Appendix: CGLB: Benchmark Tasks for Continual Graph Learning

Xikun Zhang
The University of Sydney
xzha0505@uni.sydney.edu.au

Dongjin Song
University of Connecticut
dongjin.song@uconn.edu

Dacheng Tao
The University of Sydney
dacheng.tao@gmail.com

1 Additional Details on the Experimental Settings and Benchmark Construction

In this section, we give further details on the experimental settings and details on the construction of CGLB.

1.1 Additional Details on the Experimental Settings

Additional experimental setting details include the indices of the deleted classes for each dataset, the used devices, the train-validation-test splitting for each task, and details on training and validation. For the N-CGL datasets, we remove the 41-th class of Reddit-CL, and the 47-th class of Products-CL. This aims to ensure an even number of classes for each dataset to be divided into a sequence of 2-class tasks. Moreover, the 47-th class of Products-CL contains only one node, and cannot be split for training, validation, and test. For the G-CGL datasets, classes removed from Aromaticity-CL are {2, 3, 4, 8, 35, 36, 37, 38, 39, 40, 41} since they contain less than 20 examples and are causing difficulties for model training. The other 30 classes of Aromaticity-CL are kept and constructed as 15 tasks. No classes are removed from SIDER-tIL and Tox21-tIL. The statistics in Tables 1 and 2 of the paper are obtained after the class removal. For the N-CGL datasets, the train-validation-test splitting ratios are 60%, 20%, and 20%, and transductive setting is adopted for each task. The train-validation-test splitting is obtained by random sampling, therefore the performance may be slightly different with splittings from different rounds of random sampling. We provide the set of splitting used in our experiments on our GitHub page as a reference. For the G-CGL datasets, the ratios are 80%, 10%, and 10%. In our code, we provide a framework for conveniently conduct grid-search over candidate hyper-parameters and find the best performing combination on the validation set. Then the selected model is also automatically evaluated on the testing set. Details on the usage can be found in our GitHub page. In our experiments, the hyper-parameter candidates we used are summarized in Table 1. The name of the hyper-parameters are consistent with the names in our code. Details of the parameters could be found in the original papers. Among the hyper-parameters of all the methods, the memory budget of the memory based methods (GEM and ERGNN) should be cared. Generally, larger memory budget leads to better performance. However, larger budget also consumes more space, which may not be practical in real-world applications. Therefore, unlike the other hyper-parameters that can be determined by validation, the memory budget is also determined by the available space in practical scenarios. Besides, the dataset splittings and the method hyper-parameters, the batch size and the number of training epochs are also factors influencing the continual learning performance. Smaller batch sizes or larger number of training epochs lead to more iterations to train the model, and the model is more adapted to a given task (more forgetting on previous tasks). Therefore, keeping a consistent batch size is an important factor when making comparisons across different models. In

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks.

NCGL experiments, we use full batch for the small NCGL datasets (CoraFull-CL and Arxiv-CL) and the GCGL datasets, and use 2,000 as the batch size for learning on large NCGL datasets (Reddit-CL and Products-CL). For GCGL experiments, we set the batch size as 128, and the number of training epochs as 100. For the two multi-label classification datasets (SIDER-tIL and Tox21-tIL), early stopping is applied to ensure a stable performance. All experiments are repeated 5 times on one Nvidia Titan Xp GPU.

Table 1: Hyper-parameter candidates used for grid search.

EWC [4]	memory_strength: [1, 100, 10000, 1000000]
MAS [1]	memory_strength: [1, 100, 10000, 1000000]
GEM [8]	memory_strength: [0.05,0.5,5]; n_memories: [10, 100,1000]
TWP [7]	lambda_l: [100,10000] ; lambda_t: [100,10000]; beta: [0.01,0.1]
LwF [6]	lambda_dist: [0.1,1.,10.] ; T: [0.2,2,20]
ER-GNN [12]	budget: [10,100]; d: [0.05, 0.5, 5.0]; sampler: [CM]

1.2 Additional Explanations on the Evaluation Metrics

In this subsection, we explain the evaluation metrics in details. The explanations will cover every step in the computation of AP and AF from the original performance matrix, so that the rationale behind the AP, AF and the performance matrix will be clarified.

The rationale behind AP and AF is to measure a model's average performance/forgetting on all learnt tasks after learning a sequence of tasks. In the following, we will separately explain AP and AF with details and examples.

In a performance matrix M^p, an entry $M^p_{i,j}$ denotes the model performance on a learnt task j ($j \leqslant i$) after learning the i-th task (i.e., the model has been trained over the sequence of tasks from 1 to i). Accordingly, the i-th row of the matrix M^p, i.e. $M^p_{i,j}, j = 1, ..., i$ reflects the performance on each learnt task j ($j = 1, ..., i$) after learning the i-th task.

Therefore, the average performance (AP) after learning the i-th task is defined as the average of the i-th row of the matrix ($M^p_{i,j}, j = 1, ..., i$), because it reflects the model's average performance over all learnt tasks after learning a sequence of tasks (i.e., from the 1-st task to the i-th task). As we could see, an AP can be computed after learning each new task, and the sequence of APs denotes how the overall performance varies with new tasks constantly coming in (i.e., the learning dynamics curves shown in Figure 3 of the paper).

