2024 IEEE/WIC International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) | 979-8-3315-0494-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/WI-IAT62293.2024.00036

Lifelong Graph Learning for Graph Summarization

Jonatan Frank, Marcel Hoffmann,
Nicolas Lell
Data Science and Big Data Analytics
Ulm University, Germany
firstname.lastname@uni-ulm.de

David Richerby
University of Essex, Colchester, UK
david.richerby@essex.ac.uk

Ansgar Scherp
Data Science and Big Data Analytics
Ulm University, Germany
ansgar.scherp@uni-ulm.de

***Abstract*—Summarizing web graphs is challenging due to the heterogeneity of the modeled information and its changes over time. We investigate the use of neural networks for lifelong graph summarization. After observing the web graph at a point in time, we train a network to summarize graph vertices. We apply this trained network to summarize the vertices of the changed graph at the next point in time. Subsequently, we continue training and evaluating the network to perform lifelong graph summarization. We use the GNNs Graph-MLP and GCN, as well as an MLP baseline, to summarize the temporal graphs. We compare 1-hop and 2-hop summaries. We investigate the impact of reusing parameters from a previous snapshot by measuring backward and forward transfer as well as forgetting rate. Our extensive experiments are on two series of ten weekly snapshots, from 2012 and 2022, of a web graph with over 100M edges. They show that all networks predominantly use 1-hop information to determine the summary, even when performing 2-hop summarization. Due to the heterogeneity of web graphs, in some snapshots, the 2-hop summary produces up to ten times as many vertex classes as the 1-hop summary. When using the network trained on the last snapshot from 2012 and applying it to the first snapshot of 2022, we observe a strong drop in accuracy. We attribute this drop over the ten-year time warp to the strongly increased heterogeneity of the web graph in 2022. The source code and additional resources are available at https://github.com/jofranky/Lifelong-Graph-Summarization-with-Neural-Networks.**

***Index Terms*—temporal graphs, lifelong graph learning, neural networks, graph neural networks, graph summary, RDF graph**

I. Introduction

Graph summarization is the generation of a small representation S of an input graph G, that preserves structural information necessary for a given task [1]. Calculating the graph summary itself can be computationally expensive [2]. However, the reduction in size allows tasks such as web data summary search [3] and data visualization [4], to be computed much faster than on the original graph [2]. In machine learning, graph summarization is used to improve the scalability of node embedding learning methods [5].

The information preserved by a summary is defined by the summary model and considers the vertices' k-hop neighborhoods. Thus, in a machine learning sense, graph summarization can be expressed as a vertex classification task where each vertex in the graph belongs to a certain summary [6], representing the class. This summarization operation has to be permutation invariant to the input, i. e., the vertices' k-hop neighborhood [2, 3]. Graph Neural Networks (GNNs) are designed to be permutation invariant in this way [6].

Blasi et al. [6] applied multiple GNNs to graph summarization. They considered graph summaries based on vertex equivalence classes [4] which are lossless with respect to specific features, such as the edge label of the outgoing edges of a vertex. The authors used the 6 May, 2012 snapshot of the Dynamic Linked Data Observatory (DyLDO) dataset [7]. We extend the analysis of Blasi et al. [6] with a temporal component by considering ten consecutive snapshots of DyLDO from each of 2012 and 2022. We evaluate reusing networks trained on 2012 for 2022, which we refer to as a time warp.

The problem with training a network only on the first snapshot is that performance decreases when the graph changes over time [8]. This is especially true for graph summaries, where classes may appear or disappear between snapshots [6]. One simple solution is learning the summaries for each snapshot from scratch, resulting in a separate network per snapshot without any information about summaries of previous snapshots. In this work, we address this problem by incrementally learning the network. We reuse the network trained on previous snapshots as a base for learning the summaries on the next snapshot. This is called lifelong graph learning [9].

We first calculate the summaries of the vertices of each snapshot using a crisp algorithm [2]. These summaries are used as the gold standard for training the GNNs. We continually train the networks on one snapshot after another. After training on a snapshot, we evaluate the network on that snapshot, and all past and future snapshots. We measure forward transfer, backward transfer, and forgetting rates of the networks based on classification accuracy [10, 11]. Forgetting is the performance drop of a network on snapshots trained on in earlier tasks after training on subsequent tasks [12]. Our results show that a network performs best on each snapshot when it is trained on the sequence of tasks up to and including that snapshot. This observation is more prominent for the 2-hop than the 1-hop summary. After the ten-year time warp, reusing the network parameters from 2012 neither improves nor harms the performance in 2022. In summary, our contributions are:

- We extend graph summarization using GNNs from static to temporal graphs. We experiment with two GNNs and a baseline MLP on sequences of ten weekly snapshots from 2012 and 2022 of the DyLDO web graph.
- We show that neural networks perform best on each snapshot when being trained on a sequence of tasks up to

979-8-3315-0494-6/24/$31.00 ©2024 IEEE
DOI 10.1109/WI-IAT62293.2024.00036

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

and including that snapshot. Changes in the graph reduce the performance which is more prominent in 2022.

- We show that an MLP using only 1-hop information is sufficient for 2-hop summaries.
- Reusing parameters from a network trained in 2012 has no benefit over a network trained from scratch in 2022.

The remainder of the paper is organized as follows: Below, we summarize related work. Section III introduces graph summarization. Section IV describes the experimental apparatus. Our results are presented in Section V. Section VI discusses the results, before we conclude.

