A Topology-aware Graph Coarsening Framework for Continual Graph Learning

Xiaoxue Han
Stevens Institute of Technology
`xhan26@stevens.edu`

Zhuo Feng
Stevens Institute of Technology
`zfeng12@stevens.edu`

Yue Ning
Stevens Institute of Technology
`yue.ning@stevens.edu`

Abstract

Graph Neural Networks (GNNs) experience "catastrophic forgetting" in continual learning setups, where they tend to lose previously acquired knowledge and perform poorly on old tasks. Rehearsal-based methods, which consolidate old knowledge with a replay memory buffer, are a *de facto* solution due to their straightforward workflow. However, these methods often fail to adequately capture topological information, leading to incorrect input-label mappings in replay samples. To address this, we propose **TA**$\mathbb{CO}$, a topology-aware graph coarsening and continual learning framework that stores information from previous tasks as a reduced graph. Throughout each learning period, this reduced graph expands by integrating with a new graph and aligning shared nodes, followed by a "zoom-out" reduction process to maintain a stable size. We have developed a graph coarsening algorithm based on node representation proximities to efficiently reduce a graph while preserving essential topological information. We empirically demonstrate that the learning process on the reduced graph can closely approximate that on the original graph. We compare **TA**$\mathbb{CO}$ with a wide range of state-of-the-art baselines, proving its superiority and the necessity of preserving high-quality topological information for effective replaying. Our code is available at: `https://github.com/hanxiaoxue114/TACO`.

1 Introduction

Graph neural networks (GNNs) are oblivious: they fail to consider pre-existing knowledge or context outside of the information they were trained on. In offline settings, this problem can be mitigated by making multiple passes through the dataset with batch training. However, in a *continual learning* (also known as *incremental learning* or *lifelong learning*) setup, [43, 20, 40, 39], the model learns a sequence of tasks incrementally, where each *task* is defined as a learning session on a subgraph. This problem becomes more intractable as the model has no access to previous data, resulting in drastic degradation of model performance on old tasks. To tackle the issue of *"catastrophic forgetting"* in GNNs, several approaches have been proposed. Among them, rehearsal-based methods [18, 57, 48, 55] are the most common due to their straightforward workflow and efficacy in consolidating old knowledge with affordable additional memory. When performing node classification tasks, these methods utilize memory buffers to save node samples during the rehearsal process of online graph training. However, they often fail to adequately capture topological information which is important in downstream tasks. As demonstrated by Zügner *et al.* [58], even small changes in edges can alter the expected node labels, causing the model to learn incorrect *input* *(feature, structure)-label* mappings from the replay samples which creates vulnerability and security issues in critical domains [42].

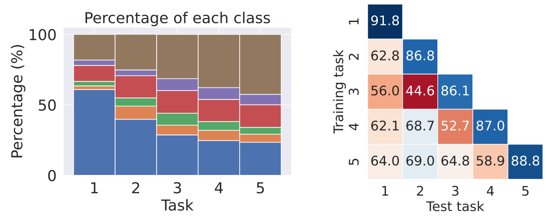
To address this, we propose a dynamic graph coarsening framework, **TA**$\mathbb{CO}$, that efficiently preserves high-quality topology information for experience replay. **TA**$\mathbb{CO}$ operates through a straightforward yet effective workflow: At the end of each training task, it reduces the current graph into a compressed form that maximally preserves its properties, allowing it to serve as a proxy for the original graph; for the next task, the reduced graph and the new graph are combined by aligning co-existing nodes, and the model is trained on this new combined graph. The former step preserves information from the *intra-task* edges (edges connecting nodes from the same task), while the latter step allows us to retrieve the *inter-task* (edges connecting nodes from different tasks) ones. Noticing that most existing graph reduction algorithms focus solely on preserving graph topology properties and are often inefficient, we propose an efficient graph reduction algorithm, **RePro**, as a component of the CGL framework. **RePro** leverages the proximities between learned node representations to effectively preserve node features and spectral properties during the reduction process. Additionally, we present a strategy, *Node Fidelity Preservation*, to ensure that certain nodes are not compressed, thereby maintaining the quality of the reduced graph. We theoretically prove that Node Fidelity Preservation can mitigate the problem of vanishing minority classes in the process of graph reduction. We claim that the simplicity of **TA**$\mathbb{CO}$ makes it highly modular and adaptable. We conduct extensive experiments and perform comprehensive ablation studies to evaluate the effectiveness of **TA**$\mathbb{CO}$ and **RePro**. We also compare our method with multiple state-of-the-art methods [20, 26, 29, 6, 57, 7, 28, 30, 55] for both CGL and graph coarsening tasks.

2 Preliminaries

2.1 Setup explanations

We notice that most existing research on CGL [26, 57, 55] focuses on either *task-incremental-learning (task-IL)* [45] or a *tranductive* [54, 3] setting, where the sequential tasks are independent graphs containing nodes from non-overlapping class sets. In this setting, the model only needs to distinguish the classes included in the current task. For instance, if there are 10 classes in total, and this experimental setting divides these classes into 5 tasks. Task 1 focuses on classifying classes 1 and 2, while task 2 classifies classes 3 and 4, and so on. Since GNNs often forget knowledge of old classes and show trivial performance, the improvement of the CGL framework can be easily highlighted with large margins. However, we argue that this setting may not accurately simulate real-life scenarios.

In this paper, we aim to tackle a more realistic *inductive* and *Generalized Class-incremental-learning (generalized class-IL)* [32] setting. In real-world graphs, nodes and edges are often associated with a time stamp indicating their appearing time, and graphs keep expanding with new nodes and edges. For instance, in a citation network, each node representing a paper cites (forms an edge with) other papers when it is published. Each year more papers are published, and the citation graph also grows rapidly. In such cases, it is necessary to train a model incrementally and dynamically because saving or retraining the model on the full graph can be prohibitively expensive in space and time. So we split a streaming graph into subgraphs based on time periods and train a model on each subgraph sequentially for the node classification task. In such cases, subgraphs are correlated with each other through the edges connecting them (e.g. a paper cites another paper from previous time stamps). The structural information represented by edges may change from previous tasks (*inductive*). Also, since the tasks are divided by time periods instead of class labels, there are overlapping classes between new tasks and old tasks, so the model needs to perform classification across all classes (*generalized class-IL*).



(a) Class distribution (b) F1 score

Figure 1: Class distributions in the Kindle dataset change over time and the model trained on new tasks tends to forget old tasks.

We further demonstrate the necessity of preserving old knowledge when learning on a streaming graph. We take the Kindle e-book co-purchasing network [14, 31] as an example. We split the graph into 5 subgraphs based on the first appearing time of each node (i.e., an e-book). We observe a gradual shift of node class distribution over time, as shown in Figure 1(a). Furthermore, even for nodes in the same class, their features and neighborhood patterns can shift [18]. Also, in real-life situations, tasks may have different class sets (e.g. new fields of study emerge and old fields of study become obsolete), which exacerbates the forgetting problem. The F1 scores of node classification tasks using a Graph Convolutional Network (GCN) [19] show that the model performs significantly worse on previous tasks when trained on new ones without strategies to alleviate forgetting, as shown in Figure 1(b). Although this paper focuses on *generalized class-IL* setting, we also conduct experiments on data splits with *traditional class-IL* setting to prove the generalizability of **TA**$\mathbb{CO}$.

2.2 Problem statement

Our main objective is to construct a continual learning framework on streaming graphs to overcome the catastrophic forgetting problem. Suppose a GNN model is trained for sequential node classification tasks with no access to previous training data, but it can utilize a memory buffer with a limited capacity to store useful information. The goal is to optimize the prediction accuracy of a model on all tasks in the end by minimizing its forgetting of previously acquired knowledge when learning new tasks. In this work, we focus on time-stamped graphs, and the tasks are defined based on the time stamps of nodes in a graph. For each task, the GNN model is trained on a subgraph where source nodes belonging to a specified time period, and all tasks are ordered in time. Node attributes and class labels are only available if the nodes belong to the current time period. The aforementioned setup closely resembles real-life scenarios. We formulate the above problems as follows.

Problem 1. Continual learning on time-stamped graphs. We are given a time-stamped expanding graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X, Y)$, where $\mathcal{V}$ denotes the node set, $\mathcal{E}$ denotes the edge set, $A \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and $X \in \mathbb{R}^{|\mathcal{V}|\times d_X}$ denote the adjacency matrix and node features, respectively; Y is a one-hot embedding matrix denoting class labels. Each node $v \in \mathcal{V}$ is assigned to a time period $\tau(v)$. We define a sequence of subgraphs, $\mathcal{G}_1, ..., \mathcal{G}_k$, such that each subgraph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, A_t, X_t)$ from $\mathcal{G}$ is formulated based on the following rules:

- For edges in $\mathcal{G}_t$: $e = (s, o) \in \mathcal{E}_t \Leftrightarrow e \in \mathcal{E}$ and $\tau(s) = t$,
- For nodes in $\mathcal{G}_t$: $v \in \mathcal{V}_t \Leftrightarrow \tau(v) = t$ or $((s, v) \in \mathcal{E}$ and $\tau(s) = t)$,

where s is a source node and o (or v) is a target node. We can assume $\tau(o) \leq \tau(s)$ for $(s, o) \in \mathcal{E}$ (e.g. in a citation network, a paper can not cite another paper published in the future). The nodes on each subgraph $\mathcal{G}_t$, are divided into three sets for training, validation, and test. We implement a GNN to perform node classification tasks and sequentially train the model on $\mathcal{G}_1, ..., \mathcal{G}_k$. When the model is trained with a new task T_t, it has no access to $\mathcal{G}_1, ..., \mathcal{G}_{t-1}$ and $\mathcal{G}_{t+1}, ..., \mathcal{G}_k$. However, a small memory buffer is allowed to preserve useful information from previous tasks. The **objective** is to optimize the overall performance of the model on test nodes from all tasks when the model is incrementally trained with new tasks.

Problem 2. Graph coarsening. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, the **goal** of graph coarsening is to reduce it to a target size n' with a specific ratio γ where $n' = \lfloor \gamma \cdot n \rfloor$, $0 < \gamma < 1$. We construct the coarsened graph $\mathcal{G}^r = (\mathcal{V}^r, \mathcal{E}^r)$ through partitioning $\mathcal{V}$ to n' disjoint clusters $(C_1, ..., C_{n'})$, so that each cluster becomes a node in $\mathcal{G}_r$. The construction of these clusters (i.e., the partition of a graph) depends on coarsening strategies. The node partitioning/clustering information can be represented by a matrix $Q \in \mathbb{B}^{n\times n'}$. If we assign every node i in cluster C_j with the same weight, then $Q_{ij} = 1$; If node i is not assigned to cluster C_j, $Q_{ij} = 0$. Let c_j be the number of node in C_j and $C = \text{diag}(c_1, ..., c_{n'})$. The normalized version of Q is $P = QC^{1/2}$. It is easy to prove P has orthogonal columns ($PP^{-1} = I$), and $P_{ij} = 1/\sqrt{c_j}$ if node i belongs to C_j; $P_{ij} = 0$ if node i does not belong to C_j. The detailed coarsening algorithm will be discussed in the next section.

3 Methodology

We propose **TA**$\mathbb{CO}$, a simple yet effective continual learning framework to consolidate graph knowledge learned from proceeding tasks by replaying previous "experiences" to the model. We observe

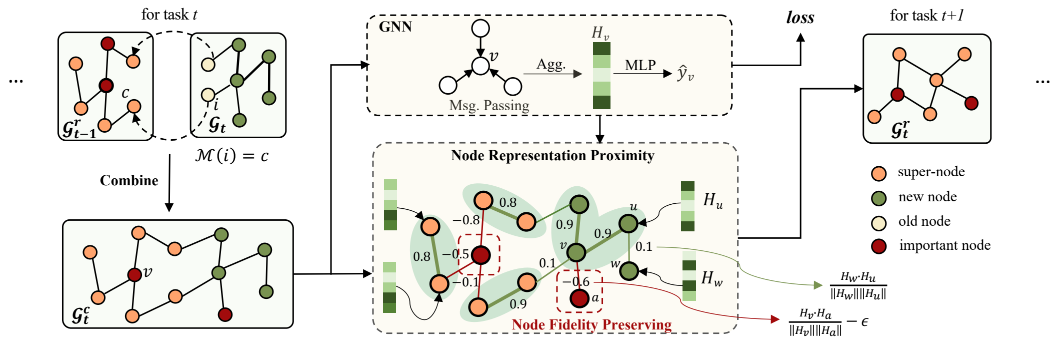


Figure 2: An overview of **TA**$\mathbb{CO}$. At t-th time period, the model takes in the coarsened graph $\mathcal{G}^r_{t-1}$ from the last time period and the original graph $\mathcal{G}_t$ from the current time period, and combine them into $\mathcal{G}^c_t$; for the same time period, the selected important node set is updated with the new nodes; the model is then trained on $\mathcal{G}^c_t$ with both the new nodes and the super-nodes from the past; finally $\mathcal{G}^c_t$ is coarsened to $\mathcal{G}^r_t$ for the next time period.

that the majority of experience replay methods, including those tailored for GNN, do not adequately maintain the intricate graph topological properties from previous tasks. Moreover, in a streaming graph setup they fail to capture the inter-dependencies among tasks that result from the presence of overlapping nodes. The inter-dependencies are essential for capturing the dynamic "receptive field" (neighborhood) of nodes and improving the performance on both new and old tasks [18]. To overcome these limitations, we design a new replay method that preserves both the node attributes and graph topology from previous tasks. Our intuition is that, if we store the original graphs from the old task, minimal old knowledge would be lost, but it is also exceedingly inefficient and goes against the initial intention of continual learning. Thus, as an alternative, we coarsen the original graphs to a much smaller size which preserves the important properties (such as node features and graph topologies) of the original graphs. We propose an efficient graph coarsening algorithm based on Node **Re**presentation **Pro**ximity as a key component of **TA**$\mathbb{CO}$. Additionally, we develop a strategy called *Node Fidelity Preservation* for selecting representative nodes to retain high-quality information. An overview of the proposed framework is provided in Figure 2. The pseudocode of **TA**$\mathbb{CO}$ is described in Algorithm 1 and **RePro** is described in Algorithm 2 in Appendix D.2.

