ON MAKING GRAPH CONTINUAL LEARNING EASY, FOOL-PROOF, AND EXTENSIVE: A BENCHMARK FRAMEWORK AND SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual Learning (CL) is the process of learning ceaselessly a sequence of tasks. Most existing CL methods deal with independent data (e.g., images and text) for which many benchmark frameworks and results under standard experimental settings are available. CL methods for graph data, however, are surprisingly underexplored because of (a) the lack of standard experimental settings, especially regarding how to deal with the dependency between instances, (b) the lack of benchmark datasets and scenarios, and (c) high complexity in implementation and evaluation due to the dependency. In this paper, regarding (a), we define four standard incremental settings (task-, class-, domain-, and time-incremental settings) for graph data, which are naturally applied to many node-, link-, and graph-level problems. Regarding (b), we provide 23 benchmark scenarios based on 14 real-world graphs. Regarding (c), we develop BEGIN, an easy and fool-proof framework for graph CL. BEGIN is easily extended since it is modularized with reusable modules for data processing, algorithm design, and evaluation. Especially, the evaluation module is completely separated from user code to eliminate potential mistakes in evaluation. Using all above, we report extensive benchmark results of seven graph CL methods. Compared to the latest benchmark for graph CL, using BEGIN, we cover 3× more combinations of incremental settings and levels of problems.

1 INTRODUCTION

Continual Learning (CL), which is also known as lifelong learning and incremental learning, is the process of learning continuously a sequence of tasks. CL aims to retain knowledge from previous tasks (i.e., knowledge consolidation) to overcome the degradation of performance (i.e., catastrophic forgetting) on previous tasks. Recently, CL has received considerable attention because of its similarity to the development of human intelligence.

Most of the existing works for CL deal with independent data, such as images and text. For example, Shin et al. (2017) aim to learn a sequence of image-classification tasks where the domains of images vary with tasks (i.e., under a domain-incremental setting), and Ermis et al. (2022) aim to learn a sequence of text-classification tasks where the possible classes of text grow over tasks (i.e., under a class-incremental setting). For CL with independent data, many datasets (Hsu et al., 2018; Goodfellow et al., 2013; Lomonaco & Maltoni, 2017) and benchmarks (Lomonaco et al., 2021; Lin et al., 2021; Pham et al., 2021; He & Sick, 2021; Shin et al., 2017) have been provided.

CL is naturally beneficial to graph data in the real world, which grows in size and diversity while being applied to more and more domains. Especially, since real-world graphs are massive and changing ceaselessly, it is computationally inefficient to train a new model per task or change, and CL is a promising remedy. However, compared to CL with independent data, CL with graph data (graph CL) (Febrinanto et al., 2022; Galke et al., 2021; Zhou & Cao, 2021) has been relatively underexplored, mainly due to the complexity caused by the dependency between instances. For example, in node classification, the class of a node is correlated not only with its features but also with its connections to other nodes, which may belong to other tasks, and their features. In addition to the complexity, major reasons of relative unpopularity of graph CL include (a) the lack of standard experimental settings, especially regarding how to deal with changes in various dimensions (e.g.,

Table 1: Compared with existing benchmarks for graph continual learning (Zhang et al., 2022; Carta et al., 2021), ours covers 7 more combinations, and it is performed thoroughly with more metrics.

			BEGIN (Proposed)			(Zhang et al., 2022)			(Carta et al., 2021)		
	Problem Level		Node	Link	Graph	Node	Link	Graph	Node	Link	Graph
Scenarios	Incremental Setting	Task	✓	✓	✓	✓	✗	✓	✗	✗	✗
		Class	✓	✓	✓	✓	✗	✓	✗	✗	✓
		Domain	✓	✓	✓	✗	✗	✗	✗	✗	✗
		Time	✓	✓	✓	✗	✗	✗	✗	✗	✗
Evaluation	# Eval. Metrics		**4**			2			1		

domains and classes) and the dependency between instances, and (b) the lack of benchmarking datasets and scenarios. In this paper, we focus on resolving these issues.

Our first contribution is to define four incremental settings for graph data. To this end, we identify and decouple four possible dimensions of changes, which are tasks, classes, domains, and time. Each of the settings is defined so that (a) the dependency between instances, which is a unique property of graph data, can be utilized, and at the same time, (b) catastrophic forgetting may happen without careful consideration of it. Then, we show that the settings can be applied to node-, link-, and graph-level problems, including node classification, link prediction, and graph classification. After that, we provide 23 benchmark scenarios for graph from 14 real-world datasets, which cover 12 combinations of the incremental settings and the levels of problems, as summarized in Table 1.

Our second contribution is BEGIN (**Be**nchmarking **G**raph Cont**in**ual Learning), which is a framework for implementation and evaluation of graph CL methods. Evaluation is complicated for graph CL due to additional dependency between instances. Specifically, instances of one task are often used also for other tasks to exploit the dependency, and thus without careful consideration, information that should not be shared among tasks can be leaked among tasks. In order to eliminate potential mistakes in evaluation, BEGIN is **fool-proof** by completely separating the evaluation module from the learning part, where users implement their own graph CL methods. The learning part only has to answer queries provided by the evaluation module after each task is processed. BEGIN is **easy-to-use**. It is easily extended since it is modularized with reusable modules for data processing, algorithm design, validation, and evaluation.

Our last contribution is extensive benchmark results of seven graph CL methods based on our proposed scenarios and framework. We use four metrics for evaluation from various perspectives. For reproducibility, we provide all source code required for reproducing the benchmark results and documents at **https://anonymous.4open.science/r/BeGin-1C33/**.

2 RELATED WORKS

Continual Learning with Independent Data. Continual learning (CL) methods have been developed mostly for independent data (e.g., images and text), and they are mainly categorized into regularization-, replay-, and parameter-isolation-based methods. Regularization-based methods (Kirkpatrick et al., 2017; Aljundi et al., 2018; Li & Hoiem, 2017) seek to consolidate knowledge from previous tasks by introducing regularization terms in the loss function. For example, Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) weights to parameters according to the diagonal of the Fisher information matrix, and Memory Aware Synapses (MAS) (Aljundi et al., 2018) computes the importance of parameters according to how sensitive the parameters are. Learning without Forgetting (LwF) (Li & Hoiem, 2017) minimizes the difference between outputs of a previous model and a new model. Replay-based methods store a sample of data for previous tasks (Chaudhry et al., 2019; Isele & Cosgun, 2018; Rolnick et al., 2019; Rebuffi et al., 2017). Then, they re-use the data while learning a new task to mitigate forgetting. For example, Gradient Episodic Memory (GEM) (Lopez-Paz & Ranzato, 2017) stores data from previous tasks and prevents the increase of losses on them while learning a new task. Parameter-isolation-based methods (Aljundi et al., 2017; Fernando et al., 2017; Mallya & Lazebnik, 2018; Serra et al., 2018) allocate different parameters to each task to prevent forgetting. For example, Progressive Neural Network (Rusu et al., 2016) freezes parameters for previous tasks while learning a new task. There exist a number of frameworks (Lomonaco et al., 2021; Lin et al., 2021; Pham et al., 2021; He & Sick, 2021; Shin et al., 2017) for implementation and evaluation of continual learning methods for independent data. However, none of them currently supports CL with graph data.

Continual Learning with Graph Data. Due to their expressiveness, graphs are widely used to model various types of data, and thus considerable attention has been paid to machine learning with graph-structured data. Since many such graphs (e.g., online social networks) evolve over time, continual learning is desirable for them, and thus several CL methods for graph-structured data have been developed (Febrinanto et al., 2022; Wang et al., 2020a; Liu et al., 2021; Zhou & Cao, 2021; Galke et al., 2021; Wang et al., 2022). For example, ER-GNN (Zhou & Cao, 2021) carefully samples nodes and uses them for re-training, and TWP (Liu et al., 2021) stabilizes parameters important in topological aggregation by graph neural networks through regularization. The replay- and regularization-based approaches are combined in CGNN (Wang et al., 2020a). Despite these efforts, CL with graph data (graph CL) is still largely underexplored, especially compared with CL with independent data, and the lack of benchmark frameworks and scenarios is one major reason, as discussed in Section 1. To the best of our knowledge, there exist only two benchmarks for graph CL (Carta et al., 2021; Zhang et al., 2022). However, Carta et al. (2021) supports only graph-level tasks under one incremental setting, and Zhang et al. (2022) supports node- and graph-level tasks under only two incremental settings, as further described in Section 3.2. Compared to them, our benchmark and framework are more extensive, as summarized in Table 1.

Comparison with Other Graph Learning Methods. In CL problems, distribution shifts may occur over tasks, and such shifts can be observed from training data for each task. There also exist works that deal with unobservable shifts between training and test distributions Li et al. (2022); Wu et al. (2021); Baek et al. (2020). For instance, Baek et al. (2020) use meta-learning to address few-shot out-of-graph link prediction. Moreover, transfer learning (TL) on graph data has been studied for various downstream tasks (e.g., link prediction Tang et al. (2016)). While the goal of CL is to train a single model for a sequence of tasks, TL aims to adapt a separate model to a new task. In addition, many works on incremental/dynamic graph learning (Rossi et al., 2020; You et al., 2022) focus on the latest snapshot of a dynamic graph to maximize performance on it. On the other hand, Graph CL, especially, under Time-IL settings, aims not only for the current snapshot but also to preserve performance on past snapshots, which can be especially useful when seasonality is present.