II. Related Work

A. Graph Neural Networks

GNNs use the structural information of a graph contained in its edges and the features of the neighboring vertices to distinguish different vertices, e. g., to classify them. Many GNNs, including graph convolutional networks (GCNs) [13], graph attention networks (GANs) [14] and GraphSAINT [15] use message passing. GraphSAINT is a scalable GNN architecture that samples smaller subgraphs to enable any GNN to be trained on large graphs. Hu et al. [16] introduced Graph-MLP, a GNN that does not use message passing. Graph-MLP consists of a multi-layer perceptron (MLP) that is trained with a cross-entropy loss and a neighbor contrastive (NContrast) loss on the graph edges.

B. Graph Summarization

FLUID [2] is a language and a generic algorithm for flexibly defining and efficiently computing graph summaries. It can express all existing lossless structural graph summaries w.r.t. the considered features, such as the vertices' edges. Computation of graph summaries is based on hash functions applied on a canonical order of the vertices features [2, 3]. The use of neural networks as a hash function has mainly been in the context of security [17, 18], but recently Blasi et al. [6] applied GCNs, GraphSAGE [19], GraphSAINT and Graph-MLP to summarizing static graphs, along with Blooom filters. We use a GraphSAINT-based sampling method by Blasi et al. [6], which considers the class distribution by sampling inverse to the number of occurrences of classes in the training set.

C. Lifelong Graph Learning

Lifelong learning [9] is a learning procedure that imitates the lifelong learning ability of humans. It adds the importance of transferring and refining knowledge to the learned network. In lifelong learning [20], a network at time t has performed a sequence of t learning tasks $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_t$ and has accumulated knowledge from these past tasks. At time $t+1$, it is faced with a new learning task $\mathcal{T}_{t+1}$. The network can use past knowledge to help with learning task $\mathcal{T}_{t+1}$.

A lifelong learning model should be able to exploit previous knowledge to learn new tasks better or faster. One issue in lifelong learning is catastrophic forgetting [12]. Catastrophic forgetting is the tendency of neural networks to drastically forget previous knowledge upon learning new information.

In lifelong graph learning, a challenge is that the graphs can grow or shrink over time. Adding and removing vertices in the graphs may also result in the inclusion of new class labels in classification tasks [8]. We address the aspect of increasing the detection accuracy of already-seen classes and adding new classes to our networks. Galke et al. [21] compare reusing an existing network (warm restart) versus training a new network from scratch (cold restart). Their experiments show that warm restarts are generally preferred for lifelong graph learning.

III. Graphs and Summarization

We define the basic notation of graphs used as input for our summary models. We introduce classical graph summarization, which is used to compute the gold standard, and graph summarization with neural networks. We describe our sampling approach and our training method. Finally, we describe how graph summarization is used in the lifelong learning setting.

A. Labeled Graphs

We consider web graphs with labeled relations of the form $G = (V, E, R)$ consisting of a set of vertices V, a set of relation types R, and a set of labeled edges $E \subseteq V \times R \times V$ that express relationships between entities in V. This allows graphs to be represented in the Resource Description Framework (RDF),[1] a W3C standard that models relations in the form of subject–predicate–object triples (s, p, o) [22]. `rdf:type` is a special predicate in the RDF standard. It assigns a vertex s a label o, i. e., a vertex type. This is done by the triple $(s, \texttt{rdf:type}, o)$. The predicates $p \neq \texttt{rdf:type}$ are called RDF properties. Our summary models focus on the properties and do not consider `rdf:type` following Gottron et al. [23]. They argue that 20% of all data providers of web graphs do not need RDF types because the vertices are already precisely described by the edges.

B. Classical Graph Summarization

A graph summary maps a graph G to a smaller representation S, which is a graph that preserves structural information of G [24]. Each vertex of the original graph G is a member of exactly one equivalence class (EQC) of the summary graph. Each EQC represents a set of vertices that share common features, such as the same set of edge labels. We formulate graph summarization as a vertex classification task following Blasi et al. [6]. Our summary models depend on the k-hop neighborhood of a vertex. We use the 1- and 2-hop versions of Attribute Collection, M_{AC1} and M_{AC2}, respectively. M_{AC1} considers a vertex's RDF properties and M_{AC2} also includes the neighbors' properties. These models use of outgoing edges and only differ in the size of the neighborhood that is considered for the summary. In $M_{\text{AC}k}$, we calculate the EQC of a vertex x by calculating the following function $h_k(x)$ that considers the k-hop neighborhood of the vertex. We set $h_1(x) = 0$ and then, for $k \geq 1$,

$$h_{k+1}(x) \quad = \bigoplus_{(x,p,y) \in E} \operatorname{hash}(p, h_k(y)) \,,$$

[1] https://www.w3.org/RDF

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

where $\bigoplus$ is the bitwise XOR operator and the function $\text{hash}(p, h_i(y))$ first concatenates p and $h_i(y)$ to a string, which is then hashed. We use the default hashing function for strings in Python, which uses SipHash [25, 26].

Figure 1 illustrates summarizing a vertex. Our model considers, at most, the 2-hop neighborhood of a vertex. In the case of vertex v_1 (center), we recursively consider edges in the 2-hop neighborhood of vertex v_1. Hence, we consider each edge in Figure 1 except the edge (v_6, p_2, v_9), which extends to the three-hop neighborhood.

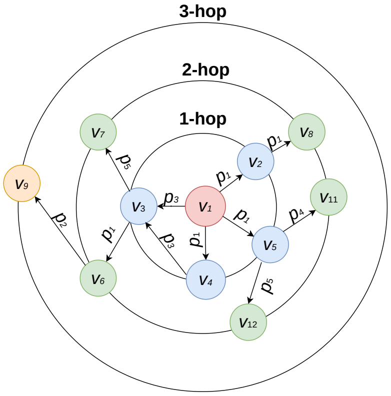


Fig. 1: 3-hop neighborhood of a vertex v_1.

