CONTINUAL LEARNING ON GRAPHS VIA GRADIENT-FREE CONDENSATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual Graph Learning (CGL) must adapt to arriving graph data while retaining past knowledge, under strict memory and computation budgets. Memory replay is the dominant strategy, and recent methods show that replaying *condensed* graphs is particularly effective. However, existing condensation-based approaches violate the basic requirements of streaming data processing: they rely on gradient-based optimization with repeated passes over the data, assume prior knowledge of future class statistics, and permit the condensed memory to grow with the stream. We propose STREAMCGL, a replay-based framework that performs *streaming graph condensation* under strict single-pass and fixed-memory constraints. Rather than learning synthetic nodes against a reference model, STREAMCGL condenses in the continuous Weisfeiler-Leman feature space—the space that bounds what any message-passing GNN can extract—which reduces condensation to distortion minimization over an edge-free point set and admits a gradient-free, one-pass algorithm with an $O(\log n \log k)$ approximation guarantee. Across three class-incremental protocols spanning chronological, imbalanced, and recurrent class arrivals, STREAMCGL attains the best average accuracy rank among all methods, condenses up to two orders of magnitude faster than state-of-the-art condensation baselines, and scales to streams on which they exhaust memory or time.

1 INTRODUCTION

Many real-world applications such as social networks, e-commerce recommendation, and financial fraud detection, involve temporally evolving graphs in which nodes, edges, and labels arrive continuously, often under distributional shift (Trivedi et al., 2019). Graph neural networks (GNNs) are the dominant paradigm for such data (Kipf, 2017; Hamilton et al., 2017b), but their standard training procedures assume a static dataset. When retrained naively on evolving snapshots, GNNs suffer catastrophic forgetting (Zhou & Cao, 2021). This has motivated *continual graph learning* (CGL), which adapts GNNs to arriving graph data while preserving past knowledge under bounded memory and computation (Zhang et al., 2024b).

Related Works. Existing CGL approaches fall into three types (Zhang et al., 2024b): *regularization-based* methods that constrain parameter updates (Xu et al., 2020; Cai et al., 2022), *parameter-isolation* techniques that allocate separate model components per task (Zhang et al., 2022a; 2023a), and *memory-replay* strategies that store representative historical data (Zhou & Cao, 2021; Wei et al., 2022; Liu et al., 2023). Replay is generally the most effective, but faces a bottleneck specific to graphs: because node representations are produced by multi-hop message passing, each stored sample drags along its computation subgraph, and the receptive field expands exponentially with GNN depth. Recent work mitigates this via subgraph sparsification (Zhang et al., 2022b), topology-aware embedding replay (Zhang et al., 2024a), and, most recently, *graph condensation* (Liu et al., 2023; 2024), which summarizes large graphs into a compact set of synthetic nodes reusable across tasks. Another related work in the *static* condensation literature, (Gao et al., 2025), uses a class-partition based method to design an efficient training-free condensation technique. We discuss detailed related works in App. A.

Limitations of existing approaches. Despite this progress, the state-of-the-art condensation methods for CGL, CaT (Liu et al., 2023) and PUMA (Liu et al., 2024), violate core tenets of streaming data processing. *(i) Assumed knowledge of the future.* Both implicitly require prior knowledge

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

of stream statistics, including the total number of classes and their relative proportions, which are unavailable in realistic streaming settings. *(ii) Unbounded memory growth.* Both allocate a fixed fraction of nodes per snapshot or class without any mechanism enforcing a global budget. Hence, the condensed graph grows monotonically with stream length, which is precisely what bounded-memory continual learning forbids. *(iii) Gradient-based, multi-pass optimization.* Both optimize synthetic nodes by gradient descent against a reference model, requiring multiple passes and making condensation the dominant computational cost.

Owing to these unrealistic design strategies, a key question remains unanswered: *can graph condensation for continual learning be performed in a fully streaming, gradient-free manner under a fixed memory budget?* We answer affirmatively. A key insight is that the two components of a message passing GNNs, feature-aggregation via message-passing and learnable parameters, can be decoupled while still maintaining sufficient representation capability. We formulate the condensation objective as a metric compression task in an architecture-agnostic space, for which single-pass, sub-linear memory algorithms with provable guaranties exist.

Contributions. Our main contributions are as follows:

- **A streaming-first reformulation of graph condensation for continual learning.** We revisit graph condensation for continual node classification under the constraints of streaming data processing: a fixed, user-specified memory budget, single-pass access, and no knowledge of future stream statistics such as task boundaries, class counts, or stream length. This departs from prior condensation methods for CGL, which implicitly assume memory that grows with the stream or repeated access to past data.
- **Condensation as metric compression in the continuous WL feature space.** Rather than *learning* synthetic nodes by optimizing against a reference model, we condense in the continuous Weisfeiler-Leman feature space (Togninalli et al., 2019). Condensation reduces to distortion minimization over an edge-free point set, with memory and update costs independent of graph size and degree, and requires no gradients and no trained model.
- **A single-pass framework with provable guarantees.** We show that minimizing distortion under a budget of k nodes is an instance of Euclidean k-means and is thus NP-hard (Aloise et al., 2009). We provide a theoretical streaming algorithm that achieves an $O(\log n \log k)$ approximation guarantee with sub-linear memory growth, which, to our knowledge, the first approximation guarantee for the CGL setting. To meet strict real-world constraints, we further develop a practical, fixed-memory implementation that uses a merge-and-reduce strategy to maintain a constant memory footprint while maintaining efficient and near-optimal condensation.
- **Benchmarks and empirical results.** We evaluate across three class-incremental protocols that vary how classes arrive over the stream—chronological, imbalanced/interleaved, and recurrent—and show that existing methods each collapse in at least one regime. STREAM-CGL attains the best average accuracy rank across all dataset–setting combinations while condensing up to two orders of magnitude faster than state-of-the-art baselines, and scales to streams on which they exhaust memory or time.

2 PROBLEM FORMULATION

Definition 1 (Graph)**.** *$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ denotes a graph over a finite, non-empty node set $\mathcal{V}$ and edge set $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$. $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times |F|}$ denotes node attributes encoded using $|F|$-dimensional feature vectors. We denote the attributes of node v as $\mathbf{x}_v$.*

We focus on *node classification* using GNNs.

Definition 2 (Node Classification)**.** *Let $\mathcal{G}_{tr}$ denote the training graph. Each node $v_i \in \mathcal{V}_{tr}$ has a ground-truth class label $y_i \in \{\mathcal{Y}_1, \ldots, \mathcal{Y}_C\}$ from C possible classes. The goal is to learn a* GNN *$\Theta : (\mathcal{G}, v) \mapsto \mathcal{Y}$ that predicts the class label of a given node.*

The problem of *graph condensation* for node classification using GNNs is defined as follows.

Definition 3 (Graph Condensation)**.** *Given a training graph $\mathcal{G}_{tr}$, a validation graph $\mathcal{G}_{val}$, and a memory budget b, synthesize a condensed graph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_x, \boldsymbol{X}_s)$ from $\mathcal{G}_{tr}$ such that $|\mathcal{V}_s| \leq b$,*

and, on validation graph the error gap between models trained on $\mathcal{G}_s$ and $\mathcal{G}_{tr}$ is minimized, i.e.,

$$\min \left\{ \left| \epsilon_{\Theta_s(\mathcal{G}_{val})} - \epsilon_{\Theta_{tr}(\mathcal{G}_{val})} \right| \right\}.$$

Here, Θ_s and Θ_{tr} denote GNN*s trained on $\mathcal{G}_s$ and $\mathcal{G}_{tr}$, respectively, and $\epsilon_{\mathcal{G}}$ represents the node classification error on the validation set when trained on graph $\mathcal{G}$.*

Error may be quantified using standard loss functions such as cross-entropy, negative log-likelihood, etc. Our objective is to perform graph condensation for *graph continual learning*.

Definition 4 (Graph Continual Learning)**.** *Graph continual learning considers a sequence of graph snapshots $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$, where each snapshot $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)}, \boldsymbol{X}^{(t)})$ corresponds to the same node classification task. At each time step t, a single* GNN *with parameters Θ_t is maintained and updated* incrementally *using only the current snapshot $\mathcal{G}^{(t)}$. Specifically, the model parameters evolve sequentially according to:*

$$\Theta_t \leftarrow \mathcal{U}(\Theta_{t-1}, \mathcal{G}^{(t)}),$$

*where $\mathcal{U}(\cdot)$ denotes an update procedure (e.g., gradient-based optimization). The objective is to obtain parameters $\{\Theta_t\}_{t=1}^{T}$ such that, after processing snapshot t, the resulting model achieves low node classification error on **all** snapshots observed so far.*

Problem 1 (Graph Condensation for Continual Learning)**.** *Given a sequence of graph snapshots $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$ and a memory budget b, the learner maintains a* single, persistent condensed graph *whose size is bounded by b throughout training. At each time step t, let $\mathcal{G}_s^{(t)} = (\mathcal{V}_s^{(t)}, \mathcal{E}_s^{(t)}, \boldsymbol{X}_s^{(t)})$ denote the current state of this condensed graph, satisfying $|\mathcal{V}_s^{(t)}| \leq b$. The condensed graph may be updated over time by modifying, replacing, or merging its nodes, but no additional historical snapshots may be stored.*

Let $\Theta_s^{(t)}$ denote a GNN *trained incrementally using $\mathcal{G}_s^{(t)}$, and let $\Theta_{\text{full}}^{(t)}$ denote a* GNN *trained incrementally using the full snapshot $\mathcal{G}^{(t)}$ under the same update procedure. The objective is to minimize the performance gap between these two models on a validation set:*

$$\min_{\{\mathcal{G}_s^{(t)}\}_{t=1}^{T}} \left| \epsilon_{\Theta_s^{(t)}}\left(\mathcal{G}_{\text{val}}^{(t)}\right) - \epsilon_{\Theta_{\text{full}}^{(t)}}\left(\mathcal{G}_{\text{val}}^{(t)}\right) \right|,$$

subject to the constraint $|\mathcal{V}_s^{(t)}| \leq b$ for all t.