3 Benchmark Scenarios: Problems, Settings, and Datasets

We introduce our benchmark scenarios by describing the considered graph-learning problems, incremental settings, and real-world datasets.

Common Notations. We denote each i-th task in a sequence by $\mathcal{T}_i$. We use $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ to denote a graph that consists of a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E}$, and node features $\mathcal{X} : \mathcal{V} \rightarrow \mathbb{R}^d$, where d is the number of node features. In some of the considered datasets, edge features are given in addition to or instead of node features, and they can be treated similarly to node features. Lastly, we use $\mathcal{Q}$ to indicate the set of queries used for evaluation.

3.1 Graph Problems of Three Levels

Our benchmark scenarios are based on various node-, link-, graph-level problems. Below, we describe node classification, link prediction, and graph classification, as examples.

Node Classification (NC). For a node classification (NC) task $\mathcal{T}_i$, the input consists of (a) a graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \mathcal{X}_i)$, (b) a labelled set $\mathcal{V}'_i \in \mathcal{V}_i$ of nodes, (c) a set of classes $\mathcal{C}_i$, and (d) the class $f(v) \in \mathcal{C}_i$ for each node $v \in \mathcal{V}'_i$. A query q on a NC task $\mathcal{T}_i$ is a node $v_q \notin \mathcal{V}'_i$ where $f(v_q) \in \mathcal{C}_i$, and its ground-truth answer is $f(v_q)$.

Link Prediction (LP). For a link prediction (LP) task $\mathcal{T}_i$, the input consists of a graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i \setminus \mathcal{E}'_i, \mathcal{X}_i)$, where $\mathcal{E}_i$ is the ground-truth set of edges, and $\mathcal{E}'_i \in \mathcal{E}_i$ is the set of missing edges among them. A query on a LP task $\mathcal{T}_i$ is a node pair $(u, v) \notin (\mathcal{E}_i \setminus \mathcal{E}'_i)$, and its ground-truth answer is $\mathbf{1}((u, v) \in \mathcal{E}'_i)$, i.e., whether there exist a missing edge between u and v or not.

Graph Classification (GC). For a graph classification (GC) task $\mathcal{T}_i$, the input consists of (a) a labelled set of graphs $\mathcal{S}_i$, (b) a set of classes $\mathcal{C}_i$, and (c) the class $f(\mathcal{G}) \in \mathcal{C}_i$ for each graph $\mathcal{G} \in \mathcal{S}_i$. A query q on a GC task $\mathcal{T}_i$ is a graph $\mathcal{G}_q \notin \mathcal{S}_i$ where $f(\mathcal{G}_q) \in \mathcal{C}_i$, and its ground-truth answer is $f(\mathcal{G}_q)$.

The problem definition of **Link Classification (LC)** problem, which is also used for our benchmark, is extended straightforwardly from that of node classification.

3.2 FOUR INCREMENTAL SETTINGS

We introduce four incremental settings for continual learning with graph data and describe how they can be applied to the above three problems. When designing them, we aim to (a) decouple changes in different dimensions (tasks, classes, domains, and time), (b) make the dependency between instances (e.g., connections between nodes) exploitable, and (c) make the input for a task (partially) lost in later tasks so that catastrophic forgetting may happen without careful attention to it.

Task-Incremental (Task-IL). In this incremental setting, the set of classes varies with tasks (i.e., $\forall i \neq j, \mathcal{C}_i \neq \mathcal{C}_j$), and they are often disjoint (i.e., $\forall i \neq j, \mathcal{C}_i \cap \mathcal{C}_j = \emptyset$). In addition, for each query in $\mathcal{Q}$, the corresponding task, which we denote by $\mathcal{T}_i$, is provided, and thus its answer is predicted among $\mathcal{C}_i$. This setting is applied to NC and GC tasks, where the sets of classes can vary with tasks, and for NC tasks, the input graph is fixed (i.e., $\forall i \neq j, \mathcal{G}_i = \mathcal{G}_j$).

Class-Incremental (Class-IL). In this incremental setting, the set of classes grows over tasks (i.e., $\forall i < j, \mathcal{C}_i \subsetneq \mathcal{C}_j$). In addition, for each query in $\mathcal{Q}$, the corresponding task is **NOT** provided, and thus its answer is predicted among all classes seen so far (i.e., $\bigcup_{j \leq i} \mathcal{C}_j$ for a current task $\mathcal{T}_i$). This setting is applied to NC and GC tasks, where the sets of classes can vary with tasks, and for NC tasks, the input graph is fixed (i.e., $\forall i \neq j, \mathcal{G}_i = \mathcal{G}_j$).

Domain-Incremental (Domain-IL). In this incremental setting, we divided entities (i.e., nodes, edges, and graphs) over tasks according to their domains, which are additionally given (see Section 3.3 for real-world examples of domains). The details for each problem is as follows:

- NC: The labelled nodes of the input graph are divided into NC tasks according to their domains. The input graph is fixed (i.e., $\forall i \neq j, \mathcal{G}_i = \mathcal{G}_j$) for all NC tasks.
- LP: The ground-truth edges are partitioned into (a) the set $\bar{\mathcal{E}}$ of base edges and (b) the set $\tilde{\mathcal{E}}$ of additional edges. The base edges are provided commonly for all LP tasks, and they are especially useful when answering queries on past tasks.[1] The additional edges $\tilde{\mathcal{E}}$ are divided further into LP tasks according to their domains. For each LP task $\mathcal{T}_i$, $\bar{\mathcal{E}} \cup \tilde{\mathcal{E}}_i$, where $\tilde{\mathcal{E}}_i$ is the additional edges assigned to $\mathcal{T}_i$, is used as the ground-truth edges (i.e., $\mathcal{E}_i = \bar{\mathcal{E}} \cup \tilde{\mathcal{E}}_i$), and the missing ground-truth edges are chosen among $\tilde{\mathcal{E}}_i$, i.e., $\mathcal{E}'_i \subset \tilde{\mathcal{E}}_i$.
- GC: The labelled graphs are divided into GC tasks, according to their domains.

Note that, as domains, we use information not directly related to labels, which we aim to infer.

Time-Incremental (Time-IL). In this incremental setting, we consider a dynamic graph evolving over time. We denote its i-th snapshot by $\mathcal{G}^{(i)} = (\mathcal{V}^{(i)}, \mathcal{E}^{(i)}, \mathcal{X}^{(i)})$. The set of classes may or may not vary across tasks, and the details for each problem is as follows:

- NC: The input graph for each NC task $\mathcal{T}_i$ is the i-th snapshot $\mathcal{G}^{(i)}$ of the dynamic graph, and labelled nodes are given among new nodes added to the snapshot (i.e., $\mathcal{V}'_i \subset \mathcal{V}^{(i)} \setminus \mathcal{V}^{(i-1)}$, where $\mathcal{V}^{(0)} = \emptyset$). The label and features of each node are fixed over time and thus for all tasks.
- LP: As in the Domain-IL setting, base edges are used. For each LP task $\mathcal{T}_i$, the set $\bar{\mathcal{E}}_i$ of base edges so far and the new edges added to the i-th snapshot $\mathcal{G}^{(i)}$ of the dynamic graph are used as the ground-truth edges (i.e., $\mathcal{E}_i = \bar{\mathcal{E}}_i \cup \mathcal{E}^{(i)} \setminus \mathcal{E}^{(i-1)}$, where $\mathcal{E}^{(0)} = \emptyset$). After each task is processed, a subset of $\mathcal{E}^{(i)} \setminus \mathcal{E}^{(i-1)} \setminus \mathcal{E}'_i$ (i.e., new edges that are not used as missing edges) are added as base edges. The features of each node are fixed over time and thus for all tasks.
- GC: The snapshots of the dynamic graph are grouped and assigned to tasks in chronological order. Specifically, for any i and j where $i < j$, every snapshot in the labelled set $\mathcal{S}_i$ of the GC task $\mathcal{T}_i$ is older than every snapshot in $\mathcal{S}_j$ of $\mathcal{T}_j$. The features of nodes may change over time.

The above settings can also be applied to the **link classification (LC)** problem straightforwardly, as they are applied to the NC problem.

Remarks: The above Task/Class-IL settings are different from those in (Zhang et al., 2022), where the edges between labelled nodes are added to the input graph together with the labelled nodes in

[1] Without base edges, queries on past tasks should be answered using only edges of different domains, which is very restrictive.

