Dynamic Mixture-of-Experts for Incremental Graph Learning

Lecheng Kong
jkong@amazon.com
Amazon

Theodore Vasiloudis
thvasilo@amazon.com
Amazon

Seongjun Yun
sjyun@amazon.com
Amazon

Han Xie
hanxie@amazon.com
Amazon

Xiang Song
xiangsx@amazon.com
Amazon

Abstract

Graph incremental learning is a learning paradigm that aims to adapt trained models to continuously incremented graphs and data over time without the need for retraining on the full dataset. However, regular graph machine learning methods suffer from catastrophic forgetting when applied to incremental learning settings, where previously learned knowledge is overridden by new knowledge. Previous approaches have tried to address this by treating the previously trained model as an inseparable unit and using regularization, experience replay, and parameter isolation to maintain old behaviors while learning new knowledge. These approaches, however, do not account for the fact that previously acquired knowledge at different timestamps contributes differently to learning new tasks. Some prior patterns can be transferred to help learn new data, while others may deviate from the new data distribution and be detrimental. Moreover, in the graph context, a node's receptive field contains neighbors from different data blocks, requiring variable processing, and an inseparable unit fails to account for such variability. To address this, we propose a dynamic mixture-of-experts (DyMoE) approach for incremental learning. Specifically, a DyMoE GNN layer adds new expert networks specialized in modeling the incoming data blocks. We design a customized regularization loss that utilizes data sequence information so existing experts can maintain their ability to solve old tasks while helping the new expert learn the new data effectively. As the number of data blocks grows over time, the computational cost of the full mixture-of-experts (MoE) model increases. To address this, we introduce a sparse MoE approach, where only the top-k most relevant experts make predictions, significantly reducing the computation time. Our model achieved 4.92% relative accuracy increase compared to the best baselines on class incremental learning, showing the model's exceptional power.

CCS Concepts

• **Computing methodologies** → **Machine learning**.

Keywords

Graph Neural Network, Continual Learning, Mixture of Experts.

KDD '25, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Lecheng Kong, Theodore Vasiloudis, Seongjun Yun, Han Xie, and Xiang Song. 2018. Dynamic Mixture-of-Experts for Incremental Graph Learning. In *Proceedings of Knowledge Discovery and Data Mining (KDD '25).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Graph neural networks (GNN) achieved great success in modeling graph data and have many applications, such as recommender systems [36], drug discovery [9], and traffic forecasting [13]. However, in many real-world settings, the graph is dynamic, starting small and expanding over time, and the training data arrive as sequences of data blocks with timestamps. Naive approaches train on the **full graph** whenever new data appears, which incurs expensive computational costs due to repetitive training on old data. On the other hand, simply finetuning conventional GNNs on the new data leads to catastrophic forgetting, where the model's prediction shifts toward the new data distribution and forgets how to handle previously learned tasks upon encountering new data [6, 39, 44, 46]. This motivated a series of *continual learning* research to tackle this problem [8, 38, 41].

Pioneering efforts focused on adapting incremental learning approaches for other data modalities to the graph domain [30, 39, 48]. However, they ignore the fact that nodes and edges are not independent and identically distributed (i.i.d.) in the graph learning scenario [34, 35]. In the vision and language domain, individual image or text data points do not affect each other, and future data blocks do not impact the data distribution of the existing data blocks. In contrast, new graph data blocks connect to existing data via edges and could significantly change existing data distribution. For example, an incoming data block can add edges between two disconnected components in an existing graph, drastically changing the graph topology and, subsequently, the learned model behavior. Such uniqueness makes graph incremental learning an even more challenging scenario than incremental learning in other domains.

Subsequent efforts tackled the problem in several ways [31, 34, 39]. For instance, PI-GNN [43] rectified the old model on the graph modified by the new data. TWP [20] identified topology-aware parameters to stabilize the model under graph structure shift. DiCGR [18] breaks relation triplets to components to better capture graph structures. RLC-CN [23] determines the optimal memory size for effective efficiency/performance trade-off. SSRM [28] minimizes structural shift loss to mitigate performance degradation on old nodes.

These methods show improvements in the graph setting compared to the naive adaptation of incremental learning methods from other domains. However, a commonality of these approaches is that

they build the new model upon an inseparable old model. Specifically, Elastic Weight Consolidation (EWC) [17] used the old model parameters as the single regularization target for all parameters; Experience Replay (ER) [48] trained the model using all saved subsets of nodes from old data blocks; Parameter Isolation (PI-GNN) [43] froze all old model parameters and used an additional network to modify the model output. Such a pattern causes inflexibility when dealing with the "stability versus plasticity dilemma" [15] commonly seen in the continual learning domain, where the model needs to effectively trade-off between maintaining old knowledge (stability) and learning new (plasticity).

For example, in Figure 1, blue, red, and green nodes represent data blocks one, two, and three that arrive in order in all three cases, and we update the model whenever a data block arrives. Block one and two are identical in all cases, while block three is isolated, connected to both blocks two and three, and connected to block two only in cases A, B, and C, respectively. After learning blocks one and two, the model needs to learn block three while maintaining old knowledge. Previous approaches tackle cases A and B, where the new block relates to old blocks by similar patterns (no correlation to all blocks in case A, and high correlation in case B), and they use the same strategy to retain knowledge from both blocks one and two. However, in case C, where the new block is partially dependent on the old blocks, they still apply the same stabilizing strength to knowledge acquired from previous blocks. While a stronger stabilizing effect is required to maintain knowledge in block one as it is more divergent from the new block, a smaller effect is desirable for knowledge in data block two as it is similar to the new block and weaker stabilizing enables better knowledge transfer and learning. Conventional approaches lack the flexibility to handle such diversity among data blocks and hence witness performance degradations.

To tackle this problem, we propose a Dynamic Mixture-of-Expert (DyMoE) module to use separate expert networks to model different data blocks with a gating mechanism to synthesize information from the most relevant experts. Specifically, the module has the same number of experts as trained data blocks, and the experts are dedicated to learning from their corresponding data blocks. DyMoE routes the input to experts through a gating mechanism, and the expert outputs are combined through the gating values by a weighted sum. This approach explicitly considers the correlation between different experts and data blocks. For the same example in Figure 1, we train three separate experts with specialization in their corresponding data blocks. We then compute the relevance of the experts to the input. The experts with higher relevance have a higher impact on the prediction. This approach dynamically adjusts the combination of knowledge from different data blocks; less impactful experts are disabled during inference to reduce misleading information. When a new data block arrives, we append a new expert dedicated to the new data block without interfering with the knowledge of existing experts during training. To ensure each expert focuses on the assigned data block, we propose a block-guided loss as a training objective that enforces a high relevance score of experts to the input from their corresponding data blocks, greatly reducing catastrophic forgetting while allowing flexible querying of old knowledge.

While DyMoE applies to most neural architectures, **it is particularly useful for graph incremental learning**, where nodes' receptive fields contains neighbor nodes from different data blocks. We apply DyMoE to graph neural networks such that different experts can process nodes from different data blocks in the same ego subgraph, largely preserving the authenticity of the node's representation. Another challenge is that nodes in future data blocks change the topology and information in the old data blocks, as they become neighbor nodes to the old data blocks. We extend the gating mechanism to distinguish future neighbors from the old, we then filter future nodes for the old experts, largely recovering the old behavior and reducing forgetting. To increase efficiency, we propose a sparse variant, inspired by Shazeer et al. [26], that only considers the most relevant experts during inference, significantly reducing the computation complexity while maintaining high accuracy. In this paper, we

- Identified the issue of existing continual learning methods that ignore the correlation between different data blocks.
- Designed a DyMoE module with specialized experts for each data block and proposed data block-guided loss to minimize the negative interference between experts.
- Interleave the DyMoE module into GNNs and use graph block-guided loss to address the data shift problem unique to graph continual learning.
- Developed a sparse version of the DyMoE module so the model is both efficient and effective.

In our empirical evaluation, DyMoE maintains the same efficiency, while significantly improving over the best baseline in class incremental learning setting. The model also demonstrates strong results in instance incremental settings.

2 Preliminaries

Graph Incremental Learning. This paper focuses on incremental learning for node classification. Specifically, we follow the widely adopted problem formulation [8, 41], and aim to incrementally learn from a graph data block sequence $D = \{G^{(1)}, ..., G^{(t)}\}$, and each data block is a graph $G^{(i)} = (V^{(i)}, E^{(i)}, \boldsymbol{y}^{(i)})$ where $V^{(i)}$ is the set of nodes, and $E^{(i)}$ is the set of edges, and $\boldsymbol{y}^{(i)}$ is the classification labels of the nodes. Future graph snapshots expand on existing graphs, and $G^{(i)}$ is a subgraph of $G^{(j)}$ for $i < j$. We additionally use $\Delta G^{(i)} = (V^{(i)} \setminus V^{(i-1)}, E^{(i)} \setminus E^{(i-1)})$ to represent the graph delta between $G^{(i)}$ and $G^{(i-1)}$. We use $b(v)$ to indicate the data block index where the data/node v first appears. In the incremental learning setting, data arrive in order, and the i-th model is only trained and evaluated on $(G^{(1)}, ..., G^{(i)})$ without any knowledge about future graphs. The goal is to maximize the overall accuracy on each data block while minimizing the performance drop on previous data blocks. If the classes in $\boldsymbol{y}^{(i)}$ persist throughout all blocks, we refer to the task as instance-incremental learning [33]. If the classes in $\boldsymbol{y}^{(i)}$ are disjoint, we refer to the task as class-incremental. In this case, new data blocks also bring in new classes [45], and the model needs to classify a sample without knowing its corresponding block during inference.