We generalize this classical graph summarization by considering graphs that change over time. For a number of hops $i \in \{1, 2\}$, the graph summary S_t of a graph $G_t = (V_t, E_t, R_t)$ at time t is a tuple (V_t^S, E_t^S, R_t^S). $V_t^S = C \cup D$, where C is the set of EQCs computed from the vertices V_t, and $D = \{d_{p,h_i(y)} \mid (x, p, y) \in E_t \text{ for some } x\}$ is a set of vertices further specifying the EQCs. These are the primary and secondary vertices of Blume et al. [2]. R_t^S is the set of predicates of the original graph G_t. For $d_{p,h_i(y)} \in D$, the outgoing edges E_t^S of the summary vertices V_t^S are given by $E_t^S = \{(v, p, d_{p,h_i(y)}) \mid (x, p, y) \in E_t \text{ and } x \in C\}$.

C. Neural Graph Summarization

When applying graph neural networks (GNNs) for graph summarization, we need to define the vertex features [6, 27]. Instead of hashing, as above, the feature vector $h_v^{(0)}$ of a vertex v is a multi-hot encoding of its outgoing edge labels, following Blasi et al. [6]. The motivation is that the multi-hot encoded feature vector can be easily extended with new elements once new predicates appear in the temporal graph. We note that multi-hot encoding is appropriate for graph summarization, as graph summary models only depend on the existence of edge or vertex labels, not their multiplicity [2].

For the 1-hop model M_{AC1}, the feature vector already contains all the information needed to calculate the summary, which is the label set of the vertex's edges. For the 2-hop model M_{AC2}, we consider the feature vector of v_1 and, in addition, consider the feature vectors of v_1's neighbors. From these feature vectors, we know the labels of the outgoing edges of v_1 and its neighbors. Note that, in classical summarization, we also know which edge labels connect to which neighboring feature vectors. Here, we know the set of edge labels and the set of neighboring feature vectors but not which edge is associated with which neighbor.

An approach to incorporate the edges is to represent an edge between a vertex and its neighbor as a vertex between them, labeled with the edge label one-hot encoded as a feature vector. In this case, we use two hidden layers in the GNN to cover the additional hop. Where this procedure is not applied, we use one hidden layer. We evaluate both approaches, i. e., test whether message passing is necessary for a 2-hop summary

For lifelong learning, we consider different snapshots of a graph G_t associated with a timestamp t. At time t, we train a neural network to predict the summary S_t of the graph G_t at time t, which we refer to as task $\mathcal{T}_t$. We compute the number of EQCs present in the summary of time t for each time t, and the number of added and deleted EQCs compared to time $t-1$. From this, we can also compute how many EQCs from time $t-1$ are still present at time t. We also count how many different EQCs in total have appeared in the snapshots $S_1, \dots, S_t$, which corresponds to the size of the neural network's output layer at time t.

We use two GNNs in our study: a GCN [13], a classical message-passing model, and Graph-MLP [16], which is an alternative to the usual message-passing GNNs.

Due to the large size of our graphs, we use the sampler of Blasi et al. [6] for training. The sampler is based on the GraphSAINT vertex sampler, with each vertex weighted according to the class distribution. This takes into account the imbalanced class distributions of the snapshots.

D. Complexity Analysis

Our approach has two steps: first, summarizing a graph w.r.t. a given summary model and, second, training the graph neural network. For the first step, we iteratively compute summaries of a snapshot-based temporal graph. This can be computed with an incremental, parallel algorithm in time $\mathcal{O}(N\,\Delta^k)$ [28], where N is the number of changes in the graph from one snapshot to the next, Δ is the maximum degree, and k is the number of hops considered in the summary model – for us, $k \in \{1, 2\}$. This applies to summary models definable in FLUID, excluding any optional preprocessing step involving inference. The approach is parallelized and scales to large graphs.

The complexity of the second step depends on the specific neural network, the sample of vertices $V' \subseteq V$, number of layers K, and hidden dimension d. Training and inference time is $\mathcal{O}(Kd(|E| + |V|d))$ for GCN [13], training time is $\mathcal{O}(Kd(|E| + |V|d))$ and inference time is $\mathcal{O}(Kd)$ for Graph-MLP [16], and $\mathcal{O}(Kd)$ for MLP [16].

IV. Experimental Apparatus

In this section, we introduce and analyze our dataset, and describe our experimental procedure. Afterward, we present how we optimize our hyperparameters. Finally, we present performance and forgetting measures for the evaluation of lifelong graph learning.

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

A. Datasets: Weekly DyLDO Snapshots from 2012 and 2022

We use the weekly snapshots of the DyLDO web crawl as foundation [7]. Starting from a seed of $90,000$ Uniform Resource Identifiers, it samples data from the web to create a snapshot. A snapshot consists line-based, plain text encoding of an Resource Description Framework (RDF) graph [29].

We create two datasets, using the following sequences of snapshots: The ten snapshots from 6 May, 2012 to 8 July, 2012 (the first ten crawls) and the ten snapshots from 25 September, 2022 to 27 November, 2022. Figures 2a and 2b show the number of EQCs in each snapshot.

For each snapshot, we create the labels of the vertices by calculating their EQCs according to the summary model. For M_{AC2} we remove all vertices with a degree over 100 from the training, validation, and test sets to speed up the experiments. This has only a small effect on the number of EQCs: for example, in the first snapshot of DyLDO in 2012 only $12,425$ out of $288,418$ EQCs are removed, which is less than 5%.