When a single numerical value is required to denote the model's overall performance (e.g., the results shown in Table 3, 4, and 5), the AP computed after learning the entire sequence of tasks (i.e., $\frac{\sum_{j=1}^{T} \mathrm{M}^p_{T,j}}{T}$ computed over the last row of the performance matrix, $M^p_{T,j}, j = 1, ..., T$) is used, which is the average model performance over all learnt tasks after the model has been trained over the sequence from the 1-st task to the final task. A high AP denotes that the model's overall performance on previous tasks is good after learning all the tasks sequentially (i.e., little forgetting issue). While a lower AP denotes that the model has more severe forgetting problem.

The average forgetting (AF) is supported by the same rationale. Specifically, a diagonal entry $M^p_{j,j}$ denotes the performance of a model when it has just learnt the task j (i.e., before the performance on task j is degraded because of the forgetting issue). The model will then keep learning the following tasks $j + 1$, $j + 2$, ..., etc.. After learning each following new task, the model is tested again on task j, and the performance on task j becomes $M^p_{j+1,j}$, $M^p_{j+2,j}$, ..., etc.. At a specific step i, when the model has just learnt task i ($i > j$), the performance on task j becomes $M^p_{i,j}$. Due to the forgetting issue (learning new tasks may interfere with the performance on task j), $M^p_{i,j}$ may be lower than $M^p_{j,j}$, and $M^p_{i,j} - M^p_{j,j}$ can quantitatively measure the forgetting. Accordingly, negative $M^p_{i,j} - M^p_{j,j}$ denotes that the performance on task j is negatively affected (forgetting on task j) by the following tasks from $j + 1$ to i, and larger $|M^p_{j+1,j} - M^p_{j,j}|$ denotes more severe forgetting. It is also possible that $M^p_{j+1,j} - M^p_{j,j}$ is positive, denoting that the learning on the following tasks has a positive influence on task j, which is rare according to the experimental results. Similar to AP, after learning each new task, we could compute an AF over all learnt tasks, and the sequence of AFs reflects the learning dynamics from the forgetting perspective. To use a single numerical value to denote the average

forgetting over all learnt tasks after learning the final task T, we could use the AF computed after learning the T-th task, i.e. $\frac{\sum_{j=1}^{T-1} \mathrm{M}^{p}_{T,j} - \mathrm{M}^{p}_{j,j}}{T-1}$.

1.3 Class Imbalance Problem

The class imbalance problem is often severe in graph data, and the model may collapse and give trivial predictions biased to the dominating classes. Moreover, due to the topological connections among the data and the severeness of the imbalance problem, it may not be practical to balance the data by choosing equal number of nodes from each class. For example, the largest class in Products-CL has 668,950 nodes, and the smallest class has 1 node. In this situation, selecting a equal number of nodes from each class will either make it necessary to delete many classes without enough number of nodes or selecting a very small amount of nodes from each class to ensure that all classes have a same size. Besides, deleting graph nodes also change the original topology and the relations among the remaining nodes, and is better avoided.

Therefore, during training, we would re-scale the loss of the nodes according to the sizes of their corresponding classes. Suppose the set of all classes of a training set is $\mathcal{C}$, and the size of each class is $\{n_c \mid c \in \mathcal{C}\}$. Then, the loss calculated on a node set $\mathbb{V}$ without balancing can be formulated as,

$$\mathcal{L} = \sum_{v \in \mathbb{V}} l(f(v), y_v), \tag{1}$$

where $f(v)$ denotes the prediction of v, and y_v denotes the label of v. The balanced loss is then formulated as,

$$\mathcal{L} = \sum_{v \in \mathbb{V}} l(f(v), y_v) \cdot s_{y_v}, \tag{2}$$

where $s_{y_v} = \frac{n_c}{\sum_{i \in \mathcal{C}} n_i}$ when $y_v = c$.

In testing, the evaluation of the model also considers the class imbalance problem. We will first calculate the performance within each class, and then average the results over all classes. In this way, the classes of different sizes contribute equally to the performance.

1.4 Classifier for Class-IL Scenario

In a standard classification task, the number of the output heads (number of output logits) of a classifier equals the number of possible classes in the dataset, and is fixed before training. However, in class-IL scenario of CGL, since new classes come in constantly, the number of output heads cannot be set beforehand but has to continually increase.

1.5 Task Statistics

Besides the dataset statistics provided in paper, in this subsection, we provide detailed statistics for each task, i.e., the numbers of nodes/graphs/edges in each class for all GCGL tasks, and the numbers of nodes/edges in each class for all NCGL tasks. The statistics are shown Table 2,3,4,5,6,7,8.

2 License

Our datasets are curated from existing public data sources, and follow their licenses. For the N-CGL datasers, the OGB-Arxiv [3] [1] is licensed under Open Data Commons Attribution License (ODC-BY), and the OGB-Products [3] [2] is licensed under Amazon license. The CoraFull dataset [9] and the Reddit dataset [2] [3][4] are two datasets built from publicly available sources (public papers and Reddit posts) without license attached by the authors. For the G-CGL datasets, the SIDER dataset [5] is