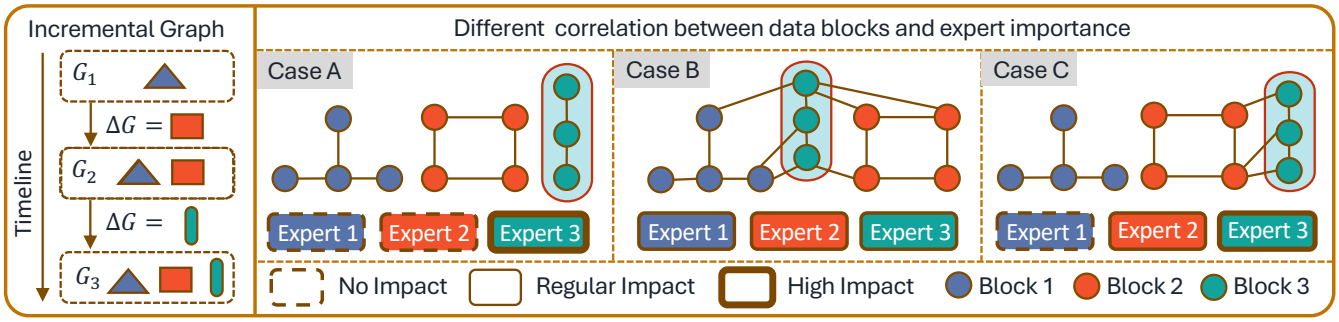

Figure 1: Left: Data blocks arrive in sequence. Right: Different connection types of three data blocks. Our proposed method activates dedicated experts when inferring relevant data blocks.

The naive solution is to train a model on the full graph $G^{(i)}$ for every block. However, this requires retraining on all old data multiple times, incurring huge computational costs. Incremental learning methods aim to train only on the graph delta while maintaining good performance on the old data.

To evaluate a model, let $a_{i,j}$ be the accuracy of all evaluation nodes in G_i, evaluated by the model after training G_j, which is a superset of evaluation nodes in $G^{(i)}$ and $i \leq j$. We evaluate the overall model performance by Average Accuracy (AA) and Average Forgetting (AF),

$$AA = \frac{1}{t}\sum_{i=1}^{t} a_{i,i}, \quad AF = \frac{1}{t}\sum_{j=1}^{t}\frac{1}{j}\sum_{i=1}^{j} a_{i,j} - a_{i,i} \tag{1}$$

where t is the number of data blocks. AA evaluates the model's average accuracy right after the model is trained on a data block, while AF evaluates the model's ability to retain knowledge from previous data blocks. The goal of an incremental learning method is to maximize AA and minimize AF.

Graph Neural Networks Graph neural networks iteratively update a node's embeddings from their neighbor nodes through message-passing layers [10]. Specifically, for a graph $G = (V, E)$, the i-th layer of a T-layer GNN is,

$$\boldsymbol{h}_v^{(i+1)} = COMB(\boldsymbol{h}_v^{(i)}, AGGR(\{\boldsymbol{h}_u^{(i)} | u \in \mathcal{N}(v)\}), \quad v \in V \tag{2}$$

where $\mathcal{N}(v)$ are the direct neighbors of v. Different GNN designs differ mainly by the combine (COMB) and aggregate (AGGR) functions.

3 Dynamic Mixture-of-Experts Graph Neural Network

This section first introduces the Dynamic Mixture-of-Experts (DyMoE) module that dynamically increases the number of experts for new data blocks. We then describe the integration of DyMoE and GNN for effective graph incremental learning. To overcome the efficiency issue with long data sequences, we propose Sparse DyMoE to reduce the complexity of our framework. The overall architecture of the framework is shown in Figure 2.

3.1 Dynamic Mixture-of-Experts Module

Conventional mixture-of-experts (MoE) models create networks of the same architecture and apply a gating mechanism to combine the networks' outputs using a weighted sum [26]. The number of experts is fixed after initialization. However, to accommodate new data blocks, MoE models suffer from the same issue as in other continual learning methods. They still need to adjust the weights of all previous experts, leading to forgetting. To mitigate this, we propose the DyMoE module, adding one expert for every new data block without modifying previously trained experts. Let $\mathcal{F}$ be a class of neural networks with the same architecture, and $f_\theta \in \mathcal{F}$ be an instance of the network parametrized by θ. Specifically,

$$\boldsymbol{h} = f_\theta(\boldsymbol{x}) \quad \boldsymbol{x} \in \mathcal{R}^n, \boldsymbol{h} \in \mathcal{R}^m, f_\theta \in \mathcal{F} \tag{3}$$

where $\boldsymbol{x}$ and $\boldsymbol{h}$ are the input and output to the network, and n and m are the input and output dimensions. Given an incremental data sequence $D = \{(X^{(1)}, \boldsymbol{y}^{(1)}), ..., (X^{(t)}, \boldsymbol{y}^{(t)})\}$, DyMoE handles the first data block like a conventional network. Specifically, it minimizes the empirical loss,

$$\arg\min_{\theta_1} \frac{1}{|X^{(1)}|} \sum_{i,\boldsymbol{x}\in X^{(1)}} \mathcal{L}(\boldsymbol{y}_i^{(1)}, f_{\theta_1}(\boldsymbol{x})) \tag{4}$$

The loss function $\mathcal{L}$ is task dependent, and we use cross-entropy loss for classification. For the second data block, we will add one expert and gating vectors to the overall model. To compute the output, we have

$$\begin{aligned} \boldsymbol{h} &= f_{\{\theta_1,\theta_2\}}(x) = \alpha_1 f_{\theta_1}(x) + \alpha_2 f_{\theta_2}(x), \\ \alpha_i &= \frac{exp(s(\boldsymbol{x}, \boldsymbol{g}_i))}{exp(s(\boldsymbol{x}, \boldsymbol{g}_1)) + exp(s(\boldsymbol{x}, \boldsymbol{g}_2))} \quad i \in \{1, 2\} \end{aligned} \tag{5}$$

where $\boldsymbol{g}$ are gating vectors associated with each expert, $s(\cdot, \cdot)$ is a similarity measure, and we use softmax on the similarities to compute the importance of each expert for the input. Note that this formulation is the same as existing MoE approaches, and the key difference is that the number of experts dynamically increases as more data arrive. Subsequent data blocks follow the same procedure, where the output is computed as,

$$\boldsymbol{h} = f_{\{\theta_1,...,\theta_t\}}(\boldsymbol{x}) = \sum_{i=1}^{t} \alpha_i f_{\theta_i}(\boldsymbol{x}), \quad \alpha_i = \frac{exp(s(\boldsymbol{x}, \boldsymbol{g}_i))}{\sum_{j=1}^{t} exp(s(\boldsymbol{x}, \boldsymbol{g}_j))} \tag{6}$$

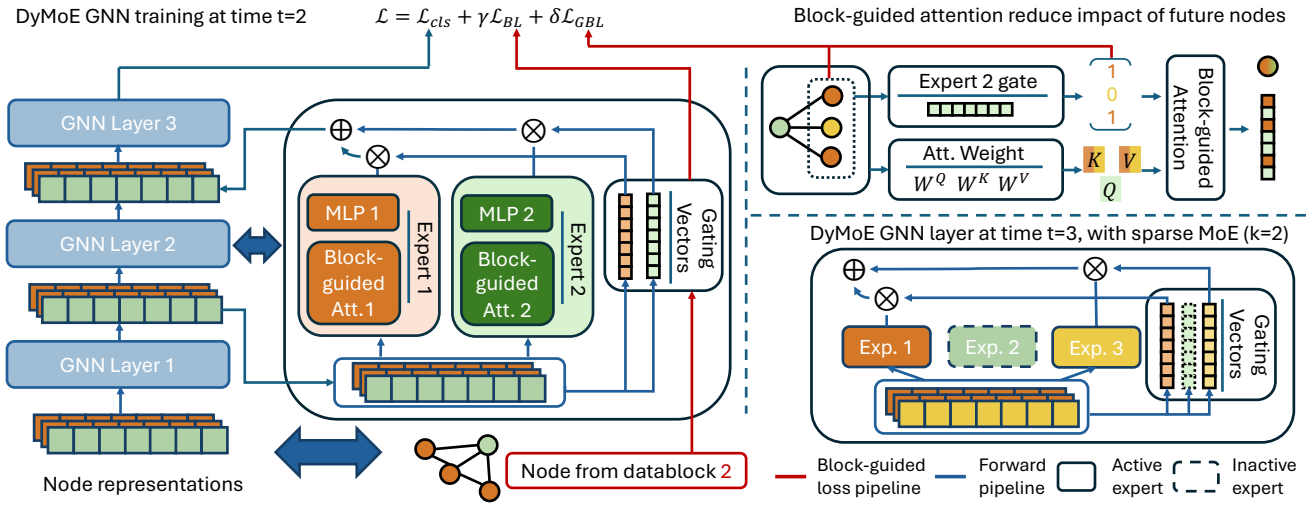

Figure 2: Pipeline of DyMoE GNN. Left: Each GNN layer has t experts with individual attention and MLP weights. We compute gating values from the node representations and the gating vectors. During training, we compute a block-guided loss between the gating values and the data block index for correct expert selection. Top-Right: the graph block-guided loss assigns additional weights to neighbor nodes and filters unrecognized nodes for experts. Bottom-Right: When a new data block arrives, we add a new expert and a gating vector to the DyMoE module. In the sparse case, only the most important experts are used.