B. Dataset Analysis

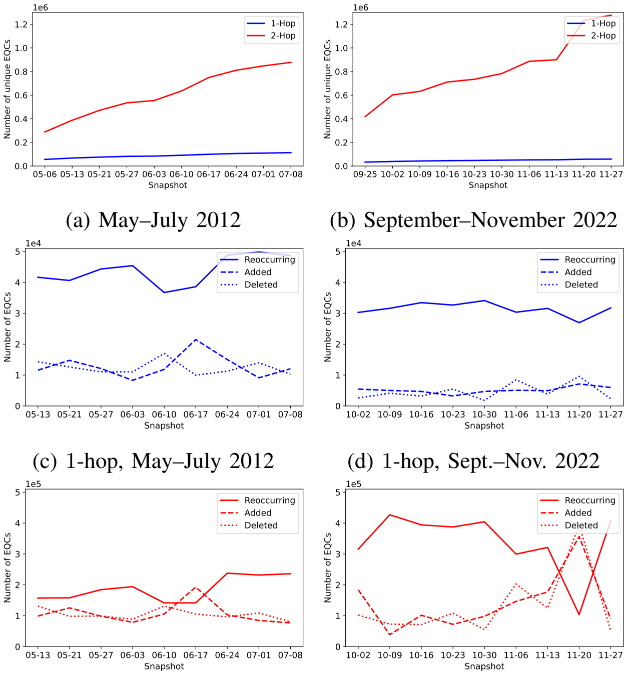
Figure 2 shows how the EQCs in each year change from snapshot to snapshot. We observe that the EQCs of the 2-hop summary change more than those of the 1-hop summary. Both changes underlie the same snapshot modification, indicating that the 2-hop summary changes more than the 1-hop summary in the same sequence of snapshots. In general, there are more EQCs in the 2-hop summaries in 2022 than in 2012. The numbers of EQCs in the 1-hop summaries are similar between the two years.

C. Procedure

a) Training of GNNs and Baselines: Based on pre-experiments, we train all networks with 100 batches, each containing up to $1,000$ vertices. Our baseline is a standard MLP with one hidden layer and ReLU-activation functions with dropout. For M_{AC1}, we use an MLP because all information is contained in the feature vector. We have two cases for M_{AC2}, one with encoded edge information and one without. In the first case, we only use GCN because the other neural networks do not use neighborhood information during inference. In the second case, we use all neural networks, i. e., GCN, MLP, and Graph-MLP.

b) Lifelong Graph Learning for Summarizing the DyLDO Snapshots: We perform the lifelong graph learning experiments once for the 2012 and once for the 2022 snapshots. We train the networks on the snapshots in chronological order. We split the vertices of each snapshot into 2% validation, 5% testing and sample the training data from the remaining 93%.

For the first snapshot of each year, the neural networks are initialized with an input size of number of predicates and output size of the number of classes. Then, the networks are trained on the snapshot for 100 iterations with maximum batch size $1,000$. After training, we increase the input and output size if the next snapshot has more classes (EQCs) or predicates. The extended neural network is trained on the next snapshot for 100 iterations. We repeat this process until the last snapshot. After training the networks on a snapshot, we evaluate them on all snapshots of the same year. We do this to calculate the lifelong learning measures (see Section IV-E).



(a) May–July 2012

(b) September–November 2022

(c) 1-hop, May–July 2012

(d) 1-hop, Sept.–Nov. 2022

(e) 2-hop, May–July 2012

(f) 2-hop, Sept.–Nov. 2022

Fig. 2: Detailed analyses of the DyLDO snapshots used in our experiments. Figures (a) and (b) show the number of unique EQCs per snapshot for a 1-hop versus 2-hop model. Figures (c)–(f) show the changes of the EQCs (addition, deletion, and recurring) compared to the previous snapshot.

c) Time Warp Experiment: In this experiment, we compare three networks on the first snapshot of 2022. One "10-year-old" network is only trained on the 2012 snapshots, one is initialized from the old network but also trained on the 2022 snapshot, and one which is trained from scratch just on the 2022 snapshot. This experiment shows whether reusing old networks of 2012 is helpful despite the time gap of ten years, compared to starting training from scratch in 2022.

D. Hyperparameter Optimization

For the hyperparameter search, we apply grid search. We tune the learning rate on the values $\{0.1, 0.01, 0.001\}$, dropout in $\{0.0, 0.2, 0.5\}$, and hidden layer size $\{32, 64\}$ for the GNN. For Graph-MLP, we optimize the weighting coefficient $\alpha \in \{1.0, 10.0, 100.0\}$, the temperature $\tau \in \{0.5, 1.0, 2.0\}$, learning rate in $\{0.1, 0.01\}$, and the hidden layer size $\{64, 256\}$. For MLP, we use a hidden layer size of $1,024$ following prior works [6, 30]. We use Adam as an optimizer.

The best configuration depends on the summary model. Details can be found in our supplementary material [31]. For the summary model M_{AC1}, we use an MLP with a hidden size

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

of $1,024$, a dropout of 0.5, and a learning rate of 0.01. For M_{AC2} with edge information, we use GCN with two hidden layers of size 32, learning rate 0.1, and no dropout. For M_{AC2} with no edge information, we use an MLP with one hidden layer of size $1,024$, dropout 0.5, and learning rate 0.01. We use Graph-MLP with hidden size 64, dropout 0.2, learning rate 0.01, $\alpha = 1$, and $\tau = 2$. GCN uses one hidden layer of dimension 64, learning rate 0.1, and no dropout.

E. Measures

We assess vertex classification performance using test accuracy, a well-known measure in graph neural networks like GCN [13]. For lifelong learning, we use the measures from Lopez-Paz and Ranzato [10] and Chaudhry et al. [11].