Table 2: **Summary of the considered real-world datasets.**

Problem Level	Dataset	# Nodes	# Edges	# Node (Edge) Features	# Classes	Incremental Settings (# Tasks)
Node	Cora	2,708	10,556	1,433 (0)	7	Task (3), Class (3)
	Citeseer	3,327	9,104	3,703 (0)	6	Task (3), Class (3)
	CoraFull	19,793	126,842	8,710 (0)	70	Task (35)
	ogbn-arxiv	169,343	2,232,486	128 (0)	40	Task (8), Class (8), Time (11)
	ogbn-proteins	132,534	39,561,252	0 (8)	2×112	Domain (8)
	ogbn-products	2,449,029	61,859,140	100 (0)	47	Class (9)
Link	Wiki-CS	11,701	431,726	300 (0)	2	Domain (10)
	Bitcoin-OTC	5,881	35,592	0 (0)	21	Task (3), Class (3), Time (7)
	ogbl-collab	235,868	1,285,465	128 (0)	2	Time (9)
Problem Level	**Dataset**	**# Graphs (Avg. # Nodes)**	**# Avg. Edges**	**# Node (Edge) Features**	**# Classes**	**Incremental Settings (# Tasks)**
Graph	MNIST	55,000 (70.6)	564.5	3 (0)	10	Task (5), Class (5)
	CIFAR10	45,000 (117.6)	941.2	5 (0)	10	Task (5), Class (5)
	Aromaticity	3,868 (29.7)	65.4	0 (0)	30	Task (10), Class (10)
	ogbg-molhiv	41,127 (25.5)	27.5	9 (3)	2	Domain (10)
	NYC-Taxi	8,760 (265.0)	1597.8	7 (1)	2	Time (12)

each task. Thus, the dynamics of the input graph, which we independently consider in the Time-IL setting, are coupled with changes in tasks or classes, without any good reason.

3.3 14 Real-world datasets and 23 Benchmark Scenarios

We describe 14 real-world datasets and 23 benchmark scenarios based on them under various incremental settings.

Datasets for Node-Level Problems.

- Cora, Citeseer (Sen et al., 2008) , and CoraFull (Bojchevski & Günnemann, 2018) are citation networks. Each node is a scientific publication, and its class is the field of the publication. For Cora and Citeseer, based on six classes in each dataset, we formulate three binary classification tasks for Task-IL and three tasks with 2, 4, and 6 classes for Class-IL. Similarly, for CoraFull, we formulate 35 binary classification tasks for Task-IL. Note that, one class in Cora is not used.
- Nodes in ogbn-proteins (Hu et al., 2020; Szklarczyk et al., 2019) are proteins, and edges indicate meaningful associations between proteins. For each protein, 112 binary classes, which indicate the presence of 112 functions, are available. Each protein belongs to one among 8 species, which are used as domains in Domain-IL. Each of the 8 task consists of 112 binary-classification problems.
- ogbn-arxiv (Hu et al., 2020; Wang et al., 2020b) is a citation network, where each node is a research paper, and its class belongs to 40 subject areas, which are divided into 8 groups for Task-IL. Similarly, the number of classes increase by 5 in each task in Class-IL. Publication years are used for the Time-IL setting.
- ogbn-products (Hu et al., 2020; Chiang et al., 2019) is a co-purchase network, where each node is a product, and its class belongs to 47 categories, which are divided into 9 groups for Class-IL. The number of classes increase by 5 in each task, and two categories are not used.

Dataset for Link-Level Problems.

- Bitcoin-OTC (Kumar et al., 2016; 2018) is a who-trust-whom network, where nodes are users of a bitcoin-trading platform. Each directed edge has an integer rating between -10 to 10 and a timestamp. The ratings are divided into 6 groups. Two of them are used separately for Task-IL and accumulated for Class-IL. For Time-IL, we formulate 7 tasks using the timestamps, where the signs of the ratings are used as binary classes. Since there is no external node feature, we use in- and out-degrees as node features, as in (Errica et al., 2020).
- Wiki-CS (Mernyei & Cangea, 2020) is a hyperlink network between computer science articles. Each article has a label indicating one of the 10 subfields that it belongs to. For Domain-IL, the node labels are used as domains, and specifically, the edges are divide into 10 groups, according to the labels of their endpoints. If the domains of its two endpoints are different, the domain considered in a later task is assigned to the edge.
- ogbl-collab (Hu et al., 2020; Wang et al., 2020b) is a co-authorship network, where nodes are authors. We use publication years for the Time-IL setting.

Datasets for Graph-Level Problems.

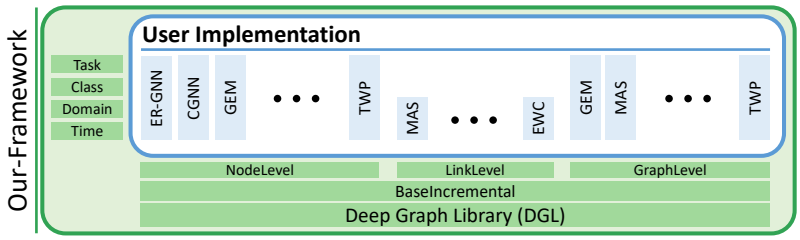

Figure 1: **Modularized structure of BEGIN**, our proposed benchmark framework for implementation and evaluation of continual learning methods for graph data.

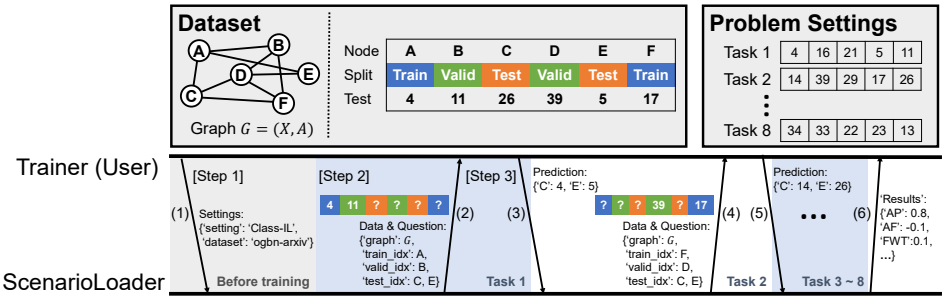

Figure 2: Example communications between the trainer (user code) and the loader of BEGIN.

- Images in MNIST and CIFAR10 (Dwivedi et al., 2020; Achanta et al., 2012) are converted to graphs of super-pixels. There are 10 classes of graphs, and they are partitioned into 5 groups, which are used separately for Task-IL and accumulated for Class-IL.
- Graphs in ogbg-molhiv (Hu et al., 2020; Wu et al., 2018; Landrum et al., 2006) are molecules consisting of atoms and their chemical bonds. The binary class of each graph indicates whether the molecule inhibits HIV virus replication or not. For Domain-IL, we divide molecules into 10 groups based on structural similarity by the scaffold splitting procedure (Landrum et al., 2006).
- Graphs in Aromaticity (Xiong et al., 2019; Wu et al., 2018) are also molecules consisting of atoms and their chemical bonds. The original dataset contains labels representing the number of aromatic atoms in each molecule. In this paper, we divide molecules into 30 groups based on the labels and formulate Task-IL and Class-IL settings with 10 tasks.
- Each graph in NYC-Taxi[2] shows the amount of taxi traffic between locations in New York City during an hour in 2021. Specifically, nodes are locations, and there exist a directed edge between two nodes if there existed a taxi customer between them during an hour. The number of such customers is used as the edge weight. The date and time of the corresponding taxi traffic are used to partition the graphs into 12 groups for Time-IL. The binary class of each graph indicates whether it indicates taxi traffic on weekdays (Mon.-Fri.) or weekends (Sat.-Sun.).

4 BEGIN: A PROPOSED BENCHMARK FRAMEWORK

In this section, we present BEGIN (**Be**nchmarking **G**raph Cont**in**ual Learning), our proposed benchmark framework for making graph continual learning (graph CL) **(a) Easy**: assisting users so that they can implement new graph CL methods with little effort, **(b) Fool-proof:** preventing potential mistakes of users in evaluation, which is complicated for graph CL, and **(c) Extensive**: supporting various benchmark scenarios, including those described in Section 3. The modularized structure of BEGIN is shown in Figure 1, and below, we focus on its three components.

4.1 SCENARIOLOADER (LOADER)

The ScenarioLoader (loader in short) of BEGIN is responsible for communicating with user code (i.e., the training part) to perform a benchmark under a desired incremental setting. First, the loader receives the entire dataset and the desired incremental setting as inputs. Then, according to the inputs, it processes the dataset into a sequence of tasks, as described in Section 3. Before each task starts, the loader provides (a) the input for the task and (b) the set $\mathcal{Q}$ of queries for evaluation to

[2]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

the user code. Once the user code is done with the current task, the loader receives the predicted answers for the queries in $\mathcal{Q}$. Lastly, if there is no more task to be performed, the loader returns the evaluation results, which are computed by the evaluator module, to the user code. In Figure 2, we provide an example of such communications between the loader and user code.

It should be noticed that the evaluation part, including the ground-truth answers of the queries, are never revealed to the user code even after all tasks are processed, in order to prevent potential mistakes and misuses by users. Also note that all queries in $\mathcal{Q}$ are asked by the loader and answered by the user code after every task is processed, even when some of the queries are on unseen tasks. Otherwise, information about the tasks that queries are on can be revealed to the user code and exploited for answering the questions, which is prohibited in Class-IL settings.

4.2 EVALUATOR

BEGIN provides the evaluator, which computes basic metrics (specifically, accuracy, AUROC, and HITS@K) based on the ground-truth and predicted answers for the queries in $\mathcal{Q}$ provided by the loader after each task is processed. The basic evaluator can easily be extended by users for additional basic metrics. The basic metrics are sent to the loader, and for each basic metric, the basic performance matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$, whre N is the number of tasks, is computed. Each (i, j)-th entry $\mathbf{M}_{i,j}$ indicates the performance on on j-th task $\mathcal{T}_j$ after the i-th task $\mathcal{T}_i$ is processed. Based on $\mathbf{M}$, the following evaluation metrics are computed when k is greater than or equal to i.