3 Proposed Method: StreamCGL

Fig. 1 in App: D presents the pipeline of StreamCGL. StreamCGL maintains a fixed-size condensed memory that serves as a running summary of all graph snapshots observed so far. StreamCGL operates in three stages: (i) For each incoming snapshot, we compute node embeddings via continuous Weisfeiler-Leman (WL) refinement (Togninalli et al., 2019), a model-free analogue of message passing that decouples feature aggregation from learning. (ii) We formulate condensation as minimizing distortion in this representation space, enabling edge-free synthetic nodes. (iii) We maintain a class-wise online summarizer that incrementally updates a compact memory under a fixed budget. Alg. 1 shows the general approach of our condensation procedure.

3.1 Static Condensation Objective: Minimizing Expected Distortion

We introduce the condensation objective in the static setting, which forms the basis of our streaming algorithm. We formulate graph condensation as constructing a small set of synthetic nodes whose representations approximate those of the original graph under a suitable embedding space. The choice of the embedding space is central to our design. To remain gradient-free, the embedding must be computable without a trained model; to be a faithful proxy for downstream learning, it must capture node features jointly with neighborhood topology; and to admit distortion minimization, it must reside in a metric space.

The continuous relaxation of the Weisfeiler-Leman kernel space (Togninalli et al., 2019) meets all three requirements. Message-passing GNNs are provably no more expressive than the 1-WL test (Xu et al., 2019); any node information they can use is captured by WL's iterative refinement. Traditional WL kernels (Shervashidze et al., 2011) implement this refinement with hash functions, requiring no

training. However, 1-WL lacks a meaningful notion of distance. By replacing the discrete hashes with smooth aggregation functions, the continuous relaxation provides a surrogate framework that is differentiable and metric-compatible, making it a natural, architecture-agnostic space in which to condense, rather than one tied to a single surrogate model. In this space, minimizing representation error directly controls prediction error under mild smoothness assumptions.

Under a fixed budget constraint, the resulting objective reduces to minimizing the average distortion between original node representations and their nearest synthetic counterparts. Importantly, the final model trained on the condensed graph is not tied to the linear refinement used for condensation: the condensed memory is an edge-free point set that any message-passing GNN can be trained on, and the WL space targets structure common to 1-WL-bounded architectures rather than idiosyncrasies of the surrogate. We validate this backbone-agnostic behavior empirically in our experiments.

Continuous WL refinement as model-free aggregation: Formally, 1-WL color refinement iteratively updates a node coloring with $c_v^{(0)} = \textsc{Hash}(\mathbf{x}_v)$,

$$c_v^{(\ell)} = \textsc{Hash}\Big(c_v^{(\ell-1)},\ \big\{\!\!\big\{\, c_u^{(\ell-1)} : u \in \mathcal{N}(v) \,\big\}\!\!\big\}\Big), \tag{1}$$

where $\mathcal{N}(v)$ denotes the neighbors of v, $\{\!\{\cdot\}\!\}$ denotes a multiset, and HASH is an injective function. The colors $c_v^{(\ell)}$ are categorical: they support equality tests but admit no meaningful notion of distance, and the injective HASH is discontinuous in the node attributes. The continuous relaxation (Togninalli et al., 2019) therefore replaces the injective hash with an average over the same arguments:

$$\mathbf{z}_v^{(\ell)} = \frac{1}{2}(\mathbf{z}_v^{(\ell-1)} + \frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} \mathbf{z}_u^{(\ell-1)}), \qquad \mathbf{z}_v^{(0)} = \mathbf{x}_v. \tag{2}$$

We adopt the same averaging-refinement principle in matrix form, with the degree-symmetric weighting standard in GNNs. Writing $\mathbf{Z}^{(0)} = \boldsymbol{X} \in \mathbb{R}^{n\times d}$ for the node features,

$$\mathbf{Z}^{(\ell)} = \Big(D'^{-\frac{1}{2}} A' D'^{-\frac{1}{2}}\Big)^{\ell} \boldsymbol{X}, \qquad A' = A + I, \tag{3}$$

where D' is the degree matrix of A' and $\mathbf{Z}^{(\ell)} \in \mathbb{R}^{n\times d}$ collects the representations after l refinement rounds. Eqs. 2 and 3 are both lazy averaging operators supported on the closed neighborhood $\mathcal{N}(v)\cup\{v\}$, differing only in the choice of averaging weights; the symmetric normalization in Eq. 3 bounds the spectral radius of the propagation operator, ensuring stable refinement (Wu et al., 2019). Importantly, the refined features absorb the full effect of message passing: the condensed graph can be represented as a set of nodes (feature vectors) with identity adjacency matrix.

Lemma 1 (Edge-Free Invariance of WL Refinement)**.** *For an adjacency matrix A, let $\boldsymbol{\mathcal{A}}_A = D'^{-\frac{1}{2}}(A+I)\,D'^{-\frac{1}{2}}$ denote the refinement operator of Eq. 3, where D' is the degree matrix of $A+I$, and let $\mathbf{Z}^{(\ell)} = R_\ell(\boldsymbol{X}, A) = \boldsymbol{\mathcal{A}}_A^{\ell}\boldsymbol{X}$ denote ℓ rounds of continuous WL refinement. Then, $R_\ell\big(\mathbf{Z}^{(\ell)}, \mathbf{0}\big) = \mathbf{Z}^{(\ell)}$ i.e., the edge-free graph $\tilde{\mathcal{G}} = (\mathbf{Z}^{(\ell)}, \mathbf{0})$ carries exactly the same WL representations as $\mathcal{G} = (\boldsymbol{X}, A)$: the edges of $\mathcal{G}$ are fully absorbed into the features. Consequently, any linear predictor of the form $g \circ R_\ell$ produces identical outputs on $\mathcal{G}$ and $\tilde{\mathcal{G}}$.*

PROOF. See App. B.1. □

The above discussion yields our central objective.

Definition 5 (Distortion Minimization)**.** *Given a distribution $\mathbf{Z}$ (aggregated features) of the data points, and fixed budget k, we want to minimize the following objective:*

$$\min_{C=\{\mathbf{c}\in\mathbb{R}^d\},\ s.t.\ |C|=k} \mathbb{E}_{\mathbf{z}\sim Z}[D(\mathbf{z}, C)^2],\ \textit{with}\ D(\mathbf{z}, C) = \min_{\mathbf{c}\in C} d(\mathbf{z}, \mathbf{c})$$

where $d(.,.)$ is the L_2 distance metric.

In other words, the objective is to find a set of k points such that the expected distance of a point sampled from the distribution is minimized.

Since the exact distribution is unknown, we resort to Monte Carlo estimation and accordingly reduce the average case distortion w.r.t. the training samples. This in fact reduces to the well-known k-means clustering objective function:

$$arg\min_C \frac{1}{n}\sum_{\mathbf{z}\in Z} D(\mathbf{z}, C)^2 = arg\min_C \frac{1}{n}\sum_{\mathbf{z}\in Z}\min_{\mathbf{c}\in C} \|\mathbf{z}-\mathbf{c}\|_2^2 \tag{4}$$

Algorithm 1: STREAMCGL: continual graph learning. Memory updates and COMPRESS are detailed in §3.3; b_y denotes the per-class budget.

procedure STREAMCGL(stream $(\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(T)})$; budget b)
  Initialize: Θ_0; $\mathcal{C}^{(0)} \leftarrow \emptyset$; a class memory $\mathcal{M}_y \leftarrow \emptyset$ is created when class y first appears
  for $t \leftarrow 1$ to T **do**
    Update class memories $\{\mathcal{M}_y^{(t)}\}$ with the WL-refined embeddings of $\mathcal{G}^{(t)}$
    $\widetilde{\mathcal{G}}^{(t)} \leftarrow \bigcup_{y \in \mathcal{C}^{(t)}} \textsc{Compress}\big(\mathcal{M}_y^{(t)},\, b_y\big)$ ▷ $|\widetilde{\mathcal{G}}^{(t)}| \le b$
    Update GNN Θ_t from Θ_{t-1} by training on $\widetilde{\mathcal{G}}^{(t)}$

Lemma 2 (Output distortion under feature approximation)**.** *Let $g(\mathbf{z}) = \mathbf{z}\,\Theta$ be a linear readout on the WL-refined features and let $\|\Theta\|$ denote its operator norm. For any representation $\mathbf{z}$ and its closest node representation $\mathbf{c}_z^*$ from the condensed set C, we have $\|g(\mathbf{c}_z^*) - g(\mathbf{z})\| \;\le\; \|\Theta\|\,\|\mathbf{c}_z^* - \mathbf{z}\|$. Consequently, the expected output distortion is bounded as:*

$$\frac{1}{n}\sum_{\mathbf{z}\in Z} \|g(\mathbf{c}_z^*) - g(\mathbf{z})\|^2 \;\le\; \|\Theta\|^2\,\phi(C),$$

where $\phi(C)$ is the objective of Eq. 4.

PROOF. See App. B.2.

Lemma 2 implies that minimizing the expected distance $\mathbb{E}_{z\sim Z}[D(z, C)^2]$ or equivalently $\phi(C)$, constrains the output discrepancy. Given a smooth, Lipschitz-bounded loss, this bound also implies a bounded gradient divergence, ensuring the condensed optimization trajectory asymptotically tracks that of the full dataset. In the limit $|C| \to |V|$, $\phi(C) \to 0$, thus the condensed set becomes a near-perfect surrogate for the full training set.

Optimally solving Eq. 4 is NP-hard, since it is an instance of Euclidean k-means clustering (Aloise et al., 2009). This motivates the streaming approximation algorithms with provable guarantees developed in § 3.2.

Lemma 3. *Let $\tilde{\mathcal{G}}_{OPT}^k$ be the condensed graph formed with k nodes that corresponds to the optimal solution of the distortion objective Eq. 4 , and ϕ_{OPT}^k be the corresponding cost. Then we have $\phi_{OPT}^{(k+1)} \le \phi_{OPT}^k \quad \forall k$,and $\phi_{OPT}^n = 0$. Training a GNN on $\tilde{\mathcal{G}}_{OPT}^n$ is equivalent to training on the full set of aggregated features $Z \in \mathbb{R}^{n\times d}$.*

PROOF. See App. B.3. □

Lemma 3 shows the condensation budget is a monotone quality knob and enlarging the condensed graph cannot increase distortion. Moreover, it recovers the full feature set at $k = n$, and replay-based training gracefully reverts to full-data training, with the synthetic-real training discrepancy shrinking as the budget grows.