[1] `https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv`

[2] `https://ogb.stanford.edu/docs/nodeprop/#ogbn-products`

[3] `https://archive.org/details/FullRedditSubmissionCorpus2006ThruAugust2015`

[4] `https://archive.org/details/2015_reddit_comments_corpus`

[5] `http://sideeffects.embl.de/`

class	0	1	2	3	4	5	6	7	8	9
# nodes	257	52	243	378	63	305	404	663	240	342
# edges	3066	976	3540	4404	966	2350	4968	12584	4488	2884
class	10	11	12	13	14	15	16	17	18	19
# nodes	141	223	102	521	341	138	115	111	80	435
# edges	2226	2892	1108	9078	3548	1468	1430	1762	416	3842
class	20	21	22	23	24	25	26	27	28	29
# nodes	420	254	414	196	334	315	284	783	113	466
# edges	6722	2040	4492	2328	4266	4266	2510	18676	1544	5606
class	30	31	32	33	34	35	36	37	38	39
# nodes	221	376	154	855	576	84	293	163	125	564
# edges	3300	4014	1708	8116	8554	592	5694	3530	1034	5490
class	40	41	42	43	44	45	46	47	48	49
# nodes	280	205	94	53	129	370	122	74	557	285
# edges	1982	2106	1240	472	1572	3716	1272	840	7384	3054
class	50	51	52	53	54	55	56	57	58	59
# nodes	72	625	501	650	99	473	324	928	212	301
# edges	1138	10772	5176	6834	750	5974	3218	8566	2488	6454
class	60	61	62	63	64	65	66	67	68	69
# nodes	116	220	165	291	147	91	137	84	15	29
# edges	2000	3434	1798	3882	1534	1068	1314	746	82	340

Table 2: Number of nodes and edges in each class of CoraFull-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	565	687	4839	2080	5862	4958	1618	589	6232	2820
# edges	4001	4697	26310	12439	46538	34558	7466	2447	41732	22296
class	10	11	12	13	14	15	16	17	18	19
# nodes	7869	750	29	2358	597	403	27321	515	749	2877
# edges	77876	5180	136	37716	4670	2668	733536	5005	4925	12172
class	20	21	22	23	24	25	26	27	28	29
# nodes	2076	393	1903	2834	22187	1257	4605	4801	21406	416
# edges	11062	1525	10036	15459	421129	12827	39656	43865	262381	2345
class	30	31	32	33	34	35	36	37	38	39
# nodes	11814	2828	411	1271	7867	127	3524	2369	1507	2029
# edges	250277	23863	2681	6243	75707	432	28697	15291	11345	11297

Table 3: Number of nodes and edges in each class of Arxiv-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	13101	3550	3302	15181	2322	3597	3952	2138	11187	2246
# edges	9333976	2207936	2170698	4435498	1284924	8838256	2876904	1981664	17567126	1577840
class	10	11	12	13	14	15	16	17	18	19
# nodes	4928	2964	1696	2731	4854	28272	1003	2639	13999	10308
# edges	4377476	3370058	3875108	1871030	9203184	23998718	1035980	3237128	16823834	7086846
class	20	21	22	23	24	25	26	27	28	29
# nodes	1596	4066	8222	12146	328	1659	4239	5962	4673	5101
# edges	1750498	5087278	2125064	4270774	148988	2005018	3190862	5055124	1505350	3147048
class	30	31	32	33	34	35	36	37	38	39
# nodes	2846	4570	1575	4960	3429	4202	4180	4233	12797	3099
# edges	2293504	8151694	836924	13468276	8032124	6091244	5871398	5330018	19629974	1190018
class	40									
# nodes	5112									
# edges	2896422									

Table 4: Number of nodes and edges in each class of Reddit-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	114294	109832	116043	151061	668950	40715	158771	172199	110796	67358
# edges	12896708	14838746	11449058	15576466	48891834	5984624	26375396	14340184	7605922	8296044
class	10	11	12	13	14	15	16	17	18	19
# nodes	52345	32937	131886	101541	3079	26911	83594	42337	49019	17438
# edges	7608804	1533830	12407580	9797510	151210	3758376	9552274	6297586	5733784	2187382
class	20	21	22	23	24	25	26	27	28	29
# nodes	22575	80795	879	3653	45406	3024	553	259	1969	1561
# edges	2247434	8065332	74388	877932	3593542	240336	78188	14760	106084	81154
class	30	31	32	33	34	35	36	37	38	39
# nodes	277	418	513	29	154	44	630	514	91	37
# edges	11672	30766	35648	3108	8066	2336	62204	44886	5972	3288
class	40	41	42	43	44	45	46			
# nodes	6	61	32500	1399	566	9	1			
# edges	876	17578	6229472	194368	122826	818	208			

Table 5: Number of nodes and edges in each class of Products-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	23472	34872	873	27456	40354	35359	44618	8211	37300	24126
# graphs	743	996	22	876	1151	997	1298	251	1024	727
class	10	11	12	13	14	15	16	17	18	19
# nodes	14057	44142	10667	6454	38814	27669	45146	7740	35994	36826
# graphs	376	1292	323	213	1108	885	1318	253	1006	1060
class	20	21	22	23	24	25	26			
# nodes	34027	31894	4694	21166	35159	45489	34019			
# graphs	1016	911	125	659	988	1304	946			

Table 6: Number of nodes and edges in each class of SIDER-tIL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	7820	5808	14428	7378	15835	7596	4002	18679	5340	7096
# graphs	309	237	768	300	793	350	186	942	264	372
class	10	11								
# nodes	20099	10151								
# graphs	918	423								

Table 7: Number of nodes and edges in each class of Tox21-tIL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	3139	0	0	0	1553	3007	3405	185	1670	3144
# graphs	148	0	0	0	60	149	150	13	83	148
class	10	11	12	13	14	15	16	17	18	19
# nodes	3364	3553	3914	3449	3686	3967	4066	4194	4629	4155
# graphs	149	150	150	144	150	150	149	149	150	149
class	20	21	22	23	24	25	26	27	28	29
# nodes	4355	4544	4824	4927	5202	4823	4820	4946	5189	3954
# graphs	149	148	149	146	145	144	136	136	132	94
class	30	31	32	33	34	35	36	37	38	39
# nodes	4817	3425	2749	1591	824	2730	1132	841	543	203
# graphs	106	74	52	29	17	15	16	6	6	1

Table 8: Number of nodes and edges in each class of Aromaticity-CL.