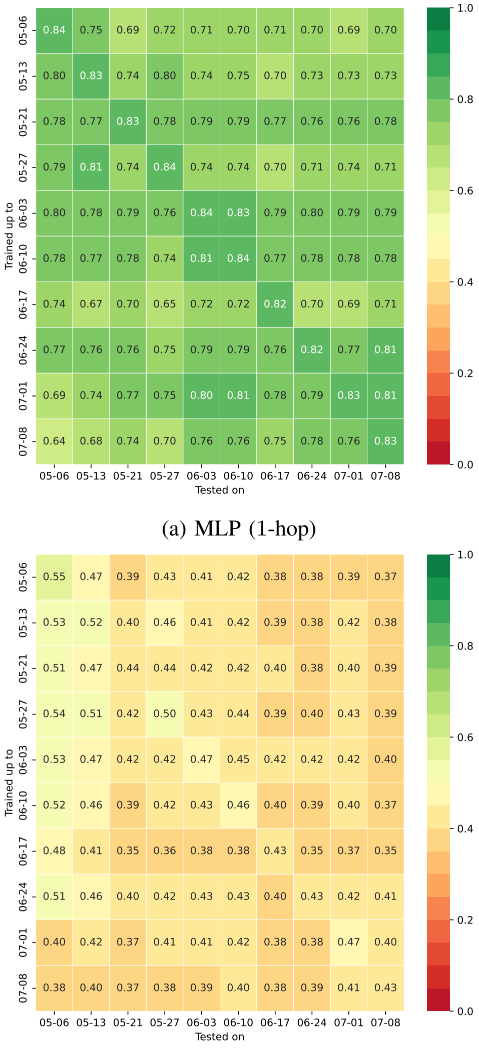
We train a network for each snapshot and test it on all other snapshots. Thus, for each of the years 2012 and 2022, we compute a result matrix $R \in \mathbb{R}^{T \times T}$ with $T = 10$. Each entry $R_{i,j}$ is the test accuracy of the network on task $\mathcal{T}_j$ after observing the last sample from task $\mathcal{T}_i$. That means the network is trained on tasks $\mathcal{T}_1$ to $\mathcal{T}_i$ and tested on $\mathcal{T}_j$. We use these matrices to calculate the following measures for each year, to measure the backward and forward transfer of the networks. The average accuracy (ACC) a network achieves on all tasks $\mathcal{T}_i$ after being trained on the last task $\mathcal{T}_T$ is given by $ACC = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}$. The backward transfer (BWT), which indicates how much knowledge of previous tasks the network keeps after being trained on the last task T is given by $BWT = \frac{1}{T-1}\sum_{i=1}^{T-1}(R_{T,i} - R_{i,i})$. The forward transfer (FWT), which shows how much knowledge of a task is reused for the next one, is based on Lopez-Paz and Ranzato [10] as $FWT = \frac{1}{T-1}\sum_{i=2}^{T}(R_{i-1,i} - R_{i,i})$.

We define the ideal performance as $\alpha_{\mathrm{ideal}} = \max_{i \in \{1,\ldots,T\}} R_{i,i}$. We define $\alpha_{i,\mathrm{all}} = \frac{1}{T}\sum_{t=1}^{T} R_{i,t}$ as the average accuracy of a network trained up to task $\mathcal{T}_i$. With this, we can define the following measures, as in [32]: $\Omega_{\mathrm{base}} = \frac{1}{T-1}\sum_{i=2}^{T}\frac{R_{i,1}}{\alpha_{\mathrm{ideal}}}$, $\Omega_{\mathrm{new}} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i,i}$, and $\Omega_{\mathrm{all}} = \frac{1}{T-1}\sum_{i=2}^{T}\frac{\alpha_{i,\mathrm{all}}}{\alpha_{\mathrm{ideal}}}$. The first snapshot is the base test set, and the best accuracy α_{ideal} in the matrix is the optimum. Ω_{base} measures how a network performs on the base test set after training on other sets compared to the optimum. Ω_{new} shows how a network performs on a snapshot after being trained on it. Ω_{all} measures how a network performs on all previous snapshots compared to the optimum.

Finally, we compute the forgetting measure [11]: $F_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_j^k$, where f_j^k is the forgetting on task j after the network is trained up to task k and is computed as: $f_j^k = \max_{\ell \in \{1,\ldots,k-1\}}(R_{\ell,j} - R_{k,j})$.

V. Results

We describe the results for the vertex classification and the lifelong learning measures. Finally, we show the results of the ten-year time warp.



(a) MLP (1-hop)

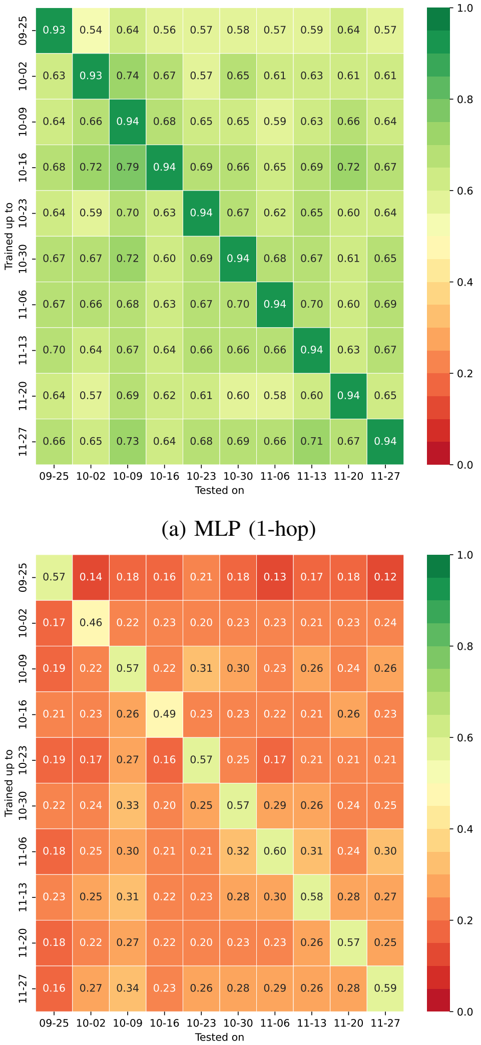
(b) MLP (2-hop)