3.2 ONLINE CONDENSATION

We extend the static condensation objective to a streaming setting in which graph snapshots arrive sequentially and past data cannot be revisited. Our key idea is to maintain a streaming coreset of WL embeddings. Instead of storing raw nodes, we retain a weighted set of representative points that approximates the distribution of embeddings observed so far.

A ONE-PASS ALGORITHM WITH GUARANTEES

We first describe an approximation streaming algorithm for the k-means objective introduced in § 3.1. The algorithm maintains an intermediate memory composed of weighted representative points, whose size grows sublinearly with respect to the number of nodes processed at any time. This intermediate representation is then compressed to a fixed number of centers for downstream GNN training. Since the intermediate memory may hold more than k centers while the final condensed graph must contain exactly k, we measure the quality of both stages in the standard bicriteria framework, which permits an algorithm to output extra centers while its cost is still compared against the optimum with k centers.

Definition 6 ((Ailon et al., 2009)). *An algorithm A is an (a, b)-approximation for the k-means problem, $a, b \geq 1$, if it outputs a set C of at most ak centers whose cost $\phi(C) = \frac{1}{n}\sum_{\mathbf{z}\in Z} D(\mathbf{z}, C)^2$, i.e., the objective of Eq. 4, satisfies $\frac{\phi(C)}{\phi_{OPT}} \leq b$, where $\phi_{OPT} = \min_{C':|C'|=k} \phi(C')$ is the optimal cost with k centers.*

The key property of bicriteria approximations is that they compose: clustering the weighted output of one approximation algorithm with another degrades the cost only in a controlled manner.

Lemma 4 ((Ailon et al., 2009), Thm. 3.1). *If there exist algorithms A and A' that are, respectively, (a, b)- and (a', b')-approximation algorithms for the k-means problem, then applying A' to the weighted output of A yields an $(a', 2b + 4b'(b + 1))$-approximation.*

The result is proven directly for the k-means objective and holds for integer-weighted inputs, as required by our setting.

Def. 6 and Lemma 4 suggest a two-stage design for a one-pass streaming algorithm: (i) a *streaming stage* that constructs an intermediate coreset with a bicriteria guarantee, and (ii) a *compression stage* that reduces this coreset to exactly k centers. The composition lemma controls the overall approximation factor, which we show to be $O(\log n \log k)$ in expectation with respect to the optimal k-means cost.

Streaming stage: At any stage, we maintain a set of centers along with weights. Each center represents a cluster of points, and its weight counts the samples assigned to it. This stage instantiates the online k-means algorithm (Liberty et al., 2016), augmented with center weights (Algorithm 2 in App. C), and serves as algorithm A; it inherits the following guarantees (Liberty et al., 2016). The expected size of this memory grows as $O(k \log n \log \gamma n)$, where n is the number of processed points and γ is the aspect ratio of the data, while its expected cost is within $O(\log n)$ of the optimal k-means cost; that is, A is an (a, b)-approximation with $ak = O(k \log n \log \gamma n)$ and $b = O(\log n)$, both in expectation.

Compression stage: To obtain a fixed-size representation, we apply a static clustering algorithm A' to the weighted centers. We use k-means++ (Arthur & Vassilvitskii, 2007), which produces exactly k centers with an expected $O(\log k)$-approximation, treating integer weights as multiplicities; hence A' is a $(1, O(\log k))$-approximation.

Combining the two stages via Lemma 4, we get:

Lemma 5. *Algorithm 2 is a one-pass streaming algorithm that computes a condensed graph $\tilde{\mathcal{G}}$ with $|\tilde{\mathcal{G}}| = k$ nodes whose expected distortion (Eq. 4) is within a factor of $O(\log n \log k)$ of the optimal k-means cost, using an intermediate memory of expected size $O(k \log n \log \gamma n)$, where γ denotes the aspect ratio of the data.*

PROOF. See App. B.4. □

3.3 THE STREAMCGL PIPELINE

We now assemble the components into the full pipeline, summarized in Fig. 1 and Alg. 1. Let $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots\}$ denote the incoming sequence of graph snapshots. When snapshot $\mathcal{G}^{(t)}$ arrives, we compute its WL-refined node representations via Eq. 3. Each representation $\mathbf{z}$ with class label y is routed to its class memory $\mathcal{M}_y^{(t)}$. $\mathcal{M}_y^{(t)}$ represents weighted set of centers that summarizes all class-y embeddings observed so far. Whenever $\mathcal{M}_y^{(t)}$ exceeds its allocated capacity, it is compressed in place (details are in App. C). Class arrivals require no prior knowledge. Let $\mathcal{C}^{(t)}$ denote the set of classes observed up to time t. Given the total budget b, each class is allocated $b_y = \lfloor b/|\mathcal{C}^{(t)}| \rfloor$ synthetic nodes, and the intermediate memory is partitioned in the same proportion. When a previously unseen class appears, the per-class budgets are recomputed and existing summaries are compressed to their reduced allocations; the stream length, the number of classes, and their relative proportions are never assumed in advance. Each class memory $\mathcal{M}_y^{(t)}$ along with class label y forms the per-class condensed graph $\widetilde{\mathcal{G}}_y^{(t)}$, and the final condensed graph $\widetilde{\mathcal{G}}^{(t)} = \bigcup_{y\in\mathcal{C}^{(t)}} \widetilde{\mathcal{G}}_y^{(t)}$. The GNN is updated incrementally, $\Theta_{t-1} \to \Theta_t$, by training on $\widetilde{\mathcal{G}}^{(t)}$.

Table 2: Recurrent class-IL. For each dataset and metric, the best result among continual-learning methods is in dark and the second best in light (higher is better for AP and AF; lower is better for Time). Full Data serves as a reference and is excluded from the ranking. OOM: out of memory.

Methods	CoraFull			Arxiv			Reddit			Products		
	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓
Full Data	76.0 ± 0.2	-2.1 ± 0.1	279.3	55.2 ± 0.2	-13.8 ± 0.1	162.0	97.7 ± 0.0	-0.3 ± 0.0	1981.2	OOM	OOM	OOM
CAT	65.4 ± 0.7	-10.4 ± 0.5	1241.0	61.6 ± 0.5	-8.6 ± 0.5	437.0	97.1 ± 0.1	-0.5 ± 0.0	810.7	64.6 ± 0.3	-5.0 ± 0.3	1043.4
PUMA	70.8 ± 0.7	-6.5 ± 1.0	1568.5	59.8 ± 0.0	-9.8 ± 0.2	459.5	97.2 ± 0.0	-0.3 ± 0.0	1098.8	OOM	OOM	OOM
ERGNN	52.1 ± 1.6	-12.7 ± 1.4	164.7	58.8 ± 0.5	-9.0 ± 0.4	80.2	85.4 ± 0.5	-1.0 ± 0.4	115.1	41.6 ± 0.1	-12.4 ± 0.5	94.1
GCDM	66.7 ± 1.1	-9.6 ± 0.5	942.3	55.9 ± 0.3	-9.9 ± 0.4	302.8	96.9 ± 0.1	-0.6 ± 0.1	454.5	63.3 ± 0.2	-5.1 ± 0.2	741.9
SSM	57.5 ± 1.3	-11.7 ± 0.9	161.7	59.2 ± 0.4	-8.7 ± 0.3	80.0	95.8 ± 0.5	-0.6 ± 0.3	114.0	56.7 ± 0.6	-6.6 ± 1.2	93.6
Bare	3.5 ± 0.2	-90.8 ± 0.5	141.3	9.0 ± 0.1	-88.4 ± 0.1	91.2	9.9 ± 0.0	-92.5 ± 0.0	399.2	5.6 ± 0.0	-92.9 ± 0.1	961.9
EWC	2.1 ± 0.0	-90.9 ± 0.2	816.8	9.2 ± 0.1	-88.0 ± 0.1	257.7	12.6 ± 0.8	-89.6 ± 0.8	533.9	10.6 ± 0.2	-84.3 ± 0.8	1123.7
MAS	3.7 ± 0.2	-64.3 ± 0.8	293.2	4.6 ± 0.0	-62.8 ± 1.1	145.9	7.8 ± 1.6	-73.9 ± 1.5	443.9	13.1 ± 0.5	-60.2 ± 0.4	993.1
GEM	3.8 ± 0.0	-89.4 ± 0.1	10707.3	9.0 ± 0.0	-71.8 ± 1.0	607.2	11.3 ± 1.3	-89.3 ± 2.5	813.1	10.2 ± 0.9	-83.8 ± 1.9	1045.2
TWP	13.9 ± 4.5	-62.9 ± 7.0	538.6	4.7 ± 2.5	-37.4 ± 7.3	203.0	OOM	OOM	OOM	OOM	OOM	OOM
LWF	2.7 ± 0.5	-91.7 ± 0.6	100.9	9.0 ± 0.0	-78.2 ± 0.6	64.6	10.0 ± 0.1	-92.4 ± 0.1	269.4	9.9 ± 0.9	-88.0 ± 0.9	610.2
STREAMCGL	68.9 ± 0.6	-6.3 ± 0.7	161.9	60.9 ± 0.6	-7.1 ± 0.5	58.9	97.5 ± 0.0	-0.4 ± 0.0	259.7	65.5 ± 0.4	-4.5 ± 1.1	194.1

Complexity Analysis: STREAMCGL processes snapshot $\mathcal{G}^{(t)}$ in $O(\ell|E^{(t)}|d + n^{(t)}Kd)$ time using $O(|\mathcal{C}^{(t)}|Kd)$ memory, independent of stream length (analysis in App. B.5).

4 EXPERIMENTAL RESULTS

Our evaluation investigates three questions:
(i) RQ1 (Quality). Under a fixed memory budget, how does gradient-free condensation compare to gradient-based condensation and to regularization- and replay-based Continual Graph Learning (CGL), across streams with different distribution shifts? **(ii) RQ2 (Efficiency and scale).** What is the computation cost of maintaining the condensed memory, and does it permit streams that defeat existing condensation methods? **(iii) RQ3 (Transferability).** Does a memory condensed in the WL feature space, without reference to any trained model, remain useful across GNN backbones?