Table 9: Performance comparisons with different backbone GNNs under task-IL without inter-task edges (↑ higher means better).

C.L.T.	GCN		GAT		GIN	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	61.7±3.8	-28.2±3.3	63.7±1.6	-27.3±1.7	65.1±1.6	-23.9±1.5
EWC [4]	78.8±2.7	-5.0±3.1	75.1±0.5	0.1±0.2	63.2±3.0	-21.5±3.2
MAS [1]	88.4±0.2	-0.0±0.0	80.5±1.5	-8.6±1.2	87.9±0.4	-0.1±0.1
GEM [8]	87.3±0.6	2.8±0.3	79.2±0.4	-5.7±0.3	80.6±0.8	-3.6±1.1
TWP [7]	86.0±0.8	-2.8±0.8	86.6±0.2	-1.5±0.3	88.3±0.6	-0.6±0.8
LwF [6]	84.2±0.5	-3.7±0.6	54.5±0.7	-36.9±0.8	84.0±1.7	-3.7±1.3
ER-GNN [12]	86.7±0.1	11.4±0.9	88.4±0.4	4.2±0.8	89.3±0.1	5.0±0.4
Joint	90.3±0.2	-	88.8±0.4	-	88.9±0.1	-

licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 License (CC BY-NC-SA 4.0). The PubChemBioAssayAromaticity [11] is constructed from the PubChem's BioAssay Database, which is licensed under the Creative Commons Attribution Non-Commercial License (CC BY-NC 3.0). The Tox21 dataset [6] was a public database constructed by the "Toxicology in the 21st Century" (Tox21) initiative for measuring toxicity of compounds without attaching a specific license. All of the above datasets are consent by the authors for academic usage and are integrated in different deep learning libraries like the Deep Graph Library (DGL) [10] [7], DGL-Lifesci [5] [8], and Open Graph Benchmark (OGB) [3] [9]. None of these datasets contains personally identifiable information or offensive content.

Our code (`https://github.com/QueuQ/CGLB`) for constructing the benchmark tasks and comparing different baseline methods is licensed under an Attribution-NonCommercial 4.0 International license (CC BY-NC 4.0). Besides, the implementations of some of the baseline continual learning methods are adapted from existing public Github repositories, including the repository of the Gradient Episodic Memory (GEM) [8] [10] and the repository of the Topology-aware Weight Preserving (TWP) [7] [11]. The implementations of the backbone GNNs are mostly based on the DGL. The construction of the datasets also benefits from several existing databases and libraries. The construction of the N-CGL datasets uses the datasets and tools from OGB and DGL. The construction of the G-CGL datasets uses the datasets and tools from DGL and DGL-Lifesci. We sincerely thank the authors of these works for sharing their codes and helping develop the community.

3 Additional Experimental Results

In this section, we provide additional results to analyze the performance of different baseline methods on CGLB. The results will be mainly based on the visualization of the performance matrices, which is the most thorough metric to evaluate a continual learning model, of different methods on different benchmark tasks.

3.1 Investigation on the Influence of GNN backbones

In this section, we investigate the influence of the GNN backbones on the performance of continual graph learning, which is not discussed in the paper due to the space limitation. To ensure a fair comparison, all GNNs are configured as 2-layer. From Table 9, our first finding is although adopting different backbones result in different continual learning performance, the relative performance comparison (ranking) of different continual learning techniques are similar across different backbones except for EWC and MAS with GAT backbone. The performance of EWC and MAS with GAT backbone exhibit a contrary pattern compared to the performance obtained with the other two backbones. Specifically, EWC exhibits more severe forgetting issues with GCN and GIN compared to GAT. On the contrary, MAS exhibits more severe forgetting issues with GAT compared to GCN and

[6] `https://tripod.nih.gov/tox21/challenge/data.jsp`
[7] `https://docs.dgl.ai/api/python/dgl.data.html`
[8] `https://lifesci.dgl.ai/api/data.html`
[9] `https://ogb.stanford.edu/`
[10] `https://github.com/facebookresearch/GradientEpisodicMemory`
[11] `https://github.com/hhliu79/TWP`