Overall framework We summarize the procedure of our framework as three steps: *combine*, *reduce*, and *generate*. At task t, we *combine* the new graph $\mathcal{G}_t$ with the reduced graph $\mathcal{G}^r_{t-1}$ from the last task. Then we *reduce* the combined graph $\mathcal{G}^c_t$ to a set of clusters. At last, we *generate* the contributions of nodes in each cluster to form a super-node in the reduced graph $\mathcal{G}^r_t$. The last step decides the new node features and the adjacency matrix of the reduced graph. We convey the details of each step below.

3.1 Step 1: Combine

We use $\mathcal{M}$ (e.g., a hash table) to denote the mapping of each original node to its assigned cluster (super-node) in a reduced graph $\mathcal{G}^r$. In the beginning, we initialize $\mathcal{G}^r_0$ as an empty undirected graph and $\mathcal{M}_0$ as an empty hash table. At task t, the model holds copies of $\mathcal{G}^r_{t-1}$, $\mathcal{M}_{t-1}$ and an original graph $\mathcal{G}_t$ for the current task. $\mathcal{G}_t$ contains both new nodes from task t and old nodes that have appeared in previous tasks. We first "combine" $\mathcal{G}_t$ with $\mathcal{G}^r_{t-1}$ to form a new graph $\mathcal{G}^c_t$ by checking the hash table $\mathcal{M}_{t-1}$ and aligning the co-existing nodes in $\mathcal{G}_t$ and $\mathcal{G}^r_{t-1}$. By doing so, we retrieve the *inter-task* edges. We train the model f to perform node classification tasks on the combined graph $\mathcal{G}^c_t = (A^c_t, X^c_t, Y^c_t)$ with the objective $\arg\min_\theta \ell(f(A^c_t, X^c_t, \theta), Y^c_t)$, where f is a L-layer GNN model (e.g. GCN), θ is trainable parameters of the model, and ℓ is the loss function. In this work, new nodes and old nodes in $\mathcal{G}^c_t$ contribute equally to the loss during the training process. However, it remains an option to assign distinct weights to these nodes to ensure a balance between learning new information and consolidating old knowledge. We describe a more detailed process in Appendix D.1.

3.2 Step 2: Reduce

We decide how nodes are grouped into clusters and each cluster forms a new super-node in the reduced graph. We propose an efficient graph coarsening method, **RePro**, by leveraging the **Re**presentation **Pro**ximities of nodes to reduce the size of a graph through merging "similar" nodes to a super-node. Node representations are automatically learned via GNN models without extra computing processes. We think two nodes are deemed similar based on three factors: 1) *feature similarity*, which evaluates the closeness of two nodes based on their features; 2) *neighbor similarity*, which evaluates two nodes based on their neighborhood characteristics; 3) *geometry closeness*, which measures the distance between two nodes in a graph (e.g., the length of the shortest path between them). Existing graph coarsening methods concentrate on preserving spectral properties, only taking graph structures into account and disregarding node features. However, estimating spectral similarity between two nodes is typically time-consuming, even with approximation algorithms, making it less scalable for our applications where graphs are dynamically expanded and coarsened. Thus, we aim to develop a more time-efficient algorithm that considers the aforementioned similarity measures.

To get started, we have the combined graph $\mathcal{G}_t^c$ to be coarsened. We train a GNN model with L ($L = 2$ in our case) layers on $\mathcal{G}_t^c$, such that the node embedding of $\mathcal{G}_t^c$ at the first layer (before the activation function) is denoted by

$$H \in \mathbb{R}^{n_t^c \times d^h} = \text{GNN}^{(1)}(A_t^c, X_t^c, \theta), \tag{1}$$

where d^h is the size of the first hidden layer in GNN. The similarity between every connected node pair $(u, v) = e \in \mathcal{E}_t^c$ is calculated based on cosine similarity as $\beta(e) = \frac{H_u \cdot H_v}{\|H_u\| \|H_v\|}$, where $\beta(e)$ is the similarity score between the two end nodes of the edge e, H_i is the embedding for node i, and $\| \cdot \|$ is the second norm of a vector. We then sort all edges of $\mathcal{G}_t^c$ such that $\beta(e_1) \geq \beta(e_2) \geq ... \geq \beta(e_{m_t^c})$. Based on this sorted list of edges, we recursively merge two end nodes (i.e., assign the nodes to the same cluster), until the target size is reached. The class label of the merged node is determined based on the majority votes of the original nodes. The new adjacency matrix and node feature matrix are discussed in Step 3. The time complexity of our coarsening process is $\mathcal{O}(d^h \cdot m_t^c)$ where m_t^c is the number of edges in the current graph $\mathcal{G}_t^c$.

Node Fidelity Preservation After multiple rounds of coarsening, the quality of a graph deteriorates as its node features and labels are repeatedly processed. Furthermore, the use of a majority vote to determine the label of a cluster can lead to the gradual loss of minority classes and cause a "vanishing minority class" problem.

Theorem 3.1. *Consider n nodes with c classes, such that the class distribution of all nodes is represented by $\mathbf{p} = p_1, p_2, ..., p_c$, where $\sum_{i=1}^{c} p_i = 1$. If these nodes are randomly partitioned into n' clusters such that $n' = \lfloor \gamma \cdot n \rfloor$, $0 < \gamma < 1$ and the class label for each cluster is determined via majority voting. The class distribution of all the clusters is $\mathbf{p}' = p'_1, p'_2, ..., p'_c$ where $\mathbf{p}'_i$ is computed as the ratio of clusters labeled as class i and $\sum_{i=1}^{c} p'_i = 1$. Let k be one of the classes, and the rest of the class are balanced $p_1 = ... = p_{k-1} = p_{k+1} = ... = p_c$. It holds that:*

1. If $p_k = 1/c$ and all classes are balanced $p_1 = p_2 = ... = p_c$, then $\mathbb{E}[p'_k] = p_k$.

2. When $p_k < 1/c$, $\mathbb{E}[p'_k] < p_k$, and $\mathbb{E}[\frac{p'_k}{p_k}]$ decreases as n' decreases. There exists a p^{min} such that $0 < p^{min} < 1$, and when $p_k < p^{min}$, $\mathbb{E}[\frac{p'_k}{p_k}]$ decrease as p_k decreases.

The proof of Theorem 3.1 is provided in Appendix D.3. Theorem 3.1 shows that as the ratio of a class decreases, its decline becomes more severe when the graph is reduced. Eventually, the class may even disappear entirely from the resulting graph. To combat these issues, we suggest preserving representative nodes in a "replay buffer" denoted as $\mathcal{V}_t^{rb}$. We adopt three strategies from [6] to select representative nodes, namely *Reservoir Sampling*, *Ring Buffer*, and *Mean of Features*. The replay buffer has a fixed capacity and is updated as new nodes are added. During the coarsening process, we prevent the selected nodes in $\mathcal{V}_t^{rb}$ from being merged by imposing a penalty on the similarity score $\beta(e)$ such that

$$\beta(e) = \frac{H_u \cdot H_v}{\|H_u\| \|H_v\|} - \mathbb{1}(u \in \mathcal{V}_t^{rb} \text{ or } v \in \mathcal{V}_t^{rb}) \cdot \epsilon, \tag{2}$$

where ϵ is a constant and $\epsilon > 0$. It is worth noting that we do not remove the edges of these nodes from the list of candidates. Instead, we assign a high value to the penalty ϵ. This is to prevent scenarios where these nodes play a critical role in connecting other nodes. Removing these nodes entirely may lead to the graph not being reduced to the desired size due to the elimination of important paths passing through them. We make the following observation:

Observation 3.1. *Node Fidelity Preservation with buffer size b can alleviate the declination of a minority class k when p_k decreases and n' decreases, and prevent class k from vanishing when p_k is small. See Appendix D.3 for further discussions.*

Note that **RePro** does not require any additional parameters or training time. It only relies on the learned node embeddings from the graph neural network in the continual learning. We train a GNN model on a combined graph at each time step for node classification tasks. The node embeddings learned from the GNN model at different layers are representative of the nodes' neighborhoods. We use this fact and propose to measure the similarity of two nodes based on the distance between their embedding vectors. However, it takes quadratic time to calculate the pair-wise distance among nodes, thus we make a constraint that only connected nodes can be merged. Since connectivity has also been used to estimate the geometry closeness of two nodes [36], by doing so, we are able to cover the three similarity measures as well as reduce the time complexity to linear time in terms of the number of edges to calculate node similarities.

Our proposed approach based on node representations seems to be distinct from spectral-based methods, but they share a similar core in terms of the preserving of graph spectral properties. See Appendix D.4 for more details.

3.3 Step 3: Generate

From the last step, we get the membership matrix $Q_t \in \mathbb{B}^{n_t^c \times n_t^r}$ where n_t^c is the number of nodes in the combined graph, $n_t^r = \lfloor \gamma \cdot n_t^c \rfloor$ is the number of nodes in the coarsened graph and γ is the coarsening ratio. $Q_t[i, j] = 1$ denotes that node i is assigned to super-node j. Otherwise, $Q_t[i, j] = 0$.

A simple way to normalize Q_t is assuming each node contributes equally to their corresponding super-node (e.g. $Q_t[i, j] = 1/\sqrt{c_j}$ for all any node i that belongs to cluster/supernode j). However, nodes might have varying contributions to a cluster depending on their significance. Intuitively, when a node is identified as a very popular one that is directly or indirectly connected with a large number of other nodes, preserving more of its attributes can potentially mitigate the effects of the inevitable "information degrading" caused by the graph coarsening procedure. To address the above issue, we propose to use two different measures to decide a node's importance score: 1) *node degree*: the number of 1-hop neighbors of the node. 2) *neighbor degree sum*: the sum of the degrees of a node's 1-hop neighbors. In this step, we propose to normalize Q_t to P_t utilizing these node importance information. We calculate the member contribution matrix $P_t \in \mathbb{R}^{n_t^c \times n_t^r}$. Let i be a node belonging to cluster C_j at timestamp t, and $s_i > 0$ be the importance score of node i (node degree or neighbor degree sum), then $p_{t,(ij)} = \sqrt{\frac{s_i}{\sum_{v \in C_j} s_v}}$. It is straightforward to prove that $P_t^\top P_t = I$ still holds. Once we have P_t, we get the new reduced graph $\mathcal{G}_t^r = (A_t^r, X_t^r, Y_t^r)$ as:

$$A_t^r = Q_t^\top A_t^c Q_t, \quad X_t^r = P_t^\top X_t^c, \quad Y_t^r = \arg\max(P_t^\top Y_t^c), \tag{3}$$

where the label for each cluster is decided by a (weighted) majority vote. Only partial nodes are labeled, and the rows of Y_t^c for unlabelled nodes are zeros and thus do not contribute to the vote.

Through training all tasks, the number of nodes in the reduced graph $\mathcal{G}^r$ is upper-bounded by $\frac{1-\gamma}{\gamma} \cdot (n_{\text{MAX}})$, where n_{MAX} is the largest number of the new nodes for each task (See Appendix D.5 for proof); when the reduction ratio γ is 0.5, the expression above is equivalent to n_{MAX}, meaning the size of the reduced graph is roughly the same size with the original graph for each task. The number of edges m is bounded by n_{MAX}^2, but we observe generally $m \ll n_{\text{MAX}}^2$ in practice. We claim that such a memory buffer size is reasonable for storing topology information. For instance, the default buffer size of the Cora-full dataset in SSM [55] is 4,200, which is approximately seven times the average graph size.

4 Empirical evaluation

We conduct experiments on time-stamped graph datasets: Kindle [14, 31], DBLP [44] and ACM [44] to evaluate the performance of **TA**$\mathbb{CO}$. See Appendix E.1 for the details of the datasets and E.2 for hyperparameter setup.

4.1 Comparison methods

We compare the performance of **TA**$\mathbb{CO}$ with SOTA continual learning methods including EWC [20], GEM [29], TWP [26], OTG [12], ERGNN [57], SSM [55], DyGrain [18], IncreGNN [48], SSRM [41], CaT [27], and DeLoMe [33]. EWC and GEM were previously not designed for graphs, so we train a GNN on new tasks but ignore the graph structure when applying continual learning strategies. ERGNN-rs, ERGNN-rb, and ERGNN-mf are ERGNN methods with different memory buffer updating strategies: Reservoir Sampling (rs), Ring Buffer (rb), and Mean of Features (mf) [6]. SSRM is an additional regularizer to be applied on top of a CGL framework; we choose ERGNN-rs as the base CGL model. Besides, finetune provides the estimated lower bound without any strategies applied to address forgetting problems, and joint-train provides an empirical upper bound where the model has access to all previous data during the training process.

We also compare **RePro** with five representative graph coarsening SOTA methods. We replace the coarsening algorithm in **TA**$\mathbb{CO}$ with different coarsening algorithms. Alge. JC [7], Aff.GS [28], Var. edges [30], and Var. neigh [30] are graph spectral-based methods; FGC [21] considers both graph spectrals and node features. We follow a standard implementation [15] for the first four methods.

4.2 Evaluation metrics

We use *Average Performance* (AP↑) and *Average Forgetting* (AF↓) [6] to evaluate the performance on test sets. AP and AF are defined as

$$\mathrm{AP} = \frac{1}{T}\sum_{j=1}^{T} a_{T,j}, \quad \mathrm{AF} = \frac{1}{T}\sum_{j=1}^{T} \max_{l\in\{1,\ldots,T\}} a_{l,j} - a_{T,j}, \tag{4}$$

where T is the total number of tasks and $a_{i,j}$ is the prediction metric of the model on the test set of task j after it is trained on task i. The prediction performance can be measured with different metrics. In this paper, we use macro F1 and balanced accuracy score (BACC). F1-AP and F1-AF indicate the AP and the AF for macro F1 and likewise for BACC-AP and BACC-AF. We calculate the macro F1 and BACC scores for multi-class classification [13].

4.3 Main results

We evaluate the performance of **TA**$\mathbb{CO}$ and other baselines on three datasets with three backbone GNN models, including GCN [19], GAT [46], and GIN [49]. We only report the node classification performance with GCN in Table 1 due to the space limit. See Appendix F.1 for results on GAT and GIN. We report the average values and the standard deviations over 10 runs. It shows that **TA**$\mathbb{CO}$ outperforms the best SOTA CGL baseline method with high statistical significance, as evidenced by p-values below 0.05 reported from a t-test. Additionally, we note that despite being Experience Replay-based methods, ER-rs, ER-rb, and ER-mf do not perform as well as SSM and **TA**$\mathbb{CO}$, highlighting the importance of retaining graph structural information when replaying experience nodes to the model. Furthermore, we infer that **TA**$\mathbb{CO}$ outperforms SSM due to its superior ability to preserve graph topology information and capture task correlations through co-existing nodes.