We establish that STREAMCGL is the only method that is robust to distributions shifts over *both* temporal drifts and class labels (Table 1) while condensing up to $111\times$ (Table 3) faster than state-of-the-art condensation baselines.

Our codebase is available in supplementary; and other reproducibility details are included in App. E.

Table 1: Robustness across all nine dataset–setting combinations (Tables 2, 3, and 4). Within each combination, methods are ranked by the metric (1 = best; ties averaged, OOM/OOT assigned the worst rank), and *gap* is the shortfall from the best value: relative for AP, in percentage points for AF. Lower is better in every column. STREAMCGL is the only method whose worst case stays within 20% (AP) and 9 points (AF) of the best; every baseline degrades by at least 49% or 26 points in some setting. Best in dark, second best in light.

Method	AP gap (%)			AF gap (pts)		
	Rank	Mean	Worst	Rank	Mean	Worst
STREAMCGL	2.4	3.8	19.8	2.5	1.9	8.9
CAT	4.5	18.2	49.8	5.6	8.1	42.7
GCDM	4.9	18.1	50.6	4.9	6.4	26.8
SSM	5.4	21.2	49.8	4.4	7.7	38.4
GEM	5.9	50.2	94.6	7.7	45.3	85.1
ERGNN	6.6	24.2	53.1	5.8	8.5	38.3
PUMA	6.6	33.9	100.0	6.6	8.6	35.8
Bare	7.3	55.1	95.6	10.2	52.1	93.9
EWC	7.3	52.9	97.0	8.5	47.9	89.9
LWF	7.6	56.1	95.3	9.6	49.6	90.6
TWP	9.4	64.1	100.0	6.4	18.5	48.8
MAS	10.0	72.1	94.8	5.8	32.3	75.4

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate on the six datasets listed in Table 5 in App. E.2. ogbn-arxiv, ogbn-MAG, and Elliptic Bitcoin carry timestamps and support temporal splits.

Tasks and continual settings. A *task* corresponds to the arrival of a new graph snapshot: at task T_k, snapshot $\mathcal{G}^{(k)}$ arrives, the model is updated using only this snapshot and the maintained memory, and it is then evaluated on the test sets of all tasks seen so far. Snapshots share no inter-task edges.

We evaluate STREAMCGL across three benchmarks. In all three benchmarks, the prediction problem is identical: after training on $\mathcal{G}^k$, the model receives a test node from any snapshot seen so far and must predict its class label from among all classes observed up to T_k. The benchmarks differ only in how the nodes are partitioned into the task sequence: (1) In *Time-IL*, each task (snapshot) contains the induced subgraphs over nodes of one specific time period (e.g., one year of Arxiv papers) and thus evaluates memory retention over temporal shifts. (2) In *Recurrent class-IL* (Liu et al.,

2023; 2024), tasks are constructed from classes rather than timestamps. Each snapshot contains nodes of two previously unseen classes. The test set, however, is a sample drawn over from the held-out set over all snapshots seen so far. Hence, this setting evaluates memory retention over node classes. (3) In *Mixed-IL*, each time period is further partitioned into sub-tasks with disjoint class sets, so the classes of a period arrive in separate, unevenly sized bursts rather than as one balanced chunk.

Baselines. We compare against three families. *(i) Condensation- and replay-based*: CaT (Liu et al., 2023), PUMA (Liu et al., 2024), ERGNN (Zhou & Cao, 2021), GCDM (Liu et al., 2022), and SSM (Zhang et al., 2022b).*(ii) Regularization and memory-based*: EWC (Kirkpatrick et al., 2017), MAS (Aljundi et al., 2018), GEM (Lopez-Paz & Ranzato, 2017), TWP (Liu et al., 2021). *(iii) Distillation-based*: LWF (Li & Hoiem, 2017).

In addition, we also include *Bare*, which performs sequential fine-tuning with no continual-learning and acts as lower-bound reference. To get an upper-bound reference, *Full Data* retrains on the complete task history at every step.

Metrics. Let $M_{k,j}$ denote the classification accuracy on the held-out test split of snapshot $\mathcal{G}^{(j)}$, measured after the model has been trained sequentially through tasks $T_1, \ldots, T_k$ ($j \leq k$), with predictions ranging over all classes observed up to task T_k. We report: *(i) Average Performance*, $AP_k = \frac{1}{k}\sum_{j=1}^{k} M_{k,j}$, the mean accuracy over all tasks seen so far. Higher is better. *(ii) Average Forgetting*, $AF_k = \frac{1}{k-1}\sum_{j=1}^{k-1}\big(M_{k,j} - M_{j,j}\big)$, also known as backward transfer (Liu et al., 2023), is the average change in accuracy on each past task relative to when it was first learned. Negative AF_K quantifies forgetting, while positive values indicate beneficial backward transfer.

Protocol. All methods train on a 2-layer GCN with hidden dimension 256. § 4.3 evaluates other backbones. The memory budget is set to $b = 10\%$ of the stream unless stated otherwise. Results are reported over three seeds.

Table 3: Time-IL: For each column, the best result is highlighted in dark and the second best in light (higher is better for AP and AF; lower is better for Time). Full Data serves as a reference and is excluded from the ranking. OOT: exceeded 6 hours.

Methods	ogbn-arxiv			ogbn-mag			elliptic bitcoin		
	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓
Full Data	52.1 ± 0.5	-5.4 ± 0.6	63.4	25.4 ± 0.0	-0.1 ± 0.0	611.8	97.2 ± 0.0	0.5 ± 0.0	321
CAT	42.7 ± 0.2	-11.9 ± 0.3	1590.5	12.2 ± 0.1	-3.2 ± 0.1	12565.5	52.2 ± 0.6	-42.8 ± 0.8	612.9
PUMA	40.7 ± 0.5	-11.7 ± 0.5	2275.4	OOT	OOT	OOT	59.7 ± 1.1	-35.9 ± 1.0	536.8
ERGNN	41.4 ± 0.2	-13.0 ± 0.4	59.7	11.4 ± 0.2	-2.6 ± 0.4	54.7	57.3 ± 0.1	-38.4 ± 0.0	107.4
GCDM	43.0 ± 0.3	-11.2 ± 0.4	541.7	12.0 ± 0.1	-2.9 ± 0.0	2109.4	63.3 ± 1.2	-26.9 ± 1.0	454.1
SSM	43.2 ± 0.3	-10.8 ± 0.5	59.9	12.2 ± 0.1	-2.9 ± 0.1	136.0	57.1 ± 0.3	-38.5 ± 0.4	107.4
Bare	45.4 ± 0.7	-11.5 ± 0.8	72.5	22.5 ± 0.0	-5.4 ± 0.0	254.3	74.7 ± 2.4	-22.5 ± 2.4	122.8
EWC	38.4 ± 1.9	-8.0 ± 0.8	163.3	24.3 ± 0.1	-3.4 ± 0.1	250.8	91.7 ± 0.9	-2.7 ± 1.2	534.3
MAS	18.6 ± 2.6	-10.9 ± 1.8	149.9	11.8 ± 0.2	-0.0 ± 0.2	509.9	67.4 ± 10.8	-0.6 ± 1.3	152.5
GEM	47.6 ± 0.2	-7.8 ± 0.9	367.3	22.0 ± 0.2	-5.3 ± 0.3	155.3	92.8 ± 0.9	-3.9 ± 0.9	1377.7
TWP	24.3 ± 4.2	-2.2 ± 0.8	133.8	20.9 ± 0.1	-0.3 ± 0.3	218.8	71.7 ± 4.9	-0.1 ± 0.2	440.6
LWF	43.8 ± 0.3	-12.7 ± 0.0	51.8	22.4 ± 0.1	-5.3 ± 0.0	158.6	74.6 ± 2.5	-22.9 ± 2.6	90.6
STREAMCGL	47.9 ± 0.6	-3.2 ± 1.1	36.3	19.5 ± 0.5	-0.6 ± 0.3	113.2	87.0 ± 1.1	-9.0 ± 1.1	46.8

4.2 BENCHMARKING RESULTS

Before we deep dive into the results at various benchmarking scenarios, in Table 1, we summarize performance across all nine scenarios, which comprehensively surfaces its clear superiority across all baselines. Its decisive property is that it never degrades sharply; its worst case is within 20% (AP) and 9 points (AF) of the best, whereas every baseline falls at least 49% or 26 points short in some setting. Robustness across regimes, rather than dominance within any one, is our central empirical claim, in addition to being orders of magnitudes faster than the closest baseline CAT.

Class-incremental Learning: Table 2 reports results under Recurrent class-IL. Three findings stand out. First, rehearsal memory is indispensable in this setting: all methods without one (Bare, EWC, MAS, GEM, TWP, LwF) collapse, with AF near -90%, since each incoming snapshot supervises

only two classes while evaluation spans all classes seen so far. Second, STREAMCGL achieves the best AP on the two largest datasets (Reddit, Products) and is within 1.9 and 0.8 points of the best gradient-based condensation method on CoraFull and Arxiv, respectively, while attaining the best or second-best AF on three of the four datasets; on Products, its AF trails the best replay baselines, but a substantially higher AP. Third, this quality comes at a fraction of the cost: STREAMCGL condenses 3–12× faster than CaT and PUMA, and it is the only condensation method to complete on Products, where PUMA runs out of memory. Notably, on Arxiv the condensation methods exceed even Full Data.

Time-incremental Learning: Table 3 presents the results. We observe that chronological splits induce a milder distribution shift than class-based splits since most classes reappear naturally across snapshots. Hence, memory-less methods remain viable and regularization-based approaches are competitive (EWC attains the best AP on ogbn-mag, GEM on Elliptic Bitcoin). It is precisely in this regime that prior condensation methods falter: CaT, PUMA, GCDM, SSM, and ERGNN trail even Bare on all three datasets. PUMA exceeds the time limit on ogbn-mag, and CaT requires over 110× the cost of STREAMCGL. In contrast, STREAMCGL is the only condensation approach that remains strong on most settings.

Mixed-IL: Mixed-IL is the most adversarial setting due to distribution shifts over both class labels and chronology (Table 4). All memory-less methods collapse (AF between −17% and −45%). Condensation baselines retain accuracy on ogbn-arxiv but at substantial forgetting (−8% to −12% AF), and fail entirely on ogbn-mag, plateauing at 11–12 AP. STREAMCGL exhibits an order of magnitude less forgetting than baselines on both datasets and is the only method that copes with ogbn-mag and recovering most of the Full Data reference at 8.5× lower cost—while CaT and PUMA spend 22× and 42× more time to reach half its accuracy.

