Stochastic Experience-Replay for Graph Continual Learning

Arnab Kumar Mondal[*]
Fujitsu Research India Private Limited
`arnabkumarmondal123@gmail.com`

Jay Nandy[*]
Fujitsu Research India Private Limited
`jayjaynandy@gmail.com`

Manohar Kaul
Fujitsu Research India Private Limited
`Manohar.Kaul@fujitsu.com`

Mahesh Chandran[†]
Fujitsu Research India Private Limited
`mahesh.chandran@fujitsu.com`

Abstract

Experience Replay (ER) methods in graph continual learning (GCL) mitigate catastrophic forgetting by storing and replaying historical tasks. However, these methods often struggle with efficiently storing tasks in a compact memory buffer, affecting scalability. While recently proposed graph condensation techniques address this by summarizing historical graphs, they often inadequately capture variations within the distribution of historical tasks. In this paper, we propose a novel framework, called *Stochastic Experience Replay for GCL (SERGCL)*, by incorporating a *stochastic memory buffer (SMB)* that parameterizes a kernel function to estimate the distribution density of condensed graphs for each historical task. This allows efficient sampling of condensed graphs, leading to better coverage of historical tasks in the memory buffer and improved experience replay. Our experimental results on four benchmark datasets demonstrate that our proposed SERGCL framework achieves up to an 8.5% improvement of the *average performance* compared to the current state-of-the-art GCL models. Our code is available at: https://github.com/jayjaynandy/sergcl

1 Introduction

Graph Continual Learning (GCL) addresses the challenge of dynamic, time-evolving graphs where new tasks associated with novel subgraphs emerge over time [1, 2]. The goal of GCL is to learn and adapt to new tasks while preserving knowledge from previously encountered ones, without accessing the historical graphs. However, these models face the problem of *catastrophic forgetting*, where the emphasis on current tasks leads to the loss of previously acquired knowledge, significantly degrading performance on older tasks [3].

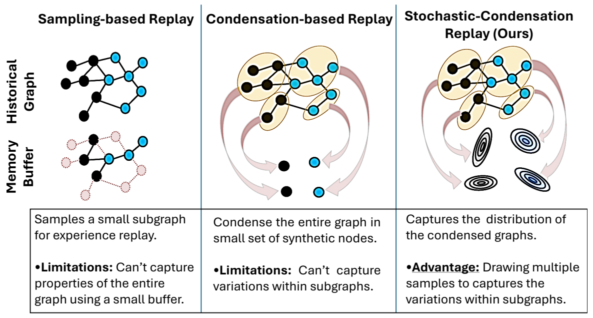

Sampling-based Replay
Condensation-based Replay
Stochastic-Condensation Replay (Ours)
Historical Graph
Memory Buffer
Samples a small subgraph for experience replay.
•Limitations: Can't capture properties of the entire graph using a small buffer.
Condense the entire graph in small set of synthetic nodes.
•Limitations: Can't capture variations within subgraphs.
Captures the distribution of the condensed graphs.
•Advantage: Drawing multiple samples to captures the variations within subgraphs.

Figure 1: Comparison of memory buffer creation strategies of different experience-replay based GCL models.

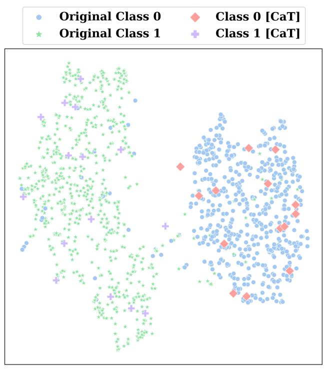
Addressing catastrophic forgetting in continual learning (CL) has been widely studied in computer vision [4–6], natural language processing [7], and reinforcement learning [8]. While Graph Continual Learning (GCL) shares the same goal of adapting over time with traditional CL, its non-Euclidean structure complicates the use of existing methods. GCL methods are broadly categorized as: *(a) Regularization-based methods* –which add penalty to preserve prior knowledge [3, 9]. *(b) Architecture-*

[*]Equal contribution.
[†]Corresponding author

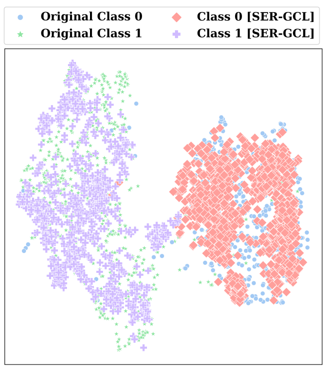
based methods – which fix parameters for historical tasks while adapting to new ones [10]. *(c) Replay-based methods* –which use a memory buffer to store and replay historical graphs [11, 12].

Several studies have demonstrated that experience replay-based models achieve the best performance among the three categories, exhibiting the lowest mean forgetting [1]. However, their ability to scale is constrained by the need to fit historical graphs within a small memory buffer (see Figure 1a). To address this, a few recent works [13, 14] have explored graph condensation techniques that synthetically create a compact "condensed graph" for each historical task, enabling more efficient experience replay. However, such a small static representative graph often fails to capture the full distribution of the historical sub-graphs. This raises an important question: ***How can we better approximate the underlying distribution of these sub-graphs within a small (stochastic) memory, from where we can easily sample for effective experience replay?***

This work introduces Stochastic Experience Replay for GCL (SERGCL), a novel framework incorporating a Stochastic Memory Buffer (SMB) to store a condensed graph distribution for each historical graph (Figure 1c). We present the condensed graph as a collection of synthetic nodes where the distributions can be captured using a learnable Gaussian kernel function for each node and stored in the SMB. It allows us to generate multiple condensed graph samples, effectively reconstructing the full range of historical nodes for improved experience replay. Figure 2 demonstrates the effectiveness of our proposed stochastic replay compared to the existing 'fixed' experience replay technique. While the 'fixed' condensation only provides the centers for each class of nodes of the historical graph, proposed 'stochastic' condensation regenerates the entire space of historical nodes for a better experience replay of historical tasks. The key contributions of our work are as follows:



(a) CaT



(b) SERGCL (ours)

Figure 2: Comparing UMAP visualization of the coverage for nodes of the historical graph corresponding to Task 0 (Arxiv dataset). (a) A Fixed, static condensed-based CaT [13] provides significantly lower coverage compared to (b) our stochastic condensation-based SERGCL.

- A stochastic memory buffer (SMB)-based framework is introduced for GCL that generalizes the fixed memory-based experience replay methods. For a fixed memory budget, SMB is shown to be an optimal strategy to maximize information storage.
- A stochastic update process (SUP) is proposed to draw multiple samples from the SMB, leading to maximal coverage of historical graphs for an effective experience replay.
- Experiments on four benchmark datasets—CoraFull, Arxiv, Reddit, and Products—demonstrate the superiority of the proposed SERGCL over prior methods. While existing approaches generally perform well in task-incremental settings, our method also achieves high performance in class-incremental settings, improving *average performance* by up to approximately 8.5% compared to current state-of-the-art techniques.

2 Background & Related Work

2.1 Preliminaries

Graph Continual Learning. Based on the downstream tasks, GCL can be categorized into *graph-level GCL (gGCL)* and *node-level GCL (nGCL)*. *gGCL* considers each graph as an independent instance and makes graph-level classification, thus resembling the traditional CL setup [1]. In contrast, *nGCL* differs significantly from the traditional CL as it focuses on node classification on a single, time-evolving graph, and the new tasks contain different disjoint node classes. In this paper, we focus on *nGCL* problem, as described below.