4.4 Ablation studies

4.4.1 Graph coarsening methods

We evaluate the performance of **RePro** by replacing the graph coarsening module of **TA**$\mathbb{CO}$ with five widely used coarsening algorithms while keeping all other components unchanged. For each method, we report the average time in seconds consumed to coarsen the graph. We also report a trade-off value which is defined as the coarsening time (in seconds) needed to increase or decrease the AP/AF by 1% compared with fine-tuning, as shown in Table 2. For instance, the trade-off for F1-AP is defined as

Table 1: Node classification performance with GCN as the backbone on three datasets (averaged over 10 trials). Standard deviation is denoted after $\pm$.

	Kindle				DBLP				ACM			
Method	F1-AP(%) $\uparrow$		F1-AF (%) $\downarrow$		F1-AP (%) $\uparrow$		F1-AF (%) $\downarrow$		F1-AP (%) $\uparrow$		F1-AF (%) $\downarrow$	
joint train	87.21	$\pm$ 0.55	0.45	$\pm$ 0.25	86.33	$\pm$ 1.38	0.77	$\pm$ 0.13	75.35	$\pm$ 1.49	1.87	$\pm$ 0.60
finetune	69.10	$\pm$ 10.85	18.99	$\pm$ 11.19	67.85	$\pm$ 8.05	20.43	$\pm$ 7.07	60.53	$\pm$ 9.35	19.09	$\pm$ 9.23
simple-reg	68.80	$\pm$ 10.02	18.21	$\pm$ 10.49	69.70	$\pm$ 9.16	18.69	$\pm$ 8.48	61.63	$\pm$ 10.09	17.83	$\pm$ 9.99
EWC	77.08	$\pm$ 8.37	10.87	$\pm$ 8.62	79.38	$\pm$ 4.86	8.85	$\pm$ 4.11	66.48	$\pm$ 6.43	12.73	$\pm$ 6.26
TWP	78.90	$\pm$ 4.71	8.99	$\pm$ 4.93	80.05	$\pm$ 3.71	8.23	$\pm$ 3.28	65.98	$\pm$ 7.26	13.33	$\pm$ 6.94
OTG	69.01	$\pm$ 10.55	18.94	$\pm$ 10.79	68.24	$\pm$ 10.12	20.12	$\pm$ 9.34	61.45	$\pm$ 9.94	18.33	$\pm$ 9.86
GEM	76.08	$\pm$ 6.70	11.01	$\pm$ 7.27	80.04	$\pm$ 3.24	7.90	$\pm$ 2.68	67.17	$\pm$ 4.24	11.69	$\pm$ 3.94
ERGNN-rs	77.63	$\pm$ 3.61	9.64	$\pm$ 4.19	78.02	$\pm$ 5.79	10.08	$\pm$ 5.16	64.82	$\pm$ 7.89	14.43	$\pm$ 7.68
ERGNN-rb	75.87	$\pm$ 6.41	11.46	$\pm$ 6.98	75.16	$\pm$ 7.24	12.85	$\pm$ 6.54	63.58	$\pm$ 8.82	15.66	$\pm$ 8.71
ERGNN-mf	77.28	$\pm$ 5.91	10.15	$\pm$ 6.31	77.42	$\pm$ 5.25	10.64	$\pm$ 4.38	64.80	$\pm$ 8.49	14.59	$\pm$ 8.41
DyGrain	69.14	$\pm$ 10.47	18.88	$\pm$ 10.72	67.52	$\pm$ 10.88	20.83	$\pm$ 10.16	61.40	$\pm$ 9.57	18.47	$\pm$ 9.50
IncreGNN	69.45	$\pm$ 10.34	18.48	$\pm$ 10.66	69.40	$\pm$ 9.60	18.92	$\pm$ 8.75	61.32	$\pm$ 9.70	18.42	$\pm$ 9.64
SSM	78.99	$\pm$ 3.13	8.19	$\pm$ 3.63	82.71	$\pm$ 1.76	4.20	$\pm$ 1.26	68.77	$\pm$ 2.93	9.50	$\pm$ 2.47
SSRM	77.37	$\pm$ 4.06	9.99	$\pm$ 4.55	77.43	$\pm$ 5.34	10.66	$\pm$ 4.47	64.39	$\pm$ 7.43	14.72	$\pm$ 7.48
CaT	75.12	$\pm$4.01	11.83	$\pm$4.22	76.24	$\pm$3.78	9.06	$\pm$3.14	63.72	$\pm$2.21	11.86	$\pm$2.32
DeLoMe	76.93	$\pm$3.83	10.16	$\pm$4.68	77.27	$\pm$2.85	8.01	$\pm$2.16	64.54	$\pm$2.42	10.75	$\pm$2.04
TA$\mathbb{CO}$	**82.97**	**$\pm$ 2.05**	**4.91**	**$\pm$ 1.90**	**84.60**	**$\pm$ 2.01**	**2.51**	**$\pm$ 1.03**	**70.96**	**$\pm$ 2.68**	**8.02**	**$\pm$ 2.33**
p-value	<0.0001		<0.0001		0.002		<0.0001		0.005		0.02	

Table 2: Coarsen runtime (seconds) and trade-off results of different coarsening methods on three datasets with GCN (average over 10 trials). Boldface indicates the best result of each column.

	Kindle		DBLP		ACM	
Method	Time (s) $\downarrow$	$\tau_{\Delta\text{F1-AP}} \downarrow$	Time (s) $\downarrow$	$\tau_{\Delta\text{F1-AP}} \downarrow$	Time (s) $\downarrow$	$\tau_{\Delta\text{F1-AP}} \downarrow$
Alge. JC	8.9	0.74	70.8	4.08	11.8	1.19
Aff. GS	65.6	7.88	237.1	14.42	96.1	12.29
Var. neigh	6.9	0.59	7.3	0.43	10.3	1.58
Var. edges	10.1	0.77	28.0	1.63	13.8	1.24
FGC	7.7	0.63	10.8	0.64	7.0	1.05
RePro	**2.3**	**0.17**	**1.1**	**0.07**	**1.4**	**0.13**

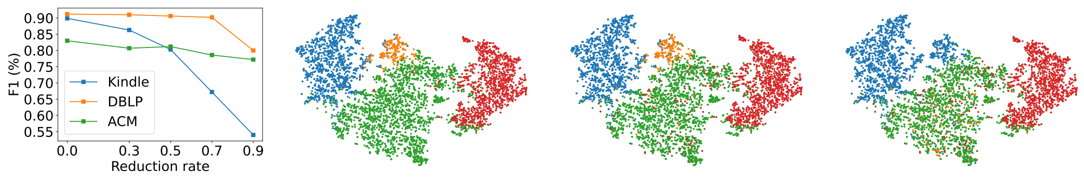
$\tau_{\Delta\text{F1-AP}} = \text{time}/(\text{F1-AP} - \text{F1-AP for fine-tuning})$. Complete results are provided in Appendix F.2. The results demonstrate that **RePro** is considerably more efficient in computing time compared to all other models and achieves the best trade-off across all metrics.

4.4.2 Graph reduction rates

We examine how **RePro** preserves original graph information. Following the setting in [15], we train GNNs from scratch on original and coarsened graphs, then compare their prediction performance on the test nodes from the original graph. Using subgraphs from the first task of three datasets as original graphs, we train GCN models on coarsened graphs with different coarsening rates $1-\gamma$. Figure 3 (a) shows that prediction performance is relatively stable as graphs are reduced for DBLP and ACM, but F1 scores are more sensitive to the reduction rate on Kindle. This may be due to overfitting on smaller datasets. Although reduced graphs may not preserve all information, they can still be used to consolidate learned knowledge and reduce forgetting in CGL paradigm. We also test if the model learns similar node embeddings on coarsened and original graphs. In Figure 3 (b)-(d), we visualize test node embeddings on the DBLP dataset for different reduction rates using t-SNE. We observe similar patterns for 0 and 0.5 reduction rates, and gradual changes as the reduction rate increases.

4.4.3 Performance after training on each task

We first investigate the performance of the model on the first task when more tasks are learned with different CGL frameworks. We also visualize the model's performance in terms of AP-F1 using the four approaches on all previous tasks after training on each task on the Kindle dataset, as shown in Figure 4. It demonstrates that the application of different CGL methods can alleviate the catastrophic



(a) GCN test F1 (b) $1 - \gamma = 0$ (c) $1 - \gamma = 0.5$ (d) $1 - \gamma = 0.9$

Figure 3: (a) The test macro-F1 scores of the GCN model trained on the coarsened graphs with different reduction rates on three datasets. (b)-(d) t-SNE visualization of node embeddings of the DBLP test graph with a reduction rate of 0, 0.5, and 0.9 on the training graph respectively.

forgetting problem on the Kindle dataset as we point out in the introduction part to varying degrees. Also, we observe that the performance of experience replay-based methods (SSM and **TA**$\mathbb{CO}$) are more stable through learning more tasks, but the regularization-based method, TWP, experiences more fluctuations. We deduce that this is caused by the fact that regularization-based methods can better prevent the model from drastically forgetting previous tasks even when the current task has more distinct distributions.

4.5 Efficiency analysis

To ensure fair comparisons, for experience replay-based methods, we adjust the memory buffer size so each model stores comparable average numbers of replay nodes. We report the average memory usage of each model on each task in Table 10 in Appendix G.3. Since TACO needs extra space to save topology information, its memory usage is slightly larger compared to methods that don't preserve graph structures. We provide memory usages of the representative methods across tasks in Table 8.

4.6 Additional studies

For those readers who are interested, we also evaluate the short-term forgetting (G.1) and the performance of the CGL frameworks when each task is assigned with shorter time intervals (G.2). We study the effectiveness of the Node Fidelity Preservation (G.4), and the effects of different important node selection strategies (G.5). We also evaluate **TA**$\mathbb{CO}$ on traditional class-incremental learning (G.6) to prove its generalizability. We present those results in Appendix G.

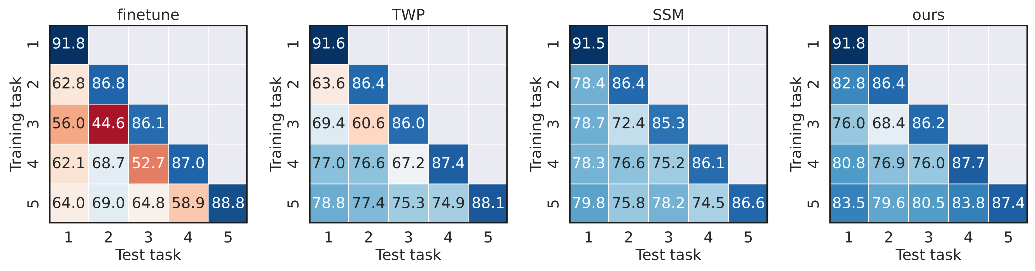


Figure 4: F1 score on the test set (x-axis) after training on each task (y-axis) on Kindle dataset.

5 Related Work

Regularization, *Expansion*, and *Rehearsal* are common approaches to overcome the catastrophic forgetting problem [43] in continual learning. *Regularization* methods [20, 4, 52, 10] penalize parameter changes that are considered important for previous tasks. Although this approach is efficient in space and computational resources, it suffers from the "brittleness" problem [43, 35] where the previously regularized parameters may become obsolete and unable to adapt to new tasks. *Expansion*-based methods [40, 51, 16, 23] assign isolated model parameters to different tasks and increase the model size when new tasks arrive. Such approaches are inherently expensive, especially when the number of tasks is large. *Rehearsal*-based methods consolidate old knowledge by replaying the model with past experiences. A common way is using a small episodic memory of previous data or generated samples from a generative model when it is trained on new tasks. While generative methods [1, 2, 8, 34] may use less space, they also struggle with catastrophic forgetting problems and over-complicated designs [38]. Experience replay-based methods [39, 5, 6, 43, 29], on the other hand, have a more concise and straightforward workflow with remarkable performance demonstrated by various implementations with a small additional working memory.

Continual graph learning Most existing CGL methods adapt *regularization*, *expansion*, or *rehearsal* methods on graphs. For instance, [26] address catastrophic forgetting by penalizing the parameters that are crucial to both task-related objectives and topology-related ones. [41] mitigate the impact of the structure shift by minimizing the input distribution shift of nodes. Rehearsal-based methods [18, 57, 48] keep a memory buffer to store old samples, which treat replayed samples as independent data points and fail to preserve their structural information. [55] preserve the topology information by storing a sparsified L-hop neighborhood of replay nodes. However, storing topology information of nodes through this method is not very efficient and the information of uncovered nodes is completely lost; also it fails to capture inter-task correlations in our setup. Besides, [12] present an approach to address both the heterophily propagation issue and forgetting problem with a triad structure replay strategy: it regularizes the distance between the nodes in selected closed triads and open triads, which is hard to be categorized into any of the two approaches. [27] uses a condensed graph as the memory buffer and employs a "Training in Memory" scheme that directly learns on the memory buffer to alleviate the data imbalance problem. However, the condensed graph contains only sampled nodes with self-loops, resulting in the loss of valuable topology information. [33] stores the learned representations of the sampled nodes as a memory store, which can preserve graph structure information. However, it still cannot handle inter-task edges, and the stored representations are inadequate for dealing with the dynamic receptive field of old nodes when new nodes form connections with them. Some CGL work [47, 50] focuses on graph-level classification where each sample (a graph) is independent of other samples, whereas our paper mainly tackles the problem of node classification where the interdependence of samples plays a pivotal role.

To better highlight the advantages of our proposed method, we provide a comparison between our method and these experience-based methods in Table 3. Our method, **TA$\mathbb{CO}$**, is the only one that can preserve graph topology, handle inter-task edges, and consider the growing receptive field of old nodes.

Table 3: Comparision between experience replay-based CGL methods.