- **Average Performance (AP):** Average performance on k tasks after learning the k-th task $\mathcal{T}_k$.
- **Average Forgetting (AF):** Average forgetting on $(k-1)$ tasks after learning the k-th task $\mathcal{T}_k$. We measure the forgetting on $\mathcal{T}_i$ by the difference between the performance on $\mathcal{T}_i$ after learning $\mathcal{T}_k$ and the performance on $\mathcal{T}_i$ right after learning $\mathcal{T}_i$. It is valid for $2 \leq k \leq N$.
- **Intransigence (INT) (Chaudhry et al., 2018):** Average intransigence on k tasks. We measure the intransigence on $\mathcal{T}_i$ by the difference between the performances of the Joint model (see Section 5.1) and the target model on $\mathcal{T}_i$ after learning $\mathcal{T}_i$.
- **Forward Transfer (FWT) (Lopez-Paz & Ranzato, 2017):** Average forward transfer on $(k-1)$ tasks. We measure the forward transfer on $\mathcal{T}_i$ by the difference between the performance on $\mathcal{T}_i$ after learning $\mathcal{T}_{i-1}$ and the performance on $\mathcal{T}_i$ without any learning. It is valid for $2 \leq k \leq N$.

Formally, the evaluation metrics are defined as follows:

$$\textbf{AP} = \frac{\sum_{i=1}^{k} \mathbf{M}_{k,i}}{k}, \ \textbf{AF} = \frac{\sum_{i=1}^{k-1} (\mathbf{M}_{i,i} - \mathbf{M}_{k,i})}{k-1}, \ \textbf{INT} = \frac{\sum_{i=1}^{k} (\mathbf{M}^{\text{Joint}}_{i,i} - \mathbf{M}_{i,i})}{k}, \ \textbf{FWT} = \frac{\sum_{i=2}^{k} (\mathbf{M}_{i-1,i} - r_i)}{k-1},$$

where $\mathbf{M}^{\text{Joint}}$ is a basic performance matrix of the Joint model, and r_i denotes the performance of a randomly initialized model on $\mathcal{T}_i$. Note that **AF** quantifies forgetting of previous tasks, and **INT** measures performance on the current task. **FWT** measures performance on future tasks, to quantify generalizable knowledge retained from previous tasks.

4.3 TRAINER

For usability, BEGIN provides the trainer, which users can extend when implementing and benchmarking new methods. It manages the overall training procedure, including preparing the dataloader, training, and validation, so that users only have to implement novel parts of their methods. As in (Lomonaco et al., 2021), the trainer divides the training procedure of continual learning as a series of events. For example, the subprocesses in the training procedure where the trainer (a) receives the input for the current task, (b) trains the model for one iteration for the current task, and (c) handles the necessary tasks before and after the training are events. Each event is modularized as a function, which users can fill out, and the trainer proceeds the training procedure with the event functions.

One thing we need to consider is that there can be cases where intermediate results generated in each event must be stored to be used in other events. For example, in the EWC method, a penalty term for preventing catastrophic forgetting should be additionally considered to compute the training loss. To compute the term, the learned parameters and the weights computed from the fisher information matrix on the previous tasks are needed, but they cannot be obtained on the current task. In order to resolve this issue, the trainer provides a dictionary where intermediate results can

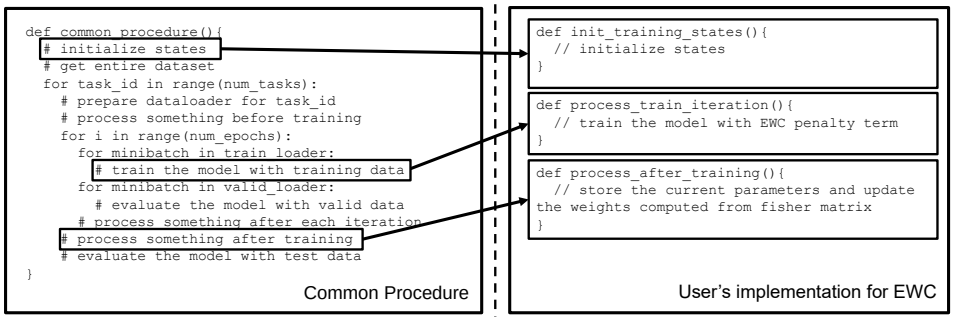

Figure 3: **An example implementation of the EWC method with BEGIN.** In order to implement and benchmark new CL methods, users only need to fill out the modularized event functions in the trainer, which then proceeds the training procedure with the event functions.

be stored and shared by events. For the aforementioned EWC method, the learned parameters and computed weights on a task are stored in the dictionary and used for computing the training loss on the following tasks.

Figure 3 shows how the EWC method for node classification can be implemented with BEGIN. In this case, users only need to fill out (a) `init_training_states` to initialize the dictionary for storing training states, (b) `process_train_iteration` to additionally consider the penalty term in the loss function, and (c) `process_after_training`, which is executed after training is done, to store learned weights and compute the weights of the parameters for the penalty term. The usability of BEGIN is compared numerically with (Zhang et al., 2022) in Table 10.

5 BENCHMARK RESULTS

In this section, we provide benchmark results of seven graph continual learning methods based on the proposed scenarios and framework.

5.1 EXPERIMENTAL SETTINGS

Machines. We performed all experiments on a Linux server with Quadro RTX 8000 GPUs.

Models. For all experiments, we used the Graph Convolutional Network (GCN) (Kipf & Welling, 2017), as the backbone model to compute node embeddings, and the Adam (Kingma & Ba, 2015) optimizer to train the model. For NC, we used a fully-connected layer right after the backbone model to compute the final output. For LC and LP, we additionally used a 3-layer MLP that receives a pair of node embeddings and outputs the final embeddings of the pair. For GC, we used mean pooling to obtain the graph-level embedding from the output of the backbone model, and feed the computed embedding to a 3-layer MLP. For NC and LC, we set the number of layers to 3 and the hidden dimension to 256. For GC, we followed the settings in (Dwivedi et al., 2020) by setting the number of layers to 4 and the hidden dimension to 146.

Training Protocol. We set the number of training epochs to $1,000$ for the Cora, Citeseer, ogbn-arxiv, and Bitcoin-OTC datasets, 200 for the ogbn-proteins, Wiki-CS, and ogbl-collab datasets, and 100 for ogbn-products and GC datasets. For all datasets except for ogbn-products, we performed full-batch training. For ogbn-products, we trained GNNs with the neighborhood sampler provided by DGL for mini-batch training. For all experiments, we used early stopping. Specifically, for NC and LC, we reduced the learning rate by a factor of 10, if the performance did not improve after 20 epochs, and stopped the experiment if the learning rate became $1,000$ times smaller than the initial learning rate. For Citeseer, we set the patience to 50 epochs and used the stopping criteria. For LP and GC, we reduced the learning rate by a factor of 10, if performance did not improve after 10 epochs, and stopped the experiment if the learning rate became 100 times ($1,000$ times in Wiki-CS) smaller the initial learning rate. For the hyperparmeters, we provided the details in Appendix B.

CL Methods. Among general CL methods, we used (1) LwF (Li & Hoiem, 2017), (2) EWC (Kirkpatrick et al., 2017), (3) MAS (Aljundi et al., 2018), and (4) GEM (Lopez-Paz & Ranzato, 2017). For CL methods designed for graph-structured data, we used (5) TWP (Liu et al., 2021), (6) ERGNN (Zhou & Cao, 2021), and (7) CGNN (Wang et al., 2020a). Note that ERGNN and CGNN, which

Table 3: **Results of Average Performance (AP, the higher, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory. N/A: methods are not applicable to the problems or scenarios. We report full results in Appendix A

Methods	Node Classification (NC)				Link Classification (LC)		Link Prediction (LP)		Graph Classification (GC)			
	Cora (Task-IL)	Citeseer (Class-IL)	ogbn-proteins (Domain-IL)	ogbn-arxiv (Time-IL)	Bitcoin-OTC (Task-IL)	Bitcoin-OTC (Class-IL)	Wiki-CS (Domain-IL)	ogbl-collab (Time-IL)	CIFAR10 (Task-IL)	MNIST (Class-IL)	ogbg-molhiv (Domain-IL)	NYC-Taxi (Time-IL)
Bare	0.903±0.018	0.447±0.040	0.690±0.022	0.687±0.002	0.648±0.071	0.243±0.030	0.125±0.035	0.449±0.047	0.646±0.074	0.194±0.005	0.735±0.031	0.804±0.006
LwF	0.915±0.012	0.464±0.039	0.714±0.020	0.696±0.002	0.704±0.032	0.242±0.029	0.136±0.037	0.475±0.036	**0.840±0.030**	0.194±0.005	0.759±0.018	**0.820±0.012**
EWC	0.912±0.013	0.452±0.037	0.761±0.011	0.689±0.002	0.682±0.046	0.242±0.030	0.138±0.038	0.425±0.024	0.784±0.042	0.193±0.006	**0.766±0.013**	0.785±0.020
MAS	**0.918±0.017**	**0.560±0.024**	0.694±0.016	0.664±0.003	**0.706±0.033**	0.244±0.023	0.152±0.042	0.259±0.025	0.762±0.046	0.192±0.006	0.759±0.021	0.764±0.016
GEM	0.882±0.031	0.482±0.033	**0.810±0.003**	0.673±0.003	0.700±0.035	**0.287±0.042**	**0.223±0.114**	**0.484±0.030**	0.769±0.030	**0.199±0.020**	0.737±0.039	0.776±0.025
TWP	0.910±0.015	0.450±0.037	O.O.M	0.682±0.001	0.673±0.046	0.243±0.028	0.134±0.045	0.422±0.039	0.788±0.044	0.190±0.007	0.764±0.012	0.769±0.008
ERGNN	0.890±0.031	0.457±0.043	N/A	0.666±0.008	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CGNN	0.911±0.015	0.531±0.035	N/A	**0.710±0.002**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Joint	0.924±0.015	0.556±0.040	0.732±0.002	0.734±0.002	0.735±0.035	0.377±0.031	0.393±0.026	0.607±0.018	0.868±0.021	0.900±0.004	0.806±0.011	0.866±0.006

