^{(t-1)} = \boldsymbol{y}_u^{(t)}$ if $u \in V_{t-1} \cap V_t$. Such changes can still be modeled by inserting a new vertex and removing the old one. The task is to predict the class labels for new vertices $V_t \setminus V_{t-1}$. Please note that these vertices may come with new unseen classes as $\mathbb{Y}_t$ may differ from $\mathbb{Y}_{t-1}$. Furthermore, we analyze the effect of a history size c, which limits the available past data. We call this past data *explicit knowledge*. In this case, we set $\tilde{\mathcal{T}}_t := (\tilde{\mathcal{G}}_t, \tilde{\boldsymbol{X}}^{(t)}, \tilde{\boldsymbol{y}}^{(t)})$ with $\tilde{\mathcal{G}}_t := \mathcal{G}_t \setminus (\mathcal{G}_1 \cup \mathcal{G}_2 \cdots \cup \mathcal{G}_{t-c-1})$, and remove corresponding features and labels to construct $\tilde{\boldsymbol{X}}_t$ and $\tilde{\boldsymbol{y}}_t$. Still, *implicit knowledge* acquired in past tasks, e. g., within the model parameters, may be used for task $\tilde{\mathcal{T}}_t$.

We further distinguish between several variants of this problem statement:

a) One-step setting: A simplified setting with only two tasks on standard benchmark datasets. We use this setup for experiments comparing transductive and inductive learning in Section VI-A.

b) Standard setting: We have a sequence of tasks that is based on real-world data. All past vertices have ground truth

labels, which can be used for training. Unseen classes are present and also considered in the evaluation, but no special methods are employed to actively detect unseen classes. This setting is reflected in the experiments of Section VI-B.

c) Limited labeled data: In this variant of the standard setting, only a fraction of ground truth labels becomes available for training, rather than the labels of all past vertices. This setting is reflected in the experiments of Section VI-C.

d) Unseen class detection: In the final variant, we seek to analyze the models' capabilities to actively detect unseen classes. In addition to the classification, the models now need to emit a binary decision per vertex, whether it belongs to a previously known class or to a new, unseen class. This setting is reflected in the experiments of Section VI-D.

IV. Methods

In the following, we introduce our proposed incremental training procedure as well as our method to harmonize window sizes. Finally, we briefly describe the base GNN models that we incrementally train for our experiments.

A. Incremental Training for Lifelong Learning on Graphs

Without loss of generality, we assume to have a finite sequence of T tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ and a model f with parameters θ. During the tasks, the graph changes, including its vertices, edges, as well as the set of classes. To address these changes, we explore a simple yet effective incremental training technique for adapting neural networks to new graph-structured tasks. As a preparation for task $\mathcal{T}_{t+1}$, we retrain f on the labels of $\mathcal{T}_t$ to obtain $\theta^{(t)}$. Whenever l new classes appear in the training data, we add a corresponding amount of parameters to the output layer of $f^{(t)}$. Therefore, we have $|\theta^{(t)}_{\text{output weights}}| = |\theta^{(t-1)}_{\text{output weights}}| + l$ and $|\theta^{(t)}_{\text{output bias}}| = |\theta^{(t-1)}_{\text{output bias}}| + l$. Those parameters that are specific to new classes are newly initialized. For the other parameters, we consider two options in our incremental training procedure: *warm restarts* and *cold restarts*. With *cold restarts*, we re-initialize $\theta^{(t)}$ and retrain from scratch. In contrast, when using *warm restarts*, we initialize the parameters for training on task $\mathcal{T}_t$ with the final parameters of the previous task $\theta^{(t-1)}$. Algorithm 1 outlines our incremental training procedure.

B. Self-Detection of Unseen Classes

A successful model for lifelong learning would not only classify new data into known classes, but also detect when an instance belongs to a previously unseen class. We seek to develop a generic method that is not specific to any particular GNN architecture. Thus, we take inspiration from the Deep Open Classification (DOC) [18] approach that has been proposed for text classification. The key idea is to replace the final softmax activation by elementwise sigmoid activation. Hence, the training objective becomes binary cross-entropy rather than categorical cross-entropy. Finally, a threshold is necessary to emit a crisp decision at test time. When the output for all classes fall below the threshold, the classification of that instance gets rejected.

Algorithm 1: Incremental training for lifelong graph learning under cold-start vs. warm-start condition

Input : Sequence of tasks $\tilde{\mathcal{T}}_0, \cdots, \tilde{\mathcal{T}}_T$, model f with parameters θ, flag for cold or warm restarts
Output: Predicted labels for new vertices of each task

known_classes $\leftarrow \emptyset$;
$\theta \leftarrow$ initialize_parameters();
for $t \leftarrow 1$ **to** T **do**
  new_classes $\leftarrow \text{set}(\tilde{\boldsymbol{y}}^{(t-1)}) \setminus$ known_classes;
  if *new_classes* $\neq \emptyset$ **then**
    $\theta' \leftarrow$ expand_output_layer(θ, |new_classes|);
  end
  $\theta' \leftarrow$ initialize_parameters();
  if $t > 1$ **and** *do_warm_restart = TRUE* **then**
    $\theta' \leftarrow$ copy_existing_parameters(θ);
  end
  $\theta' \leftarrow$ train(θ', $\tilde{\mathcal{G}}_{t-1}$, $\tilde{\boldsymbol{X}}^{(t-1)}$, $\tilde{\boldsymbol{y}}^{(t-1)}$);
  $\tilde{\boldsymbol{y}}_{\text{pred}} \leftarrow$ predict(θ', $\tilde{\mathcal{G}}_t$, $\tilde{\boldsymbol{X}}^{(t)}$) for vertices $V_t \setminus V_{t-1}$;
  known_classes $\leftarrow$ known_classes $\cup$ new_classes;
  $\theta \leftarrow \theta'$;
end

Such thresholds can be either global or class-specific. A natural choice for a global threshold is the inflection point of the sigmoid function: $\tau = 0.5$. However, estimating class-specific thresholds might further reduce the risk of wrongly rejecting the classification of a known class [18]. A strategy for estimating class-specific thresholds is consulting the standard deviation across the training data [18]. To determine a threshold for class i, we collect all model outputs for instances of class i. For all these outputs $\hat{y} \in [0, 1]$, we create a mirror point $1 + \hat{y}$, assuming a Gaussian distribution with mean 1. On this distribution, we estimate the standard deviation SD_i and assign the new class specific threshold

$$\tau_i := \max\{\tau_{\min}, 1 - \alpha \cdot \text{SD}_i\}$$

where α is a scaling factor for the standard deviation and $\tau_{\min}$ is the minimum threshold. The original DOC [18] has used a fixed minimum threshold of $\tau_{\min} = 0.5$, while we leave it open to investigate upon different values for $\tau_{\min}$. For α, the original work suggests a value of 3.

This generic method can be used for self-detection of unseen classes in conjunction with any GNN model for vertex classification. In this work, we further experiment with adjusting the loss scaling of binary cross-entropy to account for class imbalance, which is inevitable in real-world graph data. This is particularly important for unseen class detection because, here, the magnitude of all outputs is relevant for the final decision, rather than only their maximum value. We denote this variant as gDOC. In detail, if class i appears n^+ times in the training data, we multiply the loss of output i with the factor $\frac{n-n^+}{n^+}$. This is a standard weighting procedure for binary cross-entropy that increases the loss according to the fraction of positive versus negative examples within the training data.

C. k-Neighborhood Time Differences

Real-world graphs grow and change at a different pace [59]. Some graphs change quickly within a short time like social networks, while others evolve rather slowly such as citation networks. Furthermore, graphs show different change behavior, i. e., different patterns in how vertices and edges are added and removed over time. Therefore, depending on the specific graph data, a different history of the data must be used for training to take these factors into account.

To make absolute history sizes more comparable across different datasets, we introduce a dataset-level measure that depends on the time differences within the k-hop neighorhood of each node.

The k-Neighborhood Time Difference Distribution tdiff_k enumerates the distribution of time differences within the k-hop neighborhood of each vertex, which corresponds to the *receptive field* of a GNN with k graph convolutional layers.

Intuitively, we collect the time differences of all node pairs reachable which are connected by at most k edges. We aggregate these time differences such that we obtain number of times a certain time difference has occurred in the dataset. Then we form a distribution (a multiset) over these time differences, whose percentiles we use as candidate history sizes.

Definition 2 (k-Neighborhood Time Difference Distribution)**.** Given graph $\mathcal{G}$ and let $\mathcal{N}^k(u)$ be the k-hop neighborhood of u, i. e., the set of vertices that are reachable from u by traversing at most k edges. Let $\mathrm{time} : \mathcal{V} \to \mathbb{N}$ be a function that gives the time information for each vertex, e. g., the year of a publication. We define $\mathrm{tdiff}_k(\mathcal{G})$ to be the *multiset of time differences*, computed over all vertices $u \in V$ to their k-distant neighboring vertices $v \in \mathcal{N}^k(u)$ that occurred before u.

$$\mathrm{tdiff}_k(\mathcal{G}) := \{\{ d^{\#(\mathrm{time}(u)-\mathrm{time}(v)=d)} \mid u \in V \wedge v \in \mathcal{N}^k(u) \wedge \mathrm{time}(v) \leq \mathrm{time}(u) \}\}$$

where $\#\left(\mathrm{time}(u) - \mathrm{time}(v) = d\right)$ gives the multiplicity with which time difference d has occurred.

We interpret the multiset tdiff_k as a distribution over time differences, that is used to further analyze a dataset's temporal distribution (percentiles) and to make datasets comparable.

In our experiments, we compare models trained with a *limited history size* against models trained with the *full history*. We use the 25th, 50th, and 75th percentiles of this distribution as history sizes versus the 100th percentile to model the full graph to analyze the influence of explicit knowledge.

