GRAPHFLEX: STRUCTURE LEARNING FRAMEWORK FOR LARGE EXPANDING GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph structure learning is a core problem in graph-based machine learning, essential for uncovering latent relationships and ensuring model interpretability. However, most existing approaches are ill-suited for large-scale and dynamically evolving graphs, as they often require complete re-learning of the structure upon the arrival of new nodes and incur substantial computational and memory costs. In this work, we propose GraphFLEx—a unified and scalable framework for Graph Structure Learning in Large and Expanding Graphs. GraphFLEx mitigates the scalability bottlenecks by restricting edge formation to structurally relevant subsets of nodes identified through a combination of clustering and coarsening techniques. This dramatically reduces the search space and enables efficient, incremental graph updates. The framework supports 48 flexible configurations by integrating diverse choices of learning paradigms, coarsening strategies, and clustering methods, making it adaptable to a wide range of graph settings and learning objectives. Extensive experiments across 26 diverse datasets and graph neural network architectures demonstrate that GraphFLEx achieves state-of-the-art performance with significantly improved scalability. Our implementation is publicly available here.

1 INTRODUCTION

Graph representations capture relationships between entities, vital across diverse fields like biology, finance, sociology, engineering, and operations research (Zhou et al., 2020; Fout et al., 2017; Wu et al., 2020; Malik et al., 2025). While some relationships, such as social connections or sensor networks, are directly observable, many, including gene regulatory networks, scene graph generation (Gu et al., 2019), brain networks, (Zhu et al., 2021) and drug interactions, require inference (Allen et al., 2012). Even when available, graph data often contains noise, requiring denoising and recalibration. In such cases, inferring the correct graph structure becomes more crucial than the specific graph model or downstream algorithm. *Graph Structure Learning (GSL)* offers a solution, enabling the construction and refinement of graph topologies. GSL has been widely studied in both supervised and unsupervised contexts (Liu et al., 2022; Chen & Wu, 2022). In supervised GSL (s-GSL), the adjacency matrix and Graph Neural Networks (GNNs) are jointly optimized for a downstream task, such as node classification. Notable examples of s-GSL include $NodeFormer$ (Wu et al., 2022b), $Pro-GNN$ (Jin et al., 2020), $WSGNN$ (Lao et al., 2022), and $SLAPS$ (Fatemi et al., 2021). Unsupervised GSL (u-GSL), on the other hand, focuses solely on learning the underlying graph structure, typically through adjacency or laplacian matrices. Methods in this category include approximate nearest neighbours ($A-NN$) (Dong et al., 2011; Muja & Lowe, 2014), k-nearest neighbours ($k-NN$) (MacQueen et al., 1967; Wang & Zhang, 2006), covariance estimation ($emp.Cov.$) (Hsieh et al., 2011), graphical lasso ($GLasso$) (Friedman et al., 2008), $SUBLIME$ (Liu et al., 2022), and signal processing techniques like $l2$-model,log-model and $large$-model (Dong et al., 2016; Kalofolias, 2016).

Supervised graph structure learning (s-GSL) methods have demonstrated effectiveness in specific tasks; however, their reliance on labeled data and optimization for downstream objectives particularly node classification, significantly constrains their generalizability to settings where annotations are scarce or unavailable (Liu et al., 2022). Unsupervised graph structure learning (u-GSL) methods, which constitute the focus of this work, offer broader applicability. Nevertheless, both s-GSL and u-GSL approaches exhibit critical limitations in their ability to scale to large graphs or adapt efficiently to expanding datasets. To address these challenges, we introduce

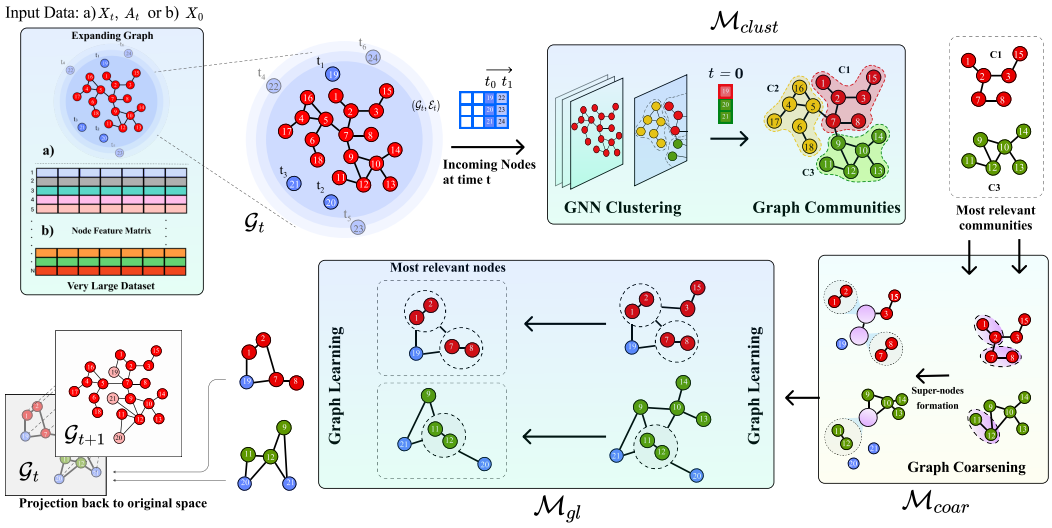
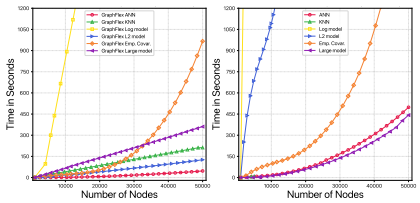


Figure 1: This figure illustrates the general pipeline of GraphFLEx, designed to efficiently handle both a) large datasets with missing structure and b) expanding graphs. Both scenarios can be modeled as expanding graphs (details in Section 3.1). GraphFLEx processes a graph ($\mathcal{G}_t$) and incoming nodes ($\mathcal{E}_{t+1}$) at time t, newly arriving nodes are shown with different timestamps and shades of blue to indicate their arrival time. Our framework comprises of three main components: i) **Clustering**, which infers $\mathcal{E}_{t+1}$ nodes to existing communities using a pre-trained model $\mathcal{M}_{\text{clust}}(\mathcal{G}_0)$ into smaller, more manageable communities; ii) Since these communities may still be large, a **Coarsening**, module is applied to further reduce their size while preserving essential structural information; and iii) Finally, a **Learning** module, where the structure associated with $\mathcal{E}_{t+1}$ nodes are learned using the coarsened graph, followed by projecting this structure onto the $\mathcal{G}_t$ graph to create graph $\mathcal{G}_{t+1}$.

GRAPHFLEX, a unified and scalable framework for *Graph Structure Learning in Large and Expanding Graphs*. GraphFLEx is built upon the coordinated integration of three foundational paradigms in graph processing: *graph clustering*, *graph coarsening*, and *structure learning*. While each of these methodologies has been studied extensively in isolation, their joint application within a single framework has remained largely unexplored. GraphFLEx combines these components in the principled manner in which they are algorithmically aligned to reinforce one another; clustering serves to localize the search space, coarsening reduces structural redundancy while preserving global properties, and structure learning operates efficiently within this refined context. This integration enables GraphFLEx to scale effectively to large datasets and accommodate dynamic graphs through *incremental updates*, eliminating the need for expensive re-training. Additionally, the framework supports **48 modular configurations**, enabling broad adaptability across datasets, learning objectives, and deployment constraints. Crucially, we establish *theoretical guarantees* on edge recovery fidelity and computational complexity, offering rigorous foundations for the framework's efficiency and reliability. As illustrated in Figure 1, GraphFLEx significantly reduces the candidate edge space by operating on structurally relevant node subsets. Empirical evaluations, summarized in Figure 2, demonstrate that GraphFLEx substantially outperforms existing baselines in both runtime and scalability.



(a) GraphFLEx (b) Vanilla

Figure 2: High computational time required to learn graph structures using existing methods, whereas GraphFLEx effectively controls computational growth, achieving near-linear scalability. Notably, Vanilla KNN failed to construct graph structures for more than 10K nodes due to memory limitations.