Table 4: **Results of Average Forgetting (AF, the lower, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory. N/A: methods are not applicable to the problems or scenarios. We report all results in Appendix A

Methods	Node Classification (NC)				Link Classification (LC)		Link Prediction (LP)		Graph Classification (GC)			
	Cora (Task-IL)	Citeseer (Class-IL)	ogbn-proteins (Domain-IL)	ogbn-arxiv (Time-IL)	Bitcoin-OTC (Task-IL)	Bitcoin-OTC (Class-IL)	Wiki-CS (Domain-IL)	ogbl-collab (Time-IL)	CIFAR10 (Task-IL)	MNIST (Class-IL)	ogbg-molhiv (Domain-IL)	NYC-Taxi (Time-IL)
Bare	0.026±0.023	0.550±0.066	0.131±0.034	-0.011±0.002	0.114±0.087	0.722±0.047	0.339±0.056	0.161±0.056	0.270±0.082	0.978±0.008	0.040±0.034	0.056±0.009
LwF	0.012±0.019	0.539±0.060	0.054±0.026	-0.015±0.002	0.035±0.021	0.726±0.047	0.300±0.079	0.179±0.040	**0.030±0.015**	0.976±0.009	0.027±0.023	0.022±0.012
EWC	0.020±0.014	0.542±0.065	0.074±0.025	-0.013±0.004	0.063±0.046	0.724±0.060	0.349±0.058	0.125±0.020	0.055±0.027	0.977±0.008	0.015±0.018	0.063±0.011
MAS	**0.005±0.007**	**0.283±0.050**	0.012±0.027	-0.011±0.002	0.029±0.025	0.726±0.049	0.203±0.034	**0.071±0.024**	0.077±0.039	0.973±0.008	**0.007±0.017**	0.045±0.016
GEM	0.060±0.057	0.507±0.053	**0.003±0.028**	**-0.027±0.002**	**0.026±0.021**	**0.579±0.100**	**0.130±0.121**	0.182±0.033	0.106±0.023	**0.866±0.080**	0.044±0.046	**0.006±0.027**
TWP	0.025±0.018	0.545±0.063	O.O.M	-0.013±0.002	0.070±0.059	0.721±0.055	0.353±0.055	0.145±0.036	0.056±0.030	0.972±0.009	0.017±0.017	0.047±0.010
ERGNN	0.052±0.059	0.518±0.058	N/A	-0.020±0.004	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CGNN	0.023±0.016	0.384±0.055	N/A	N/A	**-0.029±0.003**	N/A	N/A	N/A	N/A	N/A	N/A	N/A

were designed for node-level problems, were not applied to link- and graph-level problems. For the baseline methods without CL techniques, we used the Bare and Joint models used in (Zhang et al., 2022). The Bare model follows the incremental learning schemes, but no CL technique is applied to the model. Lastly, the Joint model trains the backbone model directly on the entire dataset, ignoring the CL procure with a sequence of tasks.

5.2 Average Performance & Average Forgetting

Our benchmark results in terms of final[3] AP and AF are shown in Table 3 and 4, respectively. In Task-IL, the replay-based methods show poor results in general. Specifically, the average rankings of GEM and ERGNN are 5.5 and 6 in terms of AP, and 6 and 6 in terms of AF, respectively. We compute the average rankings over all problems considered for the results in the tables. Interestingly, regularization-based methods are highly-ranked, outperforming CGNN, which combines replay- and regularization-based methods. For node-level problems in Class-IL, most of the graph CL methods perform better than the Bare model, and CGNN achieves the best performance in both terms of both AP and AF. However, the graph-level problems in Class-IL are challenging. We see that the most of graph CL methods show poor results in terms of both AP and AF. In Domain-IL, the replay-based method (i.e., GEM) outperforms the regularization-based methods. Specifically, on average, GEM is the best and the second-best among the five methods in terms of AP and AF, respectively. In Time-IL, only LwF outperforms the Bare model on all datasets in terms of AP. That is, it is challenging for current graph CL methods to deal with dynamic graphs evolving over time.

We also provide and discuss the benchmark results in terms of INT and FWT in Appendix A.2.

6 Conclusion

In this work, we define four incremental settings for evaluating continual learning methods for graph data (graph CL) by identifying and decoupling four possible dimensions of changes, which are tasks, classes, domains, and time. Then, we apply the settings to node-, link-, and graph-level learning problems, and as a result, we provide 23 benchmark scenarios from 14 real-world datasets, which cover 12 combinations of the four incremental settings and the three levels of problems. In addition, we propose BeGin, a fool-proof and easy-to-use benchmark framework for implementation and evaluation of graph CL methods. We support numerically that our benchmark and framework are more extensive, rigorous, and usable than the previous ones for the same purpose. For reproducibility, we provide all source code required for reproducing the benchmark results and documents at **https://anonymous.4open.science/r/BeGin-1C33/**.

[3] In the equations in Section 4.2, k is equal to N

REFERENCES

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *CVPR*, 2017.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018.

Jinheon Baek, Dong Bok Lee, and Sung Ju Hwang. Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction. In *NeurIPS*, 2020.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. *arXiv preprint arXiv:2103.11750*, 2021.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, 2018.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Beyza Ermis, Giovanni Zappella, Martin Wistuba, and Cedric Archambeau. Memory efficient continual learning for neural text classification. *arXiv preprint arXiv:2203.04640*, 2022.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

Falih Gozi Febrinanto, Feng Xia, Kristen Moore, Chandra Thapa, and Charu Aggarwal. Graph lifelong learning: A survey. *arXiv preprint arXiv:2202.10688*, 2022.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Lukas Galke, Benedikt Franke, Tobias Zielke, and Ansgar Scherp. Lifelong learning of graph neural networks for open-world node classification. In *IJCNN*, 2021.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Yujiang He and Bernhard Sick. Clear: An adaptive continual learning framework for regression tasks. *AI Perspectives*, 3(1):1–16, 2021.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: datasets for machine learning on graphs. In *NeurIPS*, 2020.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *ICDM*, 2016.

Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*, 2018.

Greg Landrum et al. Rdkit: Open-source cheminformatics. 2006, 2006.

Haoyang Li, Xin Wang, Ziwei Zhang, and Wenwu Zhu. Ood-gnn: Out-of-distribution generalized graph neural network. *TKDE*, 2022.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The clear benchmark: Continual learning on real-world imagery. In *NeurIPS*, 2021.

Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *AAAI*, 2021.

Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *CORL*, 2017.

Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M Van de Ven, et al. Avalanche: an end-to-end library for continual learning. In *ICCV*, 2021.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Quang Pham, Chenghao Liu, and Steven Hoi. Dualnet: Continual learning, fast and slow. In *NeurIPS*, 2021.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *NeurIPS*, 2019.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017.

Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

Jie Tang, Tiancheng Lou, Jon Kleinberg, and Sen Wu. Transfer learning to infer social ties across heterogeneous networks. *ACM Transactions on Information Systems*, 34(2):1–43, 2016.

Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. Lifelong graph learning. In *CVPR*, 2022.

Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. Streaming graph neural networks via continual learning. In *CIKM*, 2020a.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020b.

Qitian Wu, Chenxiao Yang, and Junchi Yan. Towards open-world feature extrapolation: An inductive graph learning approach. In *NeurIPS*, 2021.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Feisheng Zhong, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, et al. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of medicinal chemistry*, 63(16):8749–8760, 2019.

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2358–2366, 2022.

Xikun Zhang, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. In *NeurIPS*, 2022.

Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *AAAI*, 2021.

Table 5: **Performance in terms of Average Performance (AP, the higher, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory. N/A: methods are not applicable to the problems or scenarios.