Fig. 3: Accuracies for snapshots trained from May to July 2012. The accuracies for Graph-MLP (2-hop), GCN (2-hop), and GCN (2-hop and edges) are omitted because they are similar to the values of the MLP (2-hop).

a) Vertex Classification: The classification accuracies for the snapshots in 2012 and 2022 are shown in Figures 3 and 4. In general, the networks perform best on task i after they were trained up to snapshot i.

Regarding 2012, the performance of the MLP for M_{AC1} is much higher than for M_{AC2}. The difference between these two MLPs is between 0.25 and 0.41. Furthermore, the best performance of a network is observed when tested on the first snapshot independent of the training snapshots. For 2022, the results are similar to the experiments in 2012 but with overall lower accuracies. The MLP accuracies for M_{AC1} are again much higher than those for M_{AC2}. The difference is between 0.33 and 0.53. The best result for each snapshot i is achieved

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.



(a) MLP (1-hop)

(b) MLP (2-hop)

Fig. 4: Accuracies for snapshots trained from September to December 2022. Graph-MLP (2-hop), GCN (2-hop), and GCN (2-hop and edges) are omitted because they are similar to the values of the MLP (2-hop).

by the network that has been trained on snapshots 1 to i. In contrast to 2012, the results on the first snapshot are not better than in the subsequent snapshots.

b) Lifelong Learning Measures: The lifelong learning measures are shown in Table I. It shows that in 2012 and 2022, no network with any summary model or information achieved a positive forward or backward transfer in our setting. For the other measures, the MLP applied on M_{AC1} achieves the best results, and the results for M_{AC2} are similar, independent of the network and the year.

The forgetting of the networks in 2012 is shown in Figure 5. The MLP (1-hop) has the highest forgetting, with over 0.13, after being trained on the 10 June snapshot. Overall, the networks forget more after being trained on this snapshot. After the networks are trained on the last snapshot in 2012, they have a similar forgetting value.

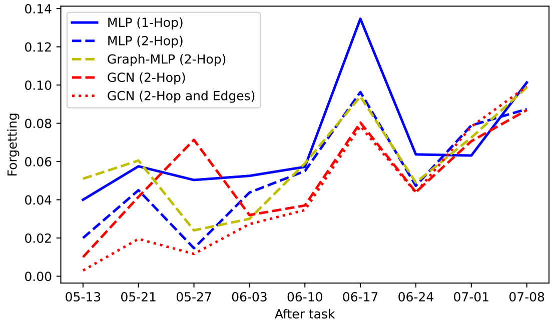


Fig. 5: The progress of the forgetting in 2012.

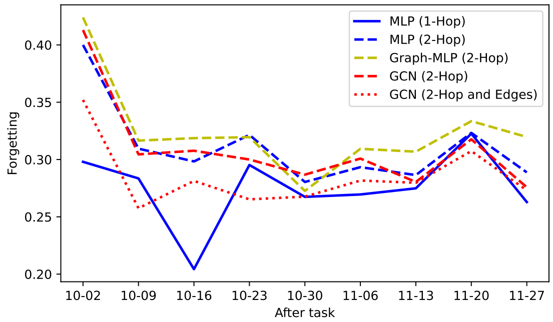


Fig. 6: The progress of the forgetting in 2022.

The forgetting of the networks in 2022, as shown in Figure 6, differs from the ones in 2012. The highest forgetting for the 2-hop networks is right in the beginning, and the lowest forgetting value is achieved by the MLP (1-hop) after being trained on the 16 October snapshot. In general, the forgetting values are higher in 2022 than in 2012. Nevertheless, after the networks have been trained on the last snapshot, they have similar forgetting values.

c) Ten-year Time Warp: We show the results of reusing a neural network from 2012 to classify vertices in 2022 in Table II. We call this setting a ten-year time warp (TW). Retraining the last network from 2012 on the first snapshot in 2022 gives the same results as training a network from scratch in 2022. Table II shows that the MLP for M_{AC1} performs a bit better in the time warp. For M_{AC2}, the time warp makes the performance of the neural network better in two of four cases, and worse in the other two. Reusing the last network from 2012 without retraining on the first 2022 snapshot gives very low accuracy – always below 0.03 (not shown in the table).

VI. Discussion

a) Vertex Classification: Our experiments show that the problem of graph summarization by vertex classification is

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Lifelong learning measures applied to the different networks and summaries of each year.