Key contributions of GraphFLEx:

- Unifies *multiple structure learning strategies* within a single, flexible framework.
- Demonstrates effectiveness in *handling growing graphs*.
- Enhances the *scalability* of graph structure learning on large-scale graphs.
- Serves as a *comprehensive framework* applicable individually for clustering, coarsening, and learning tasks.
- Provides both *theoretical guarantees and empirical evidence*, validating performance across diverse datasets.

2 Problem Formulation and Background

A graph $\mathcal{G}$ is represented using $\mathcal{G}(V, A, X)$ where $V = \{v_1, v_2 ... v_N\}$ is the set of N nodes, each node v_i has a $d-$dimensional feature vector x_i in $X \in \mathbb{R}^{N \times d}$ and $A \in \mathbb{R}^{N \times N}$ is adjacency matrix representing connection between i^{th} and j^{th} nodes when entry $A_{ij} > 0$. An expanding graph $\mathcal{E}_{\mathcal{G}}$ can be considered a variant of graph $\mathcal{G}$ where nodes v now have an associated timestamp τ_v. We can represent a expanding graph as a sequence of graphs, i.e., $\mathcal{E}_{\mathcal{G}} = \{\mathcal{G}_0, \mathcal{G}_1, ...\mathcal{G}_T\}$ where $\{\mathcal{G}_0 \subseteq \mathcal{G}_1 .... \subseteq \mathcal{G}_T\}$ at $\tau \in \{0, ...T\}$ timestamps. New nodes arriving at different timestamps are seamlessly integrating into initial graph $\mathcal{G}_0$.

Problem statement. Given a partially known or missing graph structure, our goal is to incrementally learn the whole graph (A or laplacian matrix). Specifically, we consider two unsupervised GSL goals:

Goal 1 ***Large Datasets with Missing Graph Structure:*** *Here, the graph structure is entirely unavailable, and existing methods are computationally infeasible for learning the whole graph in a single step. To address this, we first randomly partition the dataset into exclusive subsets. We then learn the initial graph $\mathcal{G}_0(V_0, X_0)$ over a small subset of nodes and incrementally expand it by integrating additional partitions, ultimately reconstructing the full graph $\mathcal{G}_T$.*

Goal 2 ***Partially Available Graph:*** *In this case, we only have access to the graph $\mathcal{G}_t$ at timestamp t, with new nodes arriving over time. The goal is to update the graph incrementally to obtain $\mathcal{G}_T$, without re-learning it from scratch at each timestamp.*

2.1 Graph Reduction

Graph reduction encompasses sparsification, clustering, coarsening, and condensation (Hashemi et al., 2024). GraphFlex employs clustering and coarsening to refine the set of relevant nodes for potential connections.

Graph Clustering. Graphs often exhibit global heterogeneity with localized homogeneity, making them well-suited for clustering (Fortunato, 2010). Clusters capture higher-order structures, aiding graph learning. Methods like DMoN (Tsitsulin et al., 2023) use GNNs for soft cluster assignments, while Spectral Clustering (SC) (Kamvar et al., 2003) and K-means (Wagstaff et al., 2001; MacQueen et al., 1967) efficiently detect communities. DiffPool (Bruna et al., 2014; Defferrard et al., 2016) applies SC for pooling in GNNs.

Graph Coarsening. Graph Coarsening (GC) reduces a graph $\mathcal{G}(V, E, X)$ with N nodes and features $X \in \mathbb{R}^{N \times d}$ into a smaller graph $\mathcal{G}_c(\widetilde{V}, \widetilde{E}, \widetilde{X})$ with $n \ll N$ nodes and $\widetilde{X} \in \mathbb{R}^{n \times d}$. This is achieved via learning a coarsening matrix $\mathcal{P} \in \mathbb{R}^{n \times N}$, mapping similar nodes in $\mathcal{G}$ to super-nodes in $\mathcal{G}_c$, ensuring $\widetilde{X} = \mathcal{P}X$ while preserving key properties (Loukas, 2019; Kataria et al., 2023; Kumar et al., 2023; Kataria et al., 2024).

2.2 Unsupervised Graph Structure Learning

Unsupervised graph learning spans from simple k-NN weighting (Wang & Zhang, 2006; Zhu et al., 2003) to advanced statistical and graph signal processing (GSP) techniques. Statistical methods, also known as probabilistic graphical models, assume an underlying graph $\mathcal{G}$ governs the joint distribution of data $X \in \mathbb{R}^{N \times d}$ (Koller & Friedman, 2009; Banerjee et al., 2008; Friedman et al., 2008). Some approaches (Dempster, 1972) prune elements in the inverse sample covariance matrix $\widehat{\Sigma} = \frac{1}{d-1} X X^T$ and sparse inverse covariance estimators, such as Graphical Lasso (GLasso) (Friedman et al., 2008): $\text{maximize}_{\Theta} \log\det\Theta - \text{tr}(\widehat{\mathbf{\Sigma}}\mathbf{\Theta}) - \rho\|\Theta\|_1$, where Θ is the inverse covariance matrix. However, these methods struggle with small sample sizes. Graph Signal Processing (GSP) techniques analyze signals on known graphs, ensuring properties like smoothness and sparsity. Signal smoothness on a graph $\mathcal{G}$ is quantified by the Laplacian quadratic form: $Q(\mathbf{L}) = \mathbf{x}^T\mathbf{L}\mathbf{x} = \frac{1}{2}\sum_{i,j} w_{ij}(\mathbf{x}(i) - \mathbf{x}(j))^2$. For a set of vectors X, smoothness is measured using the Dirichlet energy (Belkin et al., 2006): $\text{tr}(X^T L X)$. State-of-the-art methods $l2-$model, $log-$model and $large-$model (Dong et al., 2016; Kalofolias, 2016; Hu et al., 2013) optimize Dirichlet energy while enforcing sparsity or specific structural constraints. Table 8 in Appendix F compares various graph learning methods based on their formulations and time complexities. More recently, SUBLIME (Liu et al., 2022) learns graph structure in an unsupervised manner by leveraging self-supervised contrastive learning to align a learnable graph with a dynamically refined anchor graph derived from the data itself.

Remark 1 *Graph Structure Learning (GSL) differs significantly from Continual Learning (CL) (Van de Ven & Tolias, 2019; Zhang et al., 2022; Parisi et al., 2019) and Dynamic Graph Learning (DGL) (Kim et al., 2022; Wu et al., 2023; You et al., 2022), as discussed in Appendix E.*

3 GRAPHFLEX

Here, we present GraphFLEx which has three main modules: *1) Graph Clustering.* It identifies communities and extracts higher-order structural information, *2) Graph Coarsening.* It is used to coarsen down the desired community, if the community itself is large, and *3) Graph Learning.* Learns the graph's structure using a limited subset of nodes from the clustering and coarsening modules, *enabling scalability*. For pseudocode, see Algorithm 1 in Appendix B.

3.1 INCREMENTAL GRAPH LEARNING FOR LARGE DATASETS

Real-world graph data is continuously expanding. For instance, e-commerce networks accumulate new clicks and purchases daily (Xiang et al., 2010), while academic networks grow with new researchers and publications (Wang et al., 2020). To manage such growth, we propose incrementally learning the graph structure over smaller segments. Given a large dataset $\mathcal{L}(V_{\mathcal{L}}, X_{\mathcal{L}})$, where $V_{\mathcal{L}}$ is the node set and $X_{\mathcal{L}}$ represents node features, we define an *expanding dataset* setting $\mathcal{L}_{\mathcal{E}} = \{\mathcal{E}_{\tau=0}^{T}\}$. Initially, $\mathcal{L}$ is split into: (i) a *static dataset* $\mathcal{E}_0(V_0, X_0)$ and (ii) an *expanding dataset* $\mathcal{E} = \{\mathcal{E}_\tau(V_\tau, X_\tau)\}_{\tau=1}^{T}$. Both *Goal 1* (large datasets with missing graph structure) and *Goal 2* (partially available graphs with incremental updates), discussed in Section 2, share the common objective of incrementally learning and updating the graph structure as new data arrives. GraphFLEx handles these by decomposing the problem into two key components:

- *Initial Graph $\mathcal{G}_0(V_0, A_0, X_0)$:* For *Goal 1*, where the graph structure is entirely missing, $\mathcal{E}_0(V_0, X_0)$ is used to construct $\mathcal{G}_0$ from scratch using structure learning methods (Sec.2.2). For *Goal 2*, the initial graph $\mathcal{G}_0(V_0, A_0, X_0)$ is already available and serves as the starting point for incremental updates.
- *Expanding Dataset $\mathcal{E} = \{\mathcal{E}_\tau(V_\tau, X_\tau)\}_{\tau=1}^{T}$:* In both cases, $\mathcal{E}$ consists of incoming nodes and features arriving over T timestamps. These nodes are progressively integrated into the existing graph, enabling continuous adaptation and growth.