Node-level GCL. Consider an expanding graph, $\mathcal{G}$, which arrives as a sequence of sub-graphs, $\{G_1, G_2, \cdots, G_T\}$, where $G_t := \{V_t, A_t, Y_t\}$ represents the sub-graph for the t-th downstream task. Here, V_t denotes the node features, A_t is the adjacency matrix of G_t, and Y_t is the vector of node labels from the label-space, $\mathcal{Y}_t$. Each node $v \in V_t$ is associated with a class $y \in \mathcal{Y}_t$. The graph G_t is accessible only during the training process of t-th task. After receiving t-th downstream task, GNN parameters are updated from θ^{t-1} to θ^t to adapt to G_t while preserving knowledge from previous tasks. Our paper focuses on *task-incremental (task-IL)* and *class-incremental (class-IL)* settings, as described below (see Figure 3):

• ***Task-IL nGCL*** involves classifying a test node v given the task-id, t into the label space $\mathcal{Y}_t$, *i.e.*, $f : (v, t) \rightarrow \mathcal{Y}_t$. E.g., in Figure 3, if a test node with task-ID of 2, the model should classify it into one of Class 3 or Class 4.

• ***Class-IL nGCL*** classifies a test node v accross all seen classes, $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_2 \cup \cdots \mathcal{Y}_T$ where $\mathcal{Y}_t \cap \mathcal{Y}_{t'} = \phi$, *i.e.*, $f : (v) \rightarrow \mathcal{Y}$. E.g., in Figure 3, a test node in class-IL should be classified among the classes $\{1, 2, 3, 4, 5, 6\}$.

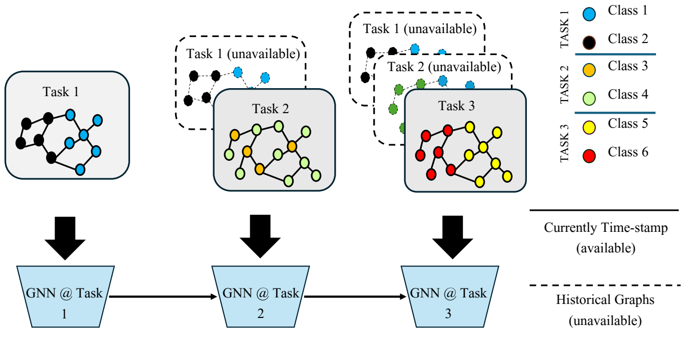


Figure 3: Illustration of Task-IL and Class-IL settings in node-level GCL: ***Task-IL*** requires classifying a test node based on its task-ID (e.g., task-ID = 1 implies classification among classes $\{1, 2\}$). ***Class-IL*** involves classifying a test node among all classes (*i.e.*, among six classes) without task-ID information.

Clearly, *class-IL nGCL* is more challenging as the classifier must predict among all possible classes without relying on the task ID. In contrast, *task-IL nGCL* restricts the classification to a smaller label space specific to the provided task ID. Existing work has generally focused on task-IL, often bypassing the complexities associated with class-IL nGCL settings.

2.2 Previous Work

2.2.1 Graph Continual Learning

Existing models for Graph Continual Learning (GCL) can be broadly classified into three categories:

(a) Regularization-based Approaches. These models incorporate regularization terms to balance between retaining knowledge of previous tasks and learning new ones, often by preserving a copy of the previous model. Regularizers may involve weight-based methods that selectively preserve network parameters based on their importance [3, 9, 15]. Examples include Elastic Weight Consolidation (EWC), which uses the Fisher information matrix to assess parameter importance [3], and Topology-Aware Weight Preserving (TWP), which maintains graph topology related to old tasks to mitigate forgetting [15]. Additionally, distillation-based regularizers [16] penalize deviations in latent representations from historical tasks [17–20].

Limitations. **(i)** Strong regularization may impede the model's ability to adapt to new tasks [1, 21]. **(ii)** Regularization-based methods often struggle with tasks that exhibit significant distributional differences. **(iii)** These methods are generally ineffective in class-incremental (class-IL) settings as they do not develop mechanisms for distinguishing between classes from different tasks.

(b) Architecture-based Approaches. These methods prevent catastrophic forgetting by isolating network parameters for each incoming task, either through task-specific parameter preservation or by expanding the original network [10, 22].

Limitations. **(i)** Determining when to share or reuse model parameters remains challenging [21], and expanding the model increases the parameter space significantly, complicating efficient training. **(ii)** Similar to regularization-based models, architecture-based methods also face difficulties in class-IL settings without mechanisms to store and replay prior information.

(c) Replay-based Approaches. These models use a lightweight memory buffer to store prior information and replay it while learning new tasks, addressing the issue of catastrophic forgetting

[11, 23–25]. Examples include methods that retrieve episodic memories linked to specific experiences, such as subsets of raw samples [11, 23, 25].

Limitations. Some methods, such as ER-GNN [11], select subgraphs based on coverage or influence maximization, while others (e.g., SSM [12]) create a sparse version of the original graph for replay purposes. Recently, DSLR [25] proposed to select a diverse set of real nodes to provide better coverage. However, identifying candidate subgraphs that effectively capture the properties of historical graphs for efficient experience replay within a compact memory remains challenging [1]. [26] introduced a single generative model to replay synthetic historical samples in the context of streaming GNNs. However, it often suffers from mode collapse and fails to fully capture data distributions. Additionally, CaT [13] and PUMA [14] investigated condensing historical input graphs to facilitate experience replay in graph continual learning.

2.2.2 Dataset condensation

Condensation techniques have been predominantly explored in computer vision [27–31]. These methods create a condensed synthetic dataset that allows models to train efficiently for downstream tasks, significantly reducing computation and storage costs. Some continual learning (CL) methods in computer vision also explored condensation to store lightweight data for replaying historical tasks [32–34]. However, graph condensation is still relatively underexplored. Recent works [35–38] have proposed gradient-matching techniques for graph condensation. SFGC [39] and GEOM [40] use trajectory matching by training a set of GNNs on original graphs to obtain their offline expert trajectory to be followed to learn their condensed graphs. Recently, CaT[13], PUMA [14] investigated graph condensation techniques for graph continual learning (GCL), demonstrating significant improvements over existing replay-based GCL models by efficiently generating condensed graphs for storing and replaying historical graphs within a limited memory budget. Notably, initial works (e.g., GCond [35]) learned synthetic edges. However, later works, such as SFGC [39] and CaT [13], indicate that structural information can be effectively captured in synthetic nodes, allowing the topology to be represented by an identity matrix. They preserve the latent representations of the original graph while improving the performance without explicit edges.

Nevertheless, these approaches generally rely on a fixed synthetic dataset for condensation, without aiming to learn the underlying distributions. Such a fixed synthetic graph often captures only the aggregated properties of subgraphs from a large historical graph. Our paper addresses this limitation by proposing a method for learning a condensed graph distribution, as detailed in Section 3.

3 Proposed Method

The proposed *Stochastic Experience Replay for Graph Continual Learning (SERGCL)* involves a two-step process, illustrated in Figure 4. The first step, ***(a) Stochastic Memory Buffer (SMB) Creation***, involves creating a memory buffer that stores the distribution over the condensed versions of historical tasks. The second step, ***(b) Stochastic Update Process (SUP)***, updates the GNN for downstream tasks using the SMB. These processes are described in detail below.

3.1 Stochastic Memory Buffer (SMB) Creation

Condensed Graph is a small, synthetic graph, $\tilde{G}_t = \{\tilde{V}_t, \tilde{A}_t, \tilde{Y}_t\}$ representing the original graph, $G_t = \{V_t, A_t, Y_t\}$ for task-ID t, such that a GNN trained using G_t or $\tilde{G}_t$ would achieve similar performance [35]. The node-labels for both G_t and $\tilde{G}_t$, denoted as Y_t and $\tilde{Y}_t$, share the same label space, $\mathcal{Y}_t$. For notational brevity, we drop the task-ID subscript t in the following discussions.

Stochastic Memory Buffer (SMB) is defined as a distribution over the condensed graph i.e., $\mathcal{P}_{\tilde{G}|y}$. Our objective is to learn such $\mathcal{P}_{\tilde{G}|y}$ for an input G to facilitate the sampling process for better coverage of historical graphs.