When training on a new data block t, we only optimize the new expert and all the gating vectors, specifically,

$$\underset{\theta_t, \{\boldsymbol{g}_1 \ldots \boldsymbol{g}_t\}}{\arg\min}\ \mathcal{L}_{cls}, \mathcal{L}_{cls} = \frac{1}{|X|} \sum_{i, \boldsymbol{x} \in X} \mathcal{L}(\boldsymbol{y}_i, f_{\{\theta_1, \ldots, \theta_t\}}(\boldsymbol{x})) \tag{7}$$

Intuitively, this training scheme completely preserves the knowledge obtained from previous data blocks. Ideally, when the gating vectors are perfectly trained to distinguish which data block a particular data point belongs to, the model can **fully recover** the output of that data point, eliminating forgetting.

Block-guided loss. While the experts can preserve learned knowledge, the new experts are randomly initialized and start with trivial predictions on all data. The model will rely on the existing trained experts to make predictions, though they may carry old, potentially suboptimal, knowledge regarding the new data block. The gating vectors, including the new one, will tend to select the old experts during training. The model can hence be trapped at the local minimum without properly training the new dedicated experts. Consequently, we need to inject the information about the correct experts for our dynamically initialized new modules. This is difficult in conventional MoE because of the lack of supervision for correct experts. However, in continual learning, data arrive in blocks, and since experts are designed to handle individual data blocks, *we know exactly which expert a particular training data point should be assigned to.* We propose a **block-guided loss** to train the gating vectors for correct expert assignment. Specifically, for an arbitrary data point $\boldsymbol{x}$, in addition to its classification loss, we add a cross-entropy loss between the gating values of all experts and the data point's corresponding data block index $b(\boldsymbol{x})$. The computation is valid because the number of experts equals the number of witnessed data blocks. The loss forces an expert's corresponding data and gating vector to have large similarities, maximizing the likelihood of using the correct expert to generate output for the data. Specifically,

$$\mathcal{L}_{BL}(x) = CE(Softmax(\alpha_1 \ldots \alpha_t), OneHot(b(\boldsymbol{x}), t)), \boldsymbol{x} \in X \tag{8}$$

where CE is cross-entropy loss, α is the gating values, $OneHot(j, t)$ generates a t-dimensional one-hot vector whose j-th entry is one. Note that if we naively take X as the new samples in the most recent data block $X^{(t)}$, all of them will have the same data block index (the last index), causing the model to always use the last expert. Hence, we store a small sample set from each data block as the memory set, $M^{(i)} \subset X^{(i)}$ and $|M^{(i)}| \ll |X^{(i)}|$, and take $X = \bigcup_{i=1}^{t-1} M^{(i)} \cup X^{(t)}$, so the model can adjust the gating values accordingly. The size of memory set $|M| = p|X|$ where $p < 1$. The memory set construction details can be found in Appendix B.

Note that we only use such information during training, and the model does not need the time information, or which block that a data point belongs to, during inference, making the model perfectly viable for difficult tasks such as class-incremental learning. The overall training loss is,

$$\mathcal{L} = \mathcal{L}_{cls} + \gamma\mathcal{L}_{BL} \tag{9}$$

where β is a hyperparameter controlling the strength of regularization. The combined framework essentially attempts to train a data-block-dedicated classifier and out-of-distribution detectors for every data block. The gating mechanism gives high weight to in-distribution experts while minimizing the impact of out-distribution experts. While this approach applies to arbitrary data modality, it is particularly critical in the graph learning setting, where a node's neighbor might be from different data blocks and require different processing. We elaborate more on this in Section 3.2. We theoretically show the advantages of our proposed model over the Parameter Isolation (PI) [43] approach, a representative architectural approach for continual learning.

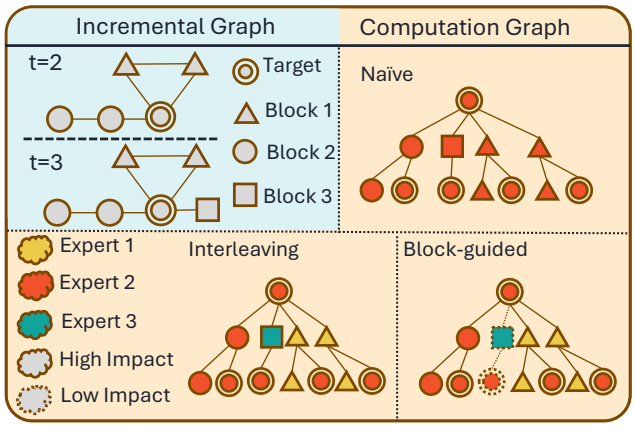

Figure 3: The computation graphs of the same target node at $t = 3$ by different approaches.

Theorem 1. *For an arbitrary continual learning problem, suppose a PI model obtains a cross-entropy classification loss $\mathcal{L}_{PI}$, there exists a parametrization of DyMoE that achieves cross-entropy classification loss $\mathcal{L}_{Dy} = \mathcal{L}_{PI}$. When the data sequence follows a mixture of Gaussian distribution, we have $\mathcal{L}_{Dy} \leq \mathcal{L}_{PI}$.*

The proof is in Appendix A. In the proof, we first show that DyMoE is at least as powerful as PI. We then show under the Gaussian Mixture assumption of the input data block sequence; the DyMoE obtains strictly lower loss, which shows the model's superiority.

In practice, the memory set is very small to ensure efficiency, but we jointly train on it with the full dataset from the new data block, which can give the model a biased understanding of the data distribution (i.e. most of the data are from the last data block). Hence, we propose a *data balancing* training procedure, where, after the regular training epochs, we collect the memory set for the new data block, combine it with all previous training memory sets, and train a few epochs on them to reflect the actual distribution of the entire input sequence. We elaborate on the data balancing training procedure in Appendix B.

3.2 Dynamic Mixture-of-Expert Graph Neural Network

We then introduce fusing the DyMoE with a graph neural network. Note that the DyMoE module does not assume any specific network architectures, and a naive solution can treat a multi-layer GNN as $\mathcal{F}$. However, this ignores the unique property of graph data in continual learning, where a target node's neighbors are from different data blocks. For example, in Figure 3, the target node is from data block two but is later connected to nodes in block three. The naive approach will assign expert two to process the target node and all of its neighbors, but its neighbor nodes are from blocks one and three, and expert two lacks the knowledge to properly handle them, leading to compromised performance. Ideally, a DyMoE model should assign neighbors to their corresponding experts. Hence, we propose interleaving the DyMoE modules into each GNN layer to correct this. We modify a transformer Graph Convolution Layer [27] to the architecture of each expert layer. Specifically,

$$f_{\theta_t}^{(i)} = \boldsymbol{h}_{v,t}^{i} = MLP(\boldsymbol{h}_v^{(i-1)} + Att(\boldsymbol{h}_v^{(i-1)}, \{\boldsymbol{h}_u^{(i-1)} | u \in \mathcal{N}(v)\})) \quad (10)$$

where Att is the attention mechanism with target node's feature as query and neighbor nodes' features as key and values, formally

$$\begin{aligned} Att(\boldsymbol{h}, U) &= softmax(\frac{\boldsymbol{q}K}{\sqrt{d}})V \\ \boldsymbol{q} = W_t^Q \boldsymbol{h}, K = W_t^K U, V = W_t^V U, &\quad W_t^Q, W_t^K, W_t^V \in \mathcal{R}^{n \times n} \end{aligned} \quad (11)$$

where Ws are the attention weights, and each expert has individual MLP and attention weights. The interleaving DyMoE design can be easily extended to other GNN architectures. The key difference between this and naive approaches is that DyMoE naturally handles the problem where neighbor nodes of the target node are from different data blocks. As long as we properly train the gating vectors, DyMoE can route each neighbor node to their corresponding expert instead of the one corresponding to the target node. This approach generates the most authentic representations.

Another critical trait of incremental graph learning is that new data can change the existing graph's overall topology. In the same example, expert two is trained to handle data block two without data from block three. However, after data block three arrives, its data changes the neighborhood of the target node, meaning that even we correctly assign the experts, we will not recover the performance and output. Essentially, the modified topology in the graph causes the old model to shift from its original prediction and cause a performance decrease.

To overcome this issue, we observe that since the gating mechanism introduced in Section 3.1 can be used to distinguish which data block a node is from, we can extend it to predict whether a node occurs before an expert is added to the model. Correctly predicting this target allows us to suppress future nodes' influence on old experts and thus maximally preserve the old behavior. Specifically, we create an additional gating vector $\boldsymbol{p}_i$ for each expert and compute gating values $\beta_{u,t}$ for node u on expert t as:

$$\beta_{u,t} = sigmoid(\boldsymbol{p}_t \cdot (W^P h_u)), \quad u \in G \quad (12)$$

where W^P is a linear projection shared by all experts in the same layer and β is independent of other experts, and it indicates whether the node u is incremented to the graph before expert t is added to the model. Since $\beta \in (0, 1)$, we can apply it to the attention computation to reduce the impact of future nodes. Specifically, the modified attention becomes,

$$Att_N(\boldsymbol{h}, U) = softmax(\frac{\boldsymbol{q}K}{\sqrt{d}} + log(\boldsymbol{\beta_t}))V \quad (13)$$

where $\boldsymbol{\beta_t}$ is the corresponding gating values $\beta_{u,t}$ of neighbor nodes $u \in U$. When $\beta_{u,t}$ approaches 0, the attention to the corresponding neighbor will become zero, and when $\beta_{u,t}$ approaches 1, the neighbor's attention is computed as normal. Hence, when the gating values are computed correctly, the attention mechanism can select the nodes that reproduce the output like the graph is not incremented with new neighbors. To properly train these gating vectors, we can use a similar supervised signal as in the block-guided loss. For a node u and its corresponding data block $b(u)$, we create a multi-hot vector l_u, whose first $b(u) - 1$ entries are zeros and last

$t - b(u) + 1$ are ones, and compute the binary-cross-entropy loss as:

$$\mathcal{L}_{GBL}(u) = \frac{1}{t}\sum_{j=1}^{t} BinaryCrossEntropy(\beta_{u,j}, l_{i,j}) \tag{14}$$

We term this loss graph block-guided loss(GBL). The loss encourages the gating value to be 1 if the experts are added after the node, and 0 otherwise. Finally, we need to accommodate the block-guided regularization loss to a more fine-grained version for the interleaving design. Instead of using the target node's corresponding data block as the regularization target, we use each neighbor node's own corresponding data block as the target. And the final loss is,

$$\mathcal{L} = \mathcal{L}_{cls} + \sum_{i=1}^{T}\sum_{v\in V} \gamma\mathcal{L}_{BL}(v) + \delta\mathcal{L}_{GBL}(v) \tag{15}$$

where γ and δ are hyperparameters. This loss ensures that (a) when the neighbor and the target nodes are from different data blocks, we still want the most relevant expert to be of higher importance; (b) when the correct expert is selected, the expert gets the input that it recognizes from training (low impact from new neighbors, and high impact from old/familiar neighbors).