	ERGNN [57]	DyGrain [18]	IncreGNN [48]	SSRM [41]	SSM [55]	SEM [56]	CaT [27]	DeLoMe [33]	PGDNN [53]	**TA$\mathbb{CO}$**
Replay buffer	Node feat.	Node feat.	Node feat.	Node feat.	Sparsified graph	Sparsified graph	Condensed graph	Node repre.	Decoupled node repre.	Coarsend graph
Graph topology	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓
Inter-task edges	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓
Dynamic reception field	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓

6 Conclusion

In this paper, we present a novel CGL framework, **TA$\mathbb{CO}$**, which stores useful information from previous tasks with a dynamically reduced graph to consolidate learned knowledge. Additionally, we propose an efficient embedding proximity-based graph coarsening method, **RePro**, that can preserve important graph properties. We present a Node Fidelity Preservation strategy and theoretically prove its capability in preventing the vanishing minority class problem. We demonstrate the effectiveness of **TA$\mathbb{CO}$** and **RePro** on real-world datasets with a realistic streaming graph setup.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work is supported in part by the US National Science Foundation under grants 2047843, 2437621, and 2212370.

References

[1] Alessandro Achille, Tom Eccles, Loïc Matthey, Christopher P. Burgess, Nick Watters, Alexander Lerchner, and Irina Higgins. Life-long disentangled representation learning with cross-domain latent homologies. *CoRR*, abs/1808.06508, 2018.

[2] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with stacked quantization module. *CoRR*, abs/1911.08019, 2019.

[3] Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. *CoRR*, abs/2103.11750, 2021.

[4] Arslan Chaudhry, Puneet Kumar Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *CoRR*, abs/1801.10112, 2018.

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *CoRR*, abs/1812.00420, 2018.

[6] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet Kumar Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *CoRR*, abs/1902.10486, 2019.

[7] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, 2011.

[8] Kamil Deja, Paweł Wawrzyński, Daniel Marczak, Wojciech Masarczyk, and Tomasz Trzciński. Binplay: A binary latent autoencoder for generative replay continual learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[9] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *CoRR*, abs/1910.02370, 2019.

[10] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *CoRR*, abs/1906.02425, 2019.

[11] Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. *CoRR*, abs/2007.02817, 2020.

[12] Kaituo Feng, Changsheng Li, Xiaolu Zhang, and Jun Zhou. Towards open temporal graph neural networks, 2023.

[13] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview, 2020.

[14] Ruining He and Julian J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *CoRR*, abs/1602.01585, 2016.

[15] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. *CoRR*, abs/2106.05150, 2021.

[16] Ghassen Jerfel, Erin Grant, Thomas L. Griffiths, and Katherine A. Heller. Online gradient-based mixtures for transfer modulation in meta-learning. *CoRR*, abs/1812.06080, 2018.

[17] Yu Jin, Andreas Loukas, and Joseph JaJa. Graph coarsening with preserved spectral properties. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4452–4462. PMLR, 26–28 Aug 2020.

[18] Seoyoon Kim, Seongjun Yun, and Jaewoo Kang. Dygrain: An incremental learning framework for dynamic graphs. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 3157–3163. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[20] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, mar 2017.

[21] Manoj Kumar, Anurag Sharma, and Sandeep Kumar. A unified framework for optimization-based graph coarsening, 2022.

[22] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 09–15 Jun 2019.

[23] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. *CoRR*, abs/2001.00689, 2020.

[24] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. MILE: A multi-level framework for scalable graph embedding. *CoRR*, abs/1802.09612, 2018.

[25] Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities, 2023.

[26] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. *CoRR*, abs/2012.06002, 2020.

[27] Yilun Liu, Ruihong Qiu, and Zi Huang. Cat: Balanced continual graph learning with graph condensation, 2023.

[28] Oren E. Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

[29] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continuum learning. *CoRR*, abs/1706.08840, 2017.

[30] Andreas Loukas. Graph reduction with spectral and cut guarantees. *CoRR*, abs/1808.10650, 2018.

[31] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. *CoRR*, abs/1506.04757, 2015.

[32] Fei Mi, Lingjing Kong, Tao Lin, Kaicheng Yu, and Boi Faltings. Generalized class incremental learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 970–974, 2020.

[33] Chaoxi Niu, Guansong Pang, and Ling Chen. Graph continual learning with debiased lossless memory replay, 2024.

[34] Oleksiy Ostapenko, Mihai Marian Puscas, Tassilo Klein, Patrick Jähnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. *CoRR*, abs/1904.03137, 2019.

[35] Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard Turner, and Mohammad Emtiyaz E Khan. Continual deep learning by functional regularisation of memorable past. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4453–4464. Curran Associates, Inc., 2020.

[36] Ying Pan, De-Hua Li, Jian-Guo Liu, and Jing-Zhang Liang. Detecting community structure in complex networks via node similarity. *Physica A: Statistical Mechanics and its Applications*, 389(14):2849–2857, 2010.

[37] Yunsheng Pang, Yunxiang Zhao, and Dongsheng Li. Graph pooling via coarsened graph infomax. *CoRR*, abs/2105.01275, 2021.

[38] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018.

[39] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[40] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[41] Junwei Su and Chuan Wu. Towards robust inductive graph incremental learning via experience replay, 2023.

[42] Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, Philip S. Yu, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, page 1–20, 2022.

[43] Binh Tang and David S. Matteson. Graph-based continual learning, 2020.

[44] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[45] Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019.

[46] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[47] Yuening Wang, Yingxue Zhang, and Mark Coates. Graph structure aware contrastive knowledge distillation for incremental learning in recommender systems. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, CIKM '21, page 3518–3522, New York, NY, USA, 2021. Association for Computing Machinery.

[48] Di Wei, Yu Gu, Yumeng Song, Zhen Song, Fangfang Li, and Ge Yu. Incregnn: Incremental graph neural network learning by considering node and parameter importance. In *Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part I*, page 739–746, Berlin, Heidelberg, 2022. Springer-Verlag.

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[50] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. *CoRR*, abs/2008.13517, 2020.

[51] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *CoRR*, abs/1708.01547, 2017.

[52] Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *CoRR*, abs/1703.04200, 2017.

[53] Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. Topology-aware embedding memory for continual learning on expanding networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 33 of *KDD '24*, page 4326–4337. ACM, August 2024.

[54] Xikun Zhang, Dongjin Song, and Dacheng Tao. CGLB: Benchmark tasks for continual graph learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[55] Xikun Zhang, Dongjin Song, and Dacheng Tao. Sparsified subgraph memory for continual graph representation learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1335–1340, 2022.

[56] Xikun Zhang, Dongjin Song, and Dacheng Tao. Ricci curvature-based graph sparsification for continual graph representation learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2023.

[57] Fan Zhou, Chengtai Cao, Ting Zhong, Kunpeng Zhang, Goce Trajcevski, and Ji Geng. Overcoming catastrophic forgetting in graph neural networks with experience replay. *CoRR*, abs/2003.09908, 2020.

[58] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery*, KDD '18. ACM, July 2018.

Appendix

A Limitations

This work only considers situations where nodes and edges are added to streaming graphs with a single relation type. In the future, we plan to investigate CGL methods when nodes and edges can be deleted or modified. Moreover, we will generalize our method to complex graphs such as multi-relation graphs and broader graph tasks such as link prediction for recommendation systems.

B Broader impacts

By alleviating the forgetting problem in graph continual learning, the proposed **TA**$\mathbb{CO}$ model retains old knowledge while integrating new information. This is crucial for maintaining accurate and reliable models in critical domains with constantly evolving data, such as healthcare, biomedical informatics, and knowledge graphs.

C Related work on Graph Coarsening

Scalability is a major concern in graph learning. Extensive studies aim to reduce the number of nodes in a graph, such that the coarsened graph approximates the original graph [17, 30, 7, 28]. In recent years, graph coarsening techniques are also applied for scalable graph representation learning [24, 9, 11, 15] and graph pooling [37, 25, 22]. Most graph coarsening methods [17, 30, 7, 28] aim to preserve certain spectral properties of graphs by merging nodes with high spectral similarity. However, such approaches usually result in high computational complexity especially when a graph needs to be repetitively reduced. Also, the aforementioned methods rely solely on graph structures but ignore node features. [21] propose to preserve both spectral properties and node features. However, it models the two objectives as separate optimization terms, thus the efficiency problem from the spectral-based methods remains.

D Supplemental Methodology

D.1 Overall framework

In the proposed framework, during the training process, we use a reduced graph $\mathcal{G}^r$ as an approximate representation of previous graphs, and a function $\mathcal{M}(\cdot)$ (e.g., a hash table) to map each original node to its assigned cluster (super-node) in $\mathcal{G}^r$ (e.g., if an original node i is assigned to cluster j, then $\mathcal{M}(i) = j$). Both $\mathcal{G}^r_t$ and $\mathcal{M}_t(\cdot)$ are updated after the t-th task.

To get started we initialize $\mathcal{G}^r_0$ as an empty undirected graph and $\mathcal{M}_0$ as an empty hash table. At task T_t, the model holds copies of $\mathcal{G}^r_{t-1}$, $\mathcal{M}_{t-1}$ and an original graph $\mathcal{G}_t$ for the current task.

We combine $\mathcal{G}_t$ with $\mathcal{G}^r_{t-1}$ to form a new graph $\mathcal{G}^c_t$ according to the following procedure:

1. Initialize the combined graph $\mathcal{G}^c_t = (A^c_t, X^c_t, Y^c_t)$ as $\mathcal{G}^r_{t-1}$ such that $A^c_t = A^r_{t-1}$, $X^c_t = X^r_{t-1}$, and $Y^c_t = Y^r_{t-1}$;
2. *Case 1: new source node and new target node.* For each edge $(s, o) \in \mathcal{E}_t$, if $\tau(s) = \tau(o) = t$, we add s and o to $\mathcal{G}^c_t$, and we add an undirected edge (s,o) to $\mathcal{G}^c_t$;
3. *Case 2: new source node and old target node.* If $\tau(s) = t$, but $\tau(o) < t$, and $o \in \mathcal{M}_{t-1}$, we add s to $\mathcal{G}^c_t$, and we add an undirected edge (s, $\mathcal{M}_{t-1}(o)$) to $\mathcal{G}^c_t$;
4. *Case 3: deleted target node.* If $\tau(o) < t$ and $o \notin \mathcal{M}_{t-1}$, we ignore the edge.
5. When a new node v is added to $\mathcal{G}^c_{t-1}$, it is also added to $\mathcal{M}_{t-1}$ and is assigned to a new cluster.

It is worth noting that we use directed edges to better explain the above procedure. In our implementation, the combined graph is undirected since the directness of the edges of the combined graph is not critical in learning node embeddings in a graph like a citation network.

D.2 Pseudocode

The pseudocode of the proposed **TA$\mathbb{CO}$** is described in Algorithm 1 and the graph coarsening process **RePro** is described in Algorithm 2

Algorithm 1 The Proposed Method **TA$\mathbb{CO}$**

```
1: Input: A sequence of graphs $\mathcal{G}_1, ..., \mathcal{G}_k$ ($\mathcal{G}_t$ is only accessible at task $t$)
2: Output: A trained GNN node classifier $f$ with parameter $\theta$
3: $\mathcal{G}^r_0, \mathcal{M}_0, \mathcal{V}^{rb}_0 \leftarrow \varnothing, \{\}, \{\}$
4: for task t = 1 to k do
5:     $(\mathcal{G}^c_t, \mathcal{M}_{t-1}) \leftarrow \mathsf{combine}(\mathcal{G}_t, \mathcal{G}^r_{t-1}, \mathcal{M}_{t-1})$
6:     for epoch=0 to num_epoch do
7:         Train $f$ on $\mathcal{G}^c_t$ and update $\theta$
8:     end for
9:     $\mathcal{V}^{rb}_t \leftarrow$ SamplingStrategy$(\mathcal{V}^{rb}_{t-1}, \mathcal{G}_t)$
10:    Compute node embedding $H_t$ based on Eq. 1
11:    $P_t, Q_t, \mathcal{P}(\cdot) \leftarrow$ GraphCoarseningAlgorithm$(\mathcal{G}^r_t, H_t, \mathcal{V}^{rb}_t)$
12:    Compute $\mathcal{G}^r_t$ based on Eq. 3.
13:    $\mathcal{M}_t \leftarrow \mathcal{M}_t(v) = \mathcal{P}(\mathcal{M}_{t-1}(v))$ for $v \in \mathcal{M}_{t-1}$
14: end for
15:
16: function COMBINE$(\mathcal{G}_t, \mathcal{G}^r_{t-1}, \mathcal{M}_{t-1})$
17:     $\mathcal{G}^c_t \leftarrow \mathcal{G}^r_{t-1}$
18:     for each edge $(s, o) \in \mathcal{E}_t$ do
19:         if $\mathcal{T}(s) = \mathcal{T}(o) = t$ then
20:             Add $s$ and $o$ to $\mathcal{G}^c_t$ and $\mathcal{M}_{t-1}$
21:             Add an undirected edge from $s$ to $o$ on $\mathcal{G}^c_t$
22:         else
23:             if $\mathcal{T}(s) = t$ and $\mathcal{T}(s) < t$ and $o \in \mathcal{M}_{t-1}$ then
24:             Add $s$ to $\mathcal{G}^c_t$ and $\mathcal{M}_{t-1}$
25:             Add an undirected edge from $s$ to $\mathcal{M}_{t-1}(o)$ on $\mathcal{G}^c_t$
26:         end if end if
27:     end for
28:     Return $\mathcal{G}^c_t, \mathcal{M}_{t-1}$
29: end function
```

D.3 Node Fidelity Preservation

Theorem 4.1. *Consider n nodes with c classes, such that the class distribution of all nodes is represented by $\mathbf{p} = p_1, p_2, ..., p_c$, where $\sum_{i=1}^{c} p_i = 1$. If these nodes are randomly partitioned into n' clusters such that $n' = \lfloor \gamma \cdot n \rfloor$, $0 < \gamma < 1$ and the class label for each cluster is determined via majority voting. The class distribution of all the clusters is $\mathbf{p}' = p'_1, p'_2, ..., p'_c$ where p'_i is computed as the ratio of clusters labeled as class i and $\sum_{i=1}^{c} p'_i = 1$. Let k be one of the classes, and the rest of the class are balanced $p_1 = ... = p_{k-1} = p_{k+1} = ... = p_c$. It holds that:*

1. If $p_k = 1/c$ and all classes are balanced $p_1 = p_2 = ... = p_c$, then $\mathbb{E}[p'_k] = p_k$.

2. When $p_k < 1/c$, $\mathbb{E}[p'_k] < p_k$, and $\mathbb{E}[\frac{p'_k}{p_k}]$ decreases as n' decreases. There exists a p^{min} such that $0 < p^{min} < 1$, and when $p_k < p^{min}$, $\mathbb{E}[\frac{p'_k}{p_k}]$ decrease as p_k decreases.