3.1.1 Graph Condensation Optimization

We can learn a condensed graph $\tilde{G}$ for input G by minimizing the divergence between their node representations [29, 30] using GNN, f, *i.e.,*

$$\min_{\tilde{G}} \underset{v\sim\mathcal{P}_{G|y},\tilde{v}\sim\mathcal{P}_{\tilde{G}|y}}{\mathrm{Div}} \big(f(v), f(\tilde{v})\big) \tag{1}$$

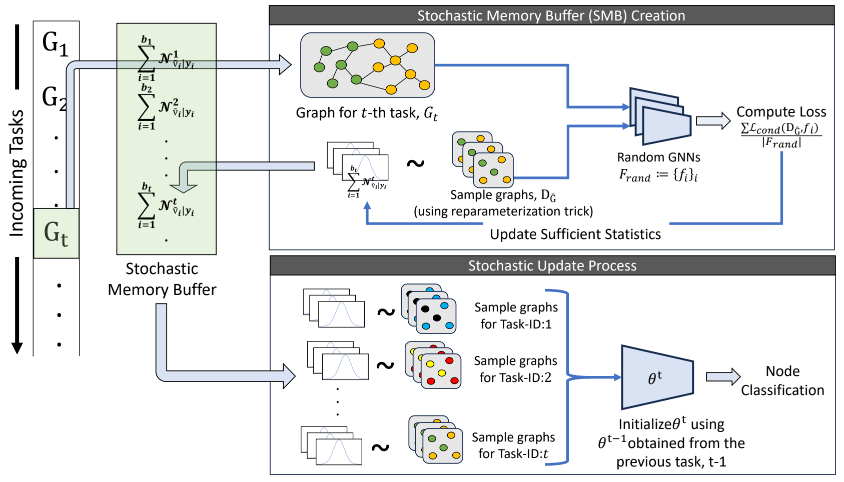

Stochastic Memory Buffer (SMB) Creation
Incoming Tasks
G_1
G_2
G_t
$\sum_{i=1}^{b_1} \mathcal{N}^1_{\tilde{v}_i|y_i}$
$\sum_{i=1}^{b_2} \mathcal{N}^2_{\tilde{v}_i|y_i}$
$\sum_{i=1}^{b_t} \mathcal{N}^t_{\tilde{v}_i|y_i}$
Stochastic Memory Buffer
Graph for t-th task, G_t
Random GNNs
$F_{rand} := \{f_i\}_i$
Compute Loss
$\frac{\sum \mathcal{L}_{cond}(D_{\tilde{G}}, f_i)}{|F_{rand}|}$
Sample graphs, $D_{\tilde{G}}$
(using reparameterization trick)
Update Sufficient Statistics
Stochastic Update Process
Sample graphs for Task-ID:1
Sample graphs for Task-ID:2
Sample graphs for Task-ID:t
θ^t
Node Classification
Initializeθ^t using θ^{t-1}obtained from the previous task, t-1

Figure 4: Proposed SERGCL framework consists of *(a) Stochastic Memory Buffer (SMB)* Creation, which stores condensed graph distributions for historical tasks, and *(b) Stochastic Update Process (SUP)*, which updates the GNN using these learned SMBs for all encountered tasks.

where $f(v)$ and $f(\tilde{v})$ are the representations for node $v \in V$ and $\tilde{v} \in \tilde{V}$of the graph G and $\tilde{G}$ respectively. The distribution of subgraphs conditioned on the node-label y from G and $\tilde{G}$ are represented by $\mathcal{P}_{G|y}$ and $\mathcal{P}_{\tilde{G}|y}$ respectively. Div is a divergence measure between the node representations of G and $\tilde{G}$.

Maximum mean discrepancy (MMD) is a well-known divergence function that minimizes the representations via the moments of the conditional distributions [30] as follows:

$$\mathcal{L}(\tilde{G}; f) = \min_{\tilde{G}} \sum_{y \in \mathcal{Y}} \gamma_y \Big|\Big| \mathop{\mathbb{E}}_{v \sim \mathcal{P}_{G|y}} [f(v)] - \mathop{\mathbb{E}}_{\tilde{v} \sim \mathcal{P}_{\tilde{G}|y}} [f(\tilde{v})] \Big|\Big|^2 \tag{2}$$

where $\gamma_y = \frac{|Y==y|}{|Y|}$ is the class-ratio, $|\cdot|$ is the cardinality of a set, and $|Y == y|$ is the number of nodes with label y.

Previous methods optimize Eq. 2 with respect to $\tilde{G}$ to obtain a fixed condensed graph for G [13, 38]. Instead, we proposed to minimize the same objective (i.e., Eq. 2) directly with respect to $\mathcal{P}_{\tilde{G}|y}$ to capture the underlying distribution of G, as described below.

3.1.2 Learning the Condensed Graphs Distribution

Challenges. Solving the optimization in Eq. 2 with respect to $\mathcal{P}_{\tilde{G}|y}$ demands sampling from the unknown distribution. However, defining and learning an analytical form of a distribution over an arbitrary graph and drawing samples is a complex and time-consuming process. We instead consider a condensed graph as a set of synthetic nodes without any edges and with pre-defined class labels *i.e.*, $\tilde{G} = \{\tilde{V}, I, \tilde{Y}\}$. Previously [13] have shown that such synthetic condensation can effectively summarize historical tasks, achieving the current state-of-the-art performance for nGCL.

Condensed Graph Distribution, $\mathcal{P}_{\tilde{G}|y}$. For a budget, b, a condensed graph is a set of synthetic nodes: $\tilde{V} = \{\tilde{v}_1, \cdots, \tilde{v}_b\}$. Such a representation of $\tilde{G}$ allows defining and sampling from independent density functions for each node $\tilde{v}_i \in \tilde{V}$ with a prefixed label $\tilde{y}_i$. For our work, we use Gaussian kernel density functions. Therefore,

$$\mathcal{P}_{\tilde{G}|y} = \frac{1}{b} \sum_{i=1}^{b} \mathcal{N}_{\tilde{v}_i|y_i}(\boldsymbol{\mu}_{\tilde{\boldsymbol{v}}_i|\tilde{\boldsymbol{y}}_i}, \boldsymbol{\Sigma}_{\tilde{\boldsymbol{v}}_i|\boldsymbol{y}_i}) \;\;\&\;\; \tilde{v}_i \sim \mathcal{N}_{\tilde{v}_i|\tilde{y}_i} \tag{3}$$

where, $\mathcal{N}_{\tilde{v}_i|y_i}$ is a Gaussian with prefixed class-label, $\tilde{y}_i$ and learnable parameters, $\boldsymbol{\mu}_{\tilde{\boldsymbol{v}}_i|\boldsymbol{y}_i}$ and $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{v}}_i|\boldsymbol{y}_i}$.

Objective Function. Now, we can present the objective to learn $\mathcal{P}_{\tilde{G}|y}$ by redefining Eq. 2 with respect to each Gaussian kernel, as follows:

$$\begin{aligned}\mathcal{L}(\{\mathcal{N}_{\tilde{v}_i|y}\}_i; f) &= \min_{\{\mathcal{N}_{\tilde{v}_i|y_i}\}_i} \sum_{y\in\mathcal{Y}} \gamma_y \Big|\Big| \mathop{\mathbb{E}}_{v\sim\mathcal{P}_{G|y}}[f(v)] - \mathop{\mathbb{E}}_{\tilde{v}\sim\mathcal{N}_{\tilde{v}_i|y}}[f(\tilde{v})] \Big|\Big|^2 \\ &\approx \min_{\{\mathcal{N}_{\tilde{v}_i|y_i}\}_i} \sum_{y\in Y_t} \gamma_y \Big|\Big| \frac{\sum\limits_{v\in G|y} f(v)}{|Y == y|} - \frac{\sum\limits_{\tilde{v}\in \mathrm{D}_{\tilde{G}|y}} f(\tilde{v})}{|\tilde{Y} == y| \times |\mathrm{D}_{\tilde{G}}|} \Big|\Big|^2\end{aligned} \tag{4}$$

where $\mathrm{D}_{\tilde{G}} := \{\tilde{G} \mid \tilde{v} \sim \mathcal{N}_{\tilde{v}_i|y}\}$ is a finite set of condensed graphs where the nodes are sampled from $\{\mathcal{N}_{\tilde{v}_i|y}\}_i$ and $\mathrm{D}_{\tilde{G}|y}$ is the set of all nodes with class-label y.