D. Properties of k-Neighborhood Time Differences

Given a graph $G = (V, E)$ with $E \subseteq V \times V$ and the time $t : V \to \mathbb{N}$ at which a vertex appears in G. A "good" measure to determine history sizes $c : (V, E, t) \mapsto \mathbb{N}$ in evolving graphs would be *equivariant to granularity*. That means:

$$c(V, E, a \cdot t \pm a) \in a \cdot c(V, E, t) \pm a \tag{1}$$

with $a \in \mathbb{R}^+$ and $a \cdot t \pm a$ being a shorthand for a function t' that satisfies $a \cdot t(u) \in t'(u) \pm a$ for all $u \in V$. We show that tdiff_k fulfills this property in Appendix A of the supplementary material.

Recall that we select the history size according to a percentile of tdiff_k. Our datasets have the granularity of years. Thus, the history size we compute is measured in years. If we changed the granularity to months on the same underlying data, we would end up with the same history sizes multiplied by a factor of 12. Still, we would obtain (nearly) the same vertices within the history as before. Furthermore, equivariance to granularity is the minimum requirement to make history sizes comparable across datasets. We cannot assume that any absolute history size on dataset A would be comparable to the same history size on dataset B. But, if we derive our measure from tdiff_k, e. g., the median of tdiff_2, we have a strategy to find comparable history sizes across datasets, even if they come from different domains, e. g., social graphs with postings on minute level vs. citation graphs with data on at least daily level. This is because the measure is equivariant to granularity and it solely relies on time differences between connected vertices.

To summarize, the k-Neighborhood time differences are a measure that we only need to compute once per dataset before training. It captures the granularity and temporal connectivity patterns of the given graph. It allows us to derive history sizes that are comparable across datasets.

E. Graph Neural Network Base Models

The previously described techniques can be applied to arbitrary base models. In the following, we describe the base models that we considered for our experiments. The success of graph convolution [4] has triggered a resurgence of interest in graph neural networks [3]. In a generic formulation, the hidden representation of vertex i in layer l is defined as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)} \right)$$

where $\mathcal{N}(\cdot)$ refers to the set of adjacent nodes and σ is a nonlinear activation function. The normalizing factor c_{ij} depends on the respective model: the original Graph Convolutional Networks (GCN) [4] use $c_{ij} = \sqrt{|\mathcal{N}(i)|} \cdot \sqrt{|\mathcal{N}(j)|}$.

For our experiments, we systematically select representative GNN architectures as well as scalable GNN techniques for our experiments on lifelong learning. For this, we consider the different types of GNNs anisotropic vs. isotropic and standard vs. scalable approaches, as outlined in Section II. Our goal is to understand how different approaches of GNNs react to situations of changing graphs and new classes.

We select **Graph Attention Networks** (GATs) [6] as representative for the class of anisotropic GNNs. In GATs, the representations in layer $l+1$ for vertex i are computed as follows: $\hat{\boldsymbol{h}}_i^{l+1} = \alpha_{ii}^l \boldsymbol{h}_i^l + \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^l \boldsymbol{h}_j^l$ and $\boldsymbol{h}_i^{l+1} = \sigma(\boldsymbol{U}^l \hat{\boldsymbol{h}}_i^{l+1})$, where $\mathcal{N}(i)$ is the set of adjacent vertices to vertex i, U^l are learnable parameters, and σ is a nonlinearity. The edge weights α_{ij} are computed by a self-attention mechanism based on h_i and h_j, i. e., the softmax of $a(\boldsymbol{U}^l \boldsymbol{h}_i || \boldsymbol{U}^l \boldsymbol{h}_j)$ over edges, where a is an MLP and $\cdot||\cdot$ is the concatenation operation.

We select **GraphSAGE-Mean** [5] as a representative for isotropic GNNs because its special treatment of the vertices' self-information has shown to be beneficial [27]. The representations of self-connections are concatenated with averaged neighbors' representations before multiplying the parameters. In GraphSAGE-Mean, the procedure to obtain representations in layer $l+1$ for vertex i is given by the equations: $\hat{\boldsymbol{h}}_i^{l+1} = \boldsymbol{h}_i^l || \frac{1}{\deg_i} \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^l$ and $\boldsymbol{h}_i^{l+1} = \sigma(\boldsymbol{U}^l \hat{\boldsymbol{h}}_i^{l+1})$,

We select **Simplified GCN** [19] as a representative for shifting the neighborhood aggregation into preprocessing. Simplified GCN is a scalable variant of Graph Convolutional Networks (GCN) [4] that admits regular mini-batch sampling. Simplified GCN removes nonlinearities and collapses consecutive weight matrices into a single one. Thus, the simplified GCN can be described by the equation $\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}(\mathbf{S}^K \mathbf{X} \boldsymbol{\Theta})$, where $\mathbf{S}$ is the normalized adjacency matrix and $\boldsymbol{\Theta}$ is the weight matrix. The hyperparameter K has a similar effect as the number of layers in regular GCNs. Instead of using multiple layers, the k-hop neighbourhood is computed by $\mathbf{S}^K$, such that $\mathbf{S}^K \boldsymbol{X}$ can be precomputed. This makes Simplified GCN efficient, while not necessarily harming the performance [19].

We use **GraphSAINT** [12] as state-of-the-art subgraph sampling technique. In GraphSAINT, entire subgraphs are sampled for training GNNs. Subgraph sampling introduces a bias which is counteracted by normalization coefficients for the loss function. We use the best-performing random-walk sampling for our experiments. The underlying GNN is exchangeable, yet the authors suggest to use **Jumping Knowledge networks** (JKNets) [60]. JKNets introduce skip-connection to GNNs: each hidden layer has a direct connection to the output layer, in which the representations are aggregated, e. g., by concatenation. To isolate the effect of GraphSAINT sampling, we also include JKNets in our experiments.

V. Datasets

Adapting models to new data is an important problem whenever machine learning models are deployed in production. However, many graph benchmark datasets are stripped off any temporal data, which is needed to divide the data into realistic partitions, i. e., tasks. We scanned the literature (e. g., [11], [27], [46]) and common collections (OpenGraphBenchmark [61], KONECT[1], and PyTorch Geometric Temporal[2]) for datasets meeting the following criteria:

- Attributed vertices
- Vertex labels
- Time information on the vertices
- *Evolving* set of vertices (and thus, also edges) over time
- *Evolving* set of classes over time

Surprisingly, graph datasets meeting these criteria are rare. In those datasets with time information, either the graph is static, or the set of classes is static. Concurrent work on lifelong learning synthesizes an ordering of the vertices in standard datasets [26]. In this work, we seek to understand how our methods deal with *naturally* evolving datasets. For our first experiment, we use two different splits on standard benchmark datasets, which are described next. For the other three experiments on lifelong learning, we construct three entirely new datasets that we describe thereafter.

[1]http://konect.cc/

[2]https://pytorch-geometric-temporal.readthedocs.io/

TABLE I
Statistics for train-test splits: few-many (A) and many-few (B) settings on the citation networks datasets: Cora, Citeseer, and Pubmed. The unseen vertices and edges are available only after the training epochs. The test samples for measuring accuracy are a subset of the unseen vertices. The label rate is the percentage of labelled vertices for training.

Dataset		**Cora**		**Citeseer**		**Pubmed**
Classes		7		6		3
Features		1,433		3,703		500
Vertices		2,708		3,327		19,717
Edges		5,278		4,552		44,324
Avg. Degree		3.90		2.77		4.50
Setting	**A**	**B**	**A**	**B**	**A**	**B**
Train Vertices	440	2,268	620	2,707	560	19,157
Train Edges	342	3,582	139	2,939	34	41,858
Unseen Vertices	2,268	440	2,707	620	19,157	560
Unseen Edges	4,936	1,696	4,413	1,613	44,290	2,466
Test Samples	1,000	440	1,000	620	1,000	560
Label Rate	16.2%	83.8%	18.6%	81.4%	2.8%	97.2%

A. Standard Datasets

We use standard citation datasets: Cora, Citeseer, and Pubmed [62] for our first experiments on transductive versus inductive learning. Vertices are research papers represented by textual features and annotated with a class label. Edges resemble citation relationships, which are represented as bidirectional edges. These datasets are often used in transductive setups [4], [6], [13]. In our experimental setup with unseen vertices, however, we cast these datasets to be inductive.

We use two different train-test splits for each dataset. Setting A is derived from the train-test split for transductive tasks [4]. It consists of few labeled vertices that induce our training set and many unlabeled vertices. Setting B instead comprises many training vertices and few test nodes. We set it up by inverting the train-test mask of Setting A and assign the edges accordingly. Setting B is motivated from applications, in which a large graph is already known and incremental changes occur over time, such as for citation recommendations, link prediction in social networks, and others [59], [63]. We refer to Table I for the details of the datasets and the two settings. We use these three datasets with two different train-test splits in our first experiment described in Section VI-A.

B. New Datasets for Lifelong Learning on Graphs

We provide three new graph datasets for lifelong learning on the basis of scientific publications: one new co-authorship graph dataset (PharmaBio) as well as two newly compiled citation graph datasets based on DBLP (DBLP-easy and DBLP-hard). For PharmaBio, the classes are journal categories. For DBLP, we use the conferences and journals of the published papers as classes. Since we select those venues with the most

publications, this serves as a proxy for a broad categorization. When new conferences and journals emerge, as they do in computer science, new classes will be introduced to the data. The datasets were generated by imposing a minimum threshold of publications per class per year: 100 for DBLP-easy, 45 for DBLP-hard, and 20 for PharmaBio. For the co-authorship graph PharmaBio we additionally require a minimum of two publications per author per year. In all datasets, vertex features are normalized TF-IDF representations of the publication title.

TABLE II
GLOBAL DATASET CHARACTERISTICS: TOTAL NUMBER OF VERTICES $|V|$, EDGES $|E|$, FEATURES D, CLASSES $|\mathbb{Y}|$ ALONG WITH # OF NEWLY APPEARING CLASSES (IN BRACES) WITHIN THE T EVALUATION TASKS

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	D	$\lvert\mathbb{Y}\rvert$	T
DBLP-easy	45,407	112,131	2,278	12 (4 new)	12
DBLP-hard	198,675	643,734	4,043	73 (23 new)	12
PharmaBio	68,068	2,1M	4,829	7	18

1) Basic Characteristics: Table II summarizes the basic characteristics of the datasets. DBLP-easy and DBLP-hard are organized into 12 annual snapshots, while PharmaBio has 18 annual snapshots. DBLP-easy has 45k vertices, 112k edges, and a feature dimension of 2,278. The nodes are assigned to one of 12 classes, of which four only appear during the sequence of snapshots, i. e., they are not present in the first snapshots. DBLP-hard has 199k classes, 644k edges, and a feature dimension of 4,043. Twenty-three of the 73 classes appear only during later snapshots. PharmaBio comes with 68k vertices, 2.1M edges, feature dimension 4,829, 7 classes and 18 snapshots. The number of edges is much higher than in the DBLP variants because PharmaBio is a co-authorship graph, which is more dense than citation graphs. Note that DBLP-easy is a subset of DBLP-hard as both where generated by applying a minimum threshold on the number of publications per class.