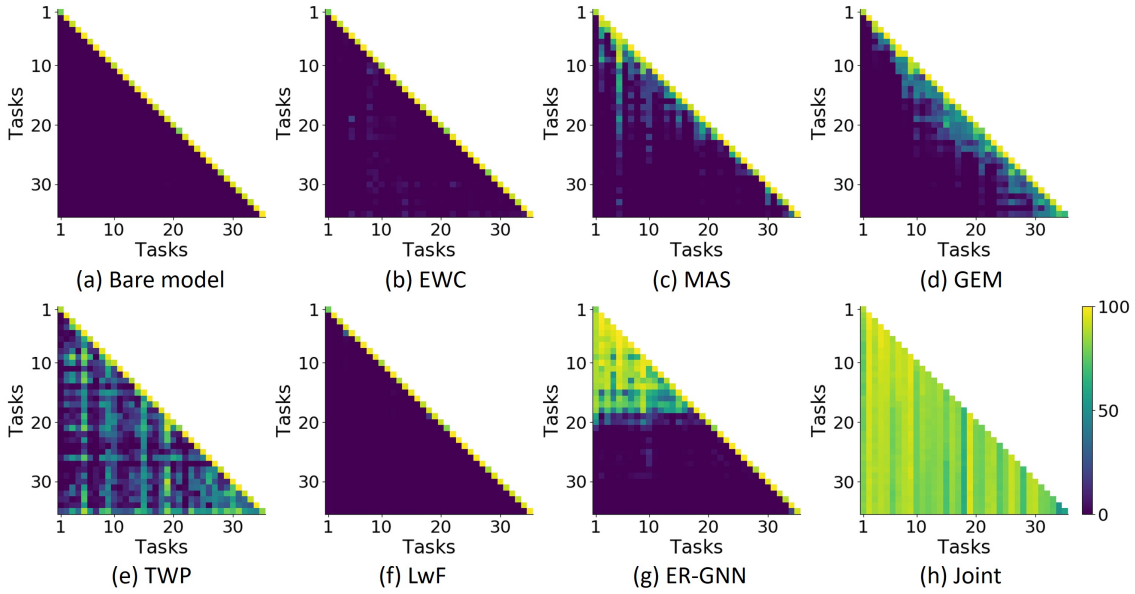

Figure 1: Visualization of the performance matrices of different methods on CoraFull-CL under class-IL setting.

GIN. This phenomenon is reasonable considering that the structure of GCN and GIN are very similar, while GAT follows a different framework. Both EWC and MAS are regularization based methods with different strategies for evaluating the importance of the model weights. This implies that for different backbones, the strategies for regularizing the model weights matter a lot and should be carefully designed. As for the bare model, bare GCN and GIN models also have similar performance, while bare GAT models perform worse. This phenomenon is closely related to the complexity of different models. The trainable part of GCN and GIN models are mainly fully connected layers, while GAT has a much more complex structure with the trainable attention mechanism. More complex models tend to overfit the new tasks with more severe forgetting issues on previous tasks. Finally, the jointly trained GCN, GAT, and GIN exhibit similar performance with tiny variations.

3.2 Additional Results on the Performance Matrix Visualization

In the paper, we visualized the performance matrices of several representative methods to show the details of the forgetting behavior under the class-IL scenario. In this subsection, we visualize the performance matrices of all different methods.

In Figure 1, the performance of the bare model is typical well for new tasks which are just learned, and bad for the old tasks since they are totally forgotten. The results of EWC exhibit certain improvement over the bare model, although not significant. MAS, which follows a similar regularization strategy, exhibits more significant improvement compared to EWC. From Figure 1(c), we could see the performance on multiple tasks (each column shows the performance change of one task along the learning process) is well maintained when a small number of tasks are learned and then decrease drastically when more tasks are learned. A similar phenomenon has been observed in the performance matrix of GEM. Among all the baselines except the joint training, TWP, which is specially designed for continual graph learning via preserving topology information, seems to be the best in maintaining the performance of existing tasks. From Figure 1(e), we observe that the performance of many tasks is maintained until the end of the learning on the task sequence (last row). Similarly, ER-GNN, which is also specially designed for CGL, exhibits significant improvement. However, ER-GNN is sensitive to the length of the task sequence. On a short task sequence (the upper part of Figure 1(g)), the performance can be well maintained, but when more tasks are learned (the lower part of Figure 1(g)), the performance decreases drastically. This trend can also be observed in the learning curve shown in Figure 3(a) of the paper. This phenomenon results from the memory mechanism of ER-GNN. ER-GNN selectively stores several nodes from each task and replays them when learning new tasks.

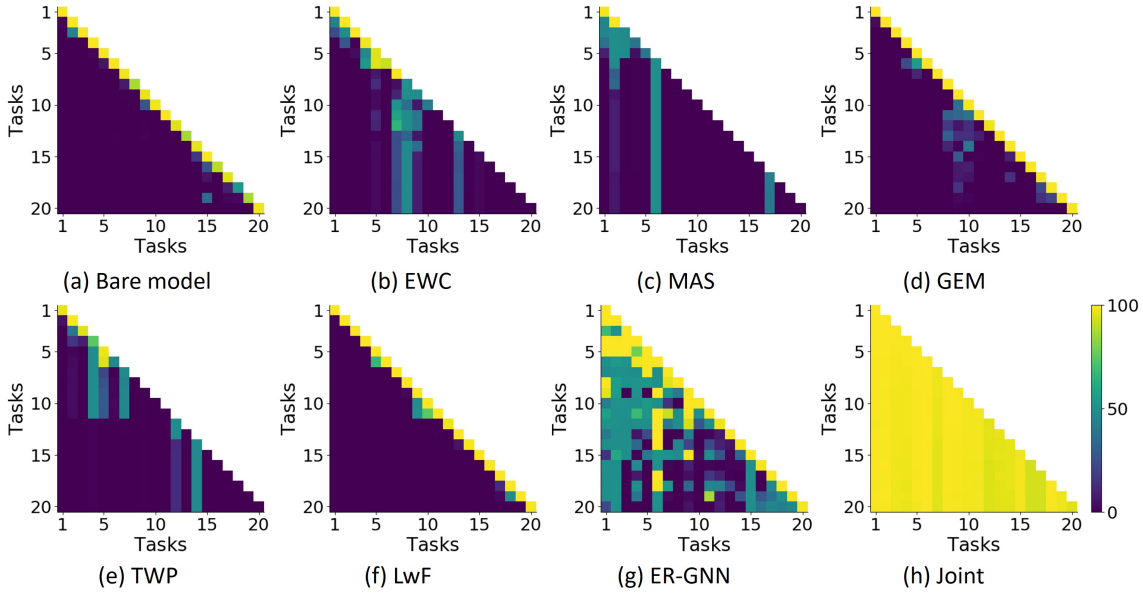

Figure 2: Visualization of the performance matrices of different methods on Reddit-CL under class-IL setting.