The partition is controlled by a parameter r, which determines the proportion of static nodes: $r = \frac{\|V_0\|}{\|V_{\mathcal{L}}\|}$. For example, $r = 0.2$ implies that 20% of $V_{\mathcal{L}}$ is treated as static, while the remaining 80% arrives incrementally over T timestamps. In our experiments, we set $r = 0.5$ and $T = 25$.

Remark 2 *We can learn $\mathcal{G}_\tau(V_\tau, A_\tau, X_\tau)$ by aggregating $\mathcal{E}_\tau$ nodes in $\mathcal{G}_{\tau-1}$ graph. We aim to learn $\mathcal{G}_T(V_T, A_T, X_T)$ after T^{th}-timestamp.*

3.2 DETECTING COMMUNITIES

From the static graph $\mathcal{G}_0$, our goal is to learn higher-order structural information, identifying potential communities to which incoming nodes ($V \in V\tau$) may belong. We train the community detection/clustering model $\mathcal{M}_{\text{clust}}$ once using $\mathcal{G}_0$, allowing subsequent inference of clusters for all incoming nodes. While our framework supports spectral and k-means clustering, our primary focus has been on Graph Neural Network (GNN)-based clustering methods. Specifically, we use DMoN (Tsitsulin et al., 2023; Bianchi, 2022), which maximizes spectral modularity. Modularity (Newman, 2006) measures the divergence between intra-cluster edges and the expected number. These methods use a GNN layer to compute the partition matrix $C = \text{softmax}(\text{MLP}(\widetilde{X}, \theta_{\text{MLP}})) \in \mathbb{R}^{N \times K}$, where K is the number of clusters and $\widetilde{X}$ is the updated feature embedding generated by one or more message-passing layers. To optimize the C matrix, we minimize the loss function $\Delta(C; A) = -\frac{1}{2m}\text{Tr}(C^T BC) + \frac{\sqrt{k}}{n}|\Sigma_i C_i^T|_F - 1$, which combines spectral modularity maximization with regularization to prevent trivial solutions, where B is the modularity matrix (Tsitsulin et al., 2023). Our static graph $\mathcal{G}_0$ and incoming nodes $\mathcal{E}$ follow Assumption 1.

Assumption 1 *Based on the well-established homophily principle, which forms the basis of most graph coarsening and learning methods. We assume that the generated graphs adhere to the Degree-Corrected Stochastic Block Model (DC-SBM) (Zhao et al., 2012), where intra-class (or intra-community) links are more likely than inter-class links. DC-SBM, an extension of SBM that accounts for degree heterogeneity, making it a more flexible and realistic choice for real-world networks (Details on DC-SBM in Appendix C).*

Lemma 1 ***$\mathcal{M}_{clust}$ Consistency.*** *We adopt the theoretical framework of (Zhao et al., 2012) for a DC-SBM with N nodes and k classes. The edge probability matrix is parameterized as $P_N = \rho_N P$, where $P \in \mathbb{R}^{k \times k}$ is a symmetric matrix containing the between/within community edge probabilities and it is independent of N, $\rho_N = \lambda_N / N$, and λ_N is the average degree of the network. Let*

$\hat{y}_N = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N]$ denote the predicted class labels, and let $\hat{C}_N$ be the corresponding $N \times k$ one-hot matrix. Let the true class label matrix is C_N, and μ is any $k \times k$ permutation matrix. Under the adjacency matrix $A^{(N)}$, the global maximum of the objective $\Delta(\cdot; A^{(N)})$ is denoted as $\hat{C}^*_N$. The consistency of class predictions is defined as:

1) Strong Consistency: $P_N\left[\min_\mu \|\hat{C}^*_N \mu - C_N\|_F^2 = 0\right] \to 1$ *as* $N \to \infty$.

2) Weak Consistency: $\forall \varepsilon > 0,\ P_N\left[\min_\mu \frac{1}{N} \|\hat{C}^*_N \mu - C_N\|_F^2 < \varepsilon\right] \to 1$ *as* $N \to \infty$.

where $\|\cdot\|_F$ is the Frobenius norm. Under theorem 3.1 from (Zhao et al., 2012), the $\mathcal{M}_{clust}$ objective is strongly consistent if $\lambda_N / \log(N) \to \infty$, and it is weakly consistent when $\lambda_N \to \infty$.

GraphFLEx focus on learning the structure within each community rather than the structure of the entire dataset at once. Strong consistency ensures perfect community recovery, meaning no inter-community edges exist representing the ideal case. Weak consistency, however, allows for a small fraction (ϵ) of inter-community edges, where ϵ is controlled by ρ_n in $P_n = \rho_n P$, influencing graph sparsity. By Lemma 1 and Assumption 1, stronger consistency leads to more precise structure learning, whereas weaker consistency permits a limited number of inter-community edges.

3.3 Learning Graph Structure on a Coarse Graph

After training $\mathcal{M}_{\text{clust}}$, we identify communities for incoming nodes, starting with $\tau = 1$. Once assigned, we determine significant communities, i.e., communities with at least one incoming node. We then learn the connections of incoming nodes only to their respective community subgraphs. For large datasets, substantial community sizes may again introduce scalability issues. To mitigate this, we first coarsen the large community graph into a smaller graph and use it to identify potential connections for incoming nodes. This process constitutes the second module of GraphFLEx, denoted as $\mathcal{M}_{\text{coar}}$, which employs LSH-based hashing for graph coarsening. The supernode index for i^{th} node is given as:

$$\mathcal{H}_i = maxOccurance\left\{\left\lfloor \frac{1}{r} \cdot (\mathcal{W} \cdot X_i + b) \right\rfloor\right\} \tag{1}$$

where r (bin width) controls the coarsened graph size, $\mathcal{W}$ represents random projection matrix, X is the feature matrix, and b is the bias term. For further details, refer to UGC (Kataria et al., 2024). After coarsening the ith community (C_i), $\mathcal{M}_{\text{coar}}(C_i) = \{\mathcal{P}_i, S_i\}$ yields a partition matrix $\mathcal{P}_i \in \mathbb{R}^{\|S_i\| \times \|C_i\|}$ and a set of coarsened supernodes (S_i), as discussed in Section 2. To identify the potential set of nodes for incoming nodes, we define their ideal connections/neighborhood as follows:

Definition 1 *The neighborhood of a set of nodes $\mathcal{E}_i$ is defined as the union of the top k most similar nodes in C_i for each node $v \in \mathcal{E}_i$, where similarity is measured by the distance function $d(v, u)$. A node $u \in C_i$ is considered part of the neighborhood if its distance $d(v, u)$ is among the k smallest distances for all $u' \in C_i$.*

$$\mathcal{N}_k(\mathcal{E}_i) = \bigcup_{v \in \mathcal{E}_i} \{u \in C_i \mid d(v, u) \leq \textit{top-}k[d(v, u') : u' \in C_i]\}$$

The neighborhood of incoming nodes $\mathcal{N}_k(\mathcal{E}_i)$ represents the ideal set of nodes where the incoming nodes $\mathcal{E}_i$ are likely to establish connections when the entire community is provided to a structure learning framework. A robust coarsening framework must reduce the number of nodes within each community C_i while ensuring that the neighborhood of the incoming nodes is preserved.