We report the datasets' label distribution, degree distribution, and the distribution over time in Figure 2. The annual number of publications grows over time. Only in PharmaBio, there is a higher amount between 1991-1997 than it is between 1998 and 2003. The global degree distributions of DBLP-easy and DBLP-hard seem to follow a power-law distribution [64] as the degree distribution is almost a straight-line except for the blurry tail. For PharmaBio, the degree distribution is more blurry, while a trend line can still be identified. Furthermore, we observe that the number of examples per class is imbalanced in all three datasets. Even though the three classes have different numbers of classes, the shape of the label distributions is similar.

2) Unseen Classes and Distribution Shift: Regarding changes in the class set, DBLP-easy has 12 venues in total, including one bi-annual conference and four new venues appearing in 2005, 2006, 2007, and 2012. DBLP-hard has 73 venues, including one discontinued, nine bi-annual, six irregular venues, and 23 new venues. To quantify changes in the class set, we compute the magnitude of the class drift as total variation distance [65], [66]:

$$\sigma_{t-1,t} = \frac{1}{2} \sum_{y \in \mathbb{Y}_{t-1} \cup \mathbb{Y}_t} |P_{t-1}(y) - P_t(y)|$$

where $P_t(y)$ is the observed class probability at time t. We visualize the drift magnitudes per dataset in Figure 3. An i.i.d dataset would have a drift magnitude of zero by definition. As expected, the drift magnitude is high (between 0.12 and 0.16) for the two datasets with new classes: DBLP-easy and DBLP-hard. On PharmaBio, which has no new classes, the drift magnitude is consistently lower than 0.07.

3) Analyzing Time Differences: Next, we analyze the k-neighborhood time differences tdiff_k, which we have introduced in Section IV-C. In Figure 4, we show the resulting distribution for three different values of $k = 1, 2, 3$. As expected, the time differences increase if we allow a higher maximum path length k. For our experiments, we will use GNN models with 2 layers, i. e., they take the two-hop neighborhood of each vertex into account. Thus, we use tdiff_2 to derive candidate history sizes. As such, we select 1, 3, 6, 25 as history sizes for DBLP-{easy,hard} and 1, 4, 8, 21 as history sizes for PharmaBio according to the 25th, 50th, 75th, and 100th percentiles of tdiff_2.

4) Setting Up Tasks for Lifelong Learning: For each dataset, we construct the sequence of tasks $\tilde{\mathcal{T}}_1, \ldots, \tilde{\mathcal{T}}_T$ on the basis of the publication year along with a history size c. For each task $\tilde{\mathcal{T}}_t$, we construct a graph with publications from time $[t-c, t]$, where publications from time t are the test vertices, and $t < c$ training vertices (transductive). In the inductive case that is used by GraphSAINT in our experiments, we train exclusively on $\tilde{\mathcal{T}}_{t-1}$, but still evaluate the test vertices of $\tilde{\mathcal{T}}_t$. We set the first evaluation task $\tilde{\mathcal{T}}_1$ to the time, at which 25% of the total number of publications are available. Thus, mapping the datasets to our problem statement (see Figure 1), our first evaluation task $t = 1$ corresponds to year 1999 in PharmaBio (total range: 1985–2016) and 2004 in DBLP-{easy,hard} (1990-2015). We continue with the next years for subsequent tasks. We will use these datasets in the experiments described in Sections VI-B, VI-C, and VI-D.

VI. EXPERIMENTS

In the following, we describe our experiments to analyze transductive vs. inductive learning, lifelong learning, open-world learning, and learning with limited labeled data. We use standard benchmark datasets (described in Section V-A) for the first experiment VI-A, and then we use our new datasets (described in Section V-B) for the experiments VI-B – VI-D.

A. Transductive versus Inductive Learning

1) Experimental Setup: We construct a dedicated experimental setup to assess the inference capabilities of graph neural networks. We include edges in the training set if and only if its source and destination vertex are both in the training set. The training process is then split in two steps. First, we pre-train the model on the labelled training set. Then, we insert the previously unseen vertices and edges into the graph and continue training for a limited amount of inference epochs.

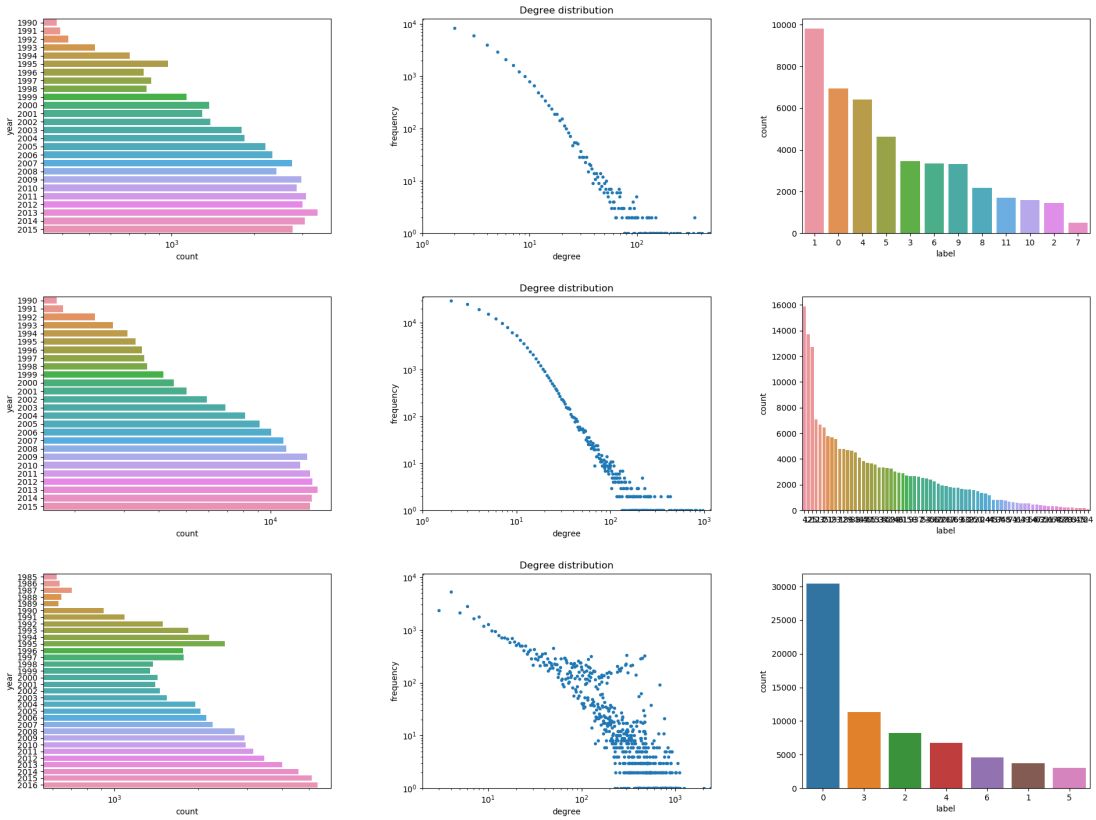

Fig. 2. Distribution of vertices per year on log scale (left column), degree distributions (middle column), label distributions (right column), for our new datasets: DBLP-easy (top row), DBLP-hard (middle row), PharmaBio (bottom row)

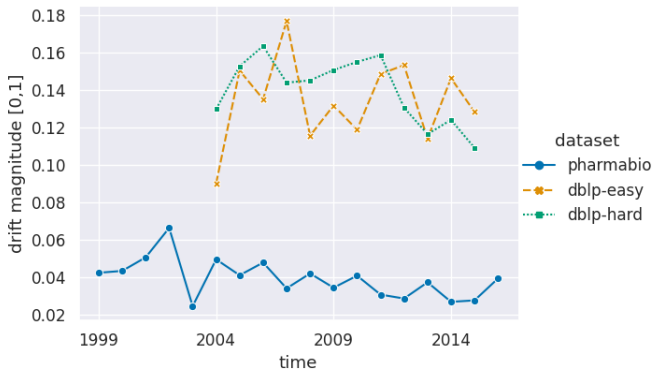

Fig. 3. Magnitude of the class drift per dataset. The drift within the PharmaBio dataset (no new classes) is lower than the drift of both DBLP variants. Independent and identically distributed data would have drift magnitude zero.

The unseen vertices do not introduce any new labels. Instead, the unseen vertices provide features and may be connected to known labelled vertices. We evaluate the accuracy on the test vertices, which are a subset of the unseen vertices, before the first and after each inference epoch. For each model, we compare using 200 pretraining epochs versus no pretraining. In the latter case, the training begins during inference, which is equivalent to retraining from scratch whenever new vertices and edges are inserted. This allows us to assess whether pretraining is helpful for applying graph neural networks on dynamic graphs.

2) Hyperparameters: All employed graph neural networks use two graph convolution layers that aggregate neighbor representations. The output dimension of the second layer corresponds to the number of classes. Thus, the features within the two-hop neighborhood of each labeled vertex are taken into account for its prediction. We adopt the same hyperparameter values as proposed in the original works. For GCN, we use 16 or 64 (denoted by GCN-64) hidden units per layer, ReLU activation, 0.5 dropout rate, along with an (initial) learning rate of 0.005 and weight decay $5 \cdot 10^{-4}$ [4]. For GAT, we use 8 hidden units per layer and 8 attention heads on the first layer. The second layer has 1 attention head (8 on Pubmed). We set the learning rate to 0.005 (0.01 on Pubmed) with weight decay 0.0005 (0.001 on Pubmed) [6]. For GraphSAGE, we use 64 hidden units per layer with mean aggregation, ReLU activation, and a dropout rate of 0.5. We set the learning rate to 0.01 with weight decay $5 \cdot 10^{-4}$ [5]. Our MLP baseline has one hidden layer with 64 hidden units, ReLU activation, a dropout rate of 0.5, learning rate 0.005 and weight decay $5 \cdot 10^{-4}$. In all cases, we use Glorot initialization [67] and Adam [68] to optimize cross-entropy. We initialize the optimizer at the beginning of the inference epochs.