3.3 Sparse Dynamic Mixture-of-Experts GNN

While the proposed DyMoE GNN allows effective knowledge preservation and updates specialized for graph data, it incurs additional computation cost for the dynamically increasing experts. With more data blocks, we can have too many experts whose computational burden overwhelms the performance benefits of the module. Inspired by previous works on Sparse MoE [26], we introduce sparsity into the system to improve its efficiency. To that end, we modify Equation 6 so that only the experts with the top-k importance score are used to generate predictions. Specifically,

$$(\alpha_1 ... \alpha_t) = Softmax(TopK(s(\boldsymbol{x}, \boldsymbol{g}_0) ... s(\boldsymbol{x}, \boldsymbol{g}_t)))$$
$$\boldsymbol{h} = \sum_{i=1}^{t} \boldsymbol{h}_i, \quad \boldsymbol{h}_i = \begin{cases} \alpha_i f_{\theta_i}(\boldsymbol{x}), & \text{if i in TopK} \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

Because we only use the top-k most essential experts, we do not need to propagate gradients and compute the output of each expert, which significantly reduces the training and inference cost. A complexity analysis can be found in Appendix C.

Since the last expert and gating are randomly initialized, the model may ignore them because they produce meaningless predictions at the beginning. To mitigate this, we follow Sparse MoE [26] to tweak the gating values during training randomly so all experts have proper selection chances, and the new experts and gates can gradually learn to correctly predict the new data block. The details about the load balancing can be found in Appendix C.

4 Related Work

Incremental Learning is extensively explored in the deep learning literature, including computer vision [17, 19, 21] and natural language processing [14, 22, 29]. The approaches can be roughly divided into three categories: **Regularization-based** methods constrain the deviation of the new model from the trained model to retain knowledge [1, 17, 42]; **Experience-Replay** approaches add a small subset of previous data blocks to the current training set as a way to maintain previous knowledge [5, 21, 24]; **Architectural** approaches maintain learned knowledge via assigning model parameters to specific data [2, 7, 19]. Our method falls into the architectural category. Some existing work also considers separate modules for each data block [2, 25], but they focus on the task-incremental scenario, while our method handles both that, and the more challenging class-incremental case. MoE architecture has been applied to solve continual learning problem [40], but it does not use data block information to account for structural shift in graph incremental learning, whereas our approach handles this well.

Graph Incremental Learning. Different from i.i.d. data, graph data suffer from distribution shifts in the incremental learning setting. To overcome this novel challenge, architectural approaches including, PI-GNN [43], FGN [34], and HPN [46], use newly initialized model components to learn new knowledge. Experience replay approaches like DyGRAIN [16], ER-GNN [48], and Continual GNN [35] explicitly retrains old nodes selected from graph-related criterion. Regularization approaches such as TWP [20], GraphSail [39], GPIL [31], and SEM [47] identify and minimize a regularization loss to mediate structural shift and correct predictions. MSCGL [4] combines architectural search and regularization to preserve learned knowledge. However, because these models treat old models as inseparable units, they ignore different interaction types between data blocks. Meanwhile, our experts are dedicated to individual data blocks, facilitating conditional adaptation to new data.

5 Experiments

We aim to answer the following research questions in the experimental evaluation: **Q1**: Does the proposed DyMoE framework achieve good empirical performance while maintaining good efficiency? **Q2**: How does the memory size impact the performance of the model? **Q3**: The framework has several components, how does each component impact its behavior? **Q4**: Does our training strategy actually encourage dedicated experts? Implementation details and data descriptions can be found in Appedix D.

5.1 Quantitative Results

To answer **Q1**, we evaluate the model performance with average accuracy (AA) and average forget (AF) on class incremental datasets (CoraFull [37], Reddit [11], Arxiv [12], DBLP-small [32]), and data incremental datasets (Paper100M[12], Elliptic [37], Arxiv, DBLP-small). We compared experience-replay baseline (ER-GNN [48]), architectural baselines (LWF [19], PI-GNN [43]), and compound baselines (continual-GNN (C-GNN) [35], RCL-CN [23], SSRM [28]). We also compared with the pretrain baseline, where we only train the model on the first date block and infer all future data blocks; the online baseline, where we directly fine-tune the old model with new data blocks; and the retrain baseline, where we retrain on all data blocks whenever new data blocks arrive. We provide the results of DyMoE module with a similar number of active parameters as baseline methods and a larger version whose individual experts are of same size as the baselines (DyMoE-L). They both have three active experts (k=3). DyMoE represents a fair setting, where our method's and baselines' parameter size and computation time are

Table 1: AA and AF of class incremental datasets. Bold represents best baseline and underline represents runner-up.

Dataset	Metric	Pretrain	Online	LWF	ER-GNN	SSRM	RCL-CN	PI-GNN	C-GNN	Retrain	DyMoE	DyMoE-L
	Params.	4.3M	4.3M	N/A	4.3M	4.3M	4.7M	N/A	4.5M	4.3M	4.6M	8.7M
CoraFull	AA	17.58±1.59	34.37±0.69	38.61±1.28	64.57±0.60	70.71±0.59	67.37±0.62	70.92±0.60	73.98±0.61	83.07±0.79	76.59±0.58	**78.16**±0.61
	AF	0.00±0.00	-14.64±0.79	-13.39±0.98	-11.92±0.57	-7.85±0.59	-11.78±0.57	-9.76±0.56	-9.28±0.54	-0.35±0.09	-7.92±0.54	**-7.31**±0.57
Reddit	AA	32.19±2.93	24.99±0.45	42.58±3.95	80.10±0.15	86.55±0.14	83.46±0.18	87.32±0.16	89.15±0.16	98.17±0.10	92.84±0.13	**94.15**±0.17
	AF	0.00±0.00	-33.91±0.19	-29.60±1.19	-6.60±0.19	-2.60±0.20	-6.36±0.19	-4.36±0.23	-4.07±0.21	0.14±0.04	-2.60±0.19	**-1.96**±0.22
Arxiv	AA	27.91±2.47	34.79±3.76	40.21±2.63	55.39±1.82	62.43±1.83	58.20±1.81	62.72±1.84	65.18±1.83	72.19±0.20	68.33±1.80	**70.15**±1.84
	AF	0.00±0.00	-32.74±2.37	-28.13±3.08	-21.06±1.75	-9.93±1.73	-20.81±1.73	-15.37±1.72	-14.44±1.75	0.47±0.18	-10.06±1.70	**-8.82**±1.74
DBLP	AA	46.03±1.89	47.52±3.63	50.48±3.30	54.81±3.03	56.56±3.06	55.80±3.03	56.35±3.02	57.62±3.04	65.59±1.27	57.75±3.01	**58.08**±3.05
	AF	0.00±0.00	-17.41±2.86	-14.28±2.46	-8.45±0.86	-5.43±0.83	-8.27±0.85	-6.69±0.83	-6.39±0.84	0.47±0.04	-5.45±0.82	**-5.07**±0.85

Table 2: AA and AF of instance incremental datasets. Bold represents best baseline and underline represents runner-up.

Dataset	Metric	Pretrain	Online	LWF	ER-GNN	SSRM	RCL-CN	PI-GNN	C-GNN	Retrain	DyMoE	DyMoE-L
	Params.	4.3M	4.3M	N/A	4.3M	4.3M	4.7M	N/A	4.5M	4.3M	4.6M	8.7M
Paper100M	AA	58.61±1.98	66.10±4.51	74.86±2.35	77.25±1.32	78.92±1.31	78.09±1.29	79.13±1.31	79.94±1.31	86.15±0.49	80.57±1.29	**81.24**±1.30
	AF	0.00±0.00	-3.97±0.43	-2.69±1.92	-4.08±0.08	-2.03±0.10	-3.96±0.09	-3.05±0.07	-2.80±0.07	-0.35±0.04	-2.08±0.05	**-1.65**±0.07
Elliptic	AA	89.91±2.41	94.37±0.13	94.79±0.16	94.37±0.05	95.11±0.02	95.17±0.05	95.67±0.05	95.64±0.05	98.13±0.03	95.42±0.01	**96.12**±0.04
	AF	0.00±0.00	-0.98±0.88	-1.96±0.14	-1.03±0.19	0.10±0.19	-0.78±0.20	-0.32±0.19	-0.26±0.19	0.14±0.02	-0.10±0.18	**0.23**±0.21
Arxiv	AA	59.81±2.32	69.05±0.39	**70.06**±0.64	66.39±0.21	67.52±0.19	67.33±0.23	68.24±0.20	68.24±0.23	73.01±0.10	68.21±0.19	68.56±0.23
	AF	0.00±0.00	-2.31±0.18	-1.70±0.42	-0.23±0.37	**0.41**±0.36	-0.13±0.39	0.20±0.37	-0.01±0.37	0.34±0.29	-0.01±0.35	0.24±0.38
DBLP	AA	55.73±2.15	63.42±1.61	65.15±1.76	64.19±0.49	66.62±0.53	65.12±0.51	66.70±0.51	67.30±0.50	68.59±1.27	67.97±0.49	**68.94**±0.50
	AF	0.00±0.00	-3.57±0.27	-2.78±0.85	-3.74±0.74	-2.51±0.76	-3.73±0.76	-3.07±0.74	-2.94±0.77	0.29±0.04	-2.39±0.73	**-2.33**±0.76

very close. We set the same memory node budget for all baselines, the memory budget for each dataset can be found in Appendix D