Algorithm 1 Steps for the SERGCL Method

Input: *(a)* G_t: t^{th} Input Graph *(b)* b_t: Budget for t^{th} task *(c)* $\mathcal{M}^{t-1}$: Learnt SMB till $(t-1)^{th}$ tasks.
Output: θ_t: Updated GNN using $\{1, \cdots, t\}$ tasks.

```
1:  ▷ Stochastic Memory Buffer Creation
2:  Initialize $\{\boldsymbol{\mathcal{N}}_{\tilde{v}_i|\tilde{y}_i}\}_{i=1}^{b_t}$ as random nodes from $G_t$
    and random diagonal/isotropic matrices respectively,
    with prefixed class-labels $\tilde{y}_i$.
3:  for each $iter \leq MaxIter$ do
4:      Sample $\mathrm{D}_{\tilde{G}} := \{\tilde{G} \sim \{\boldsymbol{\mathcal{N}}_{\tilde{v}_i|\tilde{y}_i}\}_{i=1}^{b_t}\}$.
5:      Randomly initialize GNNs: $F_{rand} = \{f_i\}_i$
6:      Optimize $\{\boldsymbol{\mathcal{N}}_{\tilde{v}_i|\tilde{y}_i}\}_{i=1}^{b_t}$ by minimizing $\mathcal{L}_{SMB}$
    using $\mathrm{D}_{\tilde{G}_t}$ and $F_{rand}$ (Eq. 5).
7:  end for
8:  ▷ Stochastic Update Process
9:  Initialize $\theta^t$ using $\theta^{t-1}$.
10: for each $iter \leq MaxIter$ do
11:     Sample $\mathrm{D}^j_{\tilde{G}} := \{\tilde{G}^j \sim \{\boldsymbol{\mathcal{N}}^j_{\tilde{v}_i|\tilde{y}_i}\}_{i=1}^{b_t}\} \forall j \leq t$.
12:     Optimize $\theta^t$ (minimizing $\mathcal{L}_{sup}$ via $\{\mathrm{D}^j_{\tilde{G}}\}_{j=1}^{t}$).
13: end for
```

CaT as a special case of SERGCL. By pushing $|\mathbf{\Sigma}_{\tilde{v}_i|y_i}| \to 0$ on Eq. 4 produces Dirac delta functions for each kernel, $\{\mathcal{N}_{\tilde{v}_i|y_i}\}_i$. It leads to a condensed graph with a fixed set of synthetic nodes, as employed in CaT [13]. Therefore, the objective function for CaT is a special case of our proposed optimization in Eq. 4. However, such a fixed condensed graph can lead to a high-variance estimation of the samples because of the specified parameterization as delta functions. In contrast, by effectively learning $\mathbf{\Sigma}_{\tilde{v}_i|y_i}$, the *SERGCL* formulation not only reduces the sample variance but also facilitates sampling of multiple condensed graph samples to provide a better estimation of historical graphs.

Overall Optimization using Random Networks. The objective, $\mathcal{L}$ in Eq. 4 can be optimized using either a pre-trained model f or by training f alongside the predictive model. However, both approaches incur additional computational overhead. An alternative is to use a set of randomly initialized GNNs, $F_{rand} = \{f_i\}_i$ [13, 30], leading to final condensation objective:

$$\mathcal{L}_{SMB} = \min_{\{\mathcal{N}_{\tilde{v}_i|y}\}_i} \frac{\sum_{f_i} \mathcal{L}(\{\mathcal{N}_{\tilde{v}_i|\tilde{y}_i}\}_i; f_i)}{|F_{rand}|} \tag{5}$$

Reparameterization Trick. Our proposed loss (Eq 4) requires a *random operation* of sampling condensed nodes. Therefore, we introduce a *'reparameterization trick'* to address this problem [41]. At each training iteration, we sample a set of noise vectors of dimension from the standard normal distribution outside the optimization process for a smooth gradient. Now, the condensed graph node samples can be produced as a fixed operation as follows:

$$\tilde{v}_i^s = \boldsymbol{\mu}_{\tilde{v}_i|\tilde{y}_i} + \epsilon_i^s \times \Sigma^{1/2}_{\tilde{v}_i|\tilde{y}_i} \tag{6}$$

where, $\epsilon_i^s \in \mathcal{N}(0, \mathcal{I})$ corresponds to i^{th} node of the s^{th} sampled Graph. We sample $|\tilde{Y}| \times |\mathrm{D}_{\tilde{G}}|$ noise vectores to obtain $|\mathrm{D}_{\tilde{G}}|$ condensed graphs at each iteration.

We expand the SMB by including the learned condensed graph distribution, $\frac{1}{b_t}\sum_i \mathcal{N}^t_{\tilde{v}_i|y}$ for G_t. The SMB is then passed to the SUP step, as discussed below.

3.2 Stochastic Update Process (SUP)

We update the GNN model, θ^t for the downstream node classification tasks using SMBs of all $\{1, \cdots, t\}$ tasks, that are received so far. We only use the condensed graphs, sampled from $\mathcal{M}^t\}$, to update θ^t without using any original graphs. Therefore, it allows us to choose equal sampling proportions for each task, removing the bias due to the mismatch of graph sizes for different tasks.

Optimizing for $j(\leq t)$-th task. We initialize the GNN, θ^t using the previous GNN, θ^{t-1} that incorporated $\{1, \cdots, t-1\}$ tasks. Now, we define the expected loss over the condensed graph distribution for each task, $1 \leq j \leq t$ as:

$$\mathcal{L}_{nGCL}(\{\mathcal{N}^j_{\tilde{v}_i|y}\}_i, \theta^t) = \min_{\theta^t} \mathbb{E}_{\tilde{G}_j \sim \frac{\sum_i \mathcal{N}^t_{\tilde{v}_i|y}}{b_j}} \left[\mathcal{L}_{nGCL}(\tilde{G}_j, \theta^t)\right] \approx \min_{\theta^t} \frac{\sum_{\tilde{G}^s_j \in \mathrm{D}^j_{\tilde{G}_j}} \mathcal{L}(\tilde{G}^j, \theta^j)}{|\mathrm{D}^j_{\tilde{G}_t}|}$$

where $\mathrm{D}^j_{\tilde{G}_j} := \{\tilde{G}^s_j \mid \tilde{v}^s_j \sim \{\mathcal{N}^j_{\tilde{v}_i|y}\}^{b_t}_{i=1}\}$ is a finite set of graph samples for the j-th task. $\mathcal{L}_{nGCL}$ is the supervised loss for node classification (*e*.g., cross-entropy).

Overall Loss. The final loss function to update the GNN parameters, θ^t incorporating t-th task is:

$$\mathcal{L}_{sup} = \min_{\theta^t} \sum_{j=1}^{t} \mathcal{L}_{nGCL}(\{\mathcal{N}^j_{\tilde{v}_i|y}\}_i, \theta^t) \tag{7}$$

Algorithm 1 presents our proposed SERGCL method with SMB creation and SUP step.