Impact of condensation budget: We further study the impact of the condensation budget in App. E.4.

Table 4: Mixed-IL: best method highlighted in dark, second best in light (higher is better for AP and AF; lower is better for Time). Full Data is a reference, excluded from ranking.

Methods	ogbn-arxiv			ogbn-mag		
	AP(%) ↑	AF(%) ↑	Time(s) ↓	AP(%) ↑	AF(%) ↑	Time(s) ↓
Full Data	49.1 ±0.2	−5.8 ±0.3	182	25.2 ± 0.0	−3.3 ± 0.0	2587.1
CAT	43.7 ± 1.0	−8.4 ± 0.4	1933.6	12.0 ± 0.1	−4.8 ± 0.1	6763.7
PUMA	41.2 ± 0.4	−11.1 ± 0.2	1708.0	11.0 ± 0.1	−4.3 ± 0.1	12876.7
ERGNN	42.0 ± 0.3	−9.3 ± 0.4	178.3	11.7 ± 0.1	−3.5 ± 0.1	247.0
GCDM	44.3 ± 0.4	−8.1 ± 0.5	928.6	11.8 ± 0.1	−4.6 ± 0.1	2140.5
SSM	43.0 ± 0.4	−8.0 ± 0.8	175.9	11.8 ± 0.0	−4.4 ± 0.2	615.4
Bare	22.1 ± 0.5	−44.9 ± 0.6	178.9	9.5 ± 0.4	−37.5 ± 0.4	275.4
EWC	21.4 ± 0.5	−37.7 ± 1.6	1083.8	10.0 ± 0.1	−42.5 ± 0.4	681.3
MAS	9.6 ± 0.6	−17.1 ± 1.3	343.6	5.8 ± 0.2	−17.6 ± 1.4	963.9
GEM	24.7 ± 0.1	−37.5 ± 0.7	2850.3	10.3 ± 0.1	−40.8 ± 0.1	1723.8
TWP	10.8 ± 0.5	−24.2 ± 0.3	900.2	9.7 ± 0.9	−23.3 ± 3.2	552.1
LWF	21.6 ± 0.3	−43.3 ± 0.4	138.6	7.3 ± 0.8	−31.3 ± 1.5	190.0
STREAMCGL	41.8 ± 0.8	−2.2 ± 1.0	77.5	20.0 ± 0.2	−1.0 ± 0.2	303.6

4.3 GENERALIZATION ACROSS GNN BACKBONES

A central implication of condensing in the WL feature space (§3.1) is that the memory should generalize on the entire family of message-passing backbones. Unlike gradient-based condensation, which optimizes synthetic nodes against a specific reference model, the memory of STREAMCGL is constructed once and reused unchanged by every backbone. Table 6 in App.E.2 shows that regardless of the GNN backbone, the gap with Full training remains consistent, validating our hypotheses.

5 CONCLUSION

We presented STREAMCGL, a graph condensation framework for continual learning under realistic streaming constraints: a fixed, user-specified memory budget, single-pass access, and no knowledge of future stream statistics. Our central move is to stop *learning* synthetic nodes against a reference model and instead condense in a continuous Weisfeiler-Leman feature space—motivated by the 1-WL bound on message-passing GNNs—which recasts condensation as *metric compression*. This formulation brings the streaming-clustering toolbox to bear, yielding a gradient-free, one-pass algorithm with an $O(\log n \log k)$ approximation guarantee. Across three benchmark protocols spanning chronological, imbalanced, and recurrent class arrivals, STREAMCGL attains the best average accuracy rank among all methods and condenses up to two orders of magnitude faster than state-of-the-art condensation baselines.

6 REPRODUCIBILITY STATEMENT

All results presented in this paper are reproducible. All code and implementation related artifacts have been made publicly available anonymously to facilitate replication and verification. Experimental setups, including training steps, model configurations, and hardware details, are described in the paper. Additionally, all datasets used in this paper are publicly available, ensuring consistent and reproducible evaluation results.

7 AI/ LLMS USAGE

In this work, generative AI tools were primarily used for assistance in writing, formatting and code implementations (includes dataset preparations (cleaning/ formatting)). We have not used generative AI tools for developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in developing proofs, proposing hypotheses, designing research methodology or experiments, supporting qualitative and thematic data analysis or interpreting results. Generation of synthetic data sets and assistance with translations are not applicable to this work. Generative AI tools were used for creating or modifying figures/ images, creating/ editing software code, drafting and editing writeups to improve readability. All AI-assisted work, including LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Kian Ahrabian, Yishi Xu, Yingxue Zhang, Jiapeng Wu, Yuening Wang, and Mark Coates. Structure aware experience replay for incremental learning in graph-based recommender systems. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2832–2836, 2021.

Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. *Advances in neural information processing systems*, 22, 2009.

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)*, pp. 139–154, 2018.

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, May 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5103-0. URL https://doi.org/10.1007/s10994-009-5103-0.

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong, and Wenwu Zhu. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*, pp. 1292–1300, 2022.

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

Xinyi Gao, Guanhua Ye, Tong Chen, Wentao Zhang, Junliang Yu, and Hongzhi Yin. Rethinking and accelerating graph condensation: A training-free approach with class partition. In *Proceedings of the ACM on Web Conference 2025*, pp. 4359–4373, 2025.

Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE transactions on knowledge and data engineering*, 15(3):515–528, 2003.

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

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

Thanh Duc Hoang, Do Viet Tung, Duy-Hung Nguyen, Bao-Sinh Nguyen, Huy Hoang Nguyen, and Hung Le. Universal graph continual learning. *arXiv preprint arXiv:2308.13982*, 2023.

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.

Mohit Kataria, Sandeep Kumar, et al. Ugc: Universal graph coarsening. *Advances in Neural Information Processing Systems*, 37:63057–63081, 2024.

Seoyoon Kim, Seongjun Yun, and Jaewoo Kang. Dygrain: An incremental learning framework for dynamic graphs. In *IJCAI*, pp. 3157–3163, 2022.

TN Kipf. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

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

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

Edo Liberty, Ram Sriharsha, and Maxim Sviridenko. An algorithm for online k-means clustering. In *2016 Proceedings of the eighteenth workshop on algorithm engineering and experiments (ALENEX)*, pp. 81–89. SIAM, 2016.

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, pp. 8653–8661, 2021.

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

Yilun Liu, Ruihong Qiu, and Zi Huang. Cat: Balanced continual graph learning with graph condensation. In *2023 IEEE International Conference on Data Mining (ICDM)*, pp. 1157–1162. IEEE, 2023.

Yilun Liu, Ruihong Qiu, Yanran Tang, Hongzhi Yin, and Zi Huang. Puma: Efficient continual graph learning for node classification with graph condensation. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

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

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

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

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

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Li Sun, Junda Ye, Hao Peng, Feiyang Wang, and Philip S Yu. Self-supervised continual graph learning in adaptive riemannian spaces. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 4633–4642, 2023.

Matteo Togninalli, M. Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten M. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. *NeurIPS*, abs/1906.01277, 2019. URL `http://arxiv.org/abs/1906.01277`.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019.

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*, pp. 1878–1888, 2022.

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

Di Wei, Yu Gu, Yumeng Song, Zhen Song, Fangfang Li, and Ge Yu. Incregnn: incremental graph neural network learning by considering node and parameter importance. In *International Conference on Database Systems for Advanced Applications*, pp. 739–746. Springer, 2022.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/wu19e.html.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

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*, pp. 2861–2868, 2020.

Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, Guojie Song, and Sunghun Kim. Continual learning on dynamic graphs via parameter isolation. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pp. 601–611, 2023a.

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, 2022a.

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

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

Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. Topology-aware embedding memory for continual learning on expanding networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4326–4337, 2024a.

Xikun Zhang, Dongjin Song, and Dacheng Tao. Continual learning on graphs: Challenges, solutions, and opportunities. *arXiv preprint arXiv:2402.11565*, 2024b.

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, pp. 4714–4722, 2021.

A Related Work

Continual graph learning (CGL) methods are commonly grouped into three paradigms Zhang et al. (2024b): regularization-based approaches, parameter-isolation methods, and memory-replay techniques. We review each, then turn to graph condensation and to the streaming-clustering foundations our method builds on.

Regularization-based methods mitigate forgetting by constraining parameter updates across tasks. Representative approaches include GraphSail Xu et al. (2020), which preserves local and global graph structure; MSCGL Cai et al. (2022), which enforces orthogonality across task updates; and LKGE Cui et al. (2023), which uses ℓ_2 regularization to retain prior knowledge. Others rely on topology preservation or knowledge distillation, such as TWP Liu et al. (2021) and RieGrace Sun et al. (2023). These methods are attractive because they require no stored data, but since they protect parameters rather than the data distribution, their effectiveness depends on how much the task distribution shifts: our experiments confirm that they remain competitive under mild temporal drift, yet degrade sharply when classes arrive in imbalanced bursts or recur after long gaps.

Parameter-isolation methods preserve past performance by allocating separate model components to different tasks. HPNs Zhang et al. (2022a) expand capacity while sharing fundamental features, and PI-GNN Zhang et al. (2023a) isolates parameters corresponding to stable and evolving graph components. These methods prevent interference by construction, but model complexity grows with the number of tasks, and most require task identities at inference—an assumption that does not hold in the class-incremental setting we target.

Memory-replay methods retrain on representative information from past tasks and are generally the most effective family for CGL. They differ in *what* they replay: ER-GNN Zhou & Cao (2021) stores representative nodes; SSM Zhang et al. (2022b) replays sparsified computation subgraphs; SEM Zhang et al. (2023b) uses Ricci-curvature-based sparsification; PDGNN-TEM Zhang et al. (2024a) replays topology-aware embeddings; and IncreGNN Wei et al. (2022) selects influential nodes. Other variants combine replay with model adaptation or generative rehearsal Rakaraddi et al. (2022); Wang et al. (2022); Ahrabian et al. (2021). *Hybrid* approaches mix paradigms, such as DyGRAIN Kim et al. (2022), which pairs replay with distillation, and UGCL Hoang et al. (2023), which additionally preserves global structure. The common bottleneck is that faithful replay of a node requires its computation subgraph, whose size grows exponentially with GNN depth.