3) Results: Figure 5 shows the results of the three models on the three datasets: Cora, Citeseer, and Pubmed. Pretrained models score consistently higher than non-pretrained models

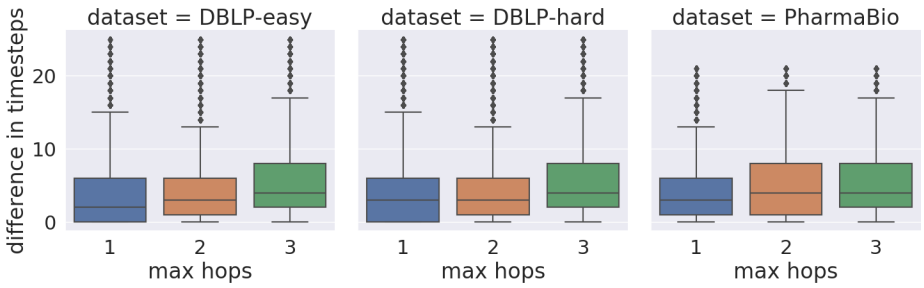

Fig. 4. Distributions of time differences tdiff_k (y-axis) for DBLP-easy (left), DBLP-hard (center) and PharmaBio (right) within the k-hop neighborhood for $k = \{1, 2, 3\}$ (x-axis).

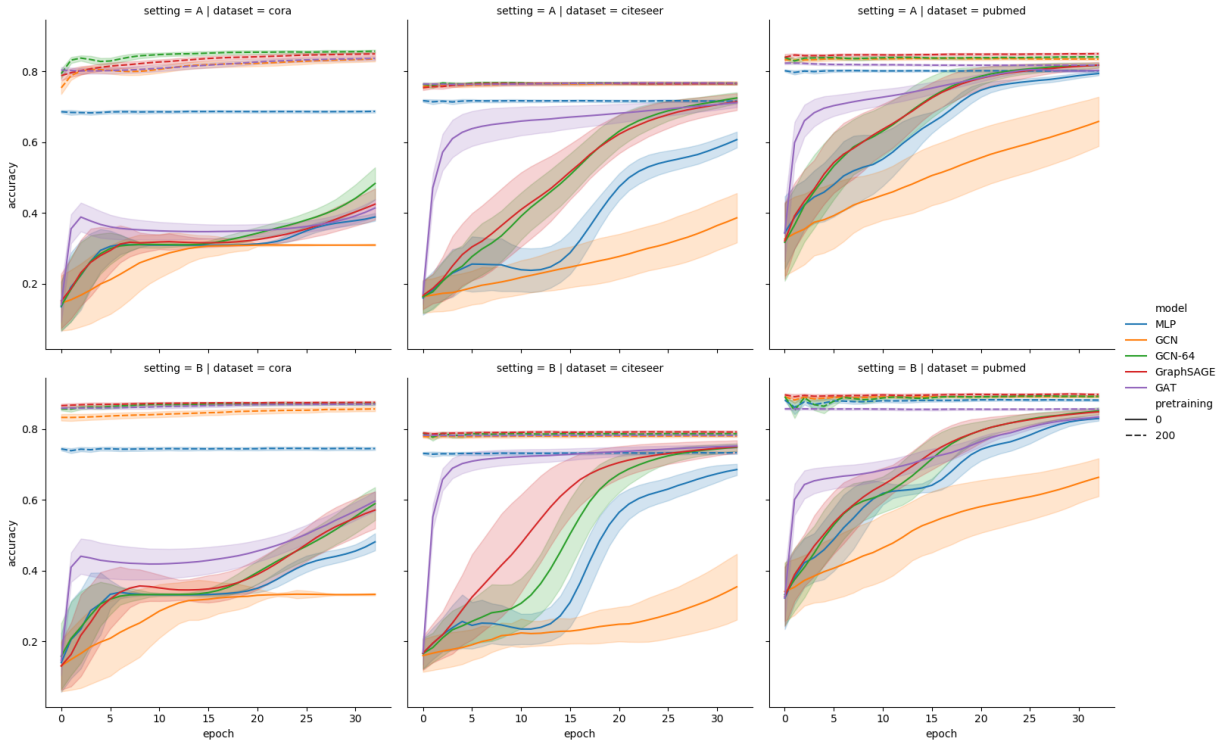

Fig. 5. Test accuracy after each inference epoch for the many-few settings A *(Top)* and few-many setting B *(Bottom)* on the datasets Cora, Citeseer, and Pubmed. Each line resembles the mean of 100 runs and its region shows the standard deviation. The dashed lines show the results with 200 pretraining. The solid lines are the results without prertraining.

while having less variance. The accuracy of the pretrained models plateaus after a few inference epochs (up to 10 on Cora-A and Pubmed-B). Without any pretraining, GAT shows the fastest learning process. The absolute scores of pretrained graph neural networks are higher than the ones of MLP. From a broad perspective, the scores of pretrained graph neural networks are all on the same level. While GCN falls behind the others on Cora-B, GAT falls behind the others on Pubmed. The scores of the many-few setting B are higher than the ones of few-many setting A by a constant margin.

We compare the results of setting A and B by measuring the Jensen-Shannon divergence [69] between the accuracy distributions. The Jenson-Shannon divergence between the two settings is lower with pretraining (between 0.0057 for GAT and 0.0115 for MLP) than it is without pretraining (between 0.0666 for GraphSAGE and 0.1013 for GCN). This shows that accuracy distributions are similar in both train-test splits.

In summary, our results show that inductive graph neural networks perform well even though we insert new *unlabeled* vertices and edges after training. For all three considered in this study, the accuracy plateaus after very few inference epochs. This observation holds for both train-test split settings: many-few and few-many. This motivates the warm restart strategy, i. e., reusing previous parameters, that we use in the following experiments on lifelong learning. In different terms, we have not observed any gain from uptraining an inductive model on extra *unlabeled* data.

B. Lifelong Learning on Graphs

So far, we have added new unlabeled data. Now, we report the results of our main experiments to explore the standard lifelong learning setting, as described in Section III. In this setting, the models have to sequentially adapt to new tasks with new labeled data including unseen classes.

1) Experimental Setup: We constrain all models to two graph convolutional layers, a comparable penultimate hidden dimension (2x32 GraphSAGE, 4x8 GAT, 2x2x16 JKNet, 64 MLP), and a 0.5 dropout rate. We fix an update step budget of 200 per task and use Adam [68] to optimize cross-entropy. We implemented GAT, GraphSAGE-mean, Simplified GCN, and JKNet with *dgl* [70] and use *torch-geometric* [71] for GraphSAINT. We had to disable GraphSAINT's norm recomputation for each task such that our experiments could finish in a reasonable time. We also optimized weight decay. The effect of weight decay was negligible. For each combination of base model, history size, and restart configuration, we tune the learning rate on DBLP-easy. Thus, we consider DBLP-easy as our development dataset to tune the learning rate, which we then apply on DBLP-hard and PharmaBio. We run our incremental training method for graph learning from Section IV-A for each of the models under warm restart and cold restart configurations. The experiments are repeated 10 times with different random seeds.

2) Evaluation Measures: Our primary evaluation measure for lifelong node classification f is accuracy. With $\mathrm{acc}_t(f^{(t)})$, we denote the accuracy of model $f^{(t)}$ on task $\mathcal{T}_t$. We aggregate accuracy scores over the sequence of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ by using their unweighted average [24]:

$$\mathrm{acc}(f) = \frac{1}{T} \sum_{t \in 1,\ldots,T} \mathrm{acc}_t(f^{(t)})$$

Following Lopez-Paz & Ranzato [24], we use Forward Transfer (FWT) to quantify the effect of reusing previous parameters. This is reflected by the accumulated differences in accuracy between the f_{warm} and f_{cold} models, defined below:

$$\mathrm{FWT}(f_{\text{warm}}, f_{\text{cold}}) = \frac{1}{T-1} \sum_{t \in 2,\ldots,T} \mathrm{acc}_t(f^{(t)}_{\text{warm}}) - \mathrm{acc}_t(f^{(t)}_{\text{cold}})$$

3) Results: Table III shows the aggregated results of 20,160 evaluation steps (48 configurations with 10 repetitions on two datasets with 12 tasks each and one dataset with 18 tasks). We consider method A better than method B when the mean accuracy of A is higher than the one of B and the 95% confidence intervals do not overlap [72]. In terms of absolute best methods per setting (= dataset $\times$ history size), we find that GraphSAGE consistently yields the highest scores except for DBLP-hard, where it is challenged by Simplified GCN.

Regarding the comparison of history sizes (i. e., *explicit knowledge*, see Section I), the highest scores are achieved in almost all cases by using an unlimited history size, i. e., using the full graph's history. However, on all datasets, the scores for training with limited window sizes larger than 1 are close to the ones of full-graph training. With history sizes that cover 50% of the GNN's receptive field, all methods achieve at least 95% relative accuracy compared to the same model under full-history training. When 75% of the receptive field is covered, the models yield at least 99% relative accuracy. To compute these percentages, we have selected the better one of either cold or warm restarts for each method.

Regarding the influence of *implicit knowledge*, we find that reusing parameters (warm restarts) leads to notably higher scores than retraining from scratch, when the history size is one (see column FWT with history size $c = 1$): The average Forward Transfer across all models and datasets with history size $c = 1$ is five accuracy points.