3.4 Graph Learning only with Potential Nodes

As we now have a coarse representation of the community (Sec. 3.3), we can employ any graph learning algorithms discussed in Section 2.2 to learn a graph between coarsened supernodes S_i and incoming nodes ($V^i_\tau \in V_\tau$). This is the third module of GraphFLEx, i.e., graph learning; we denote it as $\mathcal{M}_{gl}$. The number of supernodes in S_i is much smaller compared to the original size of the community, i.e., $\|S_i\| \ll \|C_i\|$; scalability is not an issue now. We learn a small graph first using $\mathcal{M}_{gl}(S_i, X^i_\tau) = \widetilde{\mathcal{G}}^i_\tau(V^c_\tau, A^c_\tau)$ where X^i_τ represents features of new incoming nodes belonging to i^{th} community at time τ, $\widetilde{\mathcal{G}}^i_\tau(V^c_\tau, A^c_\tau)$ representing the graph between supernodes and incoming nodes. Utilizing the partition matrix $\mathcal{P}_i$ obtained from $\mathcal{M}_{\text{coar}}$, we can precisely determine the set of nodes associated with each supernode. For every new node $V \in V^i_\tau$, we identify the connected supernodes and subsequently select nodes within those supernodes. This subset of nodes is denoted by $\omega_{V^i_\tau}$, the sub-graph associated with $\omega_{V^i_\tau}$ represented by $\mathcal{G}^i_{\tau-1}(\omega_{V^i_\tau})$ then undergoes an additional round of

Table 1: Time complexity analysis of GraphFLEx. Here, N is the number of nodes in the graph, k is the number of nodes in the static subgraph used for clustering ($k \ll N$), and c represents the number of detected communities. k_τ denotes the number of nodes at timestamp τ. Finally, $\alpha = \|S_\tau^i\| + \|\mathcal{E}_\tau^i\|$ is the sum of coarsened and incoming nodes in the relevant community at τ timestamp.

	$\mathcal{M}_{clust}$	$\mathcal{M}_{coar}$	$\mathcal{M}_{gl}$	GraphFLEx
Best (kNN-UGC-ANN)	$\mathcal{O}(k^2)$	$\mathcal{O}\left(\frac{k_\tau}{c}\right)$	$\mathcal{O}(\alpha \log \alpha)$	$\mathcal{O}(k^2 + \frac{k_\tau}{c} + \alpha \log \alpha)$
Worst (SC-FGC-GLasso)	$\mathcal{O}(k^3)$	$\mathcal{O}\left(\left(\frac{k_\tau}{c}\right)^2 \|S_\tau^i\|\right)$	$\mathcal{O}(\alpha^3)$	$\mathcal{O}(k^3 + \left(\frac{k_\tau}{c}\right)^2 \|S_\tau^i\| + \alpha^3)$

graph learning $\mathcal{M}_{gl}(\mathcal{G}_{\tau-1}^i(\omega_{V_\tau^i}), X_\tau^i)$, ultimately providing a clear and accurate connection of new nodes V_τ^i with nodes of $\mathcal{G}_{\tau-1}$, ultimately updating it to $\mathcal{G}_\tau$. This multi-step approach, characterized by coarsening, learning on coarsened graphs, and translation to the original graph, ensures scalability.

Theorem 3.1 ***Neighborhood Preservation.*** *Let $\mathcal{N}_k(\mathcal{E}_i)$ denote the neighborhood of incoming nodes $\mathcal{E}_i$ for the ith community. With partition matrix $\mathcal{P}_i$ and $\mathcal{M}_{gl}(S_i, X_\tau^i) = \mathcal{G}_\tau^c(V_\tau^c, A_\tau^c)$ we identify the supernodes connected to incoming nodes $\mathcal{E}_i$ and subsequently select nodes within those supernodes; this subset of nodes is denoted by $\omega_{V_\tau^i}$. Formally,*

$$\omega_{V_\tau^i} = \bigcup_{v \in \mathcal{E}_i} \left\{ \bigcup_{s \in S_i} \{\pi^{-1}(s) | A_\tau^c(v, s) \neq 0\} \right\}$$

Then, with probability $\Pi_{\{c \in \phi\}} p(c)$, it holds that $\mathcal{N}_k(\mathcal{E}_i) \subseteq \omega_{V_\tau^i}$ where

$$p(c) \leq 1 - \frac{2}{\sqrt{2\pi}} \frac{c}{r} \left[1 - e^{-r^2/(2c^2)}\right],$$

and ϕ is a set containing all pairwise distance values ($c = \|v - u\|$) between every node $v \in \mathcal{E}_i$ and the nodes $u \in \omega_{V_\tau^i}$. Here, $\pi^{-1}(s)$ denotes the set of nodes mapped to supernode s, r is the bin-width hyperparameter of $\mathcal{M}_{coar}$.

Proof 3.1 *The proof is deferred in Appendix D.*

Theorem 3.1 establishes that, with a constant probability of success, the neighborhood of incoming nodes $\mathcal{N}_k(\mathcal{E}_i)$ can be effectively recovered using the GraphFLEx multistep approach, which involves coarsening and learning on the coarsened graph, i.e., $\mathcal{N}_k(\mathcal{E}_i) \subseteq \omega_{V_\tau^i}$. The set $\omega_{V_\tau^i}$, estimated by GraphFLEx, identifies potential candidates where incoming nodes are likely to connect. The probability of failure can be reduced by regulating the average degree of connectivity in $\mathcal{M}_{gl}(S_i, X_\tau^i) = \mathcal{G}_\tau^c(V_\tau^c, A_\tau^c)$. While a fully connected network $\mathcal{G}_\tau^c$ ensures all nodes in the community are candidates, it significantly increases computational costs for large communities.

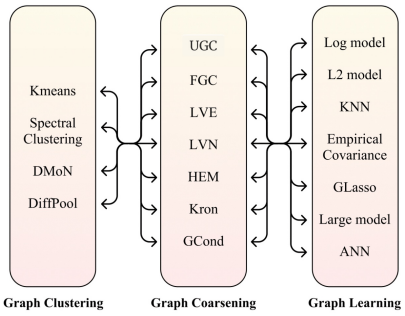


Figure 3: The versatility of GraphFlex in supporting multiple GSL methods.

3.5 GraphFLEx: Multiple GSL Frameworks

Each of the three modules of GraphFLEx, as also shown in Figure 3 controls a distinct aspect of the GSL process: clustering influences community detection, coarsening reduces graph complexity via supernodes, and the learning module governs structural inference. Altering any of these modules results in a new GSL method. Currently, we support 48 different graph learning configurations, and this number scales aggressively with the addition of new methods to any module. The number of possible frameworks is given by $\alpha \times \beta \times \gamma$, where α, β, and γ represent the number of clustering, coarsening, and learning methods, respectively.

Module Selection: Since different clustering, coarsening, and learning choices yield distinct GSL variants, we provide module selection guidelines in Appendix L.

3.6 Run Time Analysis

GraphFLEx computational time is always bounded by existing approaches, as it operates on a significantly reduced set of nodes. We evaluate the run-time complexity of GraphFLEx in two scenarios: (a) the worst-case scenario, where computationally intensive clustering and coarsening modules are selected, providing an upper bound on time complexity, and (b) the best-case scenario, where the most efficient modules are chosen. Table 1 presents a summary of this analysis for both cases. Due to space limitations, a more comprehensive analysis is provided in Appendix G.

4 EXPERIMENTS

Tasks and Datasets. To validate GraphFLEx's utility, we evaluate it across four key dimensions: (i) computational efficiency, (ii) scalability to large graphs, (iii) quality of learned structures, and (iv) adaptability to dynamically growing graphs. To validate the characteristics of GraphFLEx, we conduct extensive experiments on 26 different datasets, including (a) datasets that already have a complete graph structure (allowing comparison between the learned and the original structure), (b) datasets with missing graph structures, (c) synthetic datasets, (d) small datasets for visualizing the graph structure, and (e) large datasets, including datasets with even $2.4M$ nodes. More details about datasets and *system specifications* are presented in Table 7 in Appendix A.

4.1 COMPUTATIONAL EFFICIENCY.

Existing methods like k-NN and log-model struggle to learn graph structures even for 20k nodes due to out-of-memory (OOM) or out-of-time (OOT) issues, while $l2$-model and $large$-model struggle beyond 50k nodes. Although A-NN and emp-Covar. are faster, GraphFLEx outperforms them on sufficiently large graphs (Table 2). While traditional methods may be efficient for small graphs, GraphFLEx scales significantly better, excelling on large datasets like *Pubmed* and *Syn 5*, where most methods fail. It accelerates structure learning, making A-NN 3× faster and emp-Covar. 2× faster.