Graph condensation sidesteps this bottleneck by replacing stored samples with a small set of synthetic nodes. In the *static* setting, methods such as GCDM Liu et al. (2022) synthesize condensed graphs by distribution matching, graph coarsening merges nodes into supernodes Kataria et al. (2024), and recent work reduces cost by avoiding inner-loop training altogether Gao et al. (2025). In the *continual* setting, CaT Liu et al. (2023) aggregates balanced condensed graphs across tasks and PUMA Liu et al. (2024) improves condensation quality via pseudo-labeling; both are the current state of the art. Despite strong empirical results, these methods optimize synthetic nodes by gradient descent against a reference model, require multiple passes over each snapshot, and allocate memory per task or class using class statistics that a streaming system does not have in advance. Our work differs on all three counts: condensation is gradient-free and model-free, operates in a single pass, and respects a fixed global budget set by the user. It is also distinct from training-free *static* condensation Gao et al. (2025), which assumes full access to the graph and does not address the streaming constraints—bounded memory across an unbounded stream, unknown class arrival, and irrevocable decisions—that define the continual problem.

Streaming clustering. Our condensation objective reduces to k-means in a fixed feature space, which connects it to a mature literature on streaming and coreset-based clustering. Guha et al. Guha et al. (2003) introduced the divide-and-conquer scheme for streaming k-median; Ailon et al. Ailon et al. (2009) extended its composition analysis to k-means; and Liberty et al. Liberty et al. (2016) gave a one-pass online k-means algorithm with bicriteria guarantees. We build directly on these results, adapting them with center weights, periodic compression, and class-aware budgeting so that a fixed, user-specified memory budget is respected across an unbounded stream. To our knowledge, this connection between graph condensation and streaming clustering has not previously been exploited, and it is what yields the first approximation guarantee for condensation in the continual setting.

Positioning. Our work falls within the memory-replay paradigm and targets its central scalability bottleneck. Rather than *learning* synthetic nodes against a reference model, we condense in a continuous Weisfeiler-Leman feature space Shervashidze et al. (2011); Togninalli et al. (2019)—motivated by the fact that message-passing GNNs are bounded by the 1-WL test Xu et al. (2019); Morris et al. (2019)—which reduces condensation to metric compression and makes the streaming-clustering toolbox applicable. The result is a replay mechanism that is gradient-free, single-pass, fixed-budget, and reusable across GNN backbones.

B Theoretical Analyses and Proofs

B.1 Edge-free representations

Proof. For the edge-free graph the adjacency matrix is $\mathbf{0}$; hence $A' = \mathbf{0} + I = I$, its degree matrix is $D' = I$, and the induced operator is $\boldsymbol{\mathcal{A}}_{\mathbf{0}} = I^{-\frac{1}{2}}\, I\, I^{-\frac{1}{2}} = I$. Therefore $R_\ell(\mathbf{Z}, \mathbf{0}) = \boldsymbol{\mathcal{A}}_{\mathbf{0}}^{\ell}\, \mathbf{Z} = \mathbf{Z}$ for every $\mathbf{Z} \in \mathbb{R}^{n\times d}$; taking $\mathbf{Z} = \mathbf{Z}^{(\ell)} = R_\ell(\boldsymbol{X}, A)$ proves the identity. For the consequence, applying any linear readout g to both sides yields $g\big(R_\ell(\mathbf{Z}^{(\ell)}, \mathbf{0})\big) = g\big(R_\ell(\boldsymbol{X}, A)\big)$, i.e., identical outputs on $\tilde{\mathcal{G}}$ and $\mathcal{G}$. □

B.2 Proof of Lemma 2

Proof. The difference between the two outputs can be written as

$$\|g(\mathbf{c}_z^*) - g(\mathbf{z})\| = \|\mathbf{c}_z^*\,\Theta - \mathbf{z}\,\Theta\| = \|(\mathbf{c}_z^* - \mathbf{z})\,\Theta\|.$$

Using the submultiplicativity of matrix norms, we obtain

$$\|(\mathbf{c}_z^* - \mathbf{z})\,\Theta\| \;\le\; \|\mathbf{c}_z^* - \mathbf{z}\| \cdot \|\Theta\|,$$

which establishes the first claim.

For the aggregate bound, fix a condensed set C and, for each $\mathbf{z} \in Z$, let $\mathbf{c}_z^*$ denote its nearest synthetic node, so that $\|\mathbf{c}^* - \mathbf{z}\| = D(\mathbf{z}, C)$. Applying the first claim gives

$$\|g(\mathbf{c}^*) - g(\mathbf{z})\|^2 \;\le\; \|\Theta\|^2\, D(\mathbf{z}, C)^2.$$

Averaging over all $\mathbf{z} \in Z$ yields

$$\begin{aligned}\frac{1}{n}\sum_{\mathbf{z}\in Z}\|g(\mathbf{c}_z^*) - g(\mathbf{z})\|^2 \;&\le\; \|\Theta\|^2 \cdot \frac{1}{n}\sum_{\mathbf{z}\in Z} D(\mathbf{z}, C)^2 \\ &=\; \|\Theta\|^2\, \phi(C).\end{aligned}$$

□

B.3 Proof of Lemma 3

Proof. For a center set $C \subset \mathbb{R}^d$, write the empirical distortion of Eq. equation 4 as

$$\phi(C) = \frac{1}{n}\sum_{\mathbf{z}\in Z} D(\mathbf{z}, C)^2, \qquad D(\mathbf{z}, C) = \min_{\mathbf{c}\in C}\|\mathbf{z} - \mathbf{c}\|_2,$$

and let $\phi_{OPT}^k = \min_{C:\,|C|=k}\phi(C)$. The minimum is attained: for any fixed assignment of points to centers, the optimal centers are the cluster centroids, and there are only finitely many partitions of Z into at most k clusters.

Monotonicity. Let C_k^* with $|C_k^*| = k$ be an optimal solution, i.e., $\phi(C_k^*) = \phi_{OPT}^k$. First observe that enlarging a center set can never increase the cost: for any $C \subseteq C'$ and any $\mathbf{z}$, $D(\mathbf{z}, C') \le D(\mathbf{z}, C)$, and hence $\phi(C') \le \phi(C)$. If $Z \subseteq C_k^*$, then $\phi_{OPT}^k = 0$ and the claim is immediate. Otherwise, pick any $\mathbf{z}_0 \in Z \setminus C_k^*$ and set $C' = C_k^* \cup \{\mathbf{z}_0\}$, so that $|C'| = k+1$. Since $D(\mathbf{z}_0, C') = 0$ while the cost of every other point cannot increase,

$$\phi_{OPT}^{k+1} \;\le\; \phi(C') \;\le\; \phi(C_k^*) - \frac{1}{n}\,D(\mathbf{z}_0, C_k^*)^2 \;\le\; \phi_{OPT}^k.$$

Choosing $\mathbf{z}_0 \in \arg\max_{\mathbf{z}\in Z} D(\mathbf{z}, C_k^*)$ yields the quantitative form $\phi_{OPT}^{k+1} \leq \phi_{OPT}^k - \frac{1}{n}\max_{\mathbf{z}\in Z} D(\mathbf{z}, C_k^*)^2$.

Zero distortion at $k = n$. For $k = n$, the choice $C = Z$ is feasible and satisfies $D(\mathbf{z}, C) = 0$ for every $\mathbf{z} \in Z$; hence $\phi_{OPT}^n = 0$. Combined with monotonicity and $\phi_{OPT}^k \geq 0$, the sequence $\{\phi_{OPT}^k\}_{k=1}^n$ decreases monotonically to zero.

Training equivalence. Any zero-cost solution must contain every distinct point of Z; thus at $k = n$ the optimal center set coincides with Z itself (if Z contains duplicate points, each distinct point appears as a center and multiplicities are carried by the center weights). The condensed graph $\tilde{\mathcal{G}}_{OPT}^n$ therefore consists of exactly the aggregated feature vectors in Z, with class labels attached since condensation is performed class-wise. The training set it induces is identical to Z, and training a linear model on $\tilde{\mathcal{G}}_{OPT}^n$ is equivalent to training on the full aggregated feature set $Z \in \mathbb{R}^{n\times d}$. □

B.4 Proof of Lemma 5

Proof. By the guarantees of Liberty et al. (2016), the streaming stage of Algorithm 2 is an (a, b)-approximation for the k-means objective with $ak = O(k \log n \log \gamma n)$ and $b = O(\log n)$ in expectation; the weighted centers it maintains constitute the intermediate memory, which therefore has expected size $O(k \log n \log \gamma n)$. For the compression stage, k-means++ is a $(1, b')$-approximation with $b' = O(\log k)$ in expectation (Arthur & Vassilvitskii, 2007), and its guarantee extends to integer-weighted inputs by treating weights as multiplicities. Applying Lemma 4 to A and A' yields a $\big(1,\ 2b + 4b'(b + 1)\big)$-approximation. Since both guarantees hold in expectation, one further step is required: the bound of Lemma 4 is linear in the cost of A, and the expected guarantee of A' holds conditionally on any fixed output of A; hence, by the tower property of expectation, the composed factor holds in expectation as well, giving $2b + 4b'(b + 1) = O(\log n \log k)$. The k centers returned by the compression stage form the condensed graph $\tilde{\mathcal{G}}$ with $|\tilde{\mathcal{G}}| = k$, whose distortion is precisely the objective of Eq. 4. □

B.5 Computational Complexity

We analyze the cost of processing one snapshot $\mathcal{G}^{(t)}$, for a single class; per-class summarizers are independent, so total costs sum over $\mathcal{C}^{(t)}$. Let d denote the node-feature dimension (preserved by WL refinement), ℓ the number of refinement rounds, and K the summarizer capacity. Note that Algorithm 3 is a practical variant of Algorithm 2. It retains the core ideas of the theoretical method while offering concrete efficiency gains—namely, parallel batch processing and a constant-memory footprint. We discuss the complexity of Algorithm 3.