Proof. We prove the theorem by deriving the value of $\mathbb{E}[p'_k]$. Since $\mathbb{E}[p'_k]$ is invariant to the order of the classes, for convenience, we consider $i = 1$ without losing generability. The probability of the first class after the partitioning is:

$$\mathbb{E}[p'_1] = \frac{1}{n'} \sum_{a=1}^{n-n'} \mathbb{E}[n_a] q(a, \mathbf{p}), \tag{5}$$

Algorithm 2 The Proposed Graph Coarsening Algorithm **RePro**

```
1: Input: The original graph $\mathcal{G}$, node embedding matrix $H$, node sets $V^{rb}$, and the reduction rate $\gamma$
2: Output: partition matrix $Q$, normalized partition matrix $P$, and the mapping function $\mathcal{P}(\cdot)$
3: Initialize the mapping function $\mathcal{P}(\cdot)$ such that $\mathcal{P}(v) = v$ for $v \in \mathcal{V}$
4: $n^r \leftarrow |\mathcal{V}|$
5: $n^{\text{target}} \leftarrow \lfloor r \cdot |\mathcal{V}| \rfloor$
6: Sort all edges $e \in \mathcal{E}$ in the descending order based on their similarity scores calculated according
   to Eq. 2
7: for each edge $e = (u, v) \in \mathcal{E}$ do
8:     if $\mathcal{P}(u) \neq \mathcal{P}(v)$ then
9:         // if $u$ and $v$ are in different clusters
10:        Merge the clusters of $u$ and $v$ such that $\mathcal{P}(u) = \mathcal{P}(v)$
11:        $n^r = n^r - 1$
12:    end if
13:    if $n^r \leq n^{\text{target}}$ then
14:        Break
15:    end if
16: end for
17: Construct $Q$ with $\mathcal{P}(\cdot)$; compute $P$ based on $p_{t,(ij)} = \sqrt{\frac{s_i}{\sum_{v \in C_j} s_v}}$.
18: Return $Q$, $P$, $\mathcal{P}(\cdot)$
```

where $\mathbb{E}[n_a]$ is the expectation of the number of clusters containing a nodes, and $q(a, \mathbf{p})$ is the probability that class 1 is the majority class in a cluster with size a.

Property D.1. *The expectation of the number of clusters containing a node is*

$$\mathbb{E}[n_a] = n' \times \binom{n - n' + 1}{a - 1} \left(\frac{1}{n'}\right)^{a-1} \left(1 - \frac{1}{n'}\right)^{n-n'-(a-1)}.$$

Proof. Define I_j to be the indicator random variable for the j^{th} cluster, such that:

$$I_j = \begin{cases} 1 & \text{if the } j^{th} \text{ cluster contains exactly } a \text{ samples,} \\ 0 & \text{otherwise.} \end{cases}$$

We first compute the expected value of I_j for a single cluster. Let's calculate the probability that $a - 1$ out of the $n - n'$ samples are allocated to the j^{th} cluster:

(a) There are $\binom{n-n'}{a-1}$ ways to choose $a - 1$ samples from the $n - n'$ remaining samples.

(b) The probability that each of these $a - 1$ samples is placed in the j^{th} cluster is $\left(\frac{1}{n'}\right)^{a-1}$.

(c) The remaining $n - n' - a + 1$ samples from our original pool of $n - n'$ should not be allocated to the j^{th} cluster. The probability that all of them avoid this cluster is $\left(1 - \frac{1}{n'}\right)^{n-n'-(a-1)}$.

Thus, the probability that the j^{th} cluster contains exactly x samples is:

$$\mathbb{E}[I_j] = \binom{n - n'}{a - 1} \left(\frac{1}{n'}\right)^{a-1} \left(1 - \frac{1}{n'}\right)^{n-n'-(a-1)}. \quad (6)$$

The expected number of clusters out of all n' clusters that contain exactly a samples is:

$$\mathbb{E}[n_a] = \sum_{j=1}^{n'} \mathbb{E}[I_j]. \quad (7)$$

Given that each I_j is identically distributed, we can simplify the sum:

$$\mathbb{E}[n_a] = n' \times \mathbb{E}[I_j]. \tag{8}$$

Substituting the expression derived for $\mathbb{E}[I_j]$ from step 2, we obtain:

$$\mathbb{E}[n_a] = n' \times \binom{n-n'}{a-1} \left(\frac{1}{n'}\right)^{a-1} \left(1-\frac{1}{n'}\right)^{n-n'-(a-1)}. \tag{9}$$

$\square$

It is easy to show that when $p_1 = p_2 = ... = p_c$, it holds that $q(a, \mathbf{p}) = \frac{1}{c}$ since all classes are equivalent and have equal chances to be the majority class. Thus:

$$\begin{aligned}
\mathbb{E}[p'_1] &= \frac{1}{n'}\sum_{a=1}^{n'} \mathbb{E}[n_a] q(a,\mathbf{p}) \\
&= \frac{1}{n'}\sum_{a=1}^{n'} \mathbb{E}[n_a]\frac{1}{c} \\
&= \frac{1}{n'}\sum_{a=1}^{n'} n' \times \binom{n-n'}{a-1}\left(\frac{1}{n'}\right)^{a-1}\left(1-\frac{1}{n'}\right)^{n-n'-(a-1)}\frac{1}{c} \\
&= \frac{1}{n'}n'\frac{1}{c} \\
&= \frac{1}{c}.
\end{aligned} \tag{10}$$

To compute $q(a, \mathbf{p})$, we need to consider the situations in which class 1 is the exclusive majority class and the cases where class 1 ties with one or more other classes for having the most samples. In the second case, we roll a die to select the majority class from all the classes that have most samples.

To start, we consider the situation when class 1 has the most nodes and no other classes tie with it. We enumerate all possible combinations of class assignments and calculate the probability of each.

$$q_0(a, \mathbf{p}) = \sum_{i_1=1}^{a} \sum_{i_2=0}^{i_1-1} ... \sum_{i_c=0}^{i_1-1} \mathbb{1}\{\sum_{k=1}^{c} i_k = a\} f(\mathbf{i}, a, \mathbf{p}), \tag{11}$$

where $\mathbf{i} = i_1, i_2...i_c$, and

$$f(\mathbf{i}, a, \mathbf{p}) = \prod_{k=1}^{c} \binom{a - i_1 ... - i_k}{i_k} p_k^{i_k} (1-p_k)^{a-i_1-...-i_k}, \tag{12}$$

and

$$p_k = \begin{cases} p_1 & \text{if } k = 1 \\ \frac{1-p_1}{c-1} & \text{otherwise} \end{cases}. \tag{13}$$

We then consider the situation when class 1 ties with another class. We first select another class that ties in with class 1, and we enumerate the possibility of other classes. We multiply the whole term by $\frac{1}{2}$ as there is an equal chance to select either class as the majority class.

$$q_1(a, \mathbf{p}) = \frac{1}{2} \sum_{j_1=2}^{c} \left(\sum_{i_1=1}^{a} \sum_{i_2=0}^{i_1-1} ... \sum_{i_{j_1}=i_1}^{i_1} ... \sum_{i_c=0}^{i_1-1} \mathbb{1}\{\sum_{k=1}^{c} i_k = a\} f(\mathbf{i}, a, \mathbf{p}) \right). \tag{14}$$

We extend the above equation to a general case where class 1 ties with k classes ($1 \leq k \leq c-1$). Here we need to select k classes that tie with class 1. Class 1 now shares a $\frac{1}{k+1}$ chance to be selected as the majority class with the selected k classes.

$$\begin{aligned} q_k(a, \mathbf{p}) =& \frac{1}{k} \sum_{j_1=2}^{c-k+1} \sum_{j_2=j_1}^{c-k+2} \cdots \sum_{j_k=j_{k-1}}^{c} \\ & \cdot \left(\sum_{i_1=1}^{a} \sum_{i_2=0}^{i_1-1} \cdots \sum_{i_{j_1}=i_1}^{i_1} \cdots \sum_{i_{j_k}=i_1}^{i_1} \cdots \sum_{i_c=0}^{i_1-1} \mathbb{1}\{\sum_{k=1}^{c} i_k = a\} f(\mathbf{i}, a, \mathbf{p}) \right). \end{aligned} \tag{15}$$

Finally, we combine all the cases, and the probability that class 1 is the majority class is:

$$q(a, \mathbf{p}) = \sum_{k=0}^{c-1} q_k(a, \mathbf{p}). \tag{16}$$

The expectation of $\frac{p_1'}{p_1}$ is thus:

$$\mathbb{E}[\frac{p_1'}{p_1}] = \frac{1}{p_1} \frac{1}{n'} \sum_{a=1}^{n'} \mathbb{E}[n_a] q(a, \mathbf{p}). \tag{17}$$

To study the behavior of $\mathbb{E}[\frac{p_1'}{p_1}]$ when p_1 changes, we derive the following derivative:

$$\begin{aligned} \frac{d\mathbb{E}[\frac{p_1'}{p_1}]}{dp_1} &= \frac{d \frac{1}{p_1} \frac{1}{n'} \sum_{a=1}^{n'} \mathbb{E}[n_a] q(a, \mathbf{p})}{dp_1} \\ &= \frac{1}{n'} \sum_{a=1}^{n'} \mathbb{E}[n_a] \frac{d \frac{q(a,\mathbf{p})}{p_1}}{dp_1}, \end{aligned} \tag{18}$$

where

$$\begin{aligned} \frac{d \frac{q(a,\mathbf{p})}{p_1}}{dp_1} &= \sum_{k=0}^{c-1} \frac{d \frac{q_k(a,p)}{p_1}}{dp_1} \\ &= \frac{1}{2} \sum_{j_1=2}^{c-k+1} \sum_{j_2=j_1}^{c-k+2} \cdots \sum_{j_k=j_{k-1}}^{c} \left(\sum_{i_1=1}^{a} \sum_{i_2=0}^{i_1-1} \cdots \sum_{i_{j_1}=i_1}^{i_1} \cdots \sum_{i_{j_k}=i_1}^{i_1} \cdots \sum_{i_c=0}^{i_1-1} \mathbb{1}\{\sum_{k=1}^{c} i_k = a\} \frac{d \frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1} \right). \end{aligned} \tag{19}$$

To find $\frac{d\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1}$, we first separate the terms that are independent of p_1:

$$\begin{aligned}
\frac{f(\mathbf{i},a,\mathbf{p})}{p_1} &= \frac{1}{p_1}\prod_{k=1}^{c}\binom{a-i_1...-i_k}{i_k}p_k^{i_k}(1-p_k)^{a-i_1-...-i_k} \\
&= \frac{1}{p_1}\left(\prod_{k=1}^{c}\binom{a-i_1...-i_k}{i_k}\right)\prod_{k=1}^{c}p_k^{i_k}(1-p_k)^{a-i_1-...-i_k} \\
&= \left(\prod_{k=1}^{c}\binom{a-i_1...-i_k}{i_k}\right)p_1^{i_1-1}(1-p_1)^{a-i_1} \\
&\times (\frac{1-p_1}{c-1})^{\sum_{k=2}^{c}i_k}(1-\frac{1-p_1}{c-1})^{\sum_{k=2}^{c}a-i_2...-i_k} \\
&= u\times p_1^{i_1-1}(1-p_1)^{a-i_1}(1-p_1)^{\sum_{k=2}^{c}i_k}(p_1+c-2)^{\sum_{k=2}^{c}a-i_2...-i_k} \\
&= u\times p_1^{\overbrace{i_1-1}^{\theta}}\times(1-p_1)^{\overbrace{a-i_1+\sum_{k=2}^{c}i_k}^{\phi}} \\
&\times (p_1+c-2)^{\overbrace{\sum_{k=2}^{c}a-i_2...-i_k}^{\psi}},
\end{aligned} \tag{20}$$

where u is independent of p_1

$$u=\left(\prod_{k=1}^{c}\binom{a-i_1...-i_k}{i_k}\right)(\frac{1}{c-1})^{\sum_{k=2}^{c}i_k+\sum_{k=2}^{c}a-i_2...-i_k} \tag{21}$$

We observe that $\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}$ demonstrates different behaviors for $a=1$ and $a>1$ and discuss the two cases separately.

(1) When $a=1$, it holds that $i_1=1$, $\theta=0$, $\phi=0$, and $\psi=0$:

$$\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}=u\times p_1^0(1-p_1)^0(p_1+c-2)^0=u. \tag{22}$$

In such case, $\frac{d\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1}$ is independent with p_1 and remain constant when p_1 changes.

(2) When $a>1$, it holds that $i_1\le 1$, $\theta\ge 0$, $\phi\ge 0$, $\psi\ge 0$, and $\theta+\phi>0$:

$$\frac{d\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1}=u\cdot p^{\theta-1}(1-p)^{\phi-1}(p+c-2)^{\psi-1}\cdot v, \tag{23}$$

where

$$\begin{aligned}
v &= \theta(1-p)(p+c-2)+\psi p(1-p)-\phi p(p+c-2) \\
&= (-\theta-\phi-\psi)p^2+(\theta+\psi+(\phi-\theta)(c-2))p+\theta(c-2).
\end{aligned} \tag{24}$$

When $0<p_1<1$ and $u>0$, $\frac{d\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1}=0$ if and only if $v=0$, and the corresponding value of p_1 is:

$$p_1^0=\frac{-(\theta-\theta\cdot(c-2)+\phi-\psi\cdot n)-\sqrt{\Delta}}{2(-\theta-\phi-\psi)}, \tag{25}$$

$$p_1^1=\frac{-(\theta-\theta\cdot(c-2)+\phi-\psi\cdot n)+\sqrt{\Delta}}{2(-\theta-\phi-\psi)}, \tag{26}$$

where

$$\Delta = (\theta - \theta \cdot (c-2) + \phi - \psi \cdot (c-2))^2 - 4\left(-\theta - \phi - \psi\right)\left(\theta \cdot (c-2)\right). \tag{27}$$

It is easy to show that $\Delta > 0$, and since $(-\theta - \phi - \psi) < 0$, v is concave, $v > 0$ when $p_1^1 < p < p_1^0$. Also, it is observed that when $p_1 = 0$,

$$v = \theta(c-2) \geq 0; \tag{28}$$

when $p_1 = 1$,

$$v = -\phi(c-1) < 0. \tag{29}$$

Thus it must be held that $0 < p_1^0 < 1$ and $p_1^1 \leq 0$, and for any $(\mathbf{i}, a > 1)$, there exist a $0 < p_1(\mathbf{i}, a > 1)^0 < 1$ such that when $p_1 < p_1^0(\mathbf{i}, a > 1)$, $\frac{d\frac{f(\mathbf{i},a,\mathbf{p})}{p_1}}{dp_1} > 0$. Let

$$p_1^{\text{min}} = \min_{\forall a \in \{2,\ldots,n'\}, \mathbf{i} \in \mathbf{I}} p_1^0(\mathbf{i}, a), \tag{30}$$

where $\mathbf{I}$ is all possible $\mathbf{i}$, then it holds that $0 < p_1^{\text{min}} < 1$, and when $p_1 < p_1^{\text{min}}$, $\frac{d\frac{q(a,\mathbf{p})}{p_1}}{dp_1} > 0$.