4.2 NODE CLASSIFICATION ACCURACY

Experimental Setup. We now evaluate the prediction performance of GNN models when trained on graph structures learned from three distinct scenarios: *1) Original Structure:* GNN models trained on the original graph structure, which we refer to as the Base Structure, *2) GraphFLEx Structure:* GNN models trained on the graph structure learned from GraphFLEx, and *3)Vanilla Structure:* GNN models trained on the graph structure learned from other existing methods. For each scenario, a unique graph structure is obtained. We trained GNN models on each of these three structure. For more details on GNN model parameters, see Appendix H.

Table 2: Computational time(in seconds) for learning graph structures using GraphFLEx (GFlex) with existing methods (Vanilla referred to as Van.). The experimental setup involves treating 50% of the data as static, while the remaining 50% of nodes are treated as incoming nodes arriving in 25 different timestamps. The best times are highlighted by color Green. OOM and OOT denote out-of-memory and out-of-time, respectively.

Data	ANN		KNN		log-model		l2-model		emp-Covar.		large-model		Sublime	
	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex
Cora	335	100	8.4	36.1	869	81.6	424	55	8.6	30	2115	18.4	7187	493
Citeseer	1535	454	21.9	75	1113	64.5	977	54.0	14.7	59.2	8319	43.9	8750	670
DBLP	2731	988	OOM	270	77000	919	OOT	1470	359	343	OOT	299	OOM	831
CS	22000	12000	OOM	789	OOT	838	32000	809	813	718	OOT	1469	OOM	1049
PubMed	770	227	OOM	164	OOT	176	OOT	165	488	299	OOT	262	OOM	914
Phy.	61000	21000	OOM	903	OOT	959	OOT	908	2152	1182	OOT	2414	OOM	2731
Syn 3	95	37	OOM	30	58000	346	859	53	88	59	5416	42	6893	780
Syn 4	482	71	OOM	73	OOT	555	OOT	145	2072	1043	OOT	392	OOM	1896

Table 3: Node classification accuracies on different GNN models using GraphFLEx (GFlex) with existing Vanilla (Van.) methods. The experimental setup involves treating 70% of the data as static, while the remaining 30% of nodes are treated as new nodes coming in 25 different timestamps. The best and the second-best accuracies in each row are highlighted by dark and lighter shades of Green, respectively. GraphFLEx's structure beats all of the vanilla structures for every dataset. OOM and OOT denotes out-of-memory and out-of-time respectively.

Data	Model	ANN		KNN		log-model		l2-model		COVAR		large-model		Sublime		Base Struct.
		Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	
DBLP	GAT	34.23	67.37	OOM	69.83	OOT	69.83	OOT	68.98	50.48	68.56	OOT	66.38	OOM	68.32	70.84
	SAGE	34.23	69.58	OOM	70.28	OOT	70.28	OOT	70.68	51.47	70.51	OOT	69.32	OOM	70.28	72.57
	GCN	34.12	69.41	OOM	73.39	OOT	73.39	OOT	73.05	51.50	71.75	OOT	68.55	OOM	69.06	74.43
	GIN	34.01	69.69	OOM	68.19	OOT	68.19	OOT	73.08	52.77	72.03	OOT	71.18	OOM	71.87	73.92
CS	GAT	12.47	60.89	OOM	61.09	OOT	60.95	18.64	61.06	58.96	88.06	OOT	86.22	OOM	64.21	60.75
	SAGE	12.70	78.81	OOM	79.43	OOT	79.06	19.24	78.94	56.97	93.30	OOT	92.79	OOM	78.94	80.33
	GCN	12.59	63.81	OOM	67.94	OOT	69.33	19.21	66.01	58.35	91.07	OOT	84.85	OOM	68.92	67.43
	GIN	13.07	77.62	OOM	78.41	OOT	78.55	19.24	77.61	58.26	92.07	OOT	86.03	OOM	77.61	55.65
Pub.	GAT	49.49	83.71	OOM	84.60	OOT	84.60	OOT	84.04	72.63	83.97	OOT	81.15	OOM	82.15	84.04
	SAGE	50.43	87.27	OOM	87.34	OOT	87.34	OOT	87.42	73.57	86.68	OOT	87.34	OOM	83.45	88.88
	GCN	50.45	82.06	OOM	83.56	OOT	83.56	OOT	83.74	73.14	82.39	OOT	78.03	OOM	70.94	85.54
	GIN	51.82	83.13	OOM	84.31	OOT	84.07	OOT	82.93	73.15	83.51	OOT	82.85	OOM	80.72	86.50
Phy.	GAT	29.18	88.06	OOM	88.47	OOT	88.47	OOT	88.68	58.96	88.06	OOT	86.22	OOM	86.12	88.58
	SAGE	29.57	93.47	OOM	93.47	OOT	93.47	OOT	93.78	56.97	93.60	OOT	92.79	OOM	89.58	94.19
	GCN	27.84	91.27	OOM	91.08	OOT	91.08	OOT	91.78	58.35	91.07	OOT	84.85	OOM	88.46	91.48
	GIN	28.38	92.69	OOM	92.04	OOT	92.04	OOT	92.27	58.26	92.07	OOT	86.03	OOM	87.20	88.89

GNN Models. Graph neural networks (GNNs) such as GCN (Kipf & Welling, 2016), $GraphSage$ (Hamilton et al., 2017), GIN (Xu et al., 2018), and GAT (Velickovic et al., 2017) rely on accurate message passing, dictated by the graph structure, for effective embedding. We use these models to evaluate the above-mentioned learned structures. Table 3 reports node classification performance across all methods. Notably, GraphFLEx outperforms vanilla structures by a significant margin across all datasets, achieving accuracies close to those obtained with the original structure. Figure 8 in Appendix H illustrates $GraphSage$ classification results, highlighting GraphFLEx's superior performance. For the CS dataset, GraphFLEx ($large$-model) and GraphFLEx ($empCovar$.-model) even surpass the original structure, demonstrating its ability to preserve key structural properties while denoising edges, leading to improved accuracy.

Graph Transformers. To address limitations of message-passing GNNs (e.g., limited receptive fields and over-smoothing), recent Graph Transformers couple self-attention with structural priors to capture long-range dependencies. We also include node-classification accuracies of Graph Transformer models like GOAT (Kong et al., 2023), Nodeformer (Wu et al., 2022a), and Exoformer (Shirzad et al., 2023) when trained on graph structures learned by GraphFLEx (Table 10 in App. H).

Table 4: Runtime (sec) and Node Classification Accuracy (%) across large datasets. Each cell shows: **Time / Accuracy**. Van = Vanilla, GFlex = GraphFLEx. OOM = Out of Memory, OOT = Out of Time.

Method	ogbn-arxiv (60.13)		ogbn-products (73.72)		Flickr (44.92)		Reddit (94.15)	
	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex
Covar	OOM — –	3.7k — 60.26	OOM — –	83.1k — 68.23	2.3k — 44.65	682 — 44.34	OOM — –	6.6k — 94.13
ANN	7.8k — 60.14	4.8k — 60.22	OOM — –	89.3k — 67.91	2.5k — 44.09	705 — 44.92	12.6k — 94.14	6.1k — 94.18
knn	8.3k — 60.09	6.1k — 60.23	OOM — –	91.8k — 68.47	2.7k — 43.95	920 — 44.73	15.6k — 94.14	6.9k — 94.15
l2	OOT — –	9.1k — 58.39	OOT — –	OOT — –	93.3k — 44.90	1.2k — 44.32	OOT — –	5.1 — 93.47
log	OOT — –	45.6k — 58.72	OOT — –	OOT — –	OOT — –	18.7k — 44.59	OOT — –	60.3k — 94.13
large	OOT — –	5.6k — 60.20	OOT — –	OOT — –	OOT — –	2.2k — 44.45	OOT — –	9.3k — 93.71

4.3 Scalability of GraphFLEx on Large-Scale Graphs.

To evaluate GraphFLEx's scalability to large-scale graphs, we consider four datasets with large number of nodes: (a) *Flickr(89k nodes)* (Zeng et al., 2019), (b) *Reddit (233k nodes)* (Zeng et al., 2019), (c) *Ogbn-arxiv (169k nodes)* (Wang et al., 2020), and (d) *Ogbn-products (2.4M nodes)* (Bhatia et al., 2016). As shown in Table 4, GraphFLEx consistently demonstrates superior scalability across all datasets, outperforming all baseline methods in runtime. In particular, methods such as *log-model*, *l2-model*, and *large-model* fail to run even on *Flickr*, while GraphFLEx successfully scales them on *Flickr*, *Ogbn-arxiv*, and *Reddit*, enabling structure learning where others cannot. For the most computationally demanding dataset, *Ogbn-products*, these methods remain prohibitively expensive even for GraphFLEx. Nonetheless, GraphFLEx efficiently supports scalable structure learning on *Ogbn-products* using the *Covar*, *ANN*, and *KNN* modules. Table 4 also reports node classification accuracy, demonstrating that GraphFLEx maintains performance comparable to the original (base) structure across all datasets. These results confirm that GraphFLEx not only scales effectively, but also preserves the quality of learned structures.