Encoding. Computing $Z^{(\ell)}$ via Eq. 3 requires ℓ sparse matrix–vector products, costing $O(\ell\,|E^{(t)}|\,d)$ time for snapshot $\mathcal{G}^{(t)}$. This is the only stage that touches graph structure; it is performed once per snapshot, after which the snapshot's edges are discarded.

Summarization. For each incoming embedding $\mathbf{z}$, the summarizer performs a nearest-anchor search over $\mathcal{S}$, requiring $O(Kd)$ time since $|\mathcal{S}| \leq K$. The subsequent update—either a weighted-average merge of the running centroid $\boldsymbol{\mu}^*$ or the insertion of a new center—costs $O(d)$.

A reduction is triggered when $|\mathcal{S}|$ reaches K, and compresses the weighted set to K/m centers using weighted k-means, at a cost of $O(\frac{K}{m}Kd)$. Since only insertions grow $\mathcal{S}$, and $K(1 - 1/m)$ insertions are required between consecutive reductions, reductions occur at most once every $\Theta(K)$ insertions and their cost amortizes to $O(Kd/m)$ per processed embedding. The per-embedding cost is therefore dominated by the nearest-anchor search, giving an amortized $O(Kd)$; over a snapshot with $n^{(t)}$ nodes, summarization costs $O(n^{(t)}Kd)$.

Condensation for training. When a GNN update is required, each class memory is compressed to b_y centers by weighted k-means++, costing $O(Kb_yd)$ per class. This step is performed once per training round, not per stream element.

Memory. Each summarizer stores at most K tuples of two d-dimensional vectors and a scalar weight, i.e. $O(Kd)$ per class and $O(|\mathcal{C}^{(t)}|Kd)$ in total. Crucially, this bound depends on neither the stream length nor the size of any snapshot: the summarizer's footprint is independent of $|V^{(t)}|$,

$|E^{(t)}|$, and the degree distribution, and the condensed graph passed to training never exceeds the user-specified budget b.

Parallelism. Distances to different anchors are independent and parallelize across the K centers. Moreover, anchors $\mathbf{c}_i$ are updated lazily and remain fixed throughout a growth phase, so a mini-batch of embeddings can be processed concurrently against the same anchor set. This is an approximation of sequential execution—a point may open a new center where sequential processing would have merged it into a center created earlier in the same batch—but since all updates remain confined to a ball of radius r around each anchor, the effect on clustering quality is minor, as confirmed by our experiments.

C Pseudocode for the Online Approximation Algorithm

We provide the psuedocodes in this section. Algorithm 2 refers to the online algorithm with the theoretical guarantees. Algorithm 3 is a practical variant of Algorithm 2. It retains the core ideas of the theoretical method while offering concrete efficiency gains—namely, parallel batch processing and a bounded-memory footprint independent of the stream size.

D Online Cluster Summarizer

The online cluster summarizer (Algorithm 3) serves as an intermediate streaming surrogate for the distortion-minimization objective in Definition 5. Its purpose is to maintain, in a single pass, a compact weighted representation of an unbounded stream of WL-refined node embeddings that preserves the geometry relevant for downstream clustering. The summarizer maintains at most K *weighted representatives* at any time, where K is a hyper-parameter controlling the size of an intermediate summary used to approximate the clustering objective and is never exposed to model training. The final condensed graph is always constructed using exactly b synthetic nodes, ensuring a strict and persistent training-time memory budget.

Let $\{\mathbf{h}_1, \mathbf{h}_2, \ldots\}$ denote a stream of WL-refined node representations for a fixed class. The summarizer maintains a weighted set

$$\mathcal{S} = \{(\mathbf{c}_i, \mathbf{p}_i, w_i)\}_{i=1}^{|\mathcal{S}|}, \qquad |\mathcal{S}| \le K,$$

where $\mathbf{c}_i \in \mathbb{R}^d$ is a fixed *anchor* defining a local Voronoi region, $\mathbf{p}_i \in \mathbb{R}^d$ is the running centroid of points assigned to that region, and w_i is the corresponding weight. New tuples are initialized with $\mathbf{c}_i = \mathbf{p}_i$ and $w_i = 1$.

This representation is designed so that clustering performed on $\mathcal{S}$ (with weights) approximates clustering on the full data stream with respect to the ℓ_2 distortion objective, analogous to streaming coreset constructions for Euclidean clustering.

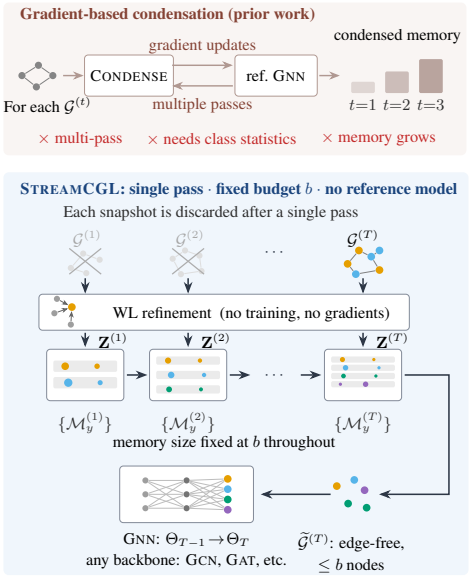

Figure 1: StreamCGL vs. gradient-based condensation. Prior methods (top) optimize synthetic nodes against a reference model over multiple passes, with memory growing over the stream; StreamCGL (bottom) encodes each snapshot once and maintains class-wise memories of fixed size b.

Algorithm. The summarizer alternates between a *growth* phase and a *reduction* phase, controlled by a distance threshold r. At initialization, the first point initializes $\mathcal{S} = \{(\mathbf{h}_1, \mathbf{h}_1, 1)\}$ and $r = r_0$. The first K points are always inserted, allowing r_0 to be set as the minimum pairwise distance among them.

For each incoming point $\mathbf{h}$, the nearest anchor

$$\mathbf{c}^* = \arg \min_{(\mathbf{c}_i, \mathbf{p}_i, w_i) \in \mathcal{S}} \|\mathbf{h} - \mathbf{c}_i\|_2$$

is identified. If $\|\mathbf{h} - \mathbf{c}^*\|_2 \leq r$, the point is merged by updating $\mathbf{p}^*$ via a weighted average and incrementing w^*. Otherwise, $\mathbf{h}$ is inserted as a new weighted center.

Crucially, anchor points $\mathbf{c}_i$ are updated *lazily*. This stabilizes region assignments and prevents excessive drift caused by early points dominating the summary, while ensuring that all updates remain confined within a ball of radius r around each anchor.

When $|\mathcal{S}|$ exceeds K, a reduction step is triggered. The weighted set is compressed to K/m representatives, where $m > 1$ is a small constant (e.g., $m = 2$), using weighted clustering or importance sampling. After reduction, the threshold r is increased by a factor $\alpha > 1$ and the growth phase resumes.

Assuming $\|\mathbf{h}\| \leq R$ for all points, the number of reduction phases is bounded by $O(\log_\alpha(R/r_0))$. This yields a single-pass algorithm with bounded memory and logarithmically many compression steps, a standard pattern in streaming approximation algorithms.

Properties. The summarizer preserves centroids locally through weighted averaging and globally through controlled reductions. As r increases, the representation transitions from fine-grained to progressively coarser summaries, reflecting the diminishing marginal utility of retaining additional centers once sufficient coverage of the feature space is achieved. At any time step, $\mathcal{S}$ constitutes a compact, online-maintained surrogate for the full data stream with respect to distortion minimization. This enables efficient approximation of the NP-hard condensation objective under strict single-pass and memory constraints, while ensuring that the final condensed graph used for training always respects the fixed budget b.

The computation complexity of the online summarizer is linear to K. The detailed analysis is presented in Appendix B.5.

E Additional Experimental Details

E.1 Hardware and Protocol

All experiments run on a Kaggle environment with a single NVIDIA T4 GPU (15 GB VRAM) and 30 GB CPU RAM. Train/validation/test splits are fixed at 60%/20%/20% across all datasets and settings. Unless stated otherwise, results are averaged over three seeds and reported with standard deviations.

E.2 Benchmark Construction

The three protocols share the same datasets and splits (§E.1) and differ only in how nodes are partitioned into the task sequence, isolating the effect of *how classes arrive* over the stream. In all cases, snapshots are presented as a stream with no inter-task edges.

Time-IL partitions each timestamped graph chronologically, so classes appear and accumulate as they naturally do. For Arxiv we follow the BeGin benchmark: task 1 contains papers published in 1997 or earlier, and task $i \geq 2$ adds papers from year $1996 + i$. For MAG, task i comprises papers from year $2009 + i$. For Elliptic Bitcoin we use the temporal snapshots provided by the source without further processing; nodes with unknown labels are retained in the graph for message passing but excluded from training and evaluation, with the 60/20/20 split applied to the labeled nodes.

Mixed-IL builds on Time-IL by further subdividing each temporal snapshot into sub-snapshots with mutually exclusive class sets, so that the classes of a period arrive in separate, unevenly sized bursts rather than as one balanced chunk. Each Arxiv snapshot is split into three sub-snapshots (e.g., a five-class snapshot yields sub-snapshots of two, two, and one class); each MAG snapshot is split into five.

Table 5: Dataset Statistics

Dataset	Nodes	Edges	Features	Classes
CoraFull McCallum et al. (2000)	19793	130,622	8,710	70
ogbn-arxiv Hu et al. (2020)	169,343	1,166,243	128	40
Reddit Hamilton et al. (2017a)	227,853	114,615,892	602	40
Products Hu et al. (2020)	2,449,028	61,859,036	100	46
ogbn-mag Hu et al. (2020)	736,389	5,416,271	128	349
Elliptic Bitcoin Weber et al. (2019)	203,769	234,355	165	2

Table 6: AP(%) across different GNN backbones (Budget: 1%). Δ denotes the performance gap between the full-data model and STREAMCGL. Note that these numbers correspond to the original class-IL setting as proposed in Liu et al. (2023; 2024) and not from the recurrent class-IL setting.