Year	Network	Type	ACC ↑	BWT ↑	FWT ↑	Ω_{base} ↑	Ω_{new} ↑	Ω_{all} ↑
2012	MLP	1-hop	0.739	−0.101	−0.062	0.898	0.829	0.817
2012	MLP	2-hop	0.392	−0.087	−0.045	0.827	0.462	0.673
2012	Graph-MLP	2-hop	0.420	−0.099	−0.049	0.831	0.499	0.675
2012	GCN	2-hop	0.376	−0.087	−0.043	0.824	0.445	0.672
2012	GCN	2-hop and edges	0.348	−0.098	−0.042	0.850	0.429	0.690
2022	MLP	1-hop	0.701	−0.263	−0.275	0.715	0.938	0.654
2022	MLP	2-hop	0.296	−0.289	−0.312	0.430	0.555	0.425
2022	Graph-MLP	2-hop	0.290	−0.320	−0.311	0.437	0.573	0.426
2022	GCN	2-hop	0.298	−0.276	−0.311	0.426	0.543	0.421
2022	GCN	2-hop and edges	0.269	−0.273	−0.282	0.431	0.514	0.426

TABLE II: Accuracy after training on the task from September 25, 2022. Trained from scratch (no time warp, no TW) or initialized with weights from July 8, 2012 (TW).

Neural network	Summary model	TW	no TW
MLP	1-hop	0.933	0.930
MLP	2-hop	0.581	0.572
Graph-MLP	2-hop	0.605	0.612
GCN	2-hop	0.565	0.572
GCN	2-hop and edges	0.526	0.515

hard when applied to a temporal graph with changing and unseen classes. The results from the snapshots in 2012 and 2022 (Figures 3 and 4) show that the classification of EQCs for M_{AC1} is easier than for M_{AC2}. This is caused by the increased number of EQCs when more hops are considered in the summary model. Figures 2a and 2b show that there are more unique EQCs for M_{AC2} than for M_{AC1}. The EQCs in M_{AC2} also change more rapidly. The 2022 snapshots have more edges than the ones from 2012. Overall, this reduces the performance. The more EQCs the network needs to learn and the more they change, the worse the network performs in general on all tasks.

b) Forward and Backward Transfer: We observe that no network has a positive forward or backward transfer between different snapshots. That means the networks perform better if they are trained on the task before being tested on it. The reason for the negative forward transfer is that new classes appear, and the reason for the negative backward transfer is that the neural network forgets classes. The results (Figures 3 and 4) confirm this statement, since a network that has been trained on $\mathcal{T}_1, \dots, \mathcal{T}_i$ always achieves its best accuracy on $\mathcal{T}_i$. This observation is confirmed by Ω_{new}, the average of the accuracies on tasks after a network has been trained, which was higher than the other accuracies. One explanation is that the EQCs of the snapshots change a lot, as can be seen in Figures 2c– 2f.

The Ω_{base} values in 2012 show that for this set of snapshots, the networks perform well on the first snapshot at each time. This is not the case for the networks for M_{AC2} in 2022, where the values are below 0.5. The networks do not perform as well as they did on the first snapshot. This shows that a network performs best on a task after being trained for it.

c) Forgetting and Reusing Networks: The ACC and the Ω_{all} value of a network indicate how well a neural network performs on previous tasks. Figures 5 and 6 show how much a network forgets over time. A higher ACC and Ω_{all} value do not necessarily mean a network forgets less. A good example is the MLP (1-hop) of 2012, which has a better ACC and Ω_{all} value than the 2-hop models, but the forgetting rate is higher.

d) Using 1-hop versus 2-hop Information in a 2-hop Summary Model: We investigate whether the information provided from a 1-hop neighborhood versus using the 2-hop neighborhood is helpful. The measures show that the networks for M_{AC2}, independent of the information used, performed similarly, and this held in both 2012 and 2022. Graph-MLP and GCN perform similarly to MLP for M_{AC2}, even though they have access to 2-hop information for the classification task and MLP does not. In summary, the results show that 2-hop information does not improve the performance for M_{AC2}. While this seems to be counter-intuitive, an explanation is that for the 2-hop summary, there are five to ten times more classes than in the 1-hop summary since the number of paths grows exponentially with the number of hops (see Figures 2a and 2b). This results in many small EQCs, i. e., many classes with low support that are hard to train on.

e) Warm versus Cold Restarts: Galke et al. [21] differentiate between warm restart and cold restart in the context of lifelong learning. A warm restart is reusing network parameters from previous tasks, and retraining from scratch is called a cold restart. Our time warp experiment (Table II) does not show a clear benefit of a warm restart compared to a cold restart over a ten-year range. However, it shows that a warm restart does not hurt a network's performance.

f) Limitations and Threat to Validity: The networks may perform better if we trained them for more iterations. However, when we increase the number of training iterations by an order of magnitude, we observe an improvement of only 2 points on the classification accuracy. Training more on the current task may also increase the forgetting of previous tasks. Furthermore, we use a hash function to compute the gold standard of the vertex labels, i. e., the EQCs. There could be collisions in the hash function. However, we chose the hash such that the probability is very low following Blasi et al. [6].

VII. Conclusion

We show that graph summarization for temporal graphs using a neural network is more challenging than the equivalent task on static graphs. The performance of the vertex classification in a temporal graph depends on the number of

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.

classes and the changes between timesteps. A problem with the summarization is that many new EQCs appear, and known ones disappear over time. We observe forgetting in lifelong graph summarization by vertex classification. We also observe that the performance of a network should not only be measured by the accuracy of the current task but also by other measures, like forward and backward transfer, which should be taken into account. The measures show that the networks for $M_{\mathrm{AC}2}$ perform similarly. With the time warp experiments, we see that reusing the network parameters does not necessarily improve lifelong learning measures.