We show the experiment results of class incremental setting in Table 1, the Params. row shows the active parameters of the baselines, LWF and PI-GNN's parameter sizes can increase indefinitely with the number of datablocks, hence we leave it as "N/A". From the results, we can see our method significantly improves over most existing baselines for both AA and AF. We reach an average of 3.18% and 4.92% relative performance improvement in AA across datasets for DyMoE and DyMoE-L, respectively. The relative improvement is computed by $\frac{AA_{DyMoE}-AA_{B1}}{AA_{B1}}$, where AA_{B1} is the best baseline result. While DyMoE does not reach unanimous superiority on AF, we observe that SSRM, obtaining better AF, has a much lower AA. DyMoE on average improves over SSRM on AA by 6.78%, whereas the decrease in AF is less than 1%. Meanwhile, DyMoE outperforms other baselines on AF by large margins. This shows that DyMoE is a more effective trade-off between stability and plasticity. The solid empirical results showed the superiority of the DyMoE design and validated our theory. Comparing DyMoE and DyMoE-L, we see that DyMoE-L achieves better AA and AF, showing that a larger parameter size is more ideal to learn new knowledge without interfering with the old. For the results of instance incremental learning in Table 2, DyMoE still achieves better results on most targets, including AA on paper100M, Elliptic, and DBLP, while maintaining competitive results on AF. On datasets

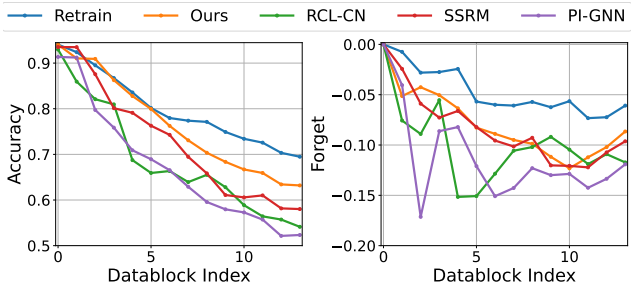

Figure 4: Performance progression on CoraFull dataset over data blocks of our models and baselines.

Table 3: Training and inference time. (Ave. Seconds/Epoch).

	CoraFull		Arxiv		Reddit	
	Train	Inference	Train	Inference	Train	Inference
Finetune	2.41	1.49	3.59	8.96	8.79	4.57
SSRM	2.46	1.48	3.94	8.79	9.86	4.65
PIGNN	2.95	2.06	4.59	12.59	10.57	5.91
Retrain	6.47	1.52	17.18	8.94	35.18	4.63
DyMoE	2.47	1.55	4.17	8.86	10.28	4.75
DyMoE-L	2.94	2.14	4.68	13.09	13.91	6.02

where DyMoE did not achieve superior results, like Arxiv, we observe that even the most naive baseline (Online) achieved higher

Table 4: Ablation study of class incremental and instance incremental datasets.

	Reddit		Cora-Full		Paper100M	
	AA	AF	AA	AF	AA	AF
DyMoE	92.84±0.13	**-2.60±0.19**	76.59±0.58	-7.92±0.54	80.57±1.29	-2.08±0.05
DyMoE-$\gamma = 0$	91.38±0.58	-3.28±0.36	74.39±0.58	-7.91±1.12	78.21±1.53	-3.31±1.58
DyMoE-$\delta = 0$	89.94±1.28	-4.13±0.42	72.56±0.65	-8.47±0.58	78.49±1.47	-3.49±1.79
DyMoE-$\gamma = 0, \delta = 0$	90.18±0.58	-5.19±1.38	72.17±1.14	-10.62±1.20	76.12±0.39	-4.34±1.76

performance than other advanced continual learning methods. We suspect that forgetting was not a severe issue in these datasets, and most continual learning methods enforce a mechanism to maintain the old knowledge, which "over-regularizes" the learning process, causing underfitting and degraded performance.

In Table 3, we show the training and inference time of the baselines and our model. Compared to the retrain baseline (performance upper-bound), both DyMoE and DyMoE-L cost significantly less training time, as the method requires significantly less memory data compared to the retrain method to maintain a competitive performance. DyMoE has very close training and inference time comparing to most effective baselines (SSRM, PIGNN), while achieving better results on most targets. Note that C-GNN is not compared here because it theoretically has the same complexity as Finetune. DyMoE-L, due to a higher parameter size, requires more computation costs during inference, however, DyMoE-L's cost is comparable to that of the architectural approach PIGNN, while maintaining higher performance. Furthermore, we plot the AA and AF with respect to the data block sequences in Figure 4. We observe that DyMoE can align with the upper-bound retrain method in the first few data blocks in AA, while demonstrating a large margin over other baselines. SSRM achieves better AF, but we can see that in the last few datablocks, DyMoE's forgetting is better, meaning that DyMoE can be more advantageous in maintaining knowledge in a longer data block sequence.

5.2 Investigation of DyMoE

To answer **Q2**, we compare our method with three other baselines, SSRM, PI-GNN, and C-GNN, that use memory nodes to help retain old knowledge. An ideal incremental learning method should only use a small memory size to obtain desirable performance. We plot the results with different memory portion in Figure 6. From the results, we can see that our approach achieves better performance with the same size of memory, especially when we only have 0.01 memory portion of the training data block on the Reddit dataset. Note that when the memory size approaches infinity, all methods become retrained, and hence we are seeing a converging pattern for the baselines. To answer **Q3**, we first investigate the effect of the gating mechanism in DyMoE. To ensure that the nodes are routed to the correct experts and experts only receive nodes that they are familiar with during message-passing, we enforce block-guided loss and graph block-guided loss to provide supervision for the gating mechanism. We study the model performance when these losses are absent from the training process. The results are in Table 4. We observe that removing either γ or δ leads to performance degradation. When both losses are absent, we usually observe the lowest

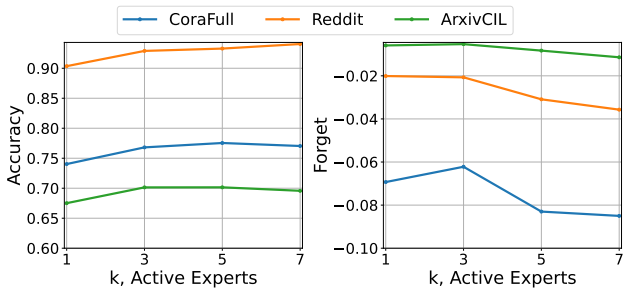

Figure 5: Performance change w.r.t. # of active experts.

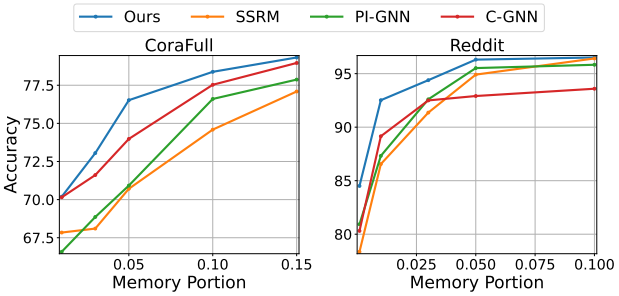

Figure 6: Model accuracy versus memory size for memory-based models.

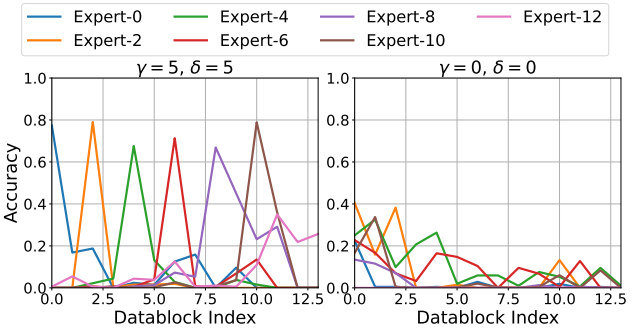

Figure 7: Experts individual performance on data blocks. Each line plot represents an expert.

performance. This shows the necessity of directly using block information as supervision, which is largely ignored in most existing approaches. We also observe that δ has a stronger impact on the model behavior, as without it the model experiences performance drop, showing the necessity of dedicated experts.

We then investigate how the model reacts to the number of active experts, and show the performance evolution as we increase the number of experts in Table 5. We can see that the AA increases from $k = 1$ to $k = 5$, while AF begins to decrease from $k = 3$. The divergence between AA and AF after $k = 3$ shows that more experts make learning easier, potentially because new experts can borrow more knowledge from the old experts. Meanwhile, it is more difficult to maintain old knowledge. We suspect that this is because extra experts inevitably introduce noises and irrelevant knowledge to old representations, causing forgetting.

Q4 validates whether our model and training procedure results in specialized experts as designed. We evaluate the performance

of each expert on individual data blocks. In Figure 7, we can easily observe that when trained with block-guided loss, the experts are specialized, and they achieve high prediction accuracy for their corresponding data blocks. When the block-guided loss is absent, the experts fail to specialize, further demonstrating the necessity of using block-information to guide training the MoE for continual learning.