	CoraFull			**Arxiv**		
Model	STREAMCGL	Full	Δ	STREAMCGL	Full	Δ
GCN	72.8	83.5	10.7	61.8	60.4	**-1.4**
GraphSAGE	71.9	82.2	10.3	54.0	65.3	1.3
SGC	70.9	83.6	12.7	58.5	60.9	2.4
GAT	71.6	81.7	**10.1**	61.5	OOM	–

Recurrent class-IL adopts the two-classes-per-task protocol of CaT and PUMA and extends it so that classes recur after a temporal gap. For each pair of consecutive snapshots $(2i, 2i{+}1)$, half of the nodes of each class are held out; the snapshots retain the other half. The held-out nodes of both snapshots are then combined into a new snapshot spanning all four of their classes, which is appended to the end of the task sequence. Each class pair is therefore encountered twice—once in its original snapshot and once, with fresh nodes, in an appended snapshot separated by many intervening tasks—testing retention across temporally distant exposures.

Table 7: Dataset Splits and Budgets

Datasets	Split Type	Budgets
CoraFull	Rec Class-IL	10%
Arxiv	Rec Class-IL, Time, Mixed(split of 3)	10%
Reddit	Rec Class-IL	10%
Products	Rec Class-IL	0.5%
Mag	Time, Mixed(split of 5)	10%
Elliptic Bitcoin	Time	10%

E.3 ADAPTING CONDENSATION BASELINES

CaT and PUMA cannot be run unmodified in our setting: both assume that no class spans multiple tasks and derive per-class condensation budgets from class-wise data ratios, which a single-pass system does not have in advance. We therefore make the minimal changes required for them to execute, leaving their condensation algorithms themselves untouched.

First, to accommodate classes that span tasks, we process the nodes of each class in two disjoint batches within the same task, condensing each batch without access to the other. No edges are removed; only labels are masked per batch. Second, to accommodate unknown class counts, we adopt adaptive budgeting: the total budget is divided evenly among all classes observed so far, and when a new class arrives, existing condensed nodes are uniformly subsampled to free its share. All methods, including STREAMCGL, operate under the same total budget throughout.

We note that Recurrent class-IL is built on CaT and PUMA's own evaluation protocol, so these baselines are assessed under the task construction they were designed for, modified only by the recurrence of class pairs. STREAMCGL requires no adaptation in any setting: class recurrence and unknown class counts are handled natively by its streaming design.

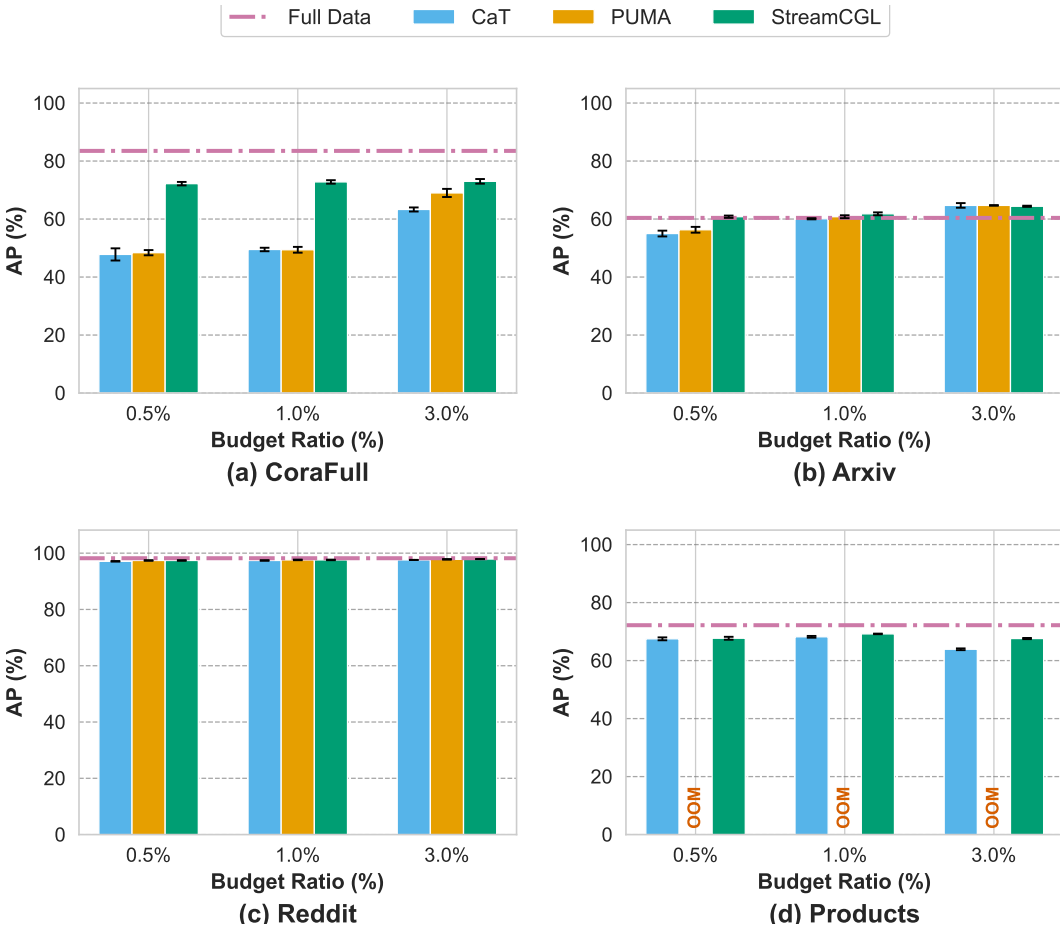

Figure 2: AP(%) comparison of methods with different budget ratios. Note that PUMA encounters Out-Of-Memory (OOM) error on the Products dataset and these numbers were recorded in the original class-IL setting which is different from the recurrence class-IL setting.

E.4 Impact of Condensation Budget

Fig. 2 reports the average performance (AP) of different methods under increasing memory budgets. As expected, performance improves for all methods as the budget increases and it shows the benefit of retaining a larger summary of past data. Across all datasets and budget ratios, our method StreamCGL consistently achieves the highest or near-highest performance, demonstrating strong robustness even in low-budget regimes. Notably, the performance gap between our method and existing condensation-based baselines is most pronounced at smaller budgets, indicating that our proposed distortion-aware condensation is more effective at utilizing limited memory. In contrast, gradient-based methods exhibit diminishing returns as the budget grows and fail to scale to larger datasets. In particular, PUMA encounters out-of-memory errors on the Products dataset for all budget settings. These results suggest that our gradient-free, edge-free condensation strategy StreamCGL scales better with higher budget.

Algorithm 2: Streaming stage (adapted from Liberty et al., 2016): online k-means with center weights, followed by compression to exactly k centers. The stream $(\mathbf{z}_1, \mathbf{z}_2, \ldots)$ consists of the WL-refined embeddings of a single class, concatenated across snapshots in order of arrival; $D(\cdot,\cdot)$ and $d(\cdot,\cdot)$ are as in Eq. 4, and n counts the points processed so far. This procedure serves as algorithm A followed by A′ in Lemma 5. Our additions to the original algorithm are the center weights and the final compression step.

procedure STREAMCONDENSE(stream $(\mathbf{z}_1, \mathbf{z}_2, \ldots)$; target number of centers k)
  $C \leftarrow$ first $k{+}1$ distinct points of the stream; $w(\mathbf{c}) \leftarrow 1$ for all $\mathbf{c} \in C$; $n \leftarrow k+1$
  $\delta \leftarrow \min_{\mathbf{c},\mathbf{c}' \in C} d(\mathbf{c},\mathbf{c}')^2 \,/\, 2$
  $q \leftarrow 0$; $f \leftarrow \delta / k$
  for each subsequent point $\mathbf{z}$ **do**
    $n \leftarrow n+1$; $p \leftarrow \min\big(D(\mathbf{z}, C)^2 / f,\ 1\big)$
    draw $u \sim \mathcal{U}(0,1)$
    if $u \le p$ **then** ▷ open a new center
      $C \leftarrow C \cup \{\mathbf{z}\}$; $w(\mathbf{z}) \leftarrow 1$; $q \leftarrow q+1$
    else ▷ assign to nearest center
      $\mathbf{c}^* \leftarrow \arg\min_{\mathbf{c} \in C} d(\mathbf{z},\mathbf{c})$; $w(\mathbf{c}^*) \leftarrow w(\mathbf{c}^*) + 1$
    if $q \ge 3k(1+\log n)$ **then** ▷ double the cost threshold
      $q \leftarrow 0$; $f \leftarrow 2f$
  $C' \leftarrow$ weighted k-means++ (Arthur & Vassilvitskii, 2007) on (C, w), returning exactly k centers
  return C'

Algorithm 3: Online Summarizer

procedure (**Input:** (Stream: $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$, m, K)
  Initialize: $r = r_0, \mathcal{S} = \{(\mathbf{h}_1, \mathbf{h_1}, 1)\}$ ▷ initialize with first element and unit weight
  for $i \leftarrow 2$ to n **do**
    $\mathbf{h} = \mathbf{h}_i$
    $(\mathbf{c}^*, \mathbf{p}^*, w^*) = \arg\min_{(\mathbf{c}_i, \mathbf{p}_i, w_i) \in \mathcal{S}} \|\mathbf{c}_i - \mathbf{h}\|_2$ ▷ find closest actual center
    if $\|\mathbf{c}^* - \mathbf{h}\|_2 \le r$ **then**
      $\mathbf{p}' \leftarrow \frac{w^* \cdot \mathbf{p}^* + \mathbf{h}}{w^*+1}$
      $w' \leftarrow w^* + 1$
      $\mathcal{S} \leftarrow \mathcal{S} \setminus (\mathbf{c}^*, \mathbf{p}^*, w^*)$
      $\mathcal{S} \leftarrow \mathcal{S} \cup (\mathbf{c}^*, \mathbf{p}', w')$ ▷ update proxy center and weight
    else if $\|\mathbf{c}^* - \mathbf{h}\|_2 > r$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup (\mathbf{h}, \mathbf{h}, 1)$ ▷ add new center
    if $|\mathcal{S}| = K$ **then** ▷ maximum capacity reached
      for $(\mathbf{c}_j, \mathbf{p}_j, w_j) \in \mathcal{S}$ **do**
        $\mathbf{c}_j \leftarrow \mathbf{p}_j$ ▷ update actual centers
      $Reduce(\mathcal{S}, K/m)$ ▷ Reduce the set of centers to K/m centers
      $r \leftarrow \alpha \cdot r$ ▷ increase distance threshold
  return S