Regarding isotropic vs anisotropic GNNs, we find that GAT and GraphSAGE perform similarly well on DBLP-easy (on which the learning rate was tuned). However, GraphSAGE-mean yields higher scores on DBLP-hard and PharmaBio, which might indicate that GraphSAGE-mean is more robust to hyperparameters than GAT.

Regarding memory-efficient methods, we observe that the scores of Simplified GCN are among the highest of all methods on DBLP-hard. To understand this result, we recall that Simplified GCN uses only one single weight matrix of shape $n_{\text{features}} \times n_{\text{outputs}}$, which leads to 300,000 learnable parameters on DBLP-hard, but only 27,000 and 34,000 on DBLP-easy and PharmaBio, respectively. For comparison, GraphSAGE has 146,000 learnable parameters on DBLP-easy, 264,000 on DBLP-hard, and 310,000 on PharmaBio. On the other hand, GraphSAINT yields high scores on PharmaBio, comparable to GraphSAGE, but lower scores on both DBLP datasets.

C. Lifelong Learning with Limited Labeled Data

Until now, we have assumed that the true label of each vertex becomes part of the training data for subsequent tasks. While this is realistic in some scenarios, there are others in which we cannot expect to gain access to the true labels after finishing each prediction task. For this reason, we artificially limit the amount of labeled data that becomes available for subsequent tasks.

1) Experimental Setup: To implement the idea of learning with only a fraction of labeled data, we randomly sample a subset of vertices, for which the true class label is available for training. We denote this fraction as the *label rate*. For the experiments, it is important to sample globally rather than on a per-task basis because we would get inconsistencies otherwise: node i could have a label in task t but not in task $t + 1$. Therefore, we sample on the entire dataset, before we split it into tasks. This way, the subset of vertices that comes with classes is fixed for the entire duration of the experiments. Furthermore, we use the same subset with all different configurations and all repetitions of the experiment. We sample uniformly at random on vertex level without any stratification across classes.

For this experiment, we use GraphSAGE-Mean as GNN model because it has achieved the best results in the previous experiment, where the label rate was not restricted. As before, we evaluate different history sizes (1, 3, 6) and both restart configurations (warm and cold). As dataset, we chose to use DBLP-hard because it has the highest number of classes and is the most challenging one.

TABLE III

ACCURACY (WITH 95% CONFIDENCE INTERVALS VIA 1.96 STANDARD ERROR OF THE MEAN) AND FORWARD TRANSFER (AVERAGED DIFFERENCE OF WARM AND COLD RESTARTS) ON OUR DATASETS WITH DIFFERENT HISTORY SIZES (COLUMN c). THE BEST METHOD PER CASE (= 1 DATASET + 1 HISTORY SIZE) IS MARKED IN BOLD, ALONG WITH METHODS WHERE THE 95% CI OVERLAPS.

		GAT			GraphSAGE-Mean			MLP (Baseline)		
		avg. accuracy		FWT	avg. accuracy		FWT	avg. accuracy		FWT
		cold	warm		cold	warm		cold	warm	
Dataset	**c**									
DBLP-easy	**1**	60.8 ± 0.5	$\mathbf{64.9 \pm 0.4}$	$+4.5$	60.4 ± 0.5	$\mathbf{65.1 \pm 0.4}$	$+5.2$	56.1 ± 0.4	62.2 ± 0.5	$+6.6$
	3	$\mathbf{68.9 \pm 0.3}$	$\mathbf{69.3 \pm 0.3}$	$+0.2$	$\mathbf{68.7 \pm 0.3}$	$\mathbf{69.3 \pm 0.3}$	$+0.7$	61.0 ± 0.5	62.9 ± 0.4	$+2.0$
	6	$\mathbf{70.3 \pm 0.4}$	70.2 ± 0.4	-0.1	$\mathbf{71.1 \pm 0.4}$	$\mathbf{70.9 \pm 0.4}$	-0.3	62.7 ± 0.3	62.7 ± 0.4	-0.2
	full	70.2 ± 0.4	70.2 ± 0.4	$+0.1$	$\mathbf{71.6 \pm 0.4}$	$\mathbf{71.4 \pm 0.3}$	-0.2	63.4 ± 0.3	61.9 ± 0.4	-1.2
DBLP-hard	**1**	39.4 ± 0.2	39.1 ± 0.2	-0.1	34.5 ± 0.4	40.0 ± 0.2	$+5.9$	31.6 ± 0.3	38.3 ± 0.3	$+7.4$
	3	44.0 ± 0.2	43.7 ± 0.2	-0.4	44.3 ± 0.2	$\mathbf{45.1 \pm 0.2}$	$+0.8$	33.7 ± 0.3	38.9 ± 0.2	$+5.6$
	6	45.1 ± 0.3	45.3 ± 0.3	$+0.2$	$\mathbf{46.5 \pm 0.3}$	$\mathbf{46.7 \pm 0.3}$	$+0.2$	39.2 ± 0.2	38.3 ± 0.2	-0.7
	full	45.6 ± 0.3	45.6 ± 0.3	-0.1	46.8 ± 0.2	47.1 ± 0.3	$+0.4$	38.2 ± 0.2	36.7 ± 0.2	-1.1
PharmaBio	**1**	61.6 ± 0.9	65.4 ± 0.9	$+3.8$	65.4 ± 0.9	$\mathbf{68.6 \pm 1.0}$	$+3.3$	62.7 ± 0.9	66.3 ± 0.9	$+3.9$
	4	64.5 ± 0.8	65.3 ± 0.9	$+0.9$	$\mathbf{68.0 \pm 0.8}$	$\mathbf{69.0 \pm 0.8}$	$+1.1$	66.3 ± 0.7	65.7 ± 0.8	-0.7
	8	65.1 ± 0.8	65.4 ± 0.8	$+0.3$	$\mathbf{68.8 \pm 0.7}$	$\mathbf{69.0 \pm 0.8}$	$+0.2$	64.2 ± 0.8	65.3 ± 0.7	$+0.9$
	full	64.3 ± 0.8	65.4 ± 0.8	$+0.2$	$\mathbf{69.0 \pm 0.7}$	$\mathbf{68.4 \pm 0.7}$	-0.7	65.4 ± 0.8	64.4 ± 0.6	-1.1

		Simplified GCN			GraphSAINT			Jumping Knowledge		
		avg. accuracy		FWT	avg. accuracy		FWT	avg. accuracy		FWT
		cold	warm		cold	warm		cold	warm	
DBLP-easy	**1**	57.1 ± 0.4	63.7 ± 0.4	$+7.2$	62.1 ± 0.3	63.2 ± 0.4	$+1.2$	56.2 ± 0.5	61.4 ± 0.5	$+5.6$
	3	66.4 ± 0.3	67.4 ± 0.3	$+1.2$	66.4 ± 0.4	65.3 ± 0.5	-0.9	65.2 ± 0.3	65.9 ± 0.5	$+1.0$
	6	69.3 ± 0.4	69.3 ± 0.4	$+0.1$	68.1 ± 0.4	65.5 ± 0.7	-2.1	68.0 ± 0.4	66.9 ± 0.6	-0.7
	full	$\mathbf{71.0 \pm 0.4}$	70.0 ± 0.4	-1.0	68.4 ± 0.5	65.7 ± 0.5	-2.8	68.7 ± 0.4	66.3 ± 0.4	-2.5
DBLP-hard	**1**	34.5 ± 0.3	$\mathbf{41.0 \pm 0.3}$	$+7.0$	35.9 ± 0.3	35.6 ± 0.4	$+0.5$	33.0 ± 0.2	35.3 ± 0.3	$+2.9$
	3	44.1 ± 0.2	$\mathbf{44.8 \pm 0.3}$	$+0.8$	39.3 ± 0.3	38.1 ± 0.5	-0.6	39.1 ± 0.3	38.8 ± 0.4	$+0.3$
	6	$\mathbf{46.9 \pm 0.3}$	46.2 ± 0.3	-0.4	40.6 ± 0.3	38.8 ± 0.6	-1.2	41.0 ± 0.3	40.1 ± 0.5	-0.3
	full	$\mathbf{48.8 \pm 0.4}$	47.5 ± 0.3	-1.2	41.0 ± 0.4	40.7 ± 0.4	-0.3	41.6 ± 0.3	40.8 ± 0.2	-0.9
PharmaBio	**1**	62.3 ± 0.9	64.5 ± 0.8	$+2.3$	65.7 ± 0.8	$\mathbf{68.6 \pm 0.8}$	$+3.0$	64.1 ± 0.9	$\mathbf{68.3 \pm 0.9}$	$+4.3$
	4	64.4 ± 0.8	64.4 ± 0.8	-0.0	67.3 ± 0.8	$\mathbf{68.4 \pm 0.7}$	$+1.0$	67.1 ± 0.8	$\mathbf{68.2 \pm 0.8}$	$+1.1$
	8	65.3 ± 0.8	64.0 ± 0.7	-1.4	$\mathbf{68.1 \pm 0.8}$	$\mathbf{68.0 \pm 0.7}$	-0.1	$\mathbf{67.8 \pm 0.8}$	$\mathbf{67.7 \pm 0.7}$	-0.3
	full	62.4 ± 0.8	61.7 ± 0.6	-0.8	$\mathbf{68.2 \pm 0.8}$	66.1 ± 0.8	-2.2	66.8 ± 0.8	64.5 ± 0.7	-2.6

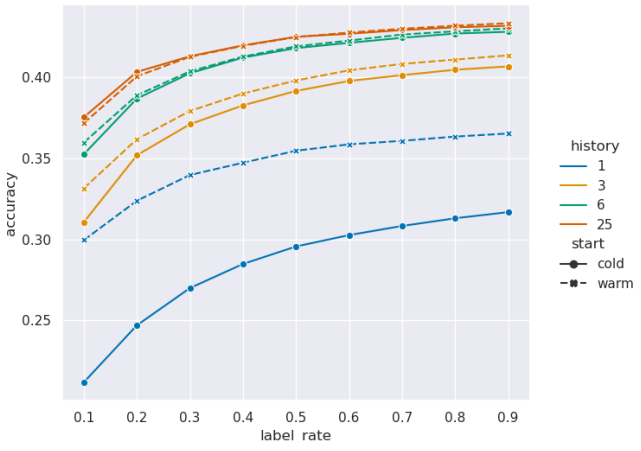

Fig. 6. Average accuracy of GraphSAGE with warm restarts across tasks on DBLP-hard under varying label rate

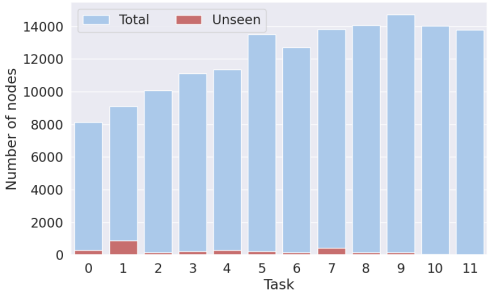

Fig. 7. Number of nodes with unseen classes per task on DBLP-hard

2) Results: In Figure 6, we plot the average accuracy across tasks as a function of label rate. We confirm that the results of our previous experiments also hold when labeled data is limited. Warm restarts yield higher scores than cold restarts. This is more pronounced when the history size is small. As expected, the absolute accuracy values decrease when the label rate becomes smaller.