Next, we show that $\mathbb{E}[\frac{p_1'}{p_1}]$ decreases as n' decreases when $p_1 < 1/c$. We first rewrite $\mathbb{E}[\frac{p_1'}{p_1}]$ as

$$\begin{aligned}\mathbb{E}[\frac{p_1'}{p_1}] =& \frac{1}{p_1}\frac{1}{n'}\sum_{a=1}^{n'} \mathbb{E}[n_a] q(a, \mathbf{p}) \\ =& \frac{1}{p_1}\sum_{a=1}^{n'} \binom{n-n'}{a-1} \left(\frac{1}{n'}\right)^{a-1} \left(1 - \frac{1}{n'}\right)^{n-n'-(a-1)} q(a, \mathbf{p})\end{aligned} \tag{31}$$

First, we show that $q(a, \mathbf{p})$ is smaller for larger a when $p_1 < 1/c$. The intuition is that when a new node is added to a cluster originally with $a-1$ nodes, the new node has a higher probability of being another class, and the likelihood of class 1 becoming the majority class decreases.

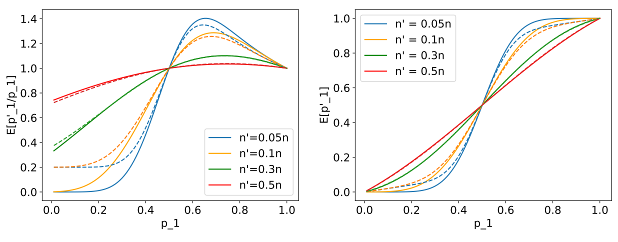
Next, we show that when n' increases, $\binom{n-n'}{a-1}\left(\frac{1}{n'}\right)^{a-1}\left(1 - \frac{1}{n'}\right)^{n-n'-(a-1)}$ becomes more left-skewed, that it gives a smaller value for large a. The intuition is that, as the value of n' increases, the average cluster size is expected to decrease. As a result, a large a becomes farther from the average cluster size, and the probability of a cluster having exactly a nodes decreases, leading to a decrease in the ratio of clusters with size a.

With the above observations, it can be concluded that when $p_1 < 1/c$, the value of $\mathbb{E}[\frac{p_1'}{p_1}]$ decreases as n' decreases. ☐

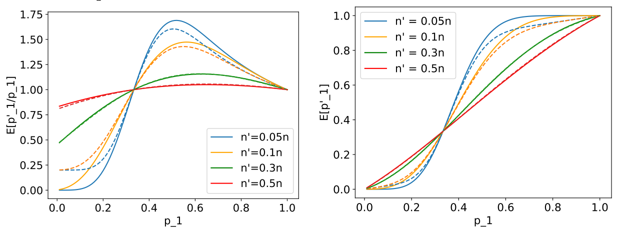
Observation 4.1. *Node Fidelity Preservation with buffer size b can alleviate the declination of a minority class k when p_k decreases and n' decreases, and prevent class k from vanishing at small when p_k is small.*

The mechanism of Node Fidelity Preservation is to "hold" b clusters such that each cluster only has 1 node. We already show that when $a = 1$, $\frac{q(a,\mathbf{p})}{q_1}$ is independent of q_1 and so $\mathbb{E}[p_1'] = p_1$. By doing so, we make sure that among the b nodes, class 1 does not decline as p_1 or n' decline.

We demonstrate the effect of Node Fidelity Preservation with examples. We assign $n = 1000$, $c = 2, 3$, and $b = \lfloor n'/5 \rfloor$. we plot change of $\mathbb{E}[\frac{p_1'}{p_1}]$ and $\mathbb{E}[p_1']$ at different n' and p_1 separately in Figure 5. We draw the trend with Node Fidelity Preservation (NFP) using dash lines and without NFP using solid lines. From the figure, we observe that without Node Fidelity Preservation being applied, the ratio $\mathbb{E}[\frac{p_1'}{p_1}]$ approaches zero when n' is small, resulting in a vanishing minority class. The application of Node Fidelity Preservation prevents the ratio from approaching zero and makes sure class 1 won't disappear when p_1 is small.



(a) $\mathbb{E}[\frac{p'_1}{p_1}]$ vs. p_1, $c = 2$ (b) $\mathbb{E}[p'_1]$ vs. p_1, $c = 2$



(c) $\mathbb{E}[\frac{p'_1}{p_1}]$ vs. p_1, $c = 3$ (d) $\mathbb{E}[p'_1]$ vs. p_1, $c = 3$

Figure 5: $\mathbb{E}[p'_1]$ and $\mathbb{E}[\frac{p'_1}{p_1}]$ against p_1 at different reduction rate γ for $c = 2$ and $c = 3$. The dashed lines represent trends with Node Fidelity Preservation (NFP), and the solid lines represent trends without NFP.

D.4 Node Representation Proximity

Spectral-based methods aim to preserve the spectral properties of a graph. Specifically, the Laplacian matrices of the original graph and the reduced graph are compared with each other [30]. The combinatorial Laplacian of a graph $\mathcal{G}$, $L \in \mathbb{R}^{n\times n}$ is defined as $L = D - A$, where $A \in \mathbb{R}^{n\times n}$ is the adjacency matrix of $\mathcal{G}$, and $D \in \mathbb{R}^{n\times n}$ is its diagonal degree matrix. The combinatorial Laplacian of the reduced graph, $L' \in \mathbb{R}^{n'\times n'}$, is calculated as $L' = P^{\mp} L P^{+}$, where P^{+} is the pseudo inverse of the normalized coarsening matrix $P \in \mathbb{R}^{n\times n'}$, and $P^{\mp}$ is the transposed pseudo inverse of P. Since L and L' have different sizes, they can not be directly compared. Instead, the following induced semi-norms are defined:

$$\|x\|_L = \sqrt{x^\top L x}, \ \|x'\|_{L'} = \sqrt{x'^\top L' x'}, \tag{32}$$

where $x \in \mathbb{R}^n$, and x' is the projection of x on n'-space such that $x' = Px$. The closeness between L to L' can be defined as how close $\|x\|_L$ is to $\|x'\|_{L'}$. L and L' are considered equivalent if it holds $\|x'\|_{L'} = \|x\|_L$ for any $x \in \mathbb{R}^n$. We next show that a partitioning made by merging nodes sharing the same neighborhood structure results in an equivalent Laplacian of the reduced graph as the original graph.

Theorem D.1. Let i and j be two nodes of $\mathcal{G}$, $A \in \mathbb{R}^{n\times n}$ be the adjacency matrix of $\mathcal{G}$ and $\tilde{A} = A + I$, D be the diagonal matrix of A and $\tilde{D} = D + I$, $P \in \mathbb{R}^{n\times n-1}$ be the normalized coarsening matrix by merging i and j to one node, L and L' be the combinatorial Laplacian of G and the reduced graph. It holds that $\tilde{A}_i = \tilde{A}_j, \tilde{D}_i = \tilde{D}_j \Rightarrow \|x'\|_{L'} = \|x\|_L$ for any $x \in \mathbb{R}^n$ and $x' = Px$, where A_i is the i-th row for a matrix A.

Proof. Given that $\tilde{A}_i = \tilde{A}_j$ and $\tilde{D}_i = \tilde{D}_j$, denote these common rows as $\tilde{A}_{ij}$ and $\tilde{D}_{ij}$, respectively. The norm $\|x\|_L$ is defined as:

$$\|x\|_L = x^T L x,$$

where $L = D - A$ is the combinatorial Laplacian of $\mathcal{G}$.

Similarly, for the coarsened graph, the norm $\|x'\|_{L'}$ is defined as:

$$\|x'\|_{L'} = (Px)^T L'(Px),$$

where L' is the combinatorial Laplacian of the coarsened graph.

We need to show that $\|x\|_L = \|x'\|_{L'}$.

First, consider the left-hand side norm:

$$\|x\|_L = x^T L x = x^T (D - A)x.$$

Given the structure of $\tilde{A}$ and $\tilde{D}$, we have:

$$\|x\|_L = \sum_k \tilde{D}_{kk} x_k^2 - \sum_{k,l} \tilde{A}_{kl} x_k x_l.$$

When merging nodes i and j, the coarsening matrix P combines these two nodes into one. Thus, $x' = Px$.

The matrix P is such that it preserves the sum of entries corresponding to the merged nodes, maintaining the structure of the Laplacian. Therefore, we have:

$$P^T P = I \quad \text{(except for the merged nodes)}.$$

The reduced Laplacian L' after merging i and j incorporates the sum of the rows and columns corresponding to i and j in $\tilde{A}$ and $\tilde{D}$. Since $\tilde{A}_i = \tilde{A}_j$ and $\tilde{D}_i = \tilde{D}_j$, the rows and columns corresponding to i and j are identical, thus preserving the overall structure and norm.

Therefore, we have:

$$\|x'\|_{L'} = (Px)^T L'(Px) = x^T L x = \|x\|_L.$$

□

D.5 Proof of the size of the reduced graph

Proof. The number of nodes of the reduced graph at task t is:

$$\begin{aligned} n &= (1-\gamma)(...((1-\gamma)n_1 + n_2)... + n_t) \\ &\leq (1-\gamma)(...((1-\gamma)n_{\text{MAX}} + n_{\text{MAX}})... + n_{\text{MAX}}) \\ &= ((1-\gamma) + (1-\gamma)^2 + ... + (1-\gamma)^t)n_{\text{MAX}} \\ &\leq \frac{1-\gamma}{\gamma} n_{\text{MAX}} \end{aligned}$$

□

D.6 Discussion on utilizing soft labels as an alternative to majority voting

Another potential solution to address the "vanishing class" problem caused by the majority voting is to use soft labels to represent the cluster label instead. However, such an approach may come with several drawbacks. First, using hard labels is memory-efficient, and requires only a single digit to represent a sample's label. In contrast, soft labels use a vector for storage, with a size equivalent to the model's output dimension. This distinction in memory usage is negligible for models with few output classes. However, in scenarios where the model predicts among a large number of classes, the increased memory demand of soft labels becomes significant and cannot be overlooked. Second, although a model can learn to predict a soft label during the training phase, most applications necessitate deterministic predictions in the testing or inference phase. We are concerned that training with soft labels might lead the model towards indeterministic and ambiguous predictions, potentially undermining its practical applicability. The last concern is that when using soft labels, instead of a classification task, the model is performing a regression task. To predict single nodes with deterministic labels, it is considered a suboptimal approach to model it as a regression task due to the unaligned optimization objectives and loss functions. Also, regression is believed to be a less difficult task to learn compared to classification task as discrete optimization is generally harder than continuous optimization. Training the model with an easier task may restrict its performance during the test phase.

Table 4: Statistics of datasets. "Interval" indicates the length of the time interval for each task. "# Item" indicates the total number of items/papers in each dataset.

Dataset	Time	Interval	#Task	#Class	#Items
Kindle	2012-2016	1 year	5	6	38,450
DBLP	1995-2014	2 years	10	4	54,265
ACM	1995-2014	2 years	10	4	77,130

E Supplemental experiment setups

E.1 Details of the datasets

The Kindle dataset contains items from the Amazon Kindle store; each node representing an item is associated with a timestamp indicating its release date, and each edge indicates a "frequently co-purchased" relation between items. The DBLP and ACM datasets are citation datasets where each node represents a paper associated with its publishing date, and a node's connection to other nodes indicates a citation relation. For the Kindle dataset, we select items from six categories: *Religion & Spirituality*, *Children's eBooks*, *Health, Fitness & Dieting*, *SciFi & Fantasy*, *Business & Money*, and *Romance*. For the DBLP dataset, we select papers published in 34 venues and divide them into four classes: *Database*, *Data Mining*, *AI*, and *Computer Vision*. For the ACM dataset, we select papers published in 66 venues and divide them into four classes: *Information Systems*, *Signal Processing*, *Applied Mathematics*, and *AI*. For each of the datasets, we select nodes from a specified time period. We build a graph and make a constraint that the timestamp of the target node is not allowed to be larger than the source node, which should naturally hold for citation datasets as one can not cite a paper published in the future. Then we split each graph into subgraphs by edges based on the timestamps of the source nodes. To simulate a real-life scenario that different tasks may have different sets of class labels, at each task for the Kindle dataset, we randomly select one or two classes and mask the labels of the nodes from selected class(s) during the training and the test phase; for the DBLP and ACM datasets, we randomly select one class and mask the labels of the nodes from the selected class during the training and the test phase. The summary of each dataset is provided in Table 4.

E.2 Hyper-parameter setting

For each task, we randomly split all nodes into training, validation, and test sets with the ratio of $30/20/50$. For the baseline CGL models the memory strengths are searched from $\{10^i|i \in [-5...5]\}$ or $\{0.1, 0.2...0.9\}$. For baselines that utilize a memory buffer, we calibrate their memory buffer sizes to ensure that their memory usage is on a similar scale to that of **TA**$\mathbb{CO}$. For **TA**$\mathbb{CO}$, by default the reduction ratio is 0.5; memory buffer size for Node Fidelity Preservation is 200; node degree is used to determine a node's importance score, and Reservoir Sampling is chosen as the node sampling strategy. We chose GCN and GAT as the GNN backbones. For both of them, we set the number of layers as 2 and the hidden layer size as 48. For GAT, we set the number of heads as 8. For each dataset, we generate 10 random seeds that split the nodes into training, validation, and test sets and select the masked class. We run all models on the same random seeds and the results are averaged over 10 runs. All experiments were conducted on a NVIDIA GeForce RTX 3090 GPU.