Methods	Cora (Task-IL)	Citeseer (Task-IL)	ogbn-arxiv (Task-IL)	CoraFull (Task-IL)	Cora (Class-IL)	Citeseer (Class-IL)	ogbn-arxiv (Class-IL)	ogbn-products (Class-IL)	ogbn-proteins (Domain-IL)	ogbn-arxiv (Time-IL)
Bare	0.903±0.018	0.836±0.029	0.650±0.076	0.759±0.019	0.541±0.036	0.447±0.040	0.120±0.004	0.105±0.008	0.690±0.022	0.687±0.002
LwF	0.915±0.012	**0.859±0.018**	**0.926±0.008**	O.O.M	0.550±0.021	0.464±0.039	0.131±0.009	0.106±0.010	0.714±0.020	0.696±0.002
EWC	0.912±0.013	0.837±0.029	0.858±0.022	0.869±0.022	0.567±0.052	0.452±0.037	0.123±0.004	0.117±0.018	0.761±0.011	0.689±0.002
MAS	**0.918±0.017**	0.843±0.023	0.918±0.008	**0.972±0.006**	**0.741±0.014**	**0.560±0.024**	0.125±0.017	0.096±0.018	0.694±0.016	0.664±0.003
GEM	0.882±0.031	0.834±0.029	0.906±0.007	0.836±0.020	0.618±0.029	0.482±0.033	**0.607±0.015**	0.250±0.040	**0.810±0.003**	0.673±0.003
TWP	0.910±0.015	0.836±0.026	0.848±0.015	0.900±0.018	0.564±0.031	0.450±0.037	0.123±0.005	0.108±0.014	O.O.M	0.682±0.001
ERGNN	0.890±0.031	0.829±0.021	0.876±0.015	0.900±0.024	0.609±0.026	0.457±0.043	0.541±0.014	**0.410±0.078**	N/A	0.666±0.008
CGNN	0.911±0.015	0.829±0.032	0.920±0.007	0.910±0.016	0.723±0.017	0.531±0.035	0.477±0.050	O.O.M	N/A	**0.710±0.002**
Joint	0.924±0.015	0.851±0.027	0.938±0.006	0.977±0.005	0.800±0.020	0.556±0.040	0.654±0.016	0.681±0.062	0.732±0.002	0.734±0.002

(a) Node Classification (NC)

Methods	Bitcoin-OTC (Task-IL)	Bitcoin-OTC (Class-IL)	Bitcoin-OTC (Time-IL)	Wiki-CS (Domain-IL)	ogbl-collab (Time-IL)
Bare	0.648±0.071	0.243±0.030	0.696±0.019	0.125±0.081	0.449±0.047
LwF	0.704±0.032	0.242±0.029	0.680±0.021	0.136±0.037	0.475±0.036
EWC	0.682±0.046	0.242±0.030	**0.702±0.021**	0.138±0.038	0.425±0.024
MAS	**0.706±0.033**	0.244±0.023	0.687±0.017	0.152±0.042	0.259±0.025
GEM	0.700±0.035	**0.287±0.042**	0.681±0.019	**0.223±0.114**	**0.484±0.030**
TWP	0.673±0.046	0.243±0.028	0.664±0.014	0.134±0.045	0.422±0.039
Joint	0.735±0.035	0.377±0.031	0.800±0.012	0.393±0.026	0.607±0.018

(b) Link Classification (LC) and Link Prediction (LP)

Methods	MNIST (Task-IL)	CIFAR10 (Task-IL)	Aromaticity (Task-IL)	MNIST (Class-IL)	CIFAR10 (Class-IL)	Aromaticity (Class-IL)	ogbg-molhiv (Domain-IL)	NYC-Taxi (Time-IL)
Bare	0.691±0.081	0.646±0.074	0.458±0.044	0.194±0.005	0.175±0.008	0.062±0.014	0.735±0.031	0.804±0.006
LwF	**0.965±0.008**	**0.840±0.030**	0.549±0.044	0.194±0.005	**0.176±0.009**	0.064±0.013	0.759±0.018	**0.820±0.012**
EWC	0.888±0.047	0.784±0.042	**0.588±0.030**	0.193±0.006	0.174±0.009	0.064±0.012	**0.766±0.013**	0.785±0.020
MAS	0.843±0.044	0.762±0.046	0.577±0.048	0.192±0.006	0.175±0.009	0.058±0.011	0.759±0.021	0.764±0.016
GEM	0.894±0.028	0.769±0.030	0.500±0.045	**0.199±0.020**	0.174±0.010	**0.065±0.013**	0.737±0.039	0.776±0.025
TWP	0.883±0.060	0.788±0.044	0.564±0.063	0.190±0.007	0.167±0.012	0.062±0.009	0.764±0.012	0.769±0.008
Joint	0.977±0.007	0.868±0.021	0.765±0.029	0.900±0.004	0.521±0.003	0.286±0.018	0.806±0.011	0.866±0.006

(c) Graph Classification (GC)

A Full Experiment Results

A.1 Average Performance & Average Forgetting

In Table 5 and Table 6, we report the results of average performance (AP) and average forgetting (AF) for all 23 benchmark scenarios.

A.2 Intransigence & Forward Transfer

We report the results of Intransigence (INT) and Forward transfer (FWT) in Table 7 and 8, respectively. In terms of INT, we can see that many methods are outperformed by the Bare model, which only focuses on learning the current task.

In terms of FWT, EWC performs best for NC and GC problems, and TWP performs best for LP problems. Note that we are able to compute FWT only in Domain-IL since we cannot infer classes among $\mathcal{C}_i$ after learning on $\mathcal{T}_{(i-1)}$ in Task-IL and Class-IL, and we cannot get the i-th task (spec., $\mathcal{G}^{(i)}$ and $\mathcal{S}^{(i)}$) in advance in Time-IL.

B Hyperparameter Settings

We performed a grid search to find the best hyperparameter settings for the backbone model (e.g., the learning rate and the dropout ratio) for each method. Specifically, we chose the setting where AP on the validation dataset was maximized and reported the results on the test dataset in the selected setting. For evaluation, we conducted ten experiments with different random seeds and averaged the performance over the ten trials. For the CL methods except for TWP, we set the initial learning rate

Table 6: **Performance in terms of Average Forgetting (AF, the lower, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory. N/A: methods are not applicable to the problems or scenarios.

Methods	Cora (Task-IL)	Citeseer (Task-IL)	ogbn-arxiv (Task-IL)	CoraFull (Task-IL)	Cora (Class-IL)	Citeseer (Class-IL)	ogbn-arxiv (Class-IL)	ogbn-products (Class-IL)	ogbn-proteins (Domain-IL)	ogbn-arxiv (Time-IL)
Bare	0.026±0.023	0.042±0.030	0.325±0.085	0.226±0.020	0.565±0.039	0.550±0.066	0.934±0.008	0.850±0.047	0.131±0.034	-0.011±0.002
LwF	0.012±0.019	**0.011±0.017**	0.010±0.004	O.O.M	0.534±0.025	0.539±0.060	0.917±0.010	0.845±0.052	0.054±0.026	-0.015±0.002
EWC	0.020±0.014	0.041±0.033	0.083±0.024	0.114±0.024	0.508±0.087	0.542±0.065	0.924±0.006	0.828±0.071	0.074±0.025	-0.013±0.004
MAS	**0.005±0.007**	0.021±0.013	**0.006±0.004**	**0.001±0.003**	**0.227±0.039**	**0.283±0.050**	0.894±0.044	0.746±0.066	0.012±0.027	-0.011±0.002
GEM	0.060±0.057	0.041±0.026	0.036±0.005	0.147±0.021	0.444±0.046	0.507±0.053	0.228±0.030	0.545±0.081	**0.003±0.028**	-0.027±0.002
TWP	0.025±0.018	0.038±0.024	0.094±0.013	0.082±0.019	0.524±0.041	0.545±0.063	0.924±0.009	0.829±0.062	O.O.M	-0.013±0.002
ERGNN	0.052±0.059	0.050±0.024	0.063±0.012	0.078±0.022	0.447±0.045	0.518±0.058	**-0.023±0.071**	**0.399±0.133**	N/A	-0.020±0.004
CGNN	0.023±0.016	0.050±0.034	0.016±0.003	0.065±0.014	0.251±0.034	0.384±0.055	0.437±0.064	O.O.M	N/A	**-0.029±0.003**

(a) Node Classification (NC)

Methods	Bitcoin-OTC (Task-IL)	Bitcoin-OTC (Class-IL)	Bitcoin-OTC (Time-IL)	Wiki-CS (Domain-IL)	ogbl-collab (Time-IL)
Bare	0.114±0.087	0.722±0.047	0.118±0.027	0.339±0.056	0.161±0.056
LwF	0.035±0.021	0.726±0.047	0.092±0.042	0.300±0.079	0.179±0.040
EWC	0.063±0.046	0.724±0.060	0.099±0.021	0.349±0.058	0.125±0.020
MAS	0.029±0.025	0.726±0.049	0.115±0.025	0.203±0.034	**0.071±0.024**
GEM	**0.026±0.021**	**0.579±0.100**	**0.042±0.033**	**0.130±0.121**	0.182±0.033
TWP	0.070±0.059	0.721±0.055	0.193±0.018	0.353±0.055	0.145±0.036

(b) Link Classification (LC) and Link Prediction (LP)