D. Self-Detection of Unseen Classes

So far, the methods had no chance to predict any of the previously unseen classes because those have never appeared in the training data. In previous experiments, unseen new classes have still been part of the evaluation to ensure a realistic setup. Now we present an experiment where the

models shall actively detect instances from unseen classes, while classifying the other instances as usual.

1) Experimental Setup: To investigate self-detection of unseen classes, we use the DBLP-hard dataset as it has the highest amount of new classes (23 in the course of the temporal evolution). We use the best-performing model GraphSAGE-mean along with the DOC extensions for unseen class detection that we have introduced in Section IV. We optimize its hyperparameters on our development dataset DBLP-easy because of the different loss function. As before, the best learning rate is selected based on the best accuracy on DBLP-easy and transferred to DBLP-hard. In Figure 7, we show how many vertices belong to unseen classes in the DBLP-hard dataset. We observe that in every task, apart from the final task, there are vertices with unseen classes. Note that we do not tune the learning rate for the best detection performance but for the best accuracy, as in previous experiments. We then compare DOC with gDOC, where the former is our baseline and the latter uses our proposed class-weighting loss function.

2) Evaluation Measures: We evaluate how good the models are at detecting unseen classes. For this purpose we use two measures: Macro-F1 with a special class for instances from unseen classes [18] and Matthews correlation coefficient (MCC). Note that Macro-F1 averages the F1 scores over classes, such that the effect of the 'unseen' class is taken into account like any of the known classes. In detail, we compute this *Open Macro-F1* as follows:

$$\boldsymbol{y}'_{\text{pred},i} := \begin{cases} \text{'unseen', if example } i \text{ is detected as o.o.d.} \\ \boldsymbol{y}_{\text{pred},i}, \text{ otherwise} \end{cases}$$

$$\boldsymbol{y}'_i := \begin{cases} \boldsymbol{y}_i \text{ if class } \boldsymbol{y}_i \text{ is known} \\ \text{'unseen', otherwise} \end{cases}$$

where $\boldsymbol{y}_{\text{pred}}$ are the predicted class labels. The $\arg\max$ of the output is replaced by a special symbol when the method has emitted an 'unseen' decision for that instance. The true labels $\boldsymbol{y}$ are pre-processed similarly, such that instances from unseen classes receive a special class symbol. Then, we average

$$\text{Open Macro-F1} := \frac{1}{T}\sum_{t=1}^{T} \text{Macro-F1}(\boldsymbol{y}'(t), \boldsymbol{y}'_{\text{pred}}(t))$$

In pre-experiments, we found that the best Open F1-Macro scores are achieved when the thresholds are high. This is because we have a high number of classes and the special class contributes only very little to the overall F1 Macro score. Thus, a large number of false rejects diminishes the overall performance in terms of F1-Macro. False rejects are vertices that should *not* be rejected (as their true class is known), but they are rejected.

The F1-Macro score is not helpful for evaluating unseen class detection, due to the class imbalance (overall there are only few vertices with an unseen class). Thus, we decided to report a further score, the Matthews correlation coefficient (MCC) of the 'unseen' class vs. all other classes (i. e., the set of known classes). MCC is a popular measure to evaluate binary classification that accounts for class imbalance [73].

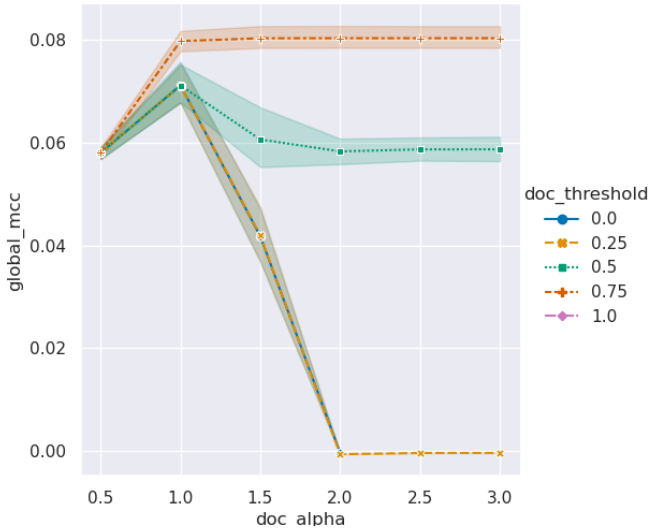

Fig. 8. Global MCC Macro-F1 of W-GS-DOC (history size 3, warm) as a function of the SD Factor α in DOC with different *minimum* threshold values.

Dealing with this class imbalance is important as the number of vertices from the known classes are much larger than the number of vertices from the unseen class. It ranges between -1 and 1, where zero corresponds to random prediction. In more detail, MCC is computed as:

$$\text{MCC} = \frac{\text{TP}\cdot\text{TN} - \text{FP}\cdot\text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}$$

where TP are true positives or correctly rejected instances, TN true negatives, FP false positives and FN false negatives. We accumulate those numbers over the entire sequence of tasks.

3) Results: The results in Table IV reveal that using class-weighted binary cross entropy (gDOC) is necessary to achieve reasonable Open-F1 macro scores. The MCC scores are also consistently higher for gDOC than for plain DOC. For the thresholds, we find that using a high threshold (0.75) is preferable over lower thresholds. We further note that the combination of warm restarts and a small history size leads to the highest MCC scores (0.09). We provide the full results of our experiments in Appendix B of the supplementary material.

In Figure 8, we show that reduction does not help to increase the performance. When a high minimum threshold is used, risk reduction only decreases the overall performance.

VII. Ablation Study: Incrementally-Trained vs Once-Trained Models

We isolate the effect of incremental training and compare once-trained trained models (static) against incrementally trained models (incremental). We train the static models for 400 epochs on the data before the first evaluation time step, which comprises 25% of the total vertices. We train incremental models for 200 epochs with history sizes of 3 time steps (4 on the PharmaBio dataset) before evaluating each task. We repeat each experiment 10 times with different random seeds. In Figure 9, we see that the accuracy of the static models decreases over time on DBLP-easy and DBLP-hard, where

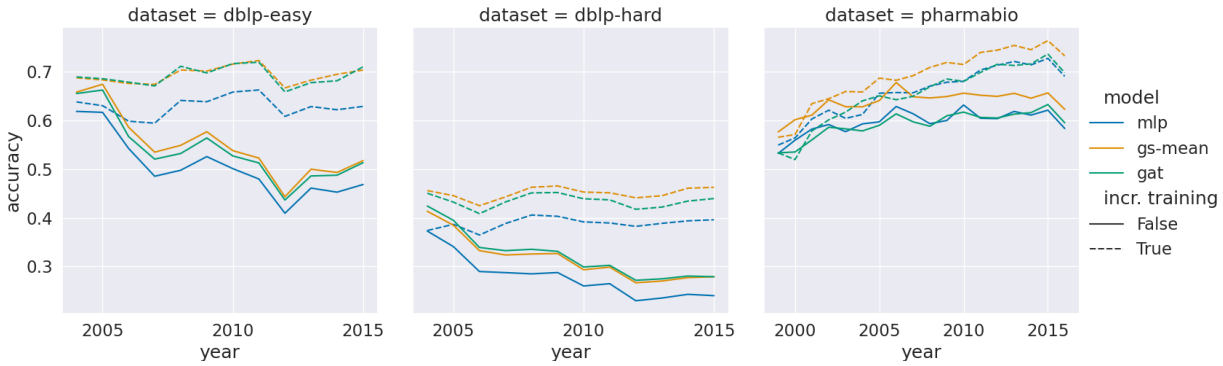

Fig. 9. Results of Ablation Study: Accuracy scores of once-trained, static models (solid lines) are lower than incrementally trained models (dashed lines).

TABLE IV
RESULTS FOR UNSEEN CLASS DETECTION ON DBLP-HARD WITH GRAPHSAGE AS BASE MODEL (AVERAGE OF 5 REPETITIONS). α INDICATES THAT RISK REDUCTION IS USED WITH THE RESPECTIVE FACTOR FOR THE STANDARD DEVIATION, τ IS THE MINIMUM THRESHOLD. RUNS NAMED GDOC ARE TRAINED WITH WEIGHTED CROSS ENTROPY. DOC IS OUR BASELINE.

		MCC		**Open F1 Macro**	
		cold	warm	cold	warm
c	**Open Learning Method**				
1	DOC ($\tau = 0.50$)	.01	.04	.01	.01
	DOC ($\tau = 0.50, \alpha = 3.0$)	.01	.02	.01	.01
	gDOC ($\tau = 0.50$)	.04	.05	**.13**	**.13**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.04	.05	**.13**	**.13**
	gDOC ($\tau = 0.75$)	.04	**.09**	**.13**	**.13**
3	DOC ($\tau = 0.50$)	.02	.03	.02	.05
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.02	.05
	gDOC ($\tau = 0.50$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.75$)	.05	**.08**	**.15**	**.15**
6	DOC ($\tau = 0.50$)	.02	.03	.05	.08
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.05	.08
	gDOC ($\tau = 0.50$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.75$)	.05	**.07**	**.16**	**.16**
∞	DOC ($\tau = 0.50$)	.02	.04	.8	.12
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.04	.08	.12
	gDOC ($\tau = 0.50$)	.04	.05	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.05	**.16**	**.16**
	gDOC ($\tau = 0.75$)	.05	**.07**	**.16**	**.16**

new classes appear over time. On PharmaBio (fixed class set), the accuracy of the static models plateaus, while the accuracy of incrementally trained models increases.