F Additional results

F.1 Main results

We present more results of the performance of **TA**$\mathbb{CO}$ and other baselines on three datasets and two additional backbone GNN models, GAT (Table 5) and GIN (Table 6). The results show that our proposed method outperforms other continual learning approaches consistently with different GNN backbone models.

Table 5: Performance comparison of node classification in terms of F1 and BACC with GAT on three datasets (average over 10 trials). Standard deviation is denoted after ±.

Dataset	Method	**F1-AP** (%) ↑	**F1-AF** (%) ↓	**BACC-AP** (%) ↑	**BACC-AF** (%) ↓
Kindle	joint train	88.54±0.70	0.35±0.27	82.71±1.02	0.62±0.46
	finetune	68.68±11.55	20.05±11.59	66.89±4.90	16.91±5.32
	simple-reg	66.20±10.89	19.26±11.18	64.61±3.98	15.24±4.48
	EWC	78.95±5.62	9.92±5.79	73.49±2.76	10.30±3.33
	TWP	78.17±6.67	10.85±6.71	73.23±3.06	10.78±3.58
	OTG	70.09±9.66	18.78±10.02	67.50±4.24	16.53±4.65
	GEM	76.46±7.14	11.86±7.61	72.52±2.56	10.97±3.49
	ERGNN-rs	78.64±4.36	9.56±4.57	72.19±2.95	10.70±3.20
	ERGNN-rb	75.60±7.14	12.66±7.34	71.86±3.16	11.55±3.32
	ERGNN-mf	78.14±5.22	10.35±5.59	72.94±3.05	10.90±3.89
	DyGrain	70.65±10.05	18.28±10.44	68.16±4.05	15.77±4.89
	IncreGNN	70.66±10.57	18.18±10.76	68.06±4.50	15.92±5.06
	SSM	81.84±2.10	6.58±2.59	74.64±3.10	8.81±2.62
	SSRM	78.09±4.54	10.20±5.15	71.99±2.46	11.17±3.39
	TA$\mathbb{CO}$	**83.66±1.93**	**4.69±1.82**	**76.58±3.07**	**6.34±2.13**
DBLP	joint train	83.43±1.81	1.08±0.31	76.97±1.94	1.79±0.64
	finetune	65.75±10.67	21.68±9.76	64.21±4.21	18.76±3.28
	simple-reg	68.85±9.68	18.49±8.58	66.21±4.01	16.81±2.75
	EWC	76.33±5.71	11.12±5.02	71.05±3.83	12.16±3.41
	TWP	76.64±4.47	10.61±3.77	70.95±3.22	12.03±2.98
	OTG	67.50±10.70	20.06±9.90	65.65±4.40	17.57±3.63
	GEM	73.64±6.07	12.76±4.77	67.53±4.47	14.42±3.24
	ERGNN-rs	75.36±5.62	11.87±4.62	70.07±3.88	12.95±3.44
	ERGNN-rb	71.65±7.32	15.20±6.34	67.16±3.86	15.37±3.07
	ERGNN-mf	74.62±6.16	12.55±5.29	68.97±4.20	13.90±3.61
	DyGrain	65.83±10.05	21.62±8.96	63.80±4.18	19.10±2.71
	IncreGNN	66.23±11.16	21.23±10.52	64.62±4.10	18.55±3.41
	SSM	81.47±2.48	4.46±1.59	74.62±2.41	6.48±1.69
	SSRM	74.31±6.00	12.63±4.68	69.15±4.18	13.57±3.41
	TA$\mathbb{CO}$	**81.63±1.06**	**2.29±0.60**	**76.08±1.67**	**2.53±1.09**
ACM	joint train	74.89±1.53	1.91±0.85	65.81±1.40	2.32±1.33
	finetune	61.59±10.85	17.79±10.44	58.05±2.18	14.24±1.91
	simple-reg	59.22±9.95	17.79±9.81	55.88±2.90	14.23±1.57
	EWC	66.75±7.00	12.20±6.93	61.49±1.84	10.44±1.75
	TWP	67.42±7.54	11.66±7.35	61.57±1.72	10.29±1.86
	OTG	62.24±10.88	16.99±11.02	58.55±2.37	13.70±1.41
	GEM	67.01±4.61	11.16±4.47	59.53±2.44	10.68±2.06
	ERGNN-rs	64.89±7.93	13.90±7.88	59.22±2.34	12.29±1.73
	ERGNN-rb	64.04±8.59	14.74±8.64	58.70±2.48	13.10±1.84
	ERGNN-mf	64.56±9.26	14.30±9.24	59.94±1.78	11.91±1.46
	DyGrain	61.66±10.58	17.72±10.42	58.15±2.49	14.07±1.80
	IncreGNN	62.25±10.70	17.22±10.60	58.37±2.41	14.01±1.55
	SSM	69.83±3.16	8.10±2.81	61.15±2.65	8.93±2.77
	SSRM	64.77±8.34	14.21±8.93	59.47±1.97	12.23±1.72
	TA$\mathbb{CO}$	**70.37±2.70**	**7.64±2.43**	**62.59±1.51**	**7.52±1.94**

F.2 Graph coarsening methods

We present the trade-offs in terms of all matrices of **TA**$\mathbb{CO}$ with its graph coarsening module **RePro** replaced by five other widely used graph coarsening algorithms with GCN as the backbone GNN model in Table 7.

Table 6: Performance comparison of node classification in terms of F1 and BACC with GIN on three datasets (average over 10 trials). Standard deviation is denoted after ±.

Dataset	Method	F1-AP(%) ↑	F1-AF (%) ↓	BACC-AP (%) ↑	BACC-AF (%) ↓
Kindle	joint train	84.39±0.84	0.55±0.34	77.17±0.83	1.45±1.07
	finetune	64.98±10.26	20.24±10.77	61.81±4.53	17.91±5.24
	simple-reg	65.04±10.74	20.04±11.13	62.95±4.16	16.62±5.14
	EWC	75.73±4.74	9.36±5.01	69.34±2.84	10.07±3.47
	TWP	76.14±4.35	9.14±4.72	69.12±3.06	10.49±3.51
	OTG	65.18±9.97	20.08±10.29	61.98±4.79	17.70±5.35
	GEM	72.10±6.29	12.40±6.80	66.76±2.92	12.05±3.67
	ERGNN-rs	73.67±3.22	10.83±3.84	66.72±3.00	11.84±3.40
	ERGNN-rb	70.25±7.35	14.39±8.01	65.53±3.14	13.70±4.38
	ERGNN-mf	72.33±6.10	12.47±6.64	67.12±3.17	12.28±4.27
	DyGrain	64.50±10.18	20.78±10.74	61.55±4.45	18.13±5.36
	IncreGNN	65.38±9.56	19.69±9.99	61.71±5.02	17.85±5.66
	SSM	76.47±3.37	8.01±3.38	67.78±2.84	10.73±3.03
	SSRM	73.75±3.25	10.79±3.56	66.55±3.25	12.07±3.63
	TA$\mathbb{CO}$	**78.71±1.76**	**6.44±1.80**	**70.76±1.86**	**8.28±2.13**
DBLP	joint train	84.42±1.47	1.63±0.28	77.69±1.07	2.65±0.68
	finetune	65.48±11.96	22.85±11.28	64.59±4.86	19.65±3.87
	simple-reg	66.84±9.64	21.85±8.97	65.04±3.77	19.76±2.80
	EWC	77.45±7.06	10.79±6.54	71.57±5.18	12.59±4.68
	TWP	77.59±4.91	10.64±4.44	71.45±4.11	12.75±3.81
	OTG	66.37±10.66	22.11±9.93	64.62±4.19	19.77±2.96
	GEM	78.71±4.45	9.10±3.47	72.24±3.86	11.39±3.27
	ERGNN-rs	76.63±3.94	11.47±3.29	70.45±3.51	13.67±3.18
	ERGNN-rb	73.23±7.29	14.77±6.50	68.79±4.52	15.22±3.91
	ERGNN-mf	75.96±5.74	12.14±4.99	70.74±3.55	13.39±3.10
	DyGrain	66.89±10.10	21.35±9.36	65.06±3.77	19.10±2.51
	IncreGNN	67.81±9.09	20.56±8.34	65.58±3.66	18.68±2.79
	SSM	80.21±7.85	6.74±7.29	73.96±5.16	8.24±4.99
	SSRM	76.60±4.89	11.55±4.30	70.89±4.29	13.30±3.90
	TA$\mathbb{CO}$	**84.03±2.08**	**3.12±0.89**	**78.09±2.19**	**3.48±1.34**
ACM	joint train	71.86±1.54	2.99±0.74	62.65±1.17	2.98±1.60
	finetune	57.20±9.03	20.31±8.97	53.50±3.43	16.14±2.57
	simple-reg	57.86±9.27	19.52±9.02	53.99±3.41	15.95±2.21
	EWC	65.18±5.86	12.06±5.68	58.64±1.78	10.90±1.84
	TWP	65.45±5.56	11.72±5.35	58.76±1.78	10.63±1.60
	OTG	58.24±9.38	19.37±9.24	54.23±3.35	15.46±2.31
	GEM	65.03±3.71	11.69±3.15	56.94±3.43	11.65±2.29
	ERGNN-rs	61.30±7.75	15.77±7.57	55.87±2.68	13.30±2.11
	ERGNN-rb	61.12±8.25	16.10±8.07	55.82±2.60	13.81±1.74
	ERGNN-mf	61.86±7.85	15.49±7.73	56.76±2.84	13.10±1.56
	DyGrain	58.09±9.46	19.43±9.27	54.26±3.06	15.36±2.45
	IncreGNN	58.21±9.17	19.43±9.03	54.10±3.15	15.78±2.03
	SSM	65.73±3.15	10.64±2.79	56.81±2.58	11.19±3.01
	SSRM	61.47±7.38	15.68±7.14	56.09±2.64	13.02±1.55
	TA$\mathbb{CO}$	**67.19±3.12**	**9.73±2.80**	**59.06±2.40**	**9.13±2.92**

G Additional ablation studies and analysis

G.1 Short-term forgetting

The average forgetting (AF) measures the decline in model performance after learning all tasks, which only captures the assess the model's long-term forgetting behavior. To evaluate the short-term forgetting of the model, we introduce a new metric, termed "short-term average forgetting" (AF-st),

Table 7: Coarsen runtime and trade-offs of **TA**$\mathbb{CO}$ variations with different coarsening methods on three datasets with GCN (average over 10 trials). Boldface indicates the best result of each column.

Dataset	**Method**	**Time** (s) ↓	$\tau_{\Delta\text{F1-AP}}$ ↓	$\tau_{\Delta\text{F1-AF}}$ ↓	$\tau_{\Delta\text{BACC-AP}}$ ↓	$\tau_{\Delta\text{BACC-AF}}$ ↓
Kindle	Alge. JC	8.9	0.74	0.69	1.17	1.02
	Aff. GS	65.6	7.88	6.96	14.48	12.08
	Var. neigh.	6.9	0.59	0.54	0.97	0.85
	Var. edges	10.1	0.77	0.70	1.23	1.08
	FGC	7.7	0.63	0.58	1.00	0.88
	RePro	**2.3**	**0.17**	**0.15**	**0.24**	**0.22**
DBLP	Alge. JC	70.8	4.08	3.90	5.65	5.20
	Aff. GS	237.1	14.42	14.2	20.89	20.74
	Var. neigh.	7.3	0.43	0.41	0.55	0.50
	Var. edges	28.0	1.63	1.53	1.98	1.77
	FGC	10.8	0.64	0.61	0.83	0.76
	RePro	**1.1**	**0.07**	**0.06**	**0.08**	**0.07**
ACM	Alge. JC	11.8	1.19	1.17	1.58	1.45
	Aff. GS	96.1	12.29	12.5	16.07	15.4
	Var. neigh.	10.3	1.58	1.61	1.82	1.79
	Var. edges	13.8	1.24	1.20	1.65	1.50
	FGC	7.0	1.05	1.06	1.15	1.11
	RePro	**1.4**	**0.13**	**0.13**	**0.19**	**0.17**

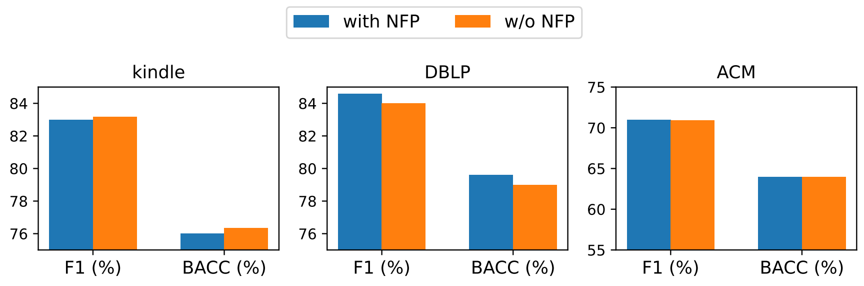


Figure 6: Average performance of the **TA**$\mathbb{CO}$ with and without Node Fidelity Preserving (NFP).

which measure the decline in model performance on the most recent task when it learns a new one:

$$\text{AF-st} = \frac{1}{T}\sum_{j=2}^{T} a_{j-1,j-1} - a_{j,j-1},$$

where T is the total number of task, and $a_{i,j}$ is the prediction metric of model on test set of task j after it is trained on task i. We report the AF-st in terms of F1 score with GCN as backbone on three datasets in Table 8.

G.2 Shorter time interval

We investigate the performance of **TA**$\mathbb{CO}$ and other baselines when each task is assigned with a shorter time interval. For the DBLP and ACM datasets, we divided them into one-year time intervals, resulting in a total of 20 tasks. We have included the AP-f1 and AF-f1 scores of all baselines utilizing GCN as their backbone in Table 9. Our findings indicate that, compared to our previous dataset splitting approach, most CGL methods exhibit a slight decline in performance, but TACO continues to outperform the other baseline models.

G.3 Efficiency Analysis

We analyze the efficiency of **TA**$\mathbb{CO}$ and other experience-replay-based CGL baselines in terms of training time and memory usage. We report the averaged total training time (including the time to learn model parameters and the time to store the memory/coarsen graphs), and the averaged memory

Table 8: The averaged short-term forgetting in terms of F1 score (%) with GCN as the backbone on three datasets (averaged over 10 trials).