6 Conclusion, Limitations, and Future Work

In this paper, we identified the drawbacks of existing graph incremental learning models and proposed the DyMoE module with a sparse version to model different interaction types between data blocks effectively and efficiently. However, we also acknowledge that our model may have trouble locating the correct experts when there are too many data blocks, resulting in compromised performance. While this can be solved by periodic retraining, we plan to extend our work to handle extremely long data sequences (over 1000 data blocks) in future work.

References

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

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. 2017. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3366–3375.

[3] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*.

[4] Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong, and Wenwu Zhu. 2022. Multimodal Continual Graph Learning with Neural Architecture Search. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 1292–1300. doi:10.1145/3485447.3512176

[5] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. 2021. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 6993–7001.

[6] Yuanning Cui, Yuxin Wang, Zequn Sun, Wenqiang Liu, Yiqiao Jiang, Kexin Han, and Wei Hu. 2023. Lifelong embedding learning and transfer for growing knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4217–4224.

[7] Sayna Ebrahimi, Franziska Meier, Roberto Calandra, Trevor Darrell, and Marcus Rohrbach. 2020. Adversarial continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*. Springer, 386–402.

[8] Falih Gozi Febrinanto, Feng Xia, Kristen Moore, Chandra Thapa, and Charu Aggarwal. 2023. Graph lifelong learning: A survey. *IEEE Computational Intelligence Magazine* 18, 1 (2023), 32–51.

[9] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B R Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L Blundell, Michael M Bronstein, and Jake P Taylor-King. 2021. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics* 22, 6 (05 2021), bbab159. doi:10.1093/bib/bbab159 arXiv:https://academic.oup.com/bib/article-pdf/22/6/bbab159/41087478/bbab159.pdf

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[12] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2021. Open Graph Benchmark: Datasets for Machine Learning on Graphs. arXiv:2005.00687 [cs.LG] https://arxiv.org/abs/2005.00687

[13] Weiwei Jiang and Jiayun Luo. 2022. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications* 207 (2022), 117921. doi:10.1016/j.eswa.2022.117921

[14] Zixuan Ke and Bing Liu. 2022. Continual learning of natural language processing tasks: A survey. *arXiv preprint arXiv:2211.12701* (2022).

[15] Dongwan Kim and Bohyung Han. 2023. On the stability-plasticity dilemma of class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20196–20204.

[16] Seoyoon Kim, Seongjun Yun, and Jaewoo Kang. 2022. DyGRAIN: An Incremental Learning Framework for Dynamic Graphs.. In *IJCAI*. 3157–3163.

[17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526. doi:10.1073/pnas.1611835114 arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114

[18] Xiaoyu Kou, Yankai Lin, Shaobo Liu, Peng Li, Jie Zhou, and Yan Zhang. 2020. Disentangle-based Continual Graph Representation Learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2961–2972.

[19] Zhizhong Li and Derek Hoiem. 2018. Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 12 (2018), 2935–2947. doi:10.1109/TPAMI.2017.2773081

[20] Huihui Liu, Yiding Yang, and Xinchao Wang. 2021. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 8653–8661.

[21] David Lopez-Paz and Marc' Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/f87522788a2be2d171666752f97ddebb-Paper.pdf

[22] Fei Mi, Liangwei Chen, Mengjie Zhao, Minlie Huang, and Boi Faltings. 2020. Continual learning for natural language generation in task-oriented dialog systems. *arXiv preprint arXiv:2010.00910* (2020).

[23] Appan Rakaraddi, Lam Siew Kei, Mahardhika Pratama, and Marcus De Carvalho. 2022. Reinforced continual learning for graphs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1666–1674.

[24] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. 2019. Experience Replay for Continual Learning. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf

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

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[27] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. 2020. Masked Label Prediction: Unified Massage Passing Model for Semi-Supervised Classification. *CoRR* abs/2009.03509 (2020). arXiv:2009.03509 https://arxiv.org/abs/2009.03509

[28] Junwei Su, Difan Zou, Zijun Zhang, and Chuan Wu. 2023. Towards robust graph incremental learning on evolving graphs. In *International Conference on Machine Learning*. PMLR, 32728–32748.

[29] Jingyuan Sun, Shaonan Wang, Jiajun Zhang, and Chengqing Zong. 2020. Distill and replay for continual language learning. In *Proceedings of the 28th international conference on computational linguistics*. 3569–3579.

[30] Li Sun, Junda Ye, Hao Peng, Feiyang Wang, and S Yu Philip. 2023. Self-supervised continual graph learning in adaptive riemannian spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4633–4642.

[31] Zhen Tan, Kaize Ding, Ruocheng Guo, and Huan Liu. 2022. Graph few-shot class-incremental learning. In *Proceedings of the fifteenth ACM international conference on web search and data mining*. 987–996.

[32] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Las Vegas, Nevada, USA) *(KDD '08)*. Association for Computing Machinery, New York, NY, USA, 990–998. doi:10.1145/1401890.1402008

[33] Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. 2022. Three types of incremental learning. *Nature Machine Intelligence* 4, 12 (2022), 1185–1197.

[34] Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. 2022. Lifelong graph learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 13719–13728.

[35] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 1515–1524.

[36] Shoujin Wang, Liang Hu, Yan Wang, Xiangnan He, Quan Z. Sheng, Mehmet A. Orgun, Longbing Cao, Francesco Ricci, and Philip S. Yu. 2021. Graph Learning based Recommender Systems: A Review. arXiv:2105.06339 [cs.IR] https://arxiv.org/abs/2105.06339

[37] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[38] Man Wu, Xin Zheng, Qin Zhang, Xiao Shen, Xiong Luo, Xingquan Zhu, and Shirui Pan. 2024. Graph Learning under Distribution Shifts: A Comprehensive Survey on Domain Adaptation, Out-of-distribution, and Continual Learning. *arXiv preprint arXiv:2402.16374* (2024).

[39] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. 2020. GraphSAIL: Graph Structure Aware Incremental Learning for Recommender Systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 2861–2868. doi:10.1145/3340531.3412754

[40] Jiazuo Yu, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. 2024. Boosting continual learning of vision-language models via mixture-of-experts adapters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 23219–23230.

[41] Qiao Yuan, Sheng-Uei Guan, Pin Ni, Tianlun Luo, Ka Lok Man, Prudence Wong, and Victor Chang. 2023. Continual graph learning: A survey. *arXiv preprint arXiv:2301.12230* (2023).

[42] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *International conference on machine learning*. PMLR, 3987–3995.

[43] Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, Guojie Song, and Sunghun Kim. 2023. Continual learning on dynamic graphs via parameter isolation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 601–611.

[44] Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. 2024. Topology-aware Embedding Memory for Continual Learning on Expanding Networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4326–4337.

[45] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2022. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems* 35 (2022), 13006–13021.

[46] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2023. Hierarchical Prototype Networks for Continual Graph Representation Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2023), 4622–4636. doi:10.1109/TPAMI.2022.3186909

[47] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2023. Ricci curvature-based graph sparsification for continual graph representation learning. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

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

A Proof of Theorem 1

We restate Theorem 1 for completeness,

Theorem 1. *For an arbitrary continual learning problem, suppose a PI model obtains a cross-entropy classification loss $\mathcal{L}_{PI}$, there exists a parametrization of DyMoE that achieves cross-entropy classification loss $\mathcal{L}_{Dy} = \mathcal{L}_{PI}$. When the data sequence follows a mixture of Gaussian distribution, we have $\mathcal{L}_{Dy} \leq \mathcal{L}_{PI}$.*

It is easy to see that DyMoE is at least as powerful as PI since we can parameterize all gating vectors with the same value; hence, the weights of all experts are the same, which makes the final output essentially a summation of each expert's output. In this case, DyMoE degenerates to PI. We then prove that under the Gaussian Mixture assumption of the data blocks, DyMoE achieves lower loss and, hence, is strictly more powerful than PI.

Proof. Consider the case with two data blocks generated from Gaussian Distributions, $X_1 = \mathcal{N}(\boldsymbol{\mu}_1, \sigma^2 I)$, and $X_2 = \mathcal{N}(\boldsymbol{\mu}_2, \sigma^2 I)$. For simplicity, we assume the same variance across coordinates and the probability of data from each distribution is the same. Let the distance between two distributions be B. The labels of data are generated depending on their distance from the mean of their coorresponding distribution, specifically, if $\boldsymbol{x} \sim X_1$,

$$y = \begin{cases} 0, & \text{if } ||\boldsymbol{x} - \boldsymbol{\mu}_1|| \leq d \\ 1, & \text{otherwise} \end{cases} \tag{17}$$

and if $x \sim X_1$,

$$y = \begin{cases} 2, & \text{if } ||\boldsymbol{x} - \boldsymbol{\mu}_2|| \leq d \\ 3, & \text{otherwise} \end{cases} \tag{18}$$

where $2d \leq B$ is a threshold distance to determine the data labels. This is a practical assumption for a mixture of two Gaussian distributions.