VIII. DISCUSSION

Our experiments show that incremental training with limited history sizes is almost as good as using the full history of the graph. With window sizes of 3 or 4 (50% receptive field coverage), GNNs achieve at least 95% accuracy compared to using all past data for incremental training. With window sizes of 6 or 8 (75% receptive field coverage), at least 99% accuracy can be retained. This result holds for standard GNN architectures and also for scalable and sampling-based approaches. This has direct consequences for lifelong learning of GNNs on evolving graphs, as it impacts how GNNs can be employed in real-world applications.

We have further investigated on reusing parameters from previous tasks (warm restarts). Using warm restarts is a viable strategy, i. e., reusing an "old" model, even though new classes appear during the sequence of tasks. We have shown that reusing parameters from previous tasks becomes more important when the history sizes are small. This is because less explicit knowledge is available. It is noteworthy that Simplified GCNs perform surprisingly well on DBLP-hard. There, the model yields the highest absolute scores, on par with GraphSAGE, despite the simplicity of the approach. Furthermore, we have shown that the methods work well, even when labeled data is limited. This is important for real-world applications because annotating data is expensive.

We acknowledge that crisp, unsupervised unseen class detection is extremely challenging. We have conducted the first step by combining graph neural networks with a DOC [18] module. Our results show that it is necessary to adjust the weights of binary cross entropy training (gDOC), when the label distribution is imbalanced. Contrary to the original DOC, we have not observed any improvements when risk reduction (via standard deviation of logits) was used. Instead, the best results were achieved with an appropriate threshold ($\tau = 0.75$) regardless of the risk reduction factor α.

Another interesting result is that the combination of warm restarts with small history sizes have led to increased MCC scores. It seems that omitting old data helps to better detect out-of-distribution examples. This might be an interesting direction of future work and might spur the development of new specialized techniques for lifelong and open-world learning in evolving graphs.

To reflect our work in the broader context of lifelong or continual learning, we reconsider the gradient episodic memory framework [24] for image data, in which examples are independent. To cast graph data into independent examples for vertex classification, certain preprocessing steps are required, such as transforming each vertex into a graph [26]. This increases the number of inference steps by $\mathcal{O}(|V|)$ compared to our approach.

Furthermore, we have shown that our approach of incremental training can be applied to various GNN architectures

and is orthogonal to sampling and preprocessing approaches. In general, our incremental training procedure can be applied to any GNN architecture with few caveats: If the GNN architecture depends on transductive learning, this constraint needs to be considered during incremental training. Similarly, any precomputation steps, such as computing normalizing constants like in GCN [4] or GraphSAINT [12], have to be performed again when adapting the model to a new task.

IX. Conclusion

We have created a new experimental framework for lifelong learning in evolving graphs, for which we contribute three new datasets. We have investigated four variants of lifelong learning in graph-structured data including learning with limited labeled data and unseen class detection. These different settings reflect numerous difficulties that practitioners face when using graph neural networks in real-world applications. In this setup, we have evaluated five representative GNN architectures as well as an MLP baseline under an incremental training procedure.

We have combined graph neural networks with a generic module for unseen class detection to tackle open-world learning problems. We have further introduced a new measure for k-neighborhood time differences tdiff_k that captures temporal connectivity patterns in graphs. By using percentiles of these time differences, we can select history sizes that are equivariant to the temporal granularity of the datasets.

Our results show that high levels of accuracy can be preserved, even when training only on a fraction of past data. In particular, using warm restarts becomes more important when few past data are available, or when few vertices are annotated with labels. Surprisingly, our results suggest that using less past data might even be beneficial for unseen class detection. As such, our work sets the ground for the development of further techniques that enhance lifelong learning on evolving graphs and close the gap between research and application.

Data Availability and Reproducibility

We published our lifelong graph learning datasets (zenodo.org/record/3764770#.YCQUOHWYXmg) along with an implementation of our experimental framework (github.com/lgalke/lifelong-learning).

Acknowledgment

Parts of this research were carried out on the computing infrastructure *bwCloud* of the State of Baden-Würtemberg and the Bioinformatics and Systems Biology group at Ulm University.

References

[1] R. Geirhos, J. Jacobsen, C. Michaelis, R. S. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut learning in deep neural networks," *Nat Mach Intell*, vol. 2, 2020.

[2] W. L. Hamilton, *Graph Representation Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, 2009.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[5] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[7] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2019.

[8] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, 2018.

[9] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NeurIPS*, 2018.

[10] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[11] X. Da, R. Chuanwei, K. Evren, K. Sushant, and A. Kannan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.

[12] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *ICLR*, 2020.

[13] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*. ACM, 2014.

[15] S. Thrun and T. M. Mitchell, "Learning one more thing," in *IJCAI*. Morgan Kaufmann, 1995.

[16] G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *KDD*. ACM, 2016.

[17] Z. Chen and B. Liu, *Lifelong Machine Learning, Second Edition*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2018.

[18] L. Shu, H. Xu, and B. Liu, "DOC: deep open classification of text documents," in *EMNLP*, M. Palmer, R. Hwa, and S. Riedel, Eds. ACL, 2017.

[19] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*. PMLR, 2019.

[20] L. Galke, B. Franke, T. Zielke, and A. Scherp, "Lifelong learning of graph neural networks for open-world node classification," in *International Joint Conference on Neural Networks*. IEEE, 2021.

[21] L. Galke, I. Vagliano, and A. Scherp, "Can graph neural networks go "online"? an analysis of pretraining and inference," in *Representation Learning on Graphs and Manifolds, ICLR Workshop*, 2019.

[22] B. Wang, Y. Chen, X. Li, and J. Chen, "Lifelong classification in open world with limited storage requirements," *Neural Comput.*, vol. 33, no. 7, pp. 1818–1852, 2021.

[23] P. Ruvolo and E. Eaton, "ELLA: an efficient lifelong learning algorithm," in *ICML*. JMLR.org, 2013.

[24] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *NIPS*, 2017.

[25] A. V. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connect. Sci.*, vol. 7, no. 2, 1995.

[26] C. Wang, Y. Qiu, and S. A. Scherer, "Lifelong graph learning," *CoRR*, vol. abs/2009.00647, 2020.

[27] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *CoRR*, 2020, abs/2003.00982.

[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[29] X. Bresson and T. Laurent, "Residual gated graph convnets," *CoRR*, vol. abs/1711.07553, 2017.

[30] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *CVPR*. IEEE, 2017.

[31] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *NeurIPS*, 2018.

[32] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD*. ACM, 2019.

[33] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. M. Bronstein, and F. Monti, "SIGN: scalable inception graph neural networks," *CoRR*, vol. abs/2004.11198, 2020.

[34] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *KDD*. ACM, 2020.

[35] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW*, 2018.

[36] J. B. Lee, G. Nguyen, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic node embeddings from edge streams," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2020.

[37] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *CoRR*, vol. abs/1805.11273, 2018.

[38] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl.-Based Syst.*, vol. 187, 2020.

[39] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," in *ICML*, 2017.

[40] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *ICONIP*. Springer, 2018.

[41] S. Kumar, X. Zhang, and J. Leskovec, "Learning dynamic embeddings from temporal interactions," *arXiv preprint arXiv:1812.02289*, 2018.

[42] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR*, 2019.

[43] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, 2020.

[44] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*, 2020.

[45] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *CoRR*, vol. abs/2006.10637, 2020.

[46] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *AAAI*, 2020.

[47] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," in *ICLR 2018*. OpenReview.net, 2018.

[48] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *NeurIPS*, 2018.

[49] D. Macêdo, T. I. Ren, C. Zanchettin, A. L. Oliveira, and T. Ludermir, "Entropic out-of-distribution detection," in *IJCNN*, 2021.

[50] D. Macêdo and T. Ludermir, "Improving entropic out-of-distribution detection using isometric distances and the minimum distance score," *arXiv preprint arXiv:2105.14399*, 2021.

[51] A. R. Dhamija, M. Günther, and T. E. Boult, "Reducing network agnostophobia," in *NeurIPS*, 2018.

[52] D. Hendrycks, M. Mazeika, and T. G. Dietterich, "Deep anomaly detection with outlier exposure," in *ICLR*. OpenReview.net, 2019.

[53] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Trans. Knowl. Data Eng.*, 2011.

[54] A. Bendale and T. E. Boult, "Towards open set deep networks," in *CVPR*. IEEE, 2016.

[55] M. Wu, S. Pan, and X. Zhu, "Openwgl: Open-world graph learning," in *ICDM*. IEEE, 2020.

[56] B. Fish and R. S. Caceres, "A task-driven approach to time scale detection in dynamic networks," in *Workshop on Mining and Learning with Graphs*, 2017.

[57] D. Ersan, C. Nishioka, and A. Scherp, "Comparison of machine learning methods for financial time series forecasting at the examples of over 10 years of daily and hourly data of DAX 30 and S&P 500," *J Comput Soc Sc*, vol. 3, 2020.

[58] S. Thrun, "Lifelong learning algorithms," in *Learning to learn*. Springer, 1998.

[59] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Comput. Surv.*, vol. 47, no. 1, May 2014.

[60] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018.

[61] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *NeurIPS*, 2020.

[62] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, 2008.

[63] L. Galke, F. Mai, I. Vagliano, and A. Scherp, "Multi-modal adversarial autoencoders for recommendations of citations and subject labels," in *26th Conference on User Modeling, Adaptation and Personalization*. ACM, 2018.

[64] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary physics*, vol. 46, no. 5, 2005.

[65] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, 2016.

[66] G. I. Webb, L. K. Lee, B. Goethals, and F. Petitjean, "Analyzing concept drift and shift from sample data," *Data Mining and Knowledge Discovery*, vol. 32, no. 5, 2018.