Method	**Kindle**	**DBLP**	**ACM**
joint train	0.33	0.37	0.69
finetune	33.15	22.25	19.57
simple-reg	29.91	19.39	19.07
EWC	22.92	13.34	15.68
TWP	21.25	14.19	15.47
OTG	32.89	20.67	19.45
GEM	17.43	9.25	10.95
ERGNN-rs	10.42	8.07	10.23
ERGNN-rb	15.78	9.30	12.72
ERGNN-mf	13.95	8.72	13.72
DyGrain	32.76	20.52	19.67
IncreGNN	32.85	21.42	19.68
SSM	12.18	5.15	10.26
SSRM	9.68	8.32	10.51
TA$\mathbb{C}\mathbb{O}$	10.26	0.18	6.44

Table 9: Node classification performance with GCN as the backbone on two datasets (averaged over 10 trials) with shorter time intervals and more tasks. Standard deviation is denoted after $\pm$.

	DBLP				**ACM**			
Method	F1-AP(%)		F1-AF (%)		F1-AP (%)		F1-AF (%)	
joint train	84.38	$\pm$1.49	1.60	$\pm$0.22	73.70	$\pm$0.71	3.04	$\pm$0.54
finetune	66.05	$\pm$11.30	22.45	$\pm$10.62	60.16	$\pm$8.58	19.56	$\pm$8.92
simple-reg	67.25	$\pm$7.80	21.50	$\pm$6.86	58.44	$\pm$7.62	21.10	$\pm$7.87
EWC	79.04	$\pm$6.15	8.84	$\pm$5.64	66.85	$\pm$4.66	11.77	$\pm$4.72
TWP	79.35	$\pm$5.83	8.56	$\pm$5.41	66.52	$\pm$4.50	12.15	$\pm$4.64
OTG	67.45	$\pm$8.31	21.03	$\pm$7.33	60.28	$\pm$8.18	19.54	$\pm$8.38
GEM	79.43	$\pm$3.66	8.41	$\pm$2.44	67.76	$\pm$3.01	10.58	$\pm$3.32
ERGNN-rs	75.08	$\pm$6.32	13.13	$\pm$5.48	61.43	$\pm$7.76	17.64	$\pm$8.16
ERGNN-rb	71.85	$\pm$7.55	16.46	$\pm$6.51	61.23	$\pm$7.67	18.09	$\pm$7.72
ERGNN-mf	74.24	$\pm$6.50	13.94	$\pm$5.37	63.13	$\pm$6.61	16.22	$\pm$6.83
DyGrain	67.96	$\pm$9.19	20.58	$\pm$8.35	61.12	$\pm$8.14	18.51	$\pm$8.45
IncreGNN	66.19	$\pm$7.88	22.34	$\pm$7.31	60.53	$\pm$8.42	19.08	$\pm$8.70
SSM	82.08	$\pm$1.99	4.37	$\pm$1.12	67.22	$\pm$1.95	10.63	$\pm$2.30
SSRM	75.35	$\pm$6.14	12.95	$\pm$5.19	62.11	$\pm$7.61	17.09	$\pm$7.91
TA$\mathbb{C}\mathbb{O}$	**83.06**	**$\pm$2.25**	**4.18**	**$\pm$1.69**	**68.31**	**$\pm$3.21**	**10.17**	**$\pm$3.63**

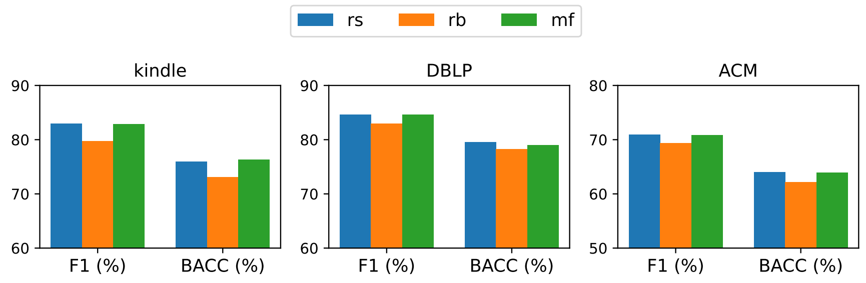
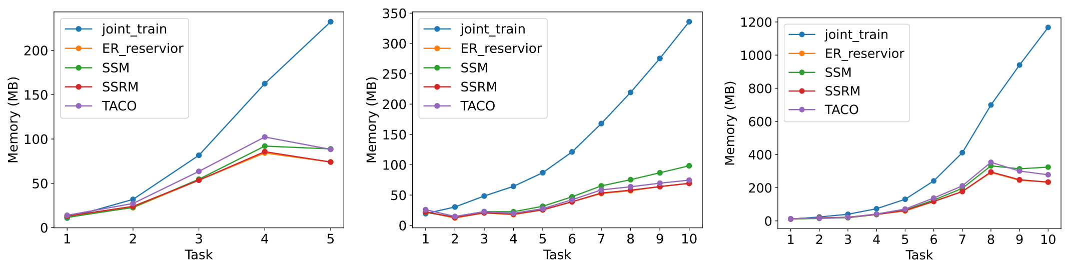


Figure 7: Average performance of the **TA**ℂ𝕆 with different node sampling strategies: Reservior-Sampling (rs), Ring Buffer (rb), and Mean Feature (mf).



(a) Kindle (b) DBLP (c) ACM

Figure 8: Memory usages of different methods over tasks.

usage of the model (including the memory to store data for current task and the memory buffer to store information of previous tasks) for each task in Table 10. We find that on average, SSM uses less memory than **TA**ℂ𝕆 on the Kindle dataset. However, on the DBLP and ACM datasets, SSM's memory usage is either more or similar. It's important to note that SSM maintains a sparsified graph that expands as additional tasks are introduced. As a result, SSM's memory continues to grow with the increasing number of tasks. In contrast, **TA**ℂ𝕆, with its dynamic coarsening algorithm, consistently maintains a relatively stable memory regardless of the number of tasks.

G.4 Node Fidelity Preservation

We investigate the effectiveness of the Node Fidelity Preservation strategy by removing it from **TA**ℂ𝕆 and compare the average performance of the variant of **TA**ℂ𝕆 with its default version. We report the average performance of the model on the three datasets in Figure 6. We observe that on DBLP dateset, the application of Node Fidelity Preservation improves the performance, while on the other two datasets, the model performs comparably or marginally better without Node Fidelity Preservation. Note that our proposal of Node Fidelity Preservation is intended as a preventative measure, not as a means of enhancing model performance. Node Fidelity Preservation aims to prevent the situation

Table 10: The averaged memory usage (MB) for each task of experience-replay-based methods.

Method	**Kindle**	**DBLP**	**ACM**
ERGNN-rs	49.5	38.0	121.1
ERGNN-rb	48.5	37.5	119.4
ERGNN-mf	48.5	37.6	119.5
DyGrain	47.9	37.1	118.1
IncreGNN	47.9	37.1	118.1
SSM	53.9	48.3	144.1
SSRM	50.0	38.4	122.6
TAℂ𝕆	59.0	41.9	144.4

Table 11: Statistics of datasets. “Interval” indicates the length of the time interval for each task. “# Item” indicates the total number of items/papers in each dataset.

Dataset	**#Class**	**#Task**	**#Class / task**	**#Items**
cora	7	3	2/3	2,708
coauthor-cs	15	5	3	18,333
corafull	70	10	7	19,793

where minority classes vanish from the reduced graph due to unbalanced class distribution. Therefore, the improvement may not be noticeable if the class distribution is relatively balanced and node degradation is insignificant. In such cases, preserving the selected nodes may prevent the graph from coarsening to the optimized structure, which could even make the performance worse.

G.5 Important node sampling strategies

We investigate how different important node sampling strategies affect the performance of the model. We report the average node classification performance of **TA**$\mathbb{CO}$ with different node sampling strategies, Reservior-Sampling, Ring Buffer, and Mean Feature on the three datasets in Figure 7. It shows that **TA**$\mathbb{CO}$ achieves better performance with Reservior-Sampling and Mean Feature. We deduce that it is caused by the fact that Ring Buffer operates on a first in, first out (FIFO) basis, that it only retains the most recent samples for each class, making it fail to preserve information from the distant past.

G.6 Performance on traditional class-incremental learning setup

We also evaluate **TA**$\mathbb{CO}$ on traditional class-incremental learning setup to prove its generability. We conduct experiments on three commonly used datasets in GCL literature including cora, coauthor-cs, and corafull. The details about their statistics and splitting are provided in Table 11. The performance of **TA**$\mathbb{CO}$ with GCN as backbone in comparison with other baselines is provided in Table 12, which shows that the forgetting could be more severe when tasks share non-overlapping class sets, and **TA**$\mathbb{CO}$’s rule on alleviating the forgetting is more evidently demonstrated.

Table 12: Node classification performance in terms of F1 and BACC with GCN on three datasets under task-IL settings (average over 10 trials). Standard deviation is denoted after $\pm$.

Dataset	**Method**	**F1-AP** (%)	**F1-AF** (%)	**BACC-AP** (%)	**BACC-AF** (%)
	joint train	82.38±1.33	5.97±0.74	73.68±2.23	9.98±0.83
	finetune	32.59±1.32	61.05±1.74	31.85±0.95	60.92±1.50
	simple-reg	39.11±4.75	54.30±4.86	37.12±4.48	55.18±4.75
	EWC	32.13±0.80	61.32±1.31	31.33±0.55	60.93±1.15
	TWP	33.34±1.85	60.02±1.85	32.29±1.74	59.92±1.69
	OTG	30.46±0.42	61.68±0.62	30.09±0.39	61.18±0.76
cora	GEM	57.86±2.42	34.15±2.42	50.50±2.81	39.09±3.02
	ERGNN-rs	74.48±2.55	14.93±2.53	65.59±2.36	19.92±2.18
	ERGNN-rb	69.05±1.81	21.49±1.60	61.62±1.97	25.76±1.80
	ERGNN-mf	69.11±1.34	21.38±1.53	61.84±1.42	25.59±1.47
	DyGrain	32.83±1.87	60.81±1.97	31.94±1.35	60.78±1.45
	IncreGNN	32.57±1.33	60.85±1.57	31.65±1.04	60.63±1.31
	SSM	59.61±5.66	29.94±5.38	48.14±5.31	37.05±4.83
	SSRM	75.16±1.38	14.17±1.65	66.15±1.44	19.48±1.70
	TAℂ𝕆	**78.16±1.29**	**9.69±1.35**	**67.75±1.25**	**14.38±1.49**
	joint train	92.30±0.24	3.11±0.31	86.43±0.61	5.55±0.45
	finetune	20.16±0.37	77.07±0.41	20.30±1.03	75.68±0.95
	simple-reg	24.88±1.54	72.01±1.46	22.98±1.42	71.68±1.38
	EWC	48.47±7.07	41.84±8.61	36.41±5.12	46.56±6.51
	TWP	45.45±11.11	38.59±9.56	33.01±9.46	42.58±9.48
	OTG	19.92±0.05	77.06±0.12	19.83±0.09	75.99±0.21
coauthor-cs	GEM	69.59±4.54	26.61±4.45	62.81±4.64	31.26±4.47
	ERGNN-rs	64.75±4.34	31.02±4.49	56.29±4.99	36.89±5.09
	ERGNN-rb	62.18±3.05	34.13±3.15	59.95±2.78	34.17±2.92
	ERGNN-mf	46.16±5.51	50.46±5.46	45.75±4.28	49.06±4.27
	DyGrain	23.62±6.34	70.43±10.61	21.26±3.35	69.56±10.47
	IncreGNN	25.30±8.82	71.03±9.61	23.77±7.43	70.48±7.76
	SSM	59.87±3.89	35.45±3.97	48.26±2.97	43.56±2.36
	SSRM	30.91±3.09	65.85±3.21	28.26±2.79	66.71±3.03
	TAℂ𝕆	**92.84±0.85**	**2.22±0.76**	**87.24±1.72**	**3.39±1.56**
	joint train	74.33±0.48	4.00±0.50	58.94±0.61	6.96±0.77
	finetune	7.24±0.35	76.53±0.57	6.35±0.29	72.52±0.76
	simple-reg	11.18±2.10	66.65±3.13	8.01±1.28	61.04±2.70
	EWC	20.47±5.27	60.30±4.66	13.45±3.17	60.18±3.14
	TWP	19.79±3.28	61.30±3.55	13.30±2.14	60.51±2.46
	OTG	7.43±0.36	77.26±0.61	6.54±0.37	73.15±0.79
corafull	GEM	26.17±6.79	58.92±6.57	16.43±4.35	62.33±4.23
	ERGNN-rs	23.82±3.88	59.97±4.24	15.11±2.48	60.57±3.05
	ERGNN-rb	18.82±2.59	65.29±3.02	15.28±1.97	61.87±2.85
	ERGNN-mf	19.19±2.54	65.29±2.36	15.28±1.95	62.18±1.95
	DyGrain	7.42±0.31	77.03±0.50	6.54±0.25	73.10±0.60
	IncreGNN	7.58±0.24	77.18±0.51	6.69±0.24	73.20±0.84
	SSM	45.77±1.76	35.59±1.85	32.37±1.60	38.53±1.73
	SSRM	22.46±2.95	61.22±2.73	14.11±1.93	61.42±1.80
	TAℂ𝕆	**55.74±1.13**	**17.14±1.13**	**39.50±1.08**	**18.13±0.81**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope, as demonstrated through both theoretical analysis and empirical study.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a separate section in the paper to discuss the limitations of our works.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We provide the full set of assumptions and a complete (and correct) proof for each theoretical result.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides all necessary implementation details, including pseudocode, to reproduce the main experimental results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We submit source code, data, and comprehensive instructions to faithfully reproduce the main experimental results.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all the training and test details in Appendix E.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report standard deviation and p-values to demonstrate statistical significance.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide information about the GPU used for the experiments and also report and analyze memory and time usage.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirm the research conducted in the paper conforms with it.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the positive social impact of this work in a separate section and do not foresee any negative social impact.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use open-source datasets and properly cite them.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We submit a zip file containing the code for our proposed method, along with detailed implementation instructions.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No human subjects were involved in the research conducted in this paper.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No human subjects were involved in the research conducted in this paper.