We then consider the procedure of Parameter Isolation (PI) and our proposed method. PI first trains a model $f_1(x)$ on X_1 and then trains a model $f_2(x)$ on X_2, both f_1 and f_2 are in R^4 for the four target classes. Hence, when making predictions, we have the logits to be:

$$y = \text{softmax}(f_1(x) + f_2(x)) \tag{19}$$

Since our approach can initialize a network with the same architecture, we can have the same network and parameters as the ones in PI, and the predictions from our model are:

$$\begin{aligned} y &= \text{softmax}(\alpha_1 f_1(\boldsymbol{x}) + \alpha_2 f_2(\boldsymbol{x})), \\ \alpha_i &= \frac{exp(-\frac{||\boldsymbol{x}-\boldsymbol{g}_i||^2}{2\sigma^2})}{exp(-\frac{||\boldsymbol{x}-\boldsymbol{g}_1||^2}{2\sigma^2}) + exp(-\frac{||\boldsymbol{x}-\boldsymbol{g}_2||^2}{2\sigma^2})} \end{aligned} \tag{20}$$

Here, we use the negative of the distance normalized by the variance as the similarity measure between the input and the gating vectors. This is a valid and tractable choice as variance can be estimated by batch normalization. Note that in the first data block, we directly set the gating vector to the empirical mean of the input, $\boldsymbol{g}_1 = \bar{\boldsymbol{x}}_1 \approx \boldsymbol{\mu}_1$. In the second data block, the block-guided loss solves the problem

$$\min_{\boldsymbol{g}_2} \frac{1}{N} \sum_{i=1}^{N} s(\boldsymbol{x}_i, \boldsymbol{g}_2) \tag{21}$$

which is minimized by $\boldsymbol{g}_2 = \boldsymbol{\mu}_2$. We can then rewrite the prediction of the model:

$$y = \text{softmax}(\alpha_1 f_1(\boldsymbol{x}) + \alpha_2 f_2(\boldsymbol{x})),$$
$$\alpha_i = \frac{exp(-\frac{||\boldsymbol{x}-\boldsymbol{\mu}_i||^2}{2\sigma^2})}{exp(-\frac{||\boldsymbol{x}-\boldsymbol{\mu}_1||^2}{2\sigma^2}) + exp(-\frac{||\boldsymbol{x}-\boldsymbol{\mu}_2||^2}{2\sigma^2})} \tag{22}$$

To show that our model achieves lower loss on this task, we only need to consider the expected loss on the D_1 as the two distributions are symmetric. We divide the problem into two cases $||\boldsymbol{x} - \boldsymbol{\mu}_1|| \leq d$, when the input is close to the distribution mean, and $||\boldsymbol{x} - \boldsymbol{\mu}_1|| > d$ when the input is farther away.

For $||\boldsymbol{x} - \boldsymbol{\mu}_1|| \leq d$, the correct label is 0. We consider the cross-entropy loss of PI,

$$\begin{aligned}\mathcal{L}_{PI} &= -log(\frac{f_1(\boldsymbol{x})_0}{f_1(\boldsymbol{x})_0 + f_1(\boldsymbol{x})_1 + f_2(\boldsymbol{x})_2 + f_2(\boldsymbol{x})_3}) \\ &= -log(\frac{\alpha_1 f_1(\boldsymbol{x})_0}{\alpha_1 f_1(\boldsymbol{x})_0 + \alpha_1 f_1(\boldsymbol{x})_1 + \alpha_1 f_2(\boldsymbol{x})_2 + \alpha_1 f_2(\boldsymbol{x})_3})\end{aligned} \tag{23}$$

and the cross-entropy loss of our method

$$\mathcal{L}_{Dy} = -log(\frac{\alpha_1 f_1(\boldsymbol{x})_0}{\alpha_1 f_1(\boldsymbol{x})_0 + \alpha_1 f_1(\boldsymbol{x})_1 + \alpha_2 f_2(\boldsymbol{x})_2 + \alpha_2 f_2(\boldsymbol{x})_3}) \tag{24}$$

since $||\boldsymbol{x} - \boldsymbol{\mu}_1|| \leq d \leq B - d \leq ||\boldsymbol{x} - \boldsymbol{\mu}_2||$, meaning that $\alpha_1 \geq \alpha_2$, and we have $\mathcal{L}_{Dy} \leq \mathcal{L}_{PI}$. Since cross-entropy is monotonic, we can obtain the minimum of $\mathcal{L}_{PI} - \mathcal{L}_{Dy}$ at $||\boldsymbol{x} - \boldsymbol{\mu}_1|| = d$ and $||\boldsymbol{x} - \boldsymbol{\mu}_2|| = B - d$. Let the minimum be $\Delta\mathcal{L}_{close}$. Note $\Delta\mathcal{L}_{close}$ increases as d increases and σ decreases.

For $||\boldsymbol{x} - \boldsymbol{\mu}_1|| > d$, the correct label is 1. Let M be the maximum absolute value that the neural network f_2 output for a logit, that is $|f_2(x)_y| \leq M$. Then, the maximum possible loss is $-f_2(x)_1 + log(C \cdot exp(M)) = log(C) + 2M$, where $C = 4$, the number of classes. Since logits of PI and our method is bounded by the same M, we have the maximum possible loss difference to be

$$|\Delta\mathcal{L}_{far}| = 4M + 2logC \tag{25}$$

We now developed a lower bound for the loss difference when $\boldsymbol{x}$ is close to $\boldsymbol{\mu}_1$ and an upper bound for the loss difference when x is far from $\boldsymbol{\mu}_2$, we then compute the probability of each case using Gaussian tail bound.

$$P[\boldsymbol{x}-\boldsymbol{\mu}_1 > d] \leq exp(-\frac{d^2}{2\sigma^2}), P[\boldsymbol{x}-\boldsymbol{\mu_1} \leq d] \geq 1-exp(-\frac{d^2}{2\sigma^2}) \tag{26}$$

Then the upper bound of the difference in expected loss when x is far is:

$$|\Delta E_{far}| \leq (4M + 2logC) \cdot exp(-\frac{d^2}{2\sigma^2}) \tag{27}$$

The lower bound of the expected loss when x is close is:

$$\Delta E_{close} \geq \Delta\mathcal{L}_{close} \cdot (1 - exp(-\frac{d^2}{2\sigma^2})) \tag{28}$$

Taking the ratio:

$$\frac{|\Delta E_{far}|}{\Delta E_{close}} \leq \frac{(4M + 2logC) \cdot exp(-\frac{d^2}{2\sigma^2})}{\Delta\mathcal{L}_{close} \cdot (1 - exp(-\frac{d^2}{2\sigma^2}))} \tag{29}$$

As $\frac{d}{\sigma}$ increases, the ratio approaches zero, hence we have the overall expected loss difference,

$$\Delta E = \Delta E_{close} + |\Delta E_{far}| \geq \Delta E_{close} - |\Delta E_{far}| \geq 0 \tag{30}$$

making the overall loss difference positive, and our approach leads to lower loss in this case. □

B Memory Set Construction

DyMoE utilizes memory sets to train gating vectors for correct data routing. To construct the memory, we first set a memory budget $0 < p < 1$, representing the portion of the full data block X that will be kept as memory set M, that is

$$|M| = p|X| \tag{31}$$

Usually, p is a small value ($p < 0.05$) to ensure the efficiency. Inspired by ER-GNN [48], we use a sample's representativeness to select memory nodes. Let X_c be all samples in X that has class c, we collect their learned representation before the final logit prediction layer, $F_c = f(x)|x \in X_c$. We then compute the representative vector $\boldsymbol{x}_c$ as

$$\boldsymbol{x}_c = \frac{1}{|F|} \sum_{\boldsymbol{c} \in F} \boldsymbol{c} \tag{32}$$

Then representativeness of sample $\boldsymbol{x}$ is determined by the norm distance between the sample representation and the representative vector, $s = -||f(\boldsymbol{x}) - \boldsymbol{x}_c||$. We sort this value in X_c, and pick the largest k_c samples to add to the memory set. k_c is determined by the class distribution.

$$k_c = \frac{|X_c|}{|X|} \tag{33}$$

A benefit of this construction strategy is that the resulting memory set $\bigcup_i M^{(i)}$ reflects the actual class distribution but with much less data. Hence, we can use it to balance the data during training. The training is divided into two stages, in the first stage, the model is trained with $\bigcup_{i=1}^{t-1} M^{(i)} \cup X^{(t)}$, so that the last expert can learn fine-grained information from the full new data block. Then, we collect memory for $X^{(t)}$ as described above and train the model with $\bigcup_{i=1}^{t} M^{(i)}$, this teaches the gating mechanism the correct class distribution of the current graph.

C Details about Sparse DyMoE

Since DyMoE in each individual timestamp can be interpreted as a conventional static sparse MoE model, we can apply the same strategy [26] to ensure experts get a good chance of being selected, including the randomly and newly initialized ones. Specifically,

$$\begin{aligned}(\alpha_1, ..., \alpha_t) &= Softmax(KeepTopK(H(\boldsymbol{x}, \boldsymbol{g}_1, \boldsymbol{q}_1), ..., H(\boldsymbol{x}, \boldsymbol{g}_t, \boldsymbol{q}_t)) \\ H(x, g_i, q_i) &= s(\boldsymbol{x}, \boldsymbol{g}_i) + StandardNormal() \cdot Softplus(s(\boldsymbol{x}, \boldsymbol{q}_i)) \\ KeepTopK(v_1, ..., v_t) &= \begin{cases} v_i, & \text{if i in TopK} \\ -\inf, & \text{otherwise} \end{cases}\end{aligned} \tag{34}$$

We add a noise term whose magnitude is determined by another learnable noise vector.

Modern MoE designs also employ load balancing design to ensure each experts get similar number of samples. However, we observe that such a strategy is not bringing performance boost to our methods, potentially because the supervised data-block signal already handles the load balancing issue.

D Experiment Details

D.1 Implementation Details

The repository for implementation can be found at the following https://github.com/amazon-science/dymoe-graph-incremental-learning. The model is implemented in PyTorch and DGL, and all experiments are conducted on 1 Nvidia A100 80GB GPU. We repeat the experiment 5 times using different random seeds and report the mean and standard deviation. We uniformly use a fan-out of 10 to extract subgraphs from each target node. The hyperparameters used during training are shown in Table 5 and Table 6, where the curly bracket represents the hyperparameters for searching, and the hyperparameters selected are marked in bold. Memory size is the per data block memory size, and it is a special hyperparameter in the continual learning setting because as it increases, all methods converge to the retrain method, which is usually the upper bound of all continual learning methods. We set a uniform ratio p of the training dataset for all methods use memory set.