4 Experimental Results

4.1 Experimental Setup

Datasets. We evaluated our SERGCL method for class- and task-incremental (IL) settings in nGCL tasks using four benchmark datasets: CoraFull [42], Arxiv [43], Reddit [44], and Products [43]. Table 3 (Appendix) provides a summary of these datasets' statistics. Our experimental setup follows the configurations used in previous GCL studies [10, 12, 13, 45]. During training, the model has access only to the current incoming graphs and the memory buffer. Unless otherwise specified, the buffer budget for memory replay-based methods is 0.01% of the total number of training nodes. During testing, the model predicts from all previous tasks. Each task consists of a binary classification problem. For our models, we report *mean*±*std.* for 5 independent runs. Please find additional details in Appendix A.1. We also provide several ablation studies in Appendix A.2.

Comparative Models. Our SERGCL is evaluated against several existing approaches: *Regularization-based* EWC [3], MAS [9], GEM [23], TWP [15], and LWF [17]. *Architecture-based* HPNs [10] and *Replay-based* [11], SSM [12], and Condensation-based CaT [13]. Results using *Finetuning* and *Joint* training strategies are also presented. The *Finetuning* approach updates the model solely with the current graph, without accessing any historical data, thus providing a lower bound for performance in comparison to nGCL models. In contrast, the *Joint* strategy, which assumes infinite memory, utilizes all historical graphs along with the current graph. Although *Joint* training can achieve high performance, it may suffer from biases due to the imbalance in graph sizes across different tasks. Furthermore, while *Joint* training is feasible for the datasets used, it may not be practical in real-world applications involving extremely large graph sizes.

Evaluation Metric. We use two well-known metrics from the continual learning literature:

(a) Average performance (AP). Average Performance (AP) evaluates how well the model retains knowledge from all previously learned tasks after incorporating a new task. It is computed as the average node-classification performance on tasks $\{1, \cdots, t\}$ after learning the t-th task: $AP_t = \frac{1}{t}\sum_{i=1}^{t} a_{t,i}$; where, $a_{t,i}$ is the performance on i-th task after receiving t-th task.

(b) Average Forgetting (AF). Average Forgetting (AF) measures the degradation in performance on previous tasks due to learning new tasks. It is computed as the average difference in performance on all previous tasks before and after learning the t-th task: $AF_t = \frac{1}{t-1}\sum_{i=1}^{t-1}(a_{t,i} - a_{i,i})$; where $a_{t,i}$ is the performance on the i-th task after learning the t-th task, and $a_{i,i}$ is the performance on the i-th task before learning the t-th task. A positive AF indicates *backward transfer*, meaning that learning new tasks improves performance on previously learned tasks. However, a higher AF does not always signify superior performance, as it is possible to achieve a high AF by focusing on historical tasks at the expense of learning new ones.

4.2 Performance Analysis

Table 1 and Table 2 present a comparative performance for Class-IL and Task-IL settings, respectively.

Table 1: Performance comparison for Class-IL without inter-task edges is shown. Replay-based models are evaluated using a 0.01 budget relative to training nodes. The best models are highlighted based on their AP scores. An upward arrow (↑) indicates that higher values are better.

Category	**Method**	**CoraFull**		**Arxiv**		**Reddit**		**Products**	
		AP % (↑)	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**
Lower Bound	Finetuning	2.2±0.0	-96.6±0.1	5.0±0.0	-96.7±0.1	5.0±0.0	-99.6±0.0	4.3±0.0	-97.2±0.1
Regularization-based	EWC [3]	2.9±0.2	-96.1±0.3	5.0±0.0	-96.8±0.1	5.3±0.6	-99.2±0.7	7.6±1.1	-91.7±1.4
	MAS [9]	2.2±0.0	-94.1±0.6	4.9±0.0	-95.0±0.7	10.7±1.4	-92.7±1.5	10.1±0.6	-89.0±0.5
	GEM [23]	2.5±0.1	-96.6±0.1	5.0±0.0	-96.8±0.1	5.3±0.5	-99.3±0.5	4.3±0.1	-96.8±0.1
	TWP [15]	21.2±3.2	-96.6±0.1	5.0±0.0	-96.8±0.1	5.3±0.5	-99.3±0.5	4.3±0.1	-96.8±0.1
	LWF [17]	2.2±0.0	-96.6±0.1	5.0±0.0	-96.8±0.1	5.0±0.0	-99.5±0.0	4.3±0.0	-96.8±0.2
Replay-based	ER-GNN [11]	4.0±0.7	-94.3±0.9	30.8±0.6	-68.3±0.7	31.8±4.0	-71.2±4.2	39.5±1.3	-48.2±1.4
	SSM [12]	16.2±2.8	-82.1±2.9	35.1±1.8	-63.7±1.9	51.6±6.4	-50.3±6.7	62.7±0.5	-22.1±0.5
	CaT [13]	61.3±2.8	-8.8±2.2	66.0±1.1	-13.1±1.0	**97.4±0.1**	**-0.5±0.0**	68.5±0.4	**-6.1±0.3**
	SERGCL (Ours)	**66.5±2.4**	**-8.5±2.0**	**67.4±0.3**	**-11.6±0.4**	**97.4±0.0**	**-0.5±0.0**	**69.0±0.4**	**-6.1±0.5**
Full Dataset	*Joint Training*	*85.3±0.1*	*-2.7±0.0*	*63.5±0.3*	*-15.7±0.4*	*98.2±0.0*	*-0.5±0.0*	*72.2±0.4*	*-5.3±0.5*

Table 2: Performance comparison for Task-IL without inter-task edges is shown. Replay-based models are evaluated using a 0.01 budget relative to training nodes. The best models are highlighted based on their AP scores. An upward arrow (↑) indicates that higher values are better.

Category	**Method**	**CoraFull**		**Arxiv**		**Reddit**		**Products**	
		AP % (↑)	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**	**AP % (↑)**	**AF % (↑)**
Lower Bound	Finetuning	51.0±3.4	-46.2±3.5	67.1±5.2	-31.3±5.6	57.1±7.4	-44.6±7.8	56.4±3.8	-42.4±4.0
Regularization-based	EWC [3]	87.4±2.2	-9.1±2.2	85.6±7.7	-11.9±8.1	85.5±3.3	-14.8±3.5	90.3±1.8	-6.8±1.9
	MAS [9]	93.0±0.3	-0.7±0.5	83.8±6.9	-12.0±7.8	99.0±0.1	0.0±0.0	95.9±0.1	0.0±0.0
	GEM [23]	94.3±0.6	-2.1±0.5	94.7±0.1	-2.3±0.2	99.3±0.1	-0.3±0.1	86.9±0.9	-10.6±0.9
	TWP [15]	87.9±1.9	-4.9±0.6	77.1±7.3	-3.5±5.4	74.1±5.5	-1.5±0.5	75.4±4.4	-4.9±6.4
	LWF [17]	64.7±1.1	-32.3±1.2	60.2±5.8	-38.6±6.2	62.4±3.5	-39.1±3.7	50.1±0.7	-49.3±0.8
Architecture-based	HPNs [10]	-	-	85.8±0.7	0.6±0.9	-	-	80.1±0.8	2.9±1.0
Replay-based	ER-GNN [11]	54.2±1.0	-43.1±1.1	92.2±0.3	-4.9±0.3	94.3±0.5	-5.6±0.5	83.5±0.4	-14.3±0.5
	SSM [12]	78.7±1.1	-17.9±1.2	93.3±0.4	-3.6±0.4	99.2±0.2	-0.5±0.2	94.6±0.5	-2.7±0.4
	CaT [13]	93.0±0.4	**-0.1±0.4**	95.8±0.2	0.2±0.1	**99.4±0.0**	**0.1±0.1**	95.6±0.3	-0.4±0.1
	SERGCL (Ours)	**93.6±0.4**	-0.3±0.4	**96.1±0.1**	**1.3±0.1**	**99.4±0.0**	**0.1±0.1**	**95.8±0.1**	**-0.3±0.3**
Full Dataset	*Joint*	*97.2±0.0*	*0.2±0.1*	*96.7±0.0*	*-0.1±0.1*	*99.7±0.0*	*0.0±0.0*	*95.7±0.7*	*-0.2±0.7*

4.2.1 Class-IL NGCL

Replay-based vs. Regularization-based models. Table 1 demonstrates that regularization-based models significantly lag behind replay-based models in AP scores. These models often fail to retain knowledge from previous tasks while focusing on new ones, thus achieving near'lower-bound,' performance. Their large negative AF scores further highlight the issue of catastrophic forgetting across all datasets. This issue arises because regularization-based models transfer knowledge effectively only when tasks have similar characteristics. In class-IL settings, these models struggle even more, as they fail to differentiate between tasks, exacerbating catastrophic forgetting.