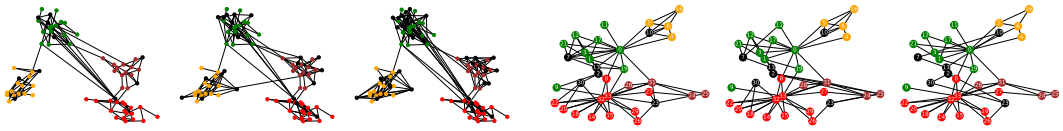
4.4 GraphFLEx for Link Prediction and Graph Classification.

To further validate the generalization, we evaluate GraphFLEx on the link prediction task. Results are in Table 5 which follows the same setting as in Table 3. The structure learned by GraphFLEx demonstrates strong predictive performance, in some cases even outperforming the base structure. This highlights the effectiveness of GraphFLEx in preserving and even enhancing relational information relevant for link prediction. While our primary focus is on structure learning for node-level

Table 5: Link predication accuracy (%) across different datasets. The best and the second-best accuracies in each row are highlighted by dark and lighter shades of Green, respectively.

Data	ANN		KNN		log-model		l2-model		COVAR		large-model		Base Struct.
	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	
DBLP	96.57	96.61	OOM	94.23	OOT	97.59	OOT	97.59	97.22	97.59	OOT	96.24	95.13
Citeseer	80.12	96.32	85.17	96.24	80.48	96.24	80.48	96.48	82.05	96.24	84.50	94.38	90.78
Cora	84.47	95.30	79.23	95.14	90.63	95.45	90.81	95.14	86.05	95.30	90.63	94.67	89.53
Pubmed	94.24	96.91	OOM	97.42	OOT	97.42	OOT	97.37	94.89	94.64	OOT	94.41	94.64
CS	94.21	95.73	OOM	96.02	OOT	93.17	OOT	93.17	93.52	92.31	OOT	95.73	95.00
Physics	95.77	91.34	OOM	94.63	OOT	90.79	OOT	94.63	92.03	90.79	OOT	92.97	93.96

tasks, we briefly discuss the applicability of GraphFLEx to graph classification. In such tasks, especially in domains like molecule or drug discovery, each data point often corresponds to a small individual subgraph. For these cases, applying clustering and coarsening is typically redundant and may introduce unnecessary computational overhead. Nevertheless, GraphFLEx remains flexible its



(a) 10 incoming nodes (b) 20 incoming nodes (c) 30 incoming nodes (d) ANN as $\mathcal{M}_{gl}$ (e) Emp. Covr. as $\mathcal{M}_{gl}$ (f) kNN as $\mathcal{M}_{gl}$

Figure 4: Figures (a), (b), and (c) illustrate the growing structure learned using GraphFLEx for *HE* synthetic dataset. Figures (d), (e), and (f) illustrate the learned structure on Zachary's karate dataset when existing methods are employed with GraphFLEx. New nodes are denoted using black color.

learning module can be directly used without the clustering or coarsening steps, making it suitable for graph classification as well. This adaptability reinforces GraphFLEx's utility across a broad range of graph learning tasks.

4.5 Clustering Quality

We measure three metrics to evaluate the resulting clusters or community assignments: a) Normalized Mutual Information (NMI) (Tsitsulin et al., 2023) between the cluster assignments and original labels; b) Conductance ($\mathcal{C}$) (Jerrum & Sinclair, 1988) which measures the fraction of total edge volume that points outside the cluster; and c) Modularity ($\mathcal{Q}$) (Newman, 2006) which measures the divergence between the intra-community edges and the expected one. Table 6 illustrates these metrics for single-cell RNA and the MNIST dataset (where the whole structure is missing), and Figure 12 in Appendix K.1 shows the PHATE (Moon et al., 2019) visualization of clusters learned using GraphFLEx's clustering module $\mathcal{M}_{clust}$. We also train the aforementioned GNN models for the node classification task in order to illustrate the efficacy of the learned structures; the accuracy values presented in Table 6, clearly highlight the significance of the learned structures, as reflected by the high accuracy values. For a comprehensive ablation study, refer to Appendix K.

Table 6: Clustering (NMI, $\mathcal{C}$, $\mathcal{Q}$) and node classification accuracy using GCN, GraphSAGE, GIN, and GAT.

Data	NMI	$\mathcal{C}$	$\mathcal{Q}$	GCN	SAGE	GIN	GAT
Bar. M.	0.716	0.057	0.741	91.2	96.2	95.1	94.9
Seger.	0.678	0.102	0.694	91.0	93.9	94.2	92.3
Mura.	0.843	0.046	0.706	96.9	97.4	97.5	96.4
Bar. H.	0.674	0.078	0.749	95.3	96.4	97.2	95.8
Xin	0.741	0.045	0.544	98.6	99.3	98.9	99.8
MNIST	0.677	0.082	0.712	92.9	94.5	94.9	82.6

4.6 Structure Visualization

We evaluate the structures generated by GraphFLEx through visualizations on four small datasets: (i) MNIST (LeCun et al., 2010), consisting of handwritten digit images, where Figure 5(a) shows that images of the same digit are mostly connected; (ii) Pre-trained GloVe embeddings (Pennington et al., 2014) of English words, with Figure 5(b) revealing that frequently used words are closely connected; (iii) A synthetic $H.E$ dataset (see Appendix A), demonstrating GraphFLEx's ability to handle expanding networks without requiring full relearning. Figure 4(a-c) shows the graph structure evolving as 30 new nodes are added over three timestamps; and (iv) Zachary's karate club network (Zachary, 1977), which highlights GraphFLEx's multi-framework capability. Figure 4(d-f) shows three distinct graph structures after altering the learning module.

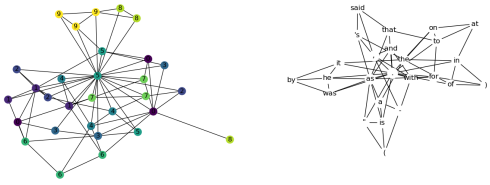


Figure 5: Effectiveness of GraphFlex in learning structure between similar MNIST digits and GloVe embeddings.

5 Conclusion

Large or expanding graphs challenge the best of graph learning approaches. GraphFLEx, introduced in this paper, seamlessly adds new nodes into an existing graph structure. It offers diverse methods for acquiring the graph's structure. GraphFLEx consists of three key modules: Clustering, Coarsening, and Learning which empowers GraphFLEx to serves as a comprehensive framework applicable individually for clustering, coarsening, and learning tasks. Empirically, GraphFLEx outperforms state-of-the-art baselines, achieving up to 3× speedup while preserving structural quality. The performance across multiple real and synthetic datasets affirms the utility and efficacy of GraphFLEx for graph structure learning.

Limitations and Future Work. GraphFLEx is designed assuming minimal inter-community connectivity, which aligns well with many real-world scenarios. However, its applicability to heterophilic graphs may require further adaptation. Future work will focus on extending the framework to supervised GSL methods and heterophilic graphs, broadening its scalability and versatility.