D.2 Dataset Details

The dataset statistics are shown in Table 7. We collect data from academic graphs (Arxiv, DBLP, Paper100M, CoraFull), social networks (Reddit), and BlockChain networks (Elliptic) to show that our model handles a wide range of datasets. We describe the construction of each dataset as follows and includes the number of new nodes and edges in Figure 8 and 9. We use the provided train/valid/test split from the dataset source. If the source does not have established split, we use 60/20/20 train valid test split.

ArXiv: Arxiv academic citation network from Open Graph Benchmark (OGB) [12] contains arxiv articles and the citation information between articles. For instance incremental learning setting, we use the first 25 timestamps in the original arxiv dataset as the first data block, as they contain significantly less data. We then split the rest of the data by year, and data in each forms a data block. For class incremental learning setting, we split the data into 8 blocks each contains 5 classes.

DBLP: DBLP is an academic network from the DBLP website containing computer science academic paper, with citation information [32]. We follow Zhang et al. [43] to sample 20000 nodes with 9 classes and 75706 edges from DBLP full data, we split it into data blocks according to the timestamps. For the class incremental setting, we split the 9 classes into 5 data block each containing 2 classes, except for the last one with only 1 class.

Paper100M: Paper100M is a citation network extracted from Microsoft Academic Graph by OGB [12]. We follow Zhang et al. [43] to sample 12 classes from the year 2009 to the year 2019 from Paper100M full data and we split it into tasks according to the timestamps.

CoraFull: CoraFull is a co-citation academic network, where nodes are papers, and the two nodes are connected if they are co-cited by other papers [3]. We use the provided CoraFull data from DGL, and split its 70 classes into 14 5-classes data blocks for class incremental learning.

Reddit: The Reddit dataset contains Reddit posts as nodes, and two nodes are connected by edges if they are posted by the same user [11]. We use the provided Reddit data from DGL, and split its 40 classes into 8 5-classes data blocks for class incremental learning.

Elliptic: The Elliptic dataset is a bitcoin transaction network, where each node represents a transaction, and each edge denotes money flow [37]. Its nodes have timestamps evenly spaced with an interval about two weeks. We use the original timestamp from the dataset for instance-incremental learning.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Table 5: Hyperparameters for class incremental learning.

	Arxiv-CIL	DBLP-CIL	CoraFull	Reddit
Learning Rate	0.0001			
Weight Decay	{0.01, **0.001**, 0.0001}			
Embedding Dimension	128			
# Epochs	40			
# Balancing Epochs	10			
γ	{0.01, 0.1, **1**, 5}	{0.01, **0.1**, 1, 5}	{0.01, 0.1, 1, **5**}	{0.01, 0.1, **1**, 5}
δ	5			
p	0.01	0.01	0.05	0.01
batch size	128			

Table 6: Hyperparameters for instance incremental learning.

	Arxiv-IIL	DBLP-IIL	Paper100M	Elliptic
Learning Rate	0.0001			
Weight Decay	0.001			
Embedding Dimension	128			
# Epochs	40			
# Balancing Epochs	5			
γ	{0.01, **0.1**, 1, 5}	{**0.01**, 0.1, 1, 5}	{0.01, **0.1**, 1, 5}	{0.01, **0.1**, 1, 5}
δ	5			
p	0.01			
batch size	128			

Table 7: Dataset statistics.

	#. Nodes	#. Edges	#. Classes	#. Data blocks	#. Classes per block
CoraFull	19793	126842	70	14	5
Arxiv-CIL	169343	2332486	40	8	5
Reddit	232965	114615892	41	9	5
DBLP-small-CIL	20000	302862	9	5	2
Paper100M-small	49459	217420	12	11	NA
Arxiv-IIL	169343	2332486	40	11	NA
DBLP-small-IIL	20000	302826	9	24	NA
Elliptic	203769	468710	2	49	NA

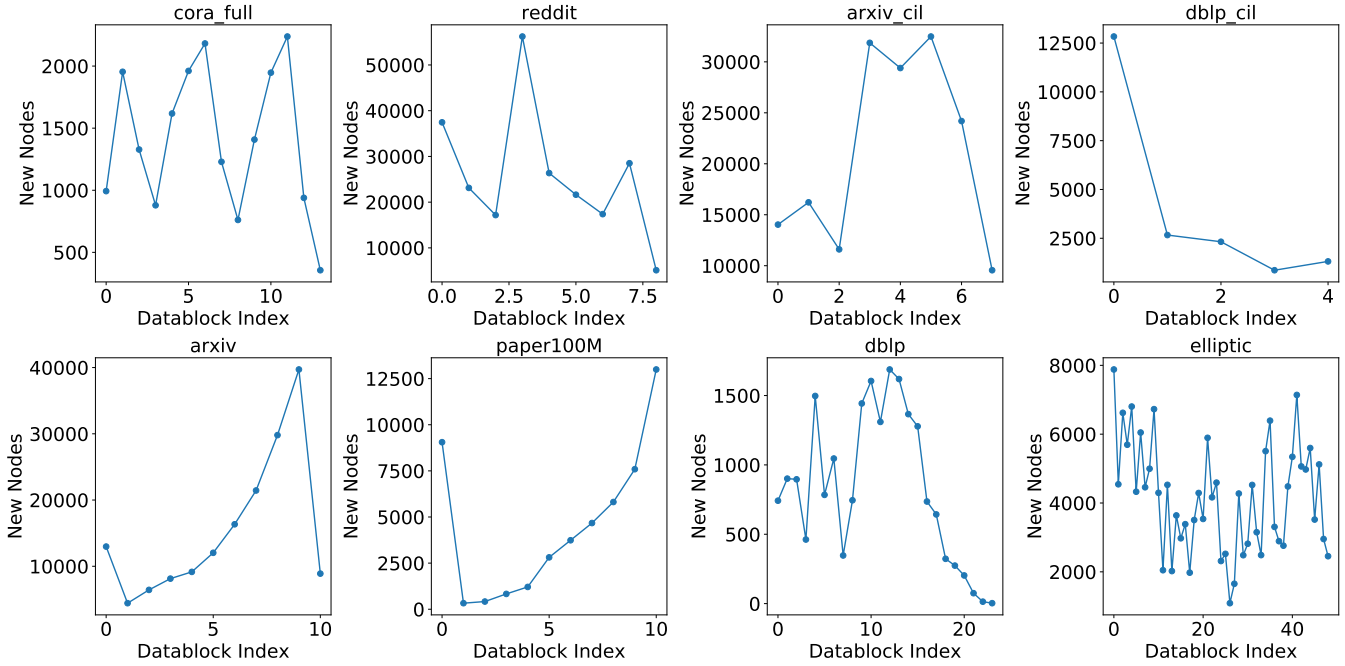

Figure 8: Number of new nodes per data block.

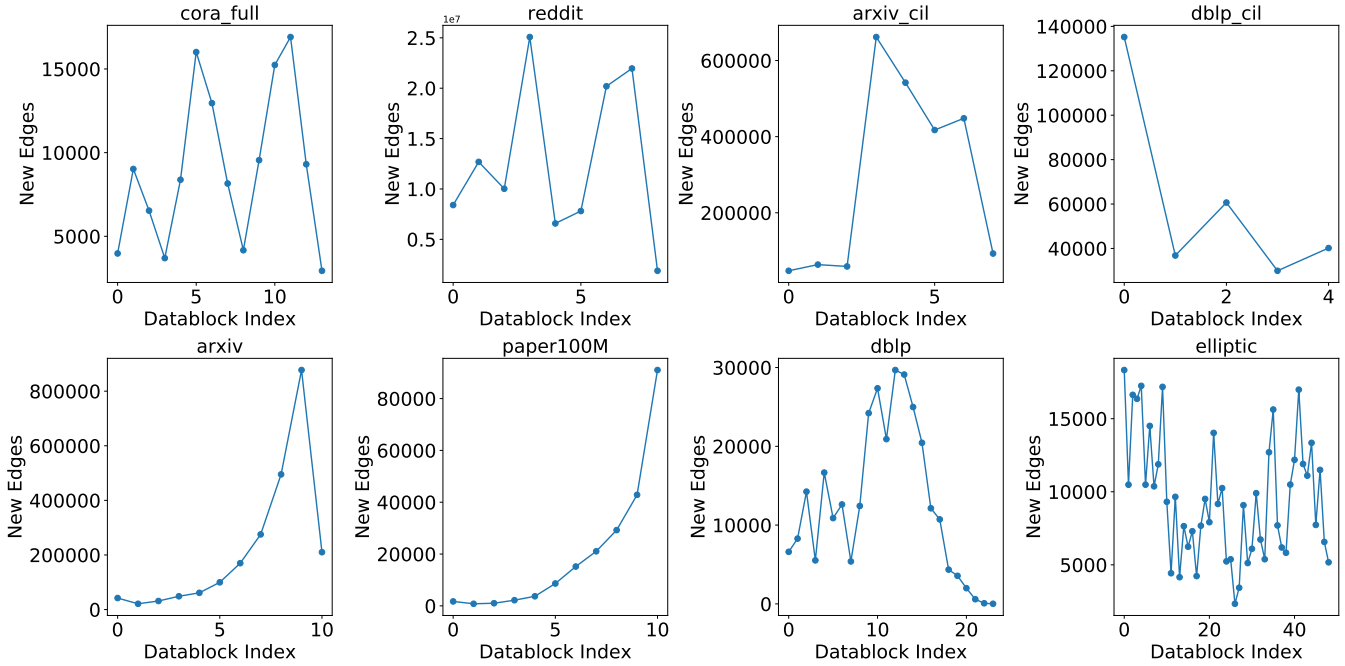

Figure 9: Number of new edges per data block.