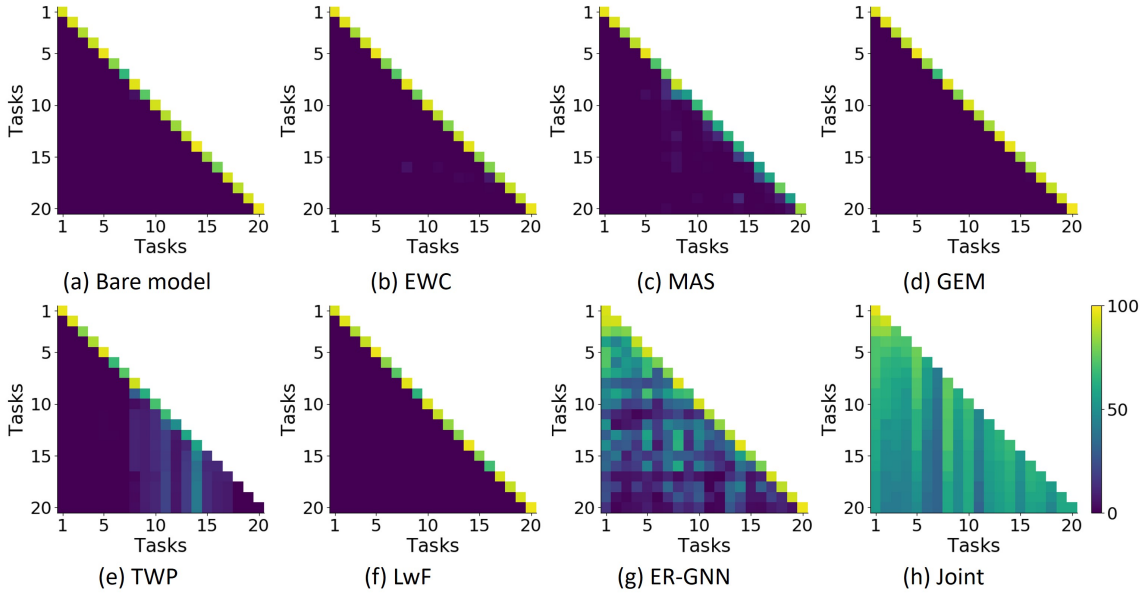

Figure 3: Visualization of the performance matrices of different methods on Arxiv-CL under class-IL setting.

The replay of the data from each previous task pushes the model parameters toward the direction that is favorable for its task. When too many tasks are stored, it would be hard for the model to find a direction to update, especially when the model has been already trained to an optimum of a certain task. This kind of forgetting pattern is not detectable solely from the final AP and AF, which only show the average results of the final row of the performance matrix.

Similar patterns can also be found in the results obtained on Reddit-CL as shown in Figure 2. The difference is that Reddit-CL is less challenging and most methods are performing better than on CoraFull-CL. Besides, since the task sequence of Reddit-CL is shorter than CoraFull-CL, ER-GNN is performing relatively well through the entire sequence.

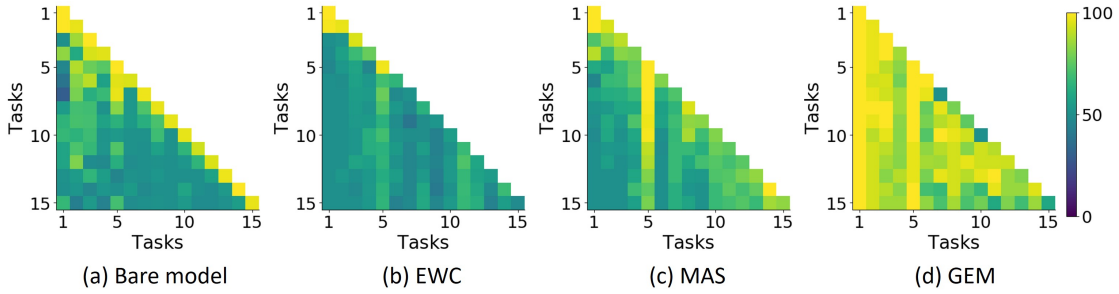

Figure 4: Visualization of the performance matrices of different methods on Aromaticity-CL under task-IL setting.

Among all the datasets, almost all baselines fail on Arxiv-CL. To further investigate this issue, we visualize the performance matrices of all baselines on Arxiv-CL, as shown in Figure 3. According to the results, the baselines perform well on each individual task (diagonal entries), and the failure comes from the severe forgetting issue. The results reveal that the task-wise interference in Arxiv-CL is strong. Specifically, Bare model, EWC, GEM, and LwF exhibit almost complete forgetting on previous tasks, and their performance on new tasks are better (diagonal entries). TWP, ERGNN, and MAS successfully preserve the performance on previous tasks to some extent, and their performance on the following tasks are lower. Specially, TWP preserves the performance on task 14 well, but fails in all the following tasks. This strong task-wise interference is finally justified again in the performance matrix of the jointly trained model (Joint), which does not suffer from the forgetting issue. Although Joint maintains a balanced performance on all tasks, its diagonal entries have obviously lower values than the methods with severe forgetting. In a word, due to the strong task-wise interference, a method cannot simultaneously maintain the performance well on both old tasks and new tasks.