Methods	MNIST (Task-IL)	CIFAR10 (Task-IL)	Aromaticity (Task-IL)	MNIST (Class-IL)	CIFAR10 (Class-IL)	Aromaticity (Class-IL)	ogbg-molhiv (Domain-IL)	NYC-Taxi (Time-IL)
Bare	0.356±0.099	0.270±0.082	0.182±0.051	0.978±0.008	0.856±0.021	0.694±0.026	0.040±0.034	0.056±0.009
LwF	**0.012±0.004**	**0.030±0.015**	0.164±0.046	0.976±0.009	0.857±0.023	0.695±0.026	0.027±0.023	0.022±0.012
EWC	0.084±0.047	0.055±0.027	0.087±0.030	0.977±0.008	0.855±0.022	0.674±0.034	0.015±0.018	0.063±0.011
MAS	0.103±0.048	0.077±0.039	**0.082±0.031**	0.973±0.008	0.849±0.026	0.647±0.040	**0.007±0.017**	0.045±0.016
GEM	0.090±0.029	0.106±0.023	0.169±0.032	**0.866±0.080**	0.843±0.022	**0.337±0.096**	0.044±0.046	**0.006±0.027**
TWP	0.096±0.062	0.056±0.030	0.107±0.077	0.972±0.009	**0.842±0.026**	0.678±0.034	0.017±0.017	0.047±0.010

(c) Graph Classification (GC)

among {1e-3, 5e-3, 1e-3}, the dropout ratio among {0, 0.25, 0.5}, and the weight decay coefficient among {0, 5e-4}. For TWP, we set the initial learning rate among {1e-3, 5e-3}, the dropout ratio among {0, 0.25, 0.5}, and the weight decay coefficient to 0. For the experiments on ogbn-proteins and ogbn-products, we fixed the learning rate to 1e-3. For Citeseer, we additionally considered the learning rate 5e-4. For ogbn-products, we set the dropout ratio to $\{0, 0.25\}$, and we restricted the number of maximum neighbors to receive messages to 5, 10, and 10 on the first layer, the second layer, and the third layer, respectively, by using the sampler.

For replay-based methods (e.g., GEM, ERGNN, and CGNN), we set the maximum size of memory the same for a fair comparison. Specifically, we set it to 12 for (a) all experiments on Cora and Citeseer, (b) 210 for CoraFull, (c) $2,000$ for ogbn-arxiv and ogbn-proteins, (d) $25,000$ for ogbn-products, (e) 500 for MNIST, CIFAR10, and ogbg-molhiv, (f) 50 for Aromaticity, (g) 180 for NYC-Taxi, (h) $4,000$ for Wiki-CS, and (i) $20,000$ for ogbl-collab. Following the original papers, we set the margin for quadratic programming to 0.5 for GEM, and we used the Coverage Maximization (CM) sampler and set the distance threshold to 0.5 for ERGNN.

For regularization methods (e.g., LwF, EWC, MAS, TWP, and CGNN), we chose the regularization coefficient λ among $\{0.1, 1.0\}$, $\{100, 10000\}$, $\{0.1, 1.0\}$, and $\{80\}$ for LwF, EWC, MAS, and CGNN, respectively. For TWP, we set beta to 0.01 and λ_l to $10,000$, and we chose λ_t among $\{100, 1000\}$. Note that CGNN combines regularization- and replay-based approaches, and thus we need to consider both the maximum size of memory and the regularization coefficients.

C Effects of the number of total tasks

We conducted additional experiments to investigate the effect of the number of total tasks N on the performance of graph CL methods. Accordingly, we changed the number of classes considered (additionally) in each task proportionally to the reciprocal of the number of tasks. Under Task- and Class-IL settings on ogbn-arxiv, we measured how the performance changes depending on the

Table 7: **Performance in terms of Intransigence (INT, the lower, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory. N/A: methods are not applicable to the problems or scenarios.

Methods	Cora (Task-IL)	Citeseer (Task-IL)	ogbn-arxiv (Task-IL)	CoraFull (Task-IL)	Cora (Class-IL)	Citeseer (Class-IL)	ogbn-arxiv (Class-IL)	ogbn-products (Class-IL)	ogbn-proteins (Domain-IL)	ogbn-arxiv (Time-IL)
Bare	0.001±0.013	-0.023±0.052	0.004±0.001	**-0.002±0.003**	**-0.064±0.018**	-0.129±0.053	**-0.168±0.031**	**-0.122±0.041**	0.012±0.001	-0.067±0.006
LwF	-0.001±0.005	**-0.026±0.052**	0.004±0.002	O.O.M	-0.052±0.020	**-0.139±0.052**	-0.164±0.030	-0.120±0.043	0.007±0.001	-0.023±0.021
EWC	-0.005±0.011	-0.024±0.054	0.008±0.002	**-0.002±0.002**	-0.052±0.023	-0.129±0.052	-0.163±0.031	-0.115±0.044	0.012±0.002	**-0.089±0.009**
MAS	-0.001±0.010	-0.016±0.050	0.014±0.002	0.004±0.004	-0.038±0.028	-0.064±0.050	-0.142±0.031	-0.021±0.056	0.035±0.002	0.034±0.026
GEM	-0.001±0.007	-0.021±0.051	**0.001±0.001**	**-0.002±0.003**	-0.060±0.018	-0.135±0.053	-0.037±0.020	0.003±0.052	0.041±0.002	-0.083±0.025
TWP	**-0.006±0.009**	-0.022±0.053	0.008±0.001	0.001±0.002	-0.059±0.021	-0.128±0.053	-0.162±0.030	-0.104±0.048	0.019±0.002	O.O.M
ERGNN	-0.003±0.011	-0.022±0.053	0.007±0.002	0.002±0.003	-0.053±0.018	-0.118±0.056	0.249±0.059	-0.027±0.055	0.041±0.005	N/A
CGNN	-0.005±0.009	-0.022±0.052	0.004±0.001	0.003±0.003	-0.037±0.011	-0.102±0.051	-0.090±0.024	O.O.M	**0.006±0.002**	N/A

(a) Node Classification (NC)

Methods	Bitcoin-OTC (Task-IL)	Bitcoin-OTC (Class-IL)	Bitcoin-OTC (Time-IL)	Wiki-CS (Domain-IL)	ogbl-collab (Time-IL)
Bare	0.005±0.014	-0.212±0.067	0.003±0.017	-0.053±0.040	0.049±0.018
LwF	**0.001±0.016**	-0.213±0.064	0.041±0.029	-0.029±0.055	0.008±0.015
EWC	0.004±0.010	-0.212±0.066	0.014±0.021	**-0.076±0.069**	0.105±0.022
MAS	0.003±0.018	**-0.215±0.070**	0.014±0.016	0.043±0.056	0.319±0.036
GEM	0.011±0.013	-0.160±0.080	0.083±0.024	0.037±0.047	**-0.005±0.014**
TWP	0.009±0.019	-0.210±0.060	**-0.030±0.011**	-0.075±0.072	0.090±0.019

(b) Link Classification (LC) and Link Prediction (LP)

Methods	MNIST (Task-IL)	CIFAR10 (Task-IL)	Aromaticity (Task-IL)	MNIST (Class-IL)	CIFAR10 (Class-IL)	Aromaticity (Class-IL)	ogbg-molhiv (Domain-IL)	NYC-Taxi (Time-IL)
Bare	**0.000±0.003**	0.004±0.003	0.106±0.034	**-0.045±0.005**	-0.203±0.016	-0.290±0.040	0.027±0.016	**-0.020±0.014**
LwF	0.001±0.001	**0.002±0.004**	**0.031±0.026**	-0.043±0.006	**-0.204±0.017**	**-0.293±0.049**	**0.015±0.016**	-0.004±0.010
EWC	0.021±0.006	0.038±0.012	0.056±0.036	-0.042±0.006	-0.201±0.017	-0.274±0.050	0.017±0.016	-0.007±0.015
MAS	0.051±0.011	0.043±0.013	0.063±0.029	-0.039±0.006	-0.197±0.018	-0.243±0.052	0.033±0.011	0.030±0.007
GEM	0.011±0.005	0.013±0.005	0.075±0.028	0.040±0.052	-0.191±0.020	0.028±0.090	0.021±0.014	0.054±0.010
TWP	0.016±0.004	0.033±0.009	0.067±0.024	-0.036±0.007	-0.184±0.022	-0.275±0.052	0.018±0.013	0.025±0.018

(c) Graph Classification (GC)

Table 8: **Performance in terms of Forward Transfer (FWT, the higher, the better)**. In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory.

Methods	ogbn-proteins (NC)	Domain-IL Wiki-CS (LP)	ogbg-molhiv (GC)
Bare	0.159±0.024	0.022±0.018	0.183±0.041
LwF	0.164±0.026	0.015±0.015	0.197±0.040
EWC	**0.171±0.029**	0.019±0.020	**0.211±0.037**
MAS	0.165±0.033	0.012±0.018	0.208±0.039
GEM	0.166±0.033	0.013±0.013	0.193±0.042
TWP	O.O.M	**0.023±0.018**	0.199±0.039

number of total tasks from 5 to 20. In Table 9, we report the benchmark final (i.e., when k is equal to N) Performance in terms of AP, AF, and INT, respectively.

Under Task-IL, most graph CL methods perform better as the number of tasks increases in terms of all 3 metrics, due to the decrease in the number of tasks considered in each task. On the contrary, under Class-IL, the performance tends to degrade in terms of AP and AF as the number of tasks increases since the distribution shift, which each model needs to adapt to, occurs more frequently. Interestingly, we find that INT tends to change positively as the number of tasks increases because INT does not consider the forgetting of previous tasks.