REFERENCES

Jeffrey D Allen, Yang Xie, Min Chen, Luc Girard, and Guanghua Xiao. Comparing statistical methods for constructing large scale gene networks. *PloS one*, 7(1):e29348, 2012. **(Cited at p. 1.)**

Onureena Banerjee, Laurent El Ghaoui, and Alexandre d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *The Journal of Machine Learning Research*, 9:485–516, 2008. **(Cited at p. 3.)**

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7 (11), 2006. **(Cited at p. 3.)**

K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html. **(Cited at pp. 8 and 16.)**

Filippo Maria Bianchi. Simplifying clustering with graph neural networks. *arXiv preprint arXiv:2207.08779*, 2022. **(Cited at p. 4.)**

J Bruna, W Zaremba, A Szlam, and Y LeCun. Spectral networks and deep locally connected networks on graphs. arxiv. *arXiv preprint arXiv:1312.6203*, 2014. **(Cited at p. 3.)**

Yu Chen and Lingfei Wu. Graph neural networks: Graph structure learning. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 297–321, 2022. **(Cited at p. 1.)**

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004. **(Cited at p. 17.)**

Michaël Defferrard, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. Pygsp: Graph signal processing in python. URL https://github.com/epfl-lts2/pygsp/. **(Cited at p. 15.)**

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. **(Cited at p. 3.)**

Arthur P Dempster. Covariance selection. *Biometrics*, pp. 157–175, 1972. **(Cited at p. 3.)**

Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pp. 577–586, 2011. **(Cited at p. 1.)**

Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23): 6160–6173, 2016. **(Cited at pp. 1 and 3.)**

Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34: 22667–22681, 2021. **(Cited at p. 1.)**

Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010. **(Cited at p. 3.)**

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems*, 30, 2017. **(Cited at p. 1.)**

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. **(Cited at pp. 1 and 3.)**

Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pp. 2331–2341, 2020. **(Cited at p. 15.)**

Jiuxiang Gu, Handong Zhao, Zhe Lin, Sheng Li, Jianfei Cai, and Mingyang Ling. Scene graph generation with external knowledge and image reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1969–1978, 2019. **(Cited at p. 1.)**

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. **(Cited at p. 8.)**

Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B Aditya Prakash, and Wei Jin. A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation. *arXiv preprint arXiv:2402.03358*, 2024. **(Cited at p. 3.)**

Cho-Jui Hsieh, Inderjit Dhillon, Pradeep Ravikumar, and Mátyás Sustik. Sparse inverse covariance matrix estimation using quadratic approximation. *Advances in neural information processing systems*, 24, 2011. **(Cited at p. 1.)**

Chenhui Hu, Lin Cheng, Jorge Sepulcre, Georges El Fakhri, Yue M Lu, and Quanzheng Li. A graph theoretical regression model for brain connectivity learning of alzheimer's disease. In *2013 IEEE 10th International Symposium on Biomedical Imaging*, pp. 616–619. IEEE, 2013. **(Cited at p. 3.)**

Mark Jerrum and Alistair Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of permanent resolved. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 235–244, 1988. **(Cited at p. 9.)**

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 66–74, 2020. **(Cited at p. 1.)**

Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*, 2021. **(Cited at p. 19.)**

Vassilis Kalofolias. How to learn a graph from smooth signals. In *Artificial intelligence and statistics*, pp. 920–929. PMLR, 2016. **(Cited at pp. 1 and 3.)**

Sepandar D Kamvar, Dan Klein, and Christopher D Manning. Spectral learning. In *IJCAI*, volume 3, pp. 561–566, 2003. **(Cited at p. 3.)**

Mohit Kataria, Aditi Khandelwal, Rocktim Das, Sandeep Kumar, and Jayadeva Jayadeva. Linear complexity framework for feature-aware graph coarsening via hashing. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023. URL https://openreview.net/forum?id=HKdsrm5nCW. **(Cited at pp. 3, 18, and 19.)**

Mohit Kataria, Sandeep Kumar, and Jayadeva Jayadeva. UGC: Universal graph coarsening. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=nN6NSd1Qds. **(Cited at pp. 3, 5, and 17.)**

Seoyoon Kim, Seongjun Yun, and Jaewoo Kang. Dygrain: An incremental learning framework for dynamic graphs. In *IJCAI*, pp. 3157–3163, 2022. **(Cited at pp. 3 and 18.)**

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. **(Cited at pp. 8 and 18.)**

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. **(Cited at p. 3.)**

Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C Bayan Bruss, and Tom Goldstein. Goat: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, pp. 17375–17390. PMLR, 2023. **(Cited at pp. 8 and 19.)**

Manoj Kumar, Anurag Sharma, and Sandeep Kumar. A unified framework for optimization-based graph coarsening. *Journal of Machine Learning Research*, 24(118):1–50, 2023. URL http://jmlr.org/papers/v24/22-1085.html. **(Cited at p. 3.)**

Danning Lao, Xinyu Yang, Qitian Wu, and Junchi Yan. Variational inference for training graph neural networks in low-data regime through joint structure-label estimation. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 824–834, 2022. **(Cited at p. 1.)**

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. **(Cited at pp. 9 and 16.)**

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference 2022*, pp. 1392–1403, 2022. **(Cited at pp. 1 and 3.)**

Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, 20(116): 1–42, 2019. **(Cited at p. 3.)**

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011. **(Cited at p. 18.)**

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pp. 281–297. Oakland, CA, USA, 1967. **(Cited at pp. 1 and 3.)**

Nikita Malik, Rahul Gupta, and Sandeep Kumar. Hyperdefender: A robust framework for hyperbolic gnns. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(18):19396–19404, Apr. 2025. doi: 10.1609/aaai.v39i18.34135. URL https://ojs.aaai.org/index.php/AAAI/article/view/34135. **(Cited at p. 1.)**

Kevin R Moon, David Van Dijk, Zheng Wang, Scott Gigante, Daniel B Burkhardt, William S Chen, Kristina Yim, Antonia van den Elzen, Matthew J Hirn, Ronald R Coifman, et al. Visualizing structure and transitions in high-dimensional biological data. *Nature biotechnology*, 37(12): 1482–1492, 2019. **(Cited at pp. 9 and 22.)**

Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014. **(Cited at p. 1.)**

Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006. **(Cited at pp. 4 and 9.)**

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019. **(Cited at pp. 3 and 18.)**

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014. **(Cited at pp. 9 and 16.)**

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. **(Cited at p. 15.)**

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pp. 31613–31632. PMLR, 2023. **(Cited at pp. 8 and 19.)**

Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023. **(Cited at pp. 3, 4, and 9.)**

Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019. **(Cited at pp. 3 and 18.)**

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017. **(Cited at p. 8.)**

Joshua T Vogelstein, William Gray Roncal, R Jacob Vogelstein, and Carey E Priebe. Graph classification using signal-subgraphs: Applications in statistical connectomics. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1539–1551, 2012. **(Cited at p. 18.)**

Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pp. 577–584, 2001. **(Cited at p. 3.)**

Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. In *Proceedings of the 23rd international conference on Machine learning*, pp. 985–992, 2006. **(Cited at pp. 1 and 3.)**

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1): 396–413, 2020. **(Cited at pp. 4, 8, and 16.)**

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440–442, 1998. **(Cited at p. 15.)**

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022a. **(Cited at pp. 8 and 19.)**

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022b. **(Cited at p. 1.)**

Tiandeng Wu, Qijiong Liu, Yi Cao, Yao Huang, Xiao-Ming Wu, and Jiandong Ding. Continual graph convolutional network for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 13754–13762, 2023. **(Cited at pp. 3 and 18.)**

Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 1054–1061, 2020. **(Cited at p. 1.)**

Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long-and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 723–732, 2010. **(Cited at p. 4.)**

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. **(Cited at p. 8.)**

Fan Yang, Wenchuan Wang, Fang Wang, Yuan Fang, Duyu Tang, Junzhou Huang, Hui Lu, and Jianhua Yao. scbert as a large-scale pretrained deep language model for cell type annotation of single-cell rna-seq data. *Nature Machine Intelligence*, 4(10):852–866, 2022. **(Cited at p. 15.)**

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016. **(Cited at p. 15.)**

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 2358–2366, 2022. **(Cited at pp. 3 and 18.)**

Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977. **(Cited at pp. 9, 16, and 23.)**

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019. **(Cited at pp. 8 and 16.)**

Xikun Zhang, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems*, 35:13006–13021, 2022. **(Cited at pp. 3 and 18.)**

Yunpeng Zhao, Elizaveta Levina, and Ji Zhu. Consistency of community detection in networks under degree-corrected stochastic block models. 2012. **(Cited at pp. 4, 5, and 16.)**

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020. **(Cited at p. 1.)**

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003. **(Cited at p. 3.)**

Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A survey on graph structure learning: Progress and opportunities. *arXiv preprint arXiv:2103.03036*, 2021. **(Cited at p. 1.)**