3.3 Further Analysis on the Model Performances on G-CGL tasks

To further demonstrate the learning dynamics on G-CGL tasks, we visualize the performance matrices of several methods in this subsection. The visualization is done for both task-IL and class-IL scenarios on the Aromaticity-CL dataset.

Figure 4 and 5 show the results on Aromaticity-CL under task-IL and class-IL scenarios, respectively. The bare model and joint training still follow similar patterns as it did in Section 2.2. In other words, the bare model forgets severely on previous tasks while joint training performs perfectly on all tasks. For the other methods under both task-IL and class-IL scenarios, we should focus more on the diagonal entries of their performance matrices. The diagonal entries of the performance matrices of the bare model still demonstrate that the bare model learns well on each new task. However, the diagonal entries of the regularization based methods like EWC, MAS, and TWP, decrease significantly along with the learning (from top to the bottom). This indicates that different tasks have significantly different distributions and preserving the parameters of the previous tasks greatly hinders the learning of new tasks. This is also consistent with the finding in the paper that GEM often meets the 'no solution' error.

3.4 Additional Results with Less Training Data

Besides the train-validation-test split used in our major experiments, our implemented pipelines also support any other splittings through specifying the corresponding arguments (details can be found on our GitHub page). In the following, we provide the results obtained on both node-level and graph-level tasks with the splitting of train (20%), validation (40%), test (40%).

As shown in Table 10 and 11, the performance change of NCGL tasks and GCGL tasks with less training data are different. On NCGL tasks, most methods exhibit performance decrease, but some methods perform better. While in GCGL, almost all methods experience significant performance decrease. The reasons are two-fold. On the one hand, less training data may decrease the performance on single tasks. On the other hand, with less training data, the models adapt less to the new tasks,

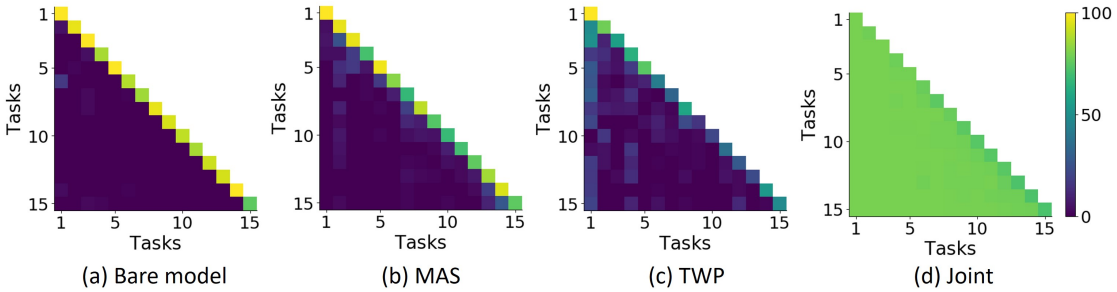

Figure 5: Visualization of the performance matrices of different methods on Aromaticity-CL under class-IL setting.

Table 10: Performance comparisons under task-IL without inter-task edges on different datasets with the splitting of train (20%), validation (40%), test (40%) (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	53.2±1.2	-40.5±1.5	60.9±6.2	-27.8±5.9	80.0±7.4	-20.2±7.7	66.0±1.9	-26.8±2.1
EWC [4]	70.1±3.7	-20.9±4.1	68.4±4.5	-16.6±5.0	92.5±6.6	-7.1±6.9	90.3±0.7	-0.6±0.3
MAS [1]	90.1±1.1	-0.5±0.5	87.3±0.2	0.0±0.0	98.9±0.3	0.0±0.0	91.8±0.7	-0.5±0.1
GEM [8]	90.0±0.1	0.0±0.4	80.4±0.1	-4.0±0.3	99.3±0.0	0.0±0.1	87.6±0.9	-3.0±0.8
TWP [7]	86.9±0.5	-2.0±0.5	86.1±0.8	-1.7±0.7	91.2±3.6	-8.5±3.8	90.3±0.3	-0.6±0.3
LwF [6]	54.2±1.6	-39.7±1.6	68.6±4.9	-20.6±5.3	78.7±4.3	-21.7±4.6	66.6±1.8	-26.9±2.0
ER-GNN [12]	83.6±0.4	-6.7±0.5	89.2±0.1	5.6±0.5	98.8±0.1	-0.4±0.1	89.3±0.1	-2.4±0.2
Joint	91.9±0.5	-	88.9±0.4	-	99.4±0.0	-	91.2±0.8	-

therefore the forgetting on previous tasks is less severe, which benefits the overall performance (AP and AF). The NCGL task sequences are longer than GCGL, and longer sequences bring more severe forgetting. Accordingly, the mitigation of the forgetting problem (by less training data) benefits the performance more in NCGL than GCGL. Therefore, the performance decreases more in GCGL than NCGL.

3.5 Studies on the Number of Classes in each Task

In the experiments reported in the paper, we set the number of classes in each task (K) as 2 so as to maximize the length of the task sequences and increase the learning difficulty. In this subsection. we further show the results obtained with multiple different Ks. As shown in Table 12, a clear pattern is that the continual learning performance of the models increase with the K. This phenomenon concerns two key factors. On the one hand, smaller K decreases the difficulty of each single task, which positively affects the performance. On the other hand, smaller K also increases the length of the task sequence and exacerbates the forgetting problem, which negatively affects the performance. Considering these two factors and the final results, we can conclude that the length of task sequence is the dominant factor determining the continual learning difficulty.