D Performance Curve (Effects of the number of learned tasks)

In Figure 4, we report the performance curves, which show how the average performance (AP) (when k is equal to n) changes depending on the number of learned tasks n. Note that most considered methods, including the Joint model, tend to perform worse, as the number of learned tasks increases, since they need to retain more previous knowledge. However, opposite trends are observed on ogbn-arxiv, ogbg-molhiv, and NYC-Taxi, which we guess have slighter or easier-to-adapt distribution shifts. Also, on Bitcoin-OTC under Task-IL, the performance of the Joint model and that of graph CL methods show different tendencies. We provide the per-

Table 9: **Effects of the number of total tasks on Average Performance, Average Forgetting, and Intransigence.** In each setting, the best score is in bold, and the second best score is underlined. O.O.M: out of memory.

Methods	ogbn-arxiv (Task-IL)			ogbn-arxiv (Class-IL)		
	$N = 5$	$N = 10$	$N = 20$	$N = 5$	$N = 10$	$N = 20$
Bare	0.649±0.065	0.650±0.076	0.813±0.035	0.185±0.010	0.120±0.004	0.053±0.007
LwF	**0.895±0.016**	**0.926±0.008**	O.O.M	0.199±0.013	0.131±0.009	0.057±0.009
EWC	0.794±0.036	0.858±0.022	0.934±0.009	0.188±0.008	0.123±0.004	0.059±0.012
MAS	0.882±0.017	0.918±0.008	**0.964±0.009**	0.187±0.014	0.125±0.017	0.048±0.007
GEM	0.873±0.018	0.906±0.007	0.959±0.009	**0.626±0.021**	**0.607±0.015**	**0.551±0.011**
TWP	0.792±0.040	0.848±0.015	0.932±0.017	0.189±0.010	0.123±0.005	0.059±0.010
ERGNN	0.813±0.022	0.876±0.015	0.949±0.010	0.538±0.023	0.541±0.014	0.441±0.088
CGNN	0.894±0.015	0.920±0.007	**0.964±0.009**	0.533±0.040	0.477±0.050	0.395±0.046
Joint	0.906±0.015	0.938±0.006	0.974±0.007	0.681±0.023	0.654±0.016	0.578±0.016

(a) Average Performance (AP, the higher, the better)

Methods	ogbn-arxiv (Task-IL)			ogbn-arxiv (Class-IL)		
	$N = 5$	$N = 10$	$N = 20$	$N = 5$	$N = 10$	$N = 20$
Bare	0.317±0.074	0.325±0.085	0.169±0.034	0.898±0.017	0.934±0.008	0.969±0.013
LwF	0.010±0.004	0.010±0.004	O.O.M	0.879±0.016	0.917±0.010	0.963±0.012
EWC	0.133±0.037	0.083±0.024	0.041±0.011	0.892±0.018	0.924±0.006	0.961±0.014
MAS	**0.000±0.001**	**0.006±0.004**	**0.000±0.001**	0.866±0.016	0.894±0.044	0.584±0.091
GEM	0.041±0.008	0.036±0.005	0.016±0.005	0.220±0.033	0.228±0.030	**0.258±0.018**
TWP	0.131±0.040	0.094±0.013	0.040±0.016	0.890±0.021	0.924±0.009	0.959±0.013
ERGNN	0.088±0.024	0.063±0.012	0.021±0.005	**-0.138±0.086**	**-0.023±0.071**	0.349±0.095
CGNN	0.016±0.003	0.016±0.003	0.011±0.004	0.367±0.061	0.437±0.064	0.511±0.052

(b) Average Forgetting (AF, the lower, the better)

Methods	ogbn-arxiv (Task-IL)			ogbn-arxiv (Class-IL)		
	$N = 5$	$N = 10$	$N = 20$	$N = 5$	$N = 10$	$N = 20$
Bare	0.005±0.001	0.004±0.001	0.000±0.001	**-0.127±0.034**	**-0.168±0.031**	**-0.251±0.034**
LwF	0.005±0.001	0.004±0.002	O.O.M	**-0.127±0.034**	-0.164±0.030	-0.249±0.034
EWC	0.005±0.001	0.008±0.002	0.000±0.002	-0.126±0.034	-0.163±0.031	-0.249±0.034
MAS	0.025±0.003	0.014±0.002	0.009±0.002	-0.105±0.033	-0.142±0.031	0.120±0.072
GEM	0.001±0.001	**0.001±0.001**	0.000±0.002	-0.026±0.018	-0.037±0.020	-0.073±0.027
TWP	0.009±0.003	0.008±0.001	0.003±0.002	-0.125±0.034	-0.162±0.030	-0.248±0.035
ERGNN	0.016±0.004	0.007±0.002	0.004±0.002	0.347±0.073	0.249±0.059	-0.050±0.057
CGNN	**0.000±0.001**	0.004±0.001	**-0.001±0.002**	-0.051±0.015	-0.090±0.024	-0.158±0.026

(c) Intransigence (INT, the lower, the better)

Table 10: **Lines of Python code to implement each algorithm (the lower, the better).** On average, with BEGIN, we can implement the same graph CL method with about 30% fewer lines of code. For ERGNN, we only considered the code for node classification with the CM sampler.

Method	LwF	EWC	MAS	GEM	TWP	ERGNN	Average
BEGIN (ours)	**130**	**176**	**162**	**267**	**272**	**129**	**189**
(Zhang et al., 2022)	171	237	251	327	299	144	238

formance curves under all considered scenarios in the online appendix, which is available at **https://anonymous.4open.science/r/BeGin-1C33/**.

E USABILITY OF BEGIN

In Table 10, we report the lines of Python code for the CL methods commonly implemented in BEGIN and CGLB (Zhang et al., 2022). For each CL method, we calculated the sum of the lines of the Python code for node classification and graph classification (if available) in Task-IL and Class-IL settings. When counting the number of lines, we ignored comments, empty lines, and unnecessary imports in the codes. It should be noticed that, with BEGIN, we can implement the same graph CL method with about 30% fewer lines of code, on average.

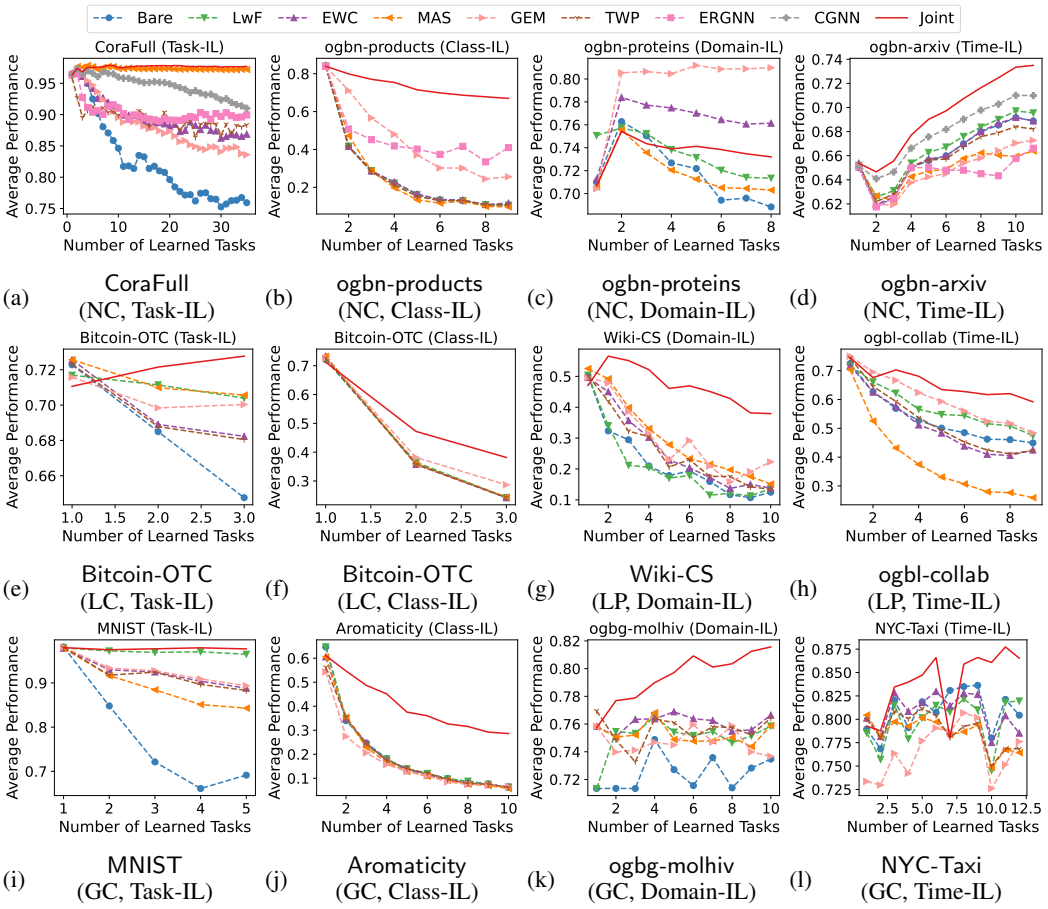

Figure 4: **Change of Average Performance (AP) during continual learning.** The full results are available at https://anonymous.4open.science/r/BeGin-1C33/.