Acknowledgements

This research is co-funded by the CodeInspector project (No. 504226141) of the DFG, German Research Foundation, and the 2LIKE project by the German Federal Ministry of Education and Research (BMBF) and the Ministry of Science, Research and the Arts Baden-Württemberg within the funding line Artificial Intelligence in Higher Education. The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

References

[1] Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou, and M. Zneika, "Summarizing semantic graphs: a survey," *VLDB J.*, 2019. [Online]. Available: https://doi.org/10.1007/s00778-018-0528-3

[2] T. Blume, D. Richerby, and A. Scherp, "FLUID: A common model for semantic structural graph summaries based on equivalence relations," *Theor. Comput. Sci.*, 2021. [Online]. Available: https://doi.org/10.1016/j.tcs.2020.12.019

[3] M. Konrath, T. Gottron, S. Staab, and A. Scherp, "SchemEX – efficient construction of a data catalogue by stream-based indexing of linked data," *J. Web Semant.*, 2012. [Online]. Available: https://doi.org/10.1016/j.websem.2012.06.002

[4] F. Goasdoué, P. Guzewicz, and I. Manolescu, "RDF graph summarization for first-sight structure discovery," *VLDB J.*, 2020. [Online]. Available: https://doi.org/10.1007/s00778-020-00611-y

[5] H. Zhou, S. Liu, D. Koutra, H. Shen, and X. Cheng, "A provable framework of learning graph embeddings via summarization," in *Proc. AAAI Conf. on Artificial Intelligence*, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i4.25621

[6] M. Blasi, M. Freudenreich, J. Horvath, D. Richerby, and A. Scherp, "Graph summarization as vertex classification task using graph neural networks vs. Bloom filter," in *9th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2022, Shenzhen, China, October 13-16, 2022*, 2022. [Online]. Available: https://doi.org/10.1109/DSAA54385.2022.10032447

[7] T. Käfer, J. Umbrich, A. Hogan, and A. Polleres, "Dyldo: Towards a dynamic linked data observatory," in *WWW2012, Lyon, France, 16 April, 2012*, 2012. [Online]. Available: http://ceur-ws.org/Vol-937/ldow2012-paper-14.pdf

[8] F. Zhou and C. Cao, "Overcoming catastrophic forgetting in graph neural networks with experience replay," in *Proc. AAAI Conf. on Artificial Intelligence*, 2021. [Online]. Available: https://doi.org/10.1609/aaai.v35i5.16602

[9] Z. Chen and B. Liu, *Lifelong Machine Learning, Second Edition*, 2018. [Online]. Available: https://doi.org/10.2200/S00832ED1V01Y201802AIM037

[10] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f87522788a2be2d171666752f97ddebb-Abstract.html

[11] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-030-01252-6_33

[12] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, 1989.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th ICLR, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th ICLR, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[15] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th ICLR, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. [Online]. Available: https://openreview.net/forum?id=BJe8pkHFwS

[16] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao, "Graph-mlp: Node classification without message passing in graph," *CoRR*, 2021. [Online]. Available: https://arxiv.org/abs/2106.04051

[17] M. Turčaník and M. Javurek, "Hash function generation by neural network," in *Proc. New Trends in Signal Processing (NTSP)*. IEEE, 2016.

[18] S. Lian, J. Sun, and Z. Wang, "One-way hash function based on neural network," *CoRR*, 2007. [Online]. Available: http://arxiv.org/abs/0707.4032

[19] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[20] G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *Proceedings of the 22nd SIGKDD, San Francisco, CA, USA, August 13-17, 2016*, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939835

[21] L. Galke, I. Vagliano, B. Franke, T. Zielke, M. Hoffmann, and A. Scherp, "Lifelong learning on evolving graphs under the constraints of imbalanced classes and new classes," *Neural Networks*, 2023. [Online]. Available: https://doi.org/10.1016/j.neunet.2023.04.022

[22] D. Beckett, "RDF 1.1 N-Triples," https://www.w3.org/TR/n-triples/, 2014.

[23] T. Gottron, M. Knauf, S. Scheglmann, and A. Scherp, "A systematic investigation of explicit and implicit schema information on the linked open data cloud," in *Proc. Extended Semantic Web Conf. (ESWC)*, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-38288-8_16

[24] A. Scherp, D. Richerby, T. Blume, M. Cochez, and J. Rau, "Structural summarization of semantic graphs using quotients," *TGDK*, 2023. [Online]. Available: https://doi.org/10.4230/TGDK.1.1.12

[25] Python, https://peps.python.org/pep-0456/\#conclusion, 2023.

[26] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input PRF," in *Intl. Conf. on Cryptology in India*. Springer, 2012.

[27] W. L. Hamilton, *Graph Representation Learning*, 2020. [Online]. Available: https://doi.org/10.2200/S01045ED1V01Y202009AIM046

[28] T. Blume, D. Richerby, and A. Scherp, "Incremental and parallel computation of structural graph summaries for evolving graphs," in *Proc. 29th ACM Intl. Conf. on Information and Knowledge Management*. ACM, 2020, pp. 75–84. [Online]. Available: https://doi.org/10.1145/3340531.3411878

[29] G. Carothers, "RDF 1.1 N-Quads," https://www.w3.org/TR/n-quads/, 2014.

[30] L. Galke and A. Scherp, "Forget me not: A gentle reminder to mind the simple multi-layer perceptron baseline for text classification," *CoRR*, 2021. [Online]. Available: https://arxiv.org/abs/2109.03777

[31] J. Frank, M. Hoffmann, N. Lell, D. Richerby, and A. Scherp, "Lifelong graph summarization with neural networks: 2012, 2022, and a time warp," 2024. [Online]. Available: https://arxiv.org/abs/2407.18042

[32] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proc AAAI Conf. on Artificial Intelligence*, 2018. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.11651

Authorized licensed use limited to: Indian Institute of Technology (Ropar). Downloaded on June 24,2025 at 05:44:16 UTC from IEEE Xplore. Restrictions apply.