3.6 Benchmark Usages & Implementing New Methods with CGLB

Table 11: Performance comparisons on different graph-level prediction datasets with the splitting of train (20%), validation (40%), test (40%) (↑ higher means better).

C.L.T.	SIDER-tIL		Tox21-tIL		Aromaticity-CL		Aromaticity-CL	
	task-IL		task-IL		task-IL		class-IL	
	AP ↑	AF ↑	AP ↑	AF ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	0.532±0.007	0.028±0.016	0.645±0.022	0.119±0.015	52.0±1.4	0.1±1.2	4.3±1.4	-5.2±2.9
EWC [4]	0.503±0.012	0.003±0.006	0.602±0.021	0.028±0.027	52.0±1.4	0.1±1.2	3.9±0.8	-10.0±2.3
MAS [1]	0.518±0.010	0.014±0.006	0.630±0.022	0.092±0.029	58.8±2.0	5.0±2.2	3.8±1.0	-8.8±2.8
GEM [8]	0.578±0.006	0.072±0.014	0.685±0.007	0.183±0.025	70.6±2.2	18.8±3.4	10.9±1.7	2.1±3.5
TWP [7]	0.505±0.009	0.009±0.004	0.593±0.010	0.046±0.025	54.2±2.7	0.9±2.9	3.5±1.1	-8.7±2.6
LwF [6]	0.531±0.009	0.027±0.008	0.641±0.017	0.105±0.049	58.7±1.1	8.2±2.4	5.4±0.4	-8.4±0.9
Joint	0.575±0.009	-	0.678±0.017	-	69.4±1.0	-	35.4±3.9	-

Table 12: Performance comparisons under class-IL on Arxiv-CL with different task splittings (↑ higher means better).

C.L.T.	$K=2$		$K=5$		$K=10$		$K=20$	
	AP/% ↑	AF/%↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	4.9±0.0	-87.0±1.5	10.5±0.1	-77.5±0.5	16.4±0.2	-63.9±0.6	26.4±0.3	-47.3±0.9
EWC [4]	4.9±0.0	-88.9±0.3	9.4±0.1	-73.7±1.1	15.7±0.3	-62.8±0.7	24.8±0.3	-47.5±0.6
MAS [1]	4.9±0.0	-86.8±0.6	10.3±0.2	-77.5±0.6	16.5±0.3	-64.0±0.5	26.3±0.6	-47.5±0.7
GEM [8]	4.8±0.5	-87.8±0.2	10.7±0.1	-81.5±0.3	18.2±0.2	-70.6±0.5	31.3±0.1	-58.5±0.2
TWP [7]	4.9±0.0	-89.0±0.4	8.3±0.4	-66.1±1.3	14.0±0.4	-57.6±1.5	22.0±0.4	-47.6±0.5
LwF [6]	4.9±0.0	-87.9±1.0	10.5±0.1	-78.9±0.3	17.4±0.3	-66.2±0.5	28.6±0.1	-52.6±0.6
ER-GNN [12]	30.3±1.5	-54.0±1.3	10.9±0.2	-77.5±0.5	19.8±1.2	-59.9±1.3	31.6±0.6	-34.8±1.3
Joint	46.4±1.4	-	47.7±0.3	-	45.2±0.7	-	43.6±0.3	-

The tutorial on using CGLB and repeating the results reported in our paper are provided on our Github page `https://github.com/QueuQ/CGLB` with examples.

Our pipelines for both node-level and graph-level tasks are highly modularized to facilitate the implementation and evaluation of new continual graph learning methods on CGLB.

We exemplify the implementation of node-level methods, and the generalization to graph-level methods is straightforward. The newly implemented method should be contained in a python script file under the directory *CGLB/NCGL/Baselines*. Suppose we are implementing a method named *A*, then an *CGLB/NCGL/Baselines/A_model.py* containing the implementation of the method should be created. The implementation is flexible as long as it satisfies the input format. Specifically, the python class of the new method should contain an *observe()* function for model training on a single task, whose input include the task configurations and the data. Details on the input format could be found in any *xxx_model.py* file under the directory *CGLB/NCGL/Baselines*.

4 Broader Impact

Our proposed benchmark can greatly facilitate the development of continual graph learning since it provides a comprehensive, fair, and standard protocol to evaluate and compare different CGL models. However, CGLB still has space for improvements, which will be targeted in our future research. For example, currently, we haven't included any task concerned with the privacy issue. Since a majority of graph related applications are concerned with privacy-related scenarios such as the social networks and the co-purchasing networks, the privacy preserving continual graph learning will inevitably become an important problem. In the future, we will keep track of the newly emerged problems on CGL and enhance CGLB by incorporating new benchmark tasks as well as new baseline results and keep improving the toolkit for result analysis.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[3] 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. *arXiv preprint arXiv:2005.00687*, 2020.

[4] 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.

[5] Mufei Li, Jinjing Zhou, Jiajing Hu, Wenxuan Fan, Yangkang Zhang, Yaxin Gu, and George Karypis. Dgl-lifesci: An open-source toolkit for deep learning on graphs in life science. *ACS Omega*, 2021.

[6] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

[7] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. *arXiv preprint arXiv:2012.06002*, 2020.

[8] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[9] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[10] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[11] Yanli Wang, Jewen Xiao, Tugba O Suzek, Jian Zhang, Jiyao Wang, Zhigang Zhou, Lianyi Han, Karen Karapetyan, Svetlana Dracheva, Benjamin A Shoemaker, et al. Pubchem's bioassay database. *Nucleic acids research*, 40(D1):D400–D412, 2012.

[12] Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4714–4722, 2021.