Comparison among different Replay-based models. The effectiveness of replay-based models hinges on their memory buffer creation strategy. We observe that the sampling methods used in ER-GNN and SSM often fall short in effectively summarizing historical graphs within a lightweight memory buffer. In contrast, CaT has made strides by introducing condensation-based replay graphs for historical tasks. However, these fixed condensed graphs still struggle to capture the full range of variations present in the historical graphs.

Performance of Proposed SERGCL. We consistently outperform existing nGCL models, demonstrating the effectiveness of using an SMB to store the distribution of condensed graphs for each historical task and sampling from this distribution to update the downstream nGCL model. This approach ensures broader coverage of historical graphs, leading to enhanced overall performance. In Appendix 4 we also study the effect of drawing different number of samples for SERGCL.

In Table 1, our SERGCL method shows an impressive 8.5% improvement in AP score on CoraFull-CL compared to the second-best CaT method. Although the AP score for the 'Joint' baseline remains substantially higher than for the GCL models, indicating room for improvement, SERGCL performs marginally better than CaT on the other three datasets and shows comparable performance to the 'Joint' baseline. The narrowing performance gap between SERGCL and the 'Joint' baseline further underscores the robustness of our approach.

4.2.2 Task-IL NGCL

Table 2 compares nGCL performance in Task-IL settings, where tasks are distinguished during inference by their Task-IDs. Hence, as the tasks are not mixed in presence of the task-ID, regularization-

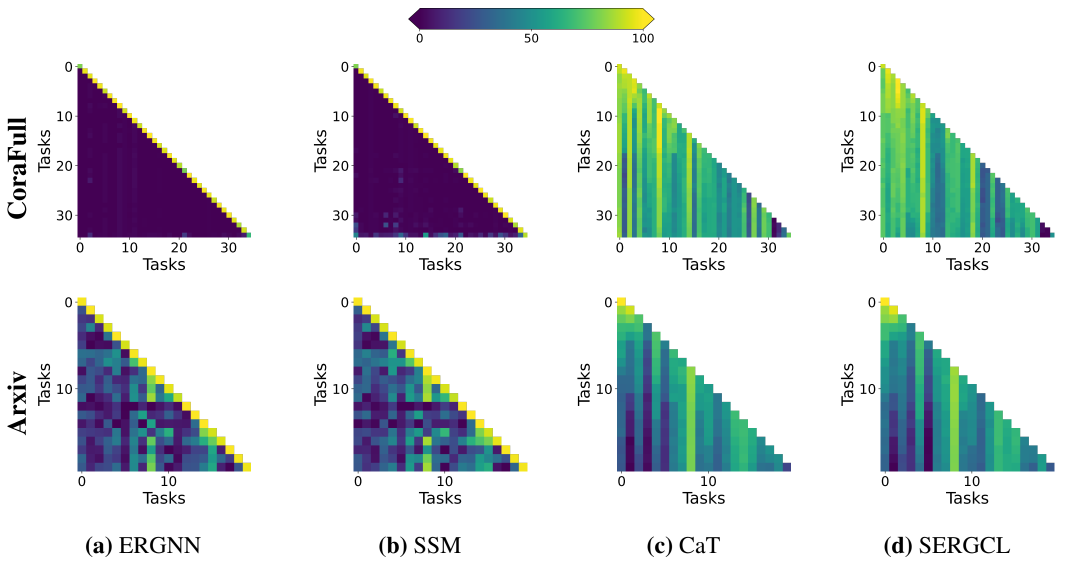


Figure 5: Visualization of Class-IL performance matrix of replay methods on CoraFull and Arxiv.

based models perform notably better than the 'Finetuning' baselines. Further, they often outperform traditional replay-based methods like ER-GNN and SSM on the CoraFull and Products datasets and achieve similar performance on Arxiv and Reddit. Graph condensation improves the performance of replay-based methods across all datasets. Finally, our SERGCL model further enhances performance in Task-IL nGCL, demonstrating clear advantages over other replay-based and GCL models.

4.3 Visualizing Performance Matrix for Class-IL setup

Figure 5 visualizes node classification performance across all tasks at each time step for different replay-based models. Regularization-based models, which show poor performance comparable to the 'lower-bound,' are included for reference. The figure reveals that SSM and ER-GNN often achieve 0% accuracy at off-diagonal entries, indicating severe catastrophic forgetting. In contrast, CaT and SERGCL mitigate this issue, with node classification performance remaining relatively stable across new tasks. SERGCL model often produces higher performance for different historical tasks, boosting the overall average performance.

5 Conclusion

We present SERGCL, a stochastic memory buffer (SMB) framework designed to store the distributions of condensed graphs for graph continual learning (GCL). We view the graphs as samples, drawn from an underlying complex distribution. By utilizing Gaussian mixture models as universal approximators, we can condense arbitrary graphs using learnable Gaussian kernels. Towards this, we leverage Gaussian mixture models (GMMs) as universal approximators to condense arbitrary graphs into sets of independent nodes represented by learnable Gaussian kernels. Experimentally, our approach achieves state-of-the-art performance across a range of graph complexities, benchmarked using carefully curated GCL datasets [45]. However, as graph complexity increases, our method may require a higher number of Gaussian kernels, resulting in increased memory demand.

We acknowledge two key challenges shared by existing graph condensation methods, including ours: (i) a heavy reliance on prior node-level supervision and (ii) diminished efficacy when handling dynamically changing labels. Further, condensation techniques are computationally heavy compared to sampling-based GCL methods, providing a tradeoff between performance and training efficiency. Addressing these challenges might serve as a potential direction for future research. Finally, while several existing works achieved better performance using structure-free condensed graphs, it would be interesting to study the impact of learning synthetic edges for GCL.

Acknowledgements

The authors thank Prof. Prathosh AP from IISc Bengaluru for his insightful feedback during various brainstorming sessions in the ideation phase of this work. His constructive feedback played a pivotal role in refining the paper's clarity and narrative.

References

[1] Falih Gozi Febrinanto, Feng Xia, Kristen Moore, Chandra Thapa, and Charu Aggarwal. Graph lifelong learning: A survey. *IEEE Computational Intelligence Magazine*, 18(1):32–51, 2023. 1, 2, 3, 4, 13

[2] Jatin Chauhan, Aravindan Raghuveer, Rishi Saket, Jay Nandy, and Balaraman Ravindran. Multivariate time series forecasting on variable subsets. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 76–86, 2022. 1

[3] 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. 1, 3, 7, 8

[4] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 220–221, 2020. 1

[5] Guile Wu, Shaogang Gong, and Pan Li Queen. Striking a balance between stability and plasticity for class-incremental learning. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1104–1113, 2021.