[67] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*. JMLR.org, 2010.

[68] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[69] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE Trans. Information Theory*, vol. 37, no. 1, 1991.

[70] M. Wang, *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[71] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[72] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016.

[73] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, no. 1, 2020.

APPENDIX A
PROOF: MEASURE tdiff_k IS EQUIVARIANT TO TEMPORAL GRANULARITY

Consider the point in time of an event as a continuous function $t : \mathcal{V} \to \mathbb{R}$. Then, we consider a measurement $t_g : \mathcal{V} \to \mathbb{N}$ that map the time into a discrete space with granularity g. For example, when $t(u)$ provides the time in (continuous) seconds, then a measurement with the precision of one minute would be $t_{\text{minutes}}(u) : \lfloor \frac{t(u)}{60} \rfloor$. A measurement with the precision of one hour would be $t_{\text{hours}}(u) = \lfloor \frac{t(u)}{3600} \rfloor$. We call these measurements with fixed precision *granularity* functions. If we only had the granularity of minutes, we could still obtain the granularity of hours: $t_{\text{hours}}(u) = \lfloor \frac{t_{\text{minutes}}(u)}{60} \rfloor$ because hours are coarser than minutes. In fact, any two measurements differ by a constant factor with one being more coarse-grained (larger denominator) than the other, or both being equal.

a) Theorem: tdiff_k is equivariant to different temporal granularity of the underlying data.

$$\mathrm{tdiff}_k(V, E, a \cdot t \pm a) \in a \cdot \mathrm{tdiff}_k(V, E, t) \pm a \tag{2}$$

with $a \in \mathbb{R}^+$ and $a \cdot t \pm a$ being a shorthand for a function t' that satisfies $a \cdot t(u) \in t'(u) \pm a \forall u \in V$.

b) Proof: Let $t_y, t_m : \mathcal{V} \to \mathbb{N}_{>0}$ be two functions whose values differ by a constant factor $a \in \mathbb{R}^+$, such that $t_y(u) = \lfloor \frac{t_m(u)}{a} \rfloor \forall u \in \mathcal{V}$. Here, we assume without loss of generality that t_m is more fine-grained than t_y, i. e., $a > 1$. Otherwise, we could swap t_y and t_m and proceed with the reciprocal of a. We will show that $a \cdot \mathrm{tdiff}_k^{(t_y)} = \mathrm{tdiff}_k^{(t_m)} \pm a$, or more precisely:

$$\forall u, v \in \mathcal{V} : a \cdot |t_y(u) - t_y(v)| \in \\ ]|t_m(u) - t_m(v)| - a, |t_m(u) - t_m(v)| + a[$$

while recalling that tdiff_k is a multi-set of time differences.

We start by transforming the prerequisite:

$$t_y(u) = \left\lfloor \frac{t_m(u)}{a} \right\rfloor \tag{3}$$

$$\Rightarrow \frac{t_m(u)}{a} \leq t_y(u) < \frac{t_m(u)}{a} + 1 \tag{4}$$

$$\Rightarrow t_m(u) \leq a \cdot t_y(u) < t_m(u) + a \tag{5}$$

Using this, we now show that:

$$a \cdot |t_y(u) - t_y(v)| > |t_m(u) - t_m(v)| - a \tag{6}$$

$$a \cdot |t_y(u) - t_y(v)| < |t_m(u) - t_m(v)| + a \tag{7}$$

We first prove the lower bound (Inequality 6) in two cases:

Case 1a: $t_y(u) = t_y(v)$. Then the left-hand side becomes zero and it remains to show that $|t_m(u) - t_m(v)| < a$. We apply Inequality 5 to find the highest possible value for $|t_m(u) - t_m(v)|$ with respect to t_y, which is $|t_y(u) + a - \epsilon - t_y(v)|$ with $\epsilon > 0$. Now as $t_y(u) = t_y(v)$, we obtain $a - \epsilon < a$.

Case 1b: $t_y(u) \neq t_y(v)$. We transform the left-hand side of Inequality 6 to $|at_y(u) - at_y(v)|$, while recalling that $t_y \in \mathbb{N}_{>0}$. Then we use Inequality 5 to obtain $|t_m(u) - t_m(v)|$ as smallest possible value of the left hand side. As $a > 1$, Inequality 6 holds.

Now, we prove the upper bound (Inequality 7) analogously in two cases:

Case 2a: $t_y(u) = t_y(v)$. The left-hand side becomes zero and $0 < |t_m(u) - t_m(v)| + a$ is trivial as $a > 1$.

Case 2b: $t_y(u) \neq t_y(v)$. Again, we transform the left-hand side of Inequality 7 to $|at_y(u) - at_y(v)|$. This time, we are interested in the highest possible value with respect to Inequality 5, which is $|t_m(u) + a - \epsilon - t_m(v)|$ with $\epsilon > 0$. Since $|a - \epsilon| \geq 0$, we obtain

$$\begin{aligned} |t_m(u) + a - \epsilon - t_m(v)| &\leq |t_m(u) - t_m(v)| + a - \epsilon \\ &< |t_m(u) - t_m(v)| + a \end{aligned}$$

which concludes the proof.

APPENDIX B
EXTENDED RESULTS FOR UNSEEN CLASS DETECTION

In Table V, We report the extended results of our experiments on unseen class detection. The measures we report are rejection MCC and Open F1 Macro. The MCC score reflects the capabilities to detect unseen classes. The Open F1 Macro score that reflects all classes including a special class for instances from unseen classes.

TABLE V
EXTENDED RESULTS FOR UNSEEN CLASS DETECTION ON DBLP-HARD WITH GRAPHSAGE AS BASE MODEL (AVERAGE OF 5 REPETITIONS). α INDICATES THAT RISK REDUCTION IS USED WITH THE RESPECTIVE FACTOR FOR THE STANDARD DEVIATION, τ IS THE MINIMUM THRESHOLD. RUNS NAMED DOC-w ARE MODELS WITH WEIGHTED TRAINING.

		MCC		**Open F1 Macro**	
		cold	warm	cold	warm
c	**Open Learning Method**				
1	DOC ($\tau = 0.50$)	.01	.04	.01	.01
	DOC ($\tau = 0.25$)	.01	.02	**.04**	**.04**
	DOC ($\tau = 0.50, \alpha = 3.0$)	.01	.02	.01	.01
	DOC ($\tau = 0.50, \alpha = 1.5$)	.01	.02	.01	.01
	DOC ($\tau = 0.25, \alpha = 1.5$)	.01	.04	**.04**	**.04**
	gDOC ($\tau = 0.75$)	.04	**.09**	**.13**	**.13**
	gDOC ($\tau = 0.50$)	.04	**.05**	**.13**	**.13**
	gDOC ($\tau = 0.25$)	.01	-	**.13**	**.13**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.04	**.05**	**.13**	**.13**
	gDOC ($\tau = 0.50, \alpha = 1.5$)	**.05**	**.05**	**.13**	**.13**
	gDOC ($\tau = 0.25, \alpha = 1.5$)	**.05**	.00	**.13**	**.13**
	gDOC ($\tau = 0.75, \alpha = 3.0$)	.04	**.09**	**.13**	**.13**
	gDOC ($\tau = 0.75, \alpha = 1.5$)	.04	**.09**	**.13**	**.13**
3	DOC ($\tau = 0.50$)	.02	.03	.02	.05
	DOC ($\tau = 0.25$)	.02	.04	.06	.12
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.02	.05
	DOC ($\tau = 0.50, \alpha = 1.5$)	.02	.03	.02	.05
	DOC ($\tau = 0.25, \alpha = 1.5$)	.02	.04	.06	.12
	gDOC ($\tau = 0.75$)	.05	**.08**	**.15**	**.15**
	gDOC ($\tau = 0.50$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.25$)	.01	.00	**.15**	**.15**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.50, \alpha = 1.5$)	.06	.06	**.15**	**.15**
	gDOC ($\tau = 0.25, \alpha = 1.5$)	.06	.04	**.15**	**.15**
	gDOC ($\tau = 0.75, \alpha = 3.0$)	.05	**.08**	**.15**	**.15**
	gDOC ($\tau = 0.75, \alpha = 1.5$)	.05	**.08**	**.15**	**.15**
6	DOC ($\tau = 0.50$)	.02	.03	.05	.08
	DOC ($\tau = 0.25$)	.02	.05	.11	.14
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.05	.08
	DOC ($\tau = 0.50, \alpha = 1.5$)	.02	.03	.05	.08
	DOC ($\tau = 0.25, \alpha = 1.5$)	.02	.05	.11	.14
	gDOC ($\tau = 0.75$)	.05	**.07**	**.16**	**.16**
	gDOC ($\tau = 0.50$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.25$)	.02	.02	**.16**	.15
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 1.5$)	.06	.06	**.16**	**.16**
	gDOC ($\tau = 0.25, \alpha = 1.5$)	.06	**.07**	**.16**	**.16**
	gDOC ($\tau = 0.75, \alpha = 3.0$)	.05	**.07**	**.16**	**.16**
	gDOC ($\tau = 0.75, \alpha = 1.5$)	.05	**.07**	**.16**	**.16**
∞	DOC ($\tau = 0.50$)	.02	.04	.08	.12
	DOC ($\tau = 0.25$)	.03	.06	.13	**.16**
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.04	.08	.12
	DOC ($\tau = 0.50, \alpha = 1.5$)	.02	.04	.08	.12
	DOC ($\tau = 0.25, \alpha = 1.5$)	.04	**.06**	.13	**.16**
	gDOC ($\tau = 0.75$)	.05	.07	**.16**	**.16**
	gDOC ($\tau = 0.50$)	.04	.05	**.16**	**.16**
	gDOC ($\tau = 0.25$)	.02	.01	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.05	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 1.5$)	.06	.06	**.16**	**.16**
	gDOC ($\tau = 0.25, \alpha = 1.5$)	.06	.06	**.16**	**.16**
	gDOC ($\tau = 0.75, \alpha = 3.0$)	.05	**.07**	**.16**	**.16**
	gDOC ($\tau = 0.75, \alpha = 1.5$)	.05	**.07**	**.16**	**.16**