[6] Dongwan Kim and Bohyung Han. On the stability-plasticity dilemma of class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20196–20204, 2023. 1

[7] Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6523–6541, 2020. 1

[8] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019. 1

[9] 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. 1, 3, 7, 8

[10] Xikun Zhang, Dongjin Song, and Dacheng Tao. Hierarchical prototype networks for continual graph representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4622–4636, 2023. doi: 10.1109/TPAMI.2022.3186909. 2, 3, 7, 8

[11] 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. 2, 4, 7, 8, 15

[12] Xikun Zhang, Dongjin Song, and Dacheng Tao. Sparsified subgraph memory for continual graph representation learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1335–1340, 2022. doi: 10.1109/ICDM54844.2022.00177. 2, 4, 7, 8, 15

[13] Yilun Liu, Ruihong Qiu, and Zi Huang. CaT: Balanced Continual Graph Learning with Graph Condensation. In *Proc. of ICDM*, 2023. 2, 4, 5, 6, 7, 8, 13, 14, 15

[14] Yilun Liu, Ruihong Qiu, Yanran Tang, Hongzhi Yin, and Zi Huang. Puma: Efficient continual graph learning with graph condensation. *arXiv preprint arXiv:2312.14439*, 2023. 2, 4

[15] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8653–8661, 2021. 3, 7, 8

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014. 3

[17] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 3, 7, 8

[18] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020.

[19] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2861–2868, 2020.

[20] Sihao Ding, Fuli Feng, Xiangnan He, Yong Liao, Jun Shi, and Yongdong Zhang. Causal incremental graph convolution for recommender system retraining. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 3

[21] Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Deep class-incremental learning: A survey. *arXiv preprint arXiv:2302.03648*, 2023. 3

[22] Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, Guojie Song, and Sunghun Kim. Continual learning on dynamic graphs via parameter isolation. *arXiv preprint arXiv:2305.13825*, 2023. 3

[23] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017. 4, 7, 8

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

[25] Seungyoon Choi, Wonjoong Kim, Sungwon Kim, Yeonjun In, Sein Kim, and Chanyoung Park. Dslr: Diversity enhancement and structure learning for rehearsal-based graph continual learning. In *Proceedings of the ACM on Web Conference 2024*, 2024. 4

[26] Junshan Wang, Wenhao Zhu, Guojie Song, and Liang Wang. Streaming graph neural networks with generative replay. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1878–1888, 2022. 4

[27] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=mSAKhLYLSsl`. 4

[28] Saehyung Lee, Sanghyuk Chun, Sangwon Jung, Sangdoo Yun, and Sungroh Yoon. Dataset condensation with contrastive signals. In *International Conference on Machine Learning*, pages 12352–12364. PMLR, 2022.

[29] Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. Cafe: Learning to condense dataset by aligning features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12196–12205, 2022. 4

[30] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023. 4, 5, 6

[31] Songhua Liu, Jingwen Ye, Runpeng Yu, and Xinchao Wang. Slimmable dataset condensation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3759–3768, 2023. 4

[32] Wojciech Masarczyk and Ivona Tautkute. Reducing catastrophic forgetting with learning on synthetic data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 252–253, 2020. 4

[33] Felix Wiewel and Bin Yang. Condensed composite memory continual learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[34] Mattia Sangermano, Antonio Carta, Andrea Cossu, and Davide Bacciu. Sample condensation in online continual learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08. IEEE, 2022. 4

[35] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *International Conference on Learning Representations*, 2022. 4, 14

[36] Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 720–730, 2022.

[37] Xinyi Gao, Tong Chen, Yilong Zang, Wentao Zhang, Quoc Viet Hung Nguyen, Kai Zheng, and Hongzhi Yin. Graph condensation for inductive node representation learning. *arXiv preprint arXiv:2307.15967*, 2023.

[38] Mengyang Liu, Shanchuan Li, Xinshi Chen, and Le Song. Graph condensation via receptive field distribution matching. *arXiv preprint arXiv:2206.13697*, 2022. 4, 5

[39] Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. In *NeurIPS*, 2023. 4

[40] Yuchen Zhang, Tianle Zhang, Kai Wang, Ziyao Guo, Yuxuan Liang, Xavier Bresson, Wei Jin, and Yang You. Navigating complexity: Toward lossless graph condensation via expanding window matching. *ICML*, 2024. 4

[41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 6

[42] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000. 7

[43] 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. *Advances in neural information processing systems*, 33:22118–22133, 2020. 7

[44] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 7

[45] Xikun Zhang, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems*, 35:13006–13021, 2022. 7, 9

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 15

[47] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. 15

A Appendix

A.1 Datasets & Implementation Details

In Table 3, we present the details of all 4 datasets used in our experiments. The datasets are split into 60% for training, 20% for validation, and 20% for testing.

Implementation details. The implementation details for the proposed SER-GCL method are as follows: First, the stochastic memory buffer (SMB) consisting of the condensed graph distributions for each historical task is created. A buffer budget as 0.01 of the total number of nodes for each historical graph is used. The mean $\boldsymbol{\mu}$ of the distributions is initialized using randomly selected node features. A diagonal covariance matrix $\boldsymbol{\Sigma}$ is considered where the diagonal entries are set to 0.001. At each iteration, a sample of 200 condensed graphs and 1000 randomly initialized GNN models is used to obtain and compare the node representations. The Adam optimizer is applied to optimize both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with learning rates of 0.001 and 0.01, respectively. For classifier training, 500 iterations are employed.

During step 2 *i.e.,* the network updation phase, we sample condensed graphs for each historical task and update the network for 500 epochs using Adam optimizer with learning rates of 0.005 and weight decay strength 0.0005.

For other baseline models, we obtain the AP and AF scores for baseline models as reported in [1]. For the recently proposed CaT method [13], the official code provided by the authors is used to reproduce the results.

Table 3: Dataset Summary

Datasets	**CoraFull**	**Arxiv**	**Reddit**	**Products**
# Nodes	19,793	1,69,343	2,27,853	24,49,028
Node Ftr. dim.	8710	128	602	100
# Edges	1,30,622	11,66,243	11,46,15,892	6,18,59,036
# Classes	70	40	40	46
# Tasks	35	20	20	23

A.2 Ablation Study

A.2.1 Varying the number of Condensed Graph Samples

Our SERGCL draws samples from the learned condensed graph distribution during both training steps i.e., SMB creation and SUP step for updating the predictive network. In Table 4, the performance of SERGCL is presented as we vary the number of samples during SMB creation and SUP step.

Table 4: Average performance (AP) for SERGCL as we vary the sample size of condensed graphs for Arxiv dataset.

		During SUP step				
		1	**10**	**20**	**50**	**100**
During SMB Creation	**1**	65.9±0.4	65.9±0.0	65.8±0.3	65.9±0.6	65.8±0.3
	10	66.4±0.1	66.4±0.1	66.2±0.1	66.5±0.0	66.5±0.3
	20	67.0±0.2	67.0±0.2	67.0±0.2	67.1±0.1	66.9±0.3
	50	67.2±0.3	67.1±0.0	67.3±0.0	67.2±0.1	**67.4±0.2**
	100	66.9±0.3	67.1±0.4	67.0±0.1	67.1±0.2	**67.4±0.3**

During SMB creation, we update the parameters of the condensed graph distribution itself at each iteration step. Understanding the coverage of this varying distribution is important to find the correct gradient direction for updating the parameters. Therefore, increasing the number of samples provides better coverage of the distribution. Hence, it leads to a better distribution for the condensed graph and improves the overall performance using the experience replay. The performance starts to converge beyond the sample size of 50.

In contrast, during the SUP step, we sample the condensed graphs from the 'fixed' distribution during each iteration step. Hence, as we execute multiple iteration SUP steps, we eventually cover the underlying distribution by drawing new samples for each iteration. Therefore, we observe only a slight increase in AP as we increase the number of condensed graph samples in this step.

A.2.2 Varying Memory Buffer Budget.

Table 5 presents a comparative performance using AP metric as we increase the memory budget for NGCL tasks in class-IL settings. Clearly, a large memory is more efficient for summarizing the original historical graphs. Therefore, it improves the performance of our SERGCL and the existing CaT models. However, our SERGCL model still consistently outperformed the CaT models on both CoraFull and Arxiv datasets irrespective of the size of the memory budget.

Table 5: Impact of buffer budget with respect to the number of nodes in the condensed graph.

Budget	**CoraFull**				**Arxiv**			
	CaT [13]		**SERGCL**		**CaT** [13]		**SERGCL**	
	AP	**AF**	**AP**	**AF**	**AP**	**AF**	**AP**	**AF**
0.01	64.5±1.4	-3.3±2.6	66.5±2.4	-8.5±2.0	66.0±1.1	-13.1±1.0	67.4±0.3	-11.6±0.4
0.05	74.5±0.4	-6.2±0.1	76.7±0.2	-5.9±0.2	65.6±1.2	-12.2±0.4	67.7±0.1	-11.2±0.8
0.10	77.4±0.2	-5.7±0.1	77.8±0.4	-6.4±0.3	65.3±0.4	-12.1±0.6	67.5±0.1	-11.4±0.3

A.2.3 SUP step using currently available graph.

Experience replay (ER) based models typically update the memory network by incorporating the memory buffer with the current graph. However, due to the small size of the memory buffer, it creates a significant data imbalance problem. It leads to overemphasizing the current task and, leads to the catastrophic forgetting problem for historical tasks. One solution is to use the same number of condensed graph samples for each task [13]. However, unlike the existing ER models, our proposed SERGCL can mitigate the data imbalance problem by drawing an appropriate number of samples for historical tasks while using the original graph for the current task.

Table 6: Impact of using the current input graph while varying the number of condensed graph samples for historical graphs during the SUP step for using the Arxiv dataset.

# Samples	**Avg. Performance**	**Avg. Forgetting**
1	35.9 ± 2.8	-63.2±3.0
5	46.9±0.3	-50.1±0.4
10	46.4±2.8	-49.8±3.2
50	54.4±1.1	-38.0±1.6
100	60.7±0.6	-28.8±0.6
500	66.2±0.4	-7.8±0.5
750	65.8±0.5	-1.6±0.4
1000	65.8±0.5	3.6±0.4

In Table 6, we present the performance as we update the network using the entire current input graph and vary the number of samples for historical tasks for the Arxiv dataset. Since the size of the original current graph is significantly large, multiple samples of the historical tasks need to be drawn from SMB to address the imbalance problem. Drawing fewer samples leads to over-emphasis on the current tasks as before which can be corrected by increasing the number of samples for historical tasks to ensure balance between the historical and current tasks. This improves the overall performance of the models. Oversampling the historical tasks leads to underestimating the current tasks and the model produces better performances on historical tasks, improving the AF metric. This comes at the cost of reduced performance on the current task.

A.2.4 Generating condensed graphs using Optimized f

In the following, we explore the performance difference as we use optimized f instead of randomly initialized GNNs, F_{rand} at each iteration in Eq. 5. In Table 7, we report the results when the SMB creation process is applied using a single network, iteratively optimized while learning the condensed graph distributions. Here, we apply a similar optimization strategy as in [35]. However, we observe that AP scores consistently degrade compared to our original SERGCL framework using random networks.

We hypothesize that this performance drop occurs as an optimized network primarily focuses on learning the current tasks without adequately preserving the broader graph properties that can be essential for distinguishing historical or future tasks. In contrast, random projection, as an

Table 7: Performance comparison for Class-IL as we use random networks vs. a single optimized model to produce condensed graphs for our SERGCL models.

Method	CoraFull		Arxiv	
	AP (%)	AF (%)	AP (%)	AF (%)
SERGCL (random networks)	66.5	-8.5	67.4	-11.6
SERGCL (single Optimized f)	34.4	-24.2	61.6	-12.8

unsupervised approach, can capture the general characteristics of the graph and provide a better balance for continual learning.

A.2.5 Additional Memory Overhead for SERGCL

While our SERGCL method provides better coverage of the historical graphs, it requires additional memory for storing the $\mathbf{\Sigma}$ matrices of the Gaussian kernels (Eq. 3). The diagonal entries of $\mathbf{\Sigma}$ store the importance of each feature dimension of μ while the off-diagonal entries capture their correlations. However, storing a full rank $\mathbf{\Sigma}$ would require $O(d^2)$ additional memory. Instead, we can use a diagonal $\mathbf{\Sigma}$ with an additional memory cost of $O(d)$. As we do not store edge-related information for the condensed graphs, our additional memory compensates with most of the previous sampling-based experience replay methods that require storing edges [11, 12]. However, we still require an extra memory overhead compared to CaT [13]. Alternatively, we can learn an isotropic matrix *i.e.,* $\mathbf{\Sigma} = \sigma I$ for the Gaussian kernels. It only requires an additional memory of $O(1)$ for storing the scaler σ.

Table 8: Comparative performance of SERGCL with diagonal and isotropic Σ for Gaussian kernels.

	CaT [13]	**SERGCL (w/ diag Σ)**	**SERGCL (w/ isotropic Σ)**
Arxiv-CL	61.3 ± 2.8	66.5 ± 2.4	**66.6 ± 0.2**
Corafull-CL	66.0 ± 1.1	**67.4 ± 0.3**	66.9 ± 2.0
Reddit-CL	**97.4 ± 0.1**	**97.4 ± 0.0**	**97.4 ± 0.0**
Products-CL	68.5 ± 0.4	69.0 ± 0.4	**69.1 ± 0.3**

Table 8 reports the results as we select SERGCL with diagonal and isotropic $\mathbf{\Sigma}$ matrix. We observe that both sets of SERGCL models achieve comparable performance and still outperform 'fixed' condensation graph-based CaT models.

Table 9: Computation Time & Memory Footprint for Larger Graphs

	Reddit				**Product**			
	ERGNN [11]	**SSM** [12]	**CaT** [13]	**SERGCL**	**ERGNN** [11]	**SSM** [12]	**CaT** [13]	**SERGCL**
Peak Mem. (Buffer Creation)	1350 MiB	643.5 KiB	1026 MiB	1029 MiB	5534 MiB	147.5 KiB	5005 MiB	5019 MiB
Memory Buffer Creation Time	1 s	1 s	14 m 59 s	15 m 12 s	2s	1s	13 m 5 s	13 m 26 s
Training Time	7 m 43 s	7 m 36 s	2 m 4 s	2 m 19 s	16 m 8 s	16 m 4 s	2 m 12 s	2 m 59 s
Performance	31.8	51.6	97.4	97.4	39.5	62.7	68.5	69

A.2.6 Computational Details & Complexity Analysis

The model is implemented using PyTorch [46] and PyTorch Geometric libraries [47], and all experiments were performed on a single NVIDIA A100 GPU. In Table 9, the training time and memory footprint for different models during the sampling/condensation phase and the training phase are presented.

Memory Footprint: SSM requires the least peak memory since it creates the memory buffer with sparsification using a probability distribution. In contrast, other methods require additional memory for their heuristic calculations for ER-GNN or solving optimizations as in SERGCL and CaT.

Computation time: (a) The buffer creation time for SERGCL and CaT is significantly higher. These methods require solving the optimization to condense the historical graphs. In comparison, previous sampling-based traditional experience replay models (i.e., ER-GNN, SSM) require less time to sample and select the sub-graphs for producing their memory buffer. (b) However, the training time for ER-GNN and SSM are significantly large as they take the entire incoming graph for the current task.

Complexity Analysis: Given a L layer GNN, original graph mini-batch with $O(N)$ nodes and $O(n)$ synthetic nodes with self-loops, and total optimization iterations, I, the time complexity for both CaT and SERGCL remains the same as $O(IL)$, while space complexity becomes $O(N^2 + n)$ and $O(N^2 + nk)$ for CaT and SERGCL; where k is the number of samples drawn at each iteration for SERGCL; $O(N^2)$ denotes the edges for original graph mini-batch. Since $N >> n$, both $O(N^2 + n)$ and $O(N^2 + nk)$ become $O(N^2)$, i.e., the same time and space complexity for both CaT and SERGCL.