Appendix

A DATASETS

Datasets used in our experiments vary in size, with nodes ranging from 1k to 60k. Table 7 lists all the datasets we used in our work. We evaluate our proposed framework $GraphFlex$ on real-world datasets *Cora ,Citeseer, Pubmed* (Yang et al., 2016), *CS, Physics* (Shchur et al., 2018), *DBLP* (Fu et al., 2020), all of which include graph structures. These datasets allow us to compare the learned structures with the originals. Additionally, we utilize single-cell RNA pancreas datasets (Yang et al., 2022), including Baron, Muraro, Segerstolpe, and Xin, where the graph structure is missing. The Baron dataset was downloaded from the Gene Expression Omnibus (GEO) (accession no. GSE84133). The Muraro dataset was downloaded from GEO (accession no. GSE85241). The Segerstolpe dataset was accessed from ArrayExpress (accession no. E-MTAB-5061). The Xin dataset was downloaded from GEO (accession no. GSE81608). We simulate the expanding graph scenario by splitting the original dataset across different $\mathcal{T}$ timestamps. We assumed 50% of the nodes were static, with the remaining nodes arriving as incoming nodes at different timestamps.

Synthetic datasets: Different data generation techniques validate that our results are generalized to

Category	Data	Nodes	Edges	Feat.	Class	Type
Original Structure Known	Cora	2,708	5,429	1,433	7	Citation network
	Citeseer	3,327	9,104	3,703	6	Citation network
	DBLP	17,716	52.8k	1,639	4	Research paper
	CS	18,333	163.7k	6,805	15	Co-authorship network
	PubMed	19,717	44.3k	500	3	Citation network
	Physics	34,493	247.9k	8,415	5	Co-authorship network
Original Structure Not Known	Xin	1,449	NA	33,889	4	Human Pancreas
	Baron Mouse	1,886	NA	14,861	13	Mouse Pancreas
	Muraro	2,122	NA	18,915	9	Human Pancreas
	Segerstolpe	2,133	NA	22,757	13	Human Pancreas
	Baron Human	8,569	NA	17,499	14	Human Pancreas
Synthetic	Syn 1	2,000	8,800	150	4	SW
	Syn 2	5,000	22k	150	4	SW
	Syn 3	10,000	44k	150	7	SW
	Syn 4	50,000	220k	150	7	SW
	Syn 5	400	1,520	100	4	PyG-G
	Syn 6	2,500	9,800	100	4	PyG-S
	Syn 7	1,000	9,990	150	4	HE
	Syn 8	2,000	40k	150	4	HE
Visulization Datasets	MNIST	60,000	NA	784	10	Images
	Zachary's karate	34	156	34	4	Karate club network
	Glove	2,000	NA	50	NA	GloVe embeddings
Large dataset	Flickr	89,250	899,756	500	7	-
	Reddit	232,965	11.60M	602	41	-
	Ogbn-arxiv	169,343	1.16M	128	40	-
	Ogbn-products	2,449,029	61.85M	100	47	-

Table 7: Summary of the datasets.

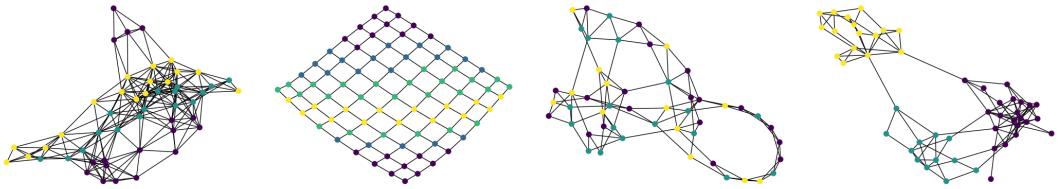
different settings. Please refer to Table 7 for more details about the number of nodes, edges, features, and classes, Syn denotes the type of synthetic datasets. Figure 6 shows graphs generated using different methods. We have employed three different ways to generate synthetic datasets which are mentioned below:

- **PyGSP(PyGsp):** We used synthetic graphs created by PyGSP (Defferrard et al.) library. PyG-G and PyG-S denotes grid and sensor graphs from PyGSP.
- **Watts–Strogatz's small world(SW):** (Watts & Strogatz, 1998) proposed a generation model that produces graphs with small-world properties, including short average path lengths and high clustering.

- **Heterophily(HE):** We propose a method for creating synthetic datasets to explore graph behavior across a heterophily spectrum by manipulating heterophilic factor α, and classes. α is determined by dividing the number of edges connecting nodes from different classes by the total number of edges in the graph.

Visulization Datasets: To evaluate, the learned graph structure, we have also included three datasets: (i) MNIST (LeCun et al., 2010), consisting of handwritten digit images; (ii) Pre-trained GloVe embeddings (Pennington et al., 2014) of English words; and (iii) Zachary's karate club network (Zachary, 1977).
Large Datasets: To comprehensively evaluate GraphFLEx's scalability to large-scale graphs, we consider four datasets with a high number of nodes: (a) *Flickr(89k nodes)* (Zeng et al., 2019), (b) *Reddit (233k nodes)* (Zeng et al., 2019), (c) *Ogbn-arxiv (169k nodes)* (Wang et al., 2020), and (d) *Ogbn-products (2.4M nodes)* (Bhatia et al., 2016).

System Specifications: All the experiments conducted for this work were performed on an Intel Xeon W-295 CPU with 64GB of RAM desktop using the Python environment.



(a) PyGSP-Sensor, N = 50, α=3 (b) PyGSP-Grid, N = 80, α=3 (c) SW, N = 50, α=3 (d) HE, N = 50, α=3

Figure 6: This figure illustrates different types of synthetic graphs generated using i)PyGSP, ii) Watts–Strogatz's small world(SW), and iii) Heterophily(HE). N denotes the number of nodes, while α denotes the number of classes.

B ALGORITHM

In this section we include the pseudocode of GraphFLEx; see Algorithm 1.

C DEGREE-CORRECTED STOCHASTIC BLOCK MODEL(DC-SBM)

The DC-SBM is one of the most commonly used models for networks with communities and postulates that, given node labels $\mathbf{c} = c_1, ...c_n$, the edge variables $A'_{ij}s$ are generated via the formula

$$E[A_{ij}] = \theta_i \theta_j P_{c_i} P_{c_j}$$

, where θ_i is a "degree parameter" associated with node i, reflecting its individual propernsity to form ties, and P is a $K \times K$ symmetric matrix containing the between/withincommunity edge probabilities and $P_{c_i} P_{c_j}$ denotes the edge probabilities between community c_i and c_j.
For DC-SBM model (Zhao et al., 2012) assumed P_n on n nodes with k classes, each node v_i is given a label/degree pair(c_i, θ_i), drawn from a discrete joint distribution $\Pi_{K\times m}$ which is fixed and does not depend on n. This implies that each θ_i is one of a fixed set of values $0 \leq x_1 \leq .... \leq x_m$. To facilitate analysis of asymptotic graph sparsity, we parameterize the edge probability matrix P as $P_n = \rho_n P$ where P is independent of n, and $\rho_n = \lambda_n / n$ where λ_n is the average degree of the network.

D NEIGHBOURHOOD PRESERVATION

Theorem D.1 ***Neighborhood Preservation.*** *Let $\mathcal{N}_k(\mathcal{E}_i)$ denote the neighborhood of incoming nodes $\mathcal{E}_i$ for the ith community. With partition matrix $\mathcal{P}_i$ and $\mathcal{M}_{gl}(S_i, X^i_\tau) = \mathcal{G}^c_\tau(V^c_\tau, A^c_\tau)$ we identify the supernodes connected to incoming nodes $\mathcal{E}_i$ and subsequently select nodes within those supernodes; this subset of nodes is denoted by $\omega_{V^i_\tau}$. Formally,*

$$\omega_{V^i_\tau} = \bigcup_{v \in \mathcal{E}_i} \Big\{ \bigcup_{s \in S_i} \{\pi^{-1}(s) | A^c_\tau(v, s) \neq 0\} \Big\}$$

Algorithm 1 GraphFlex: A Unified Structure Learning framework for expanding and Large Scale Graphs

Input: Graph $G_0(X_0, A_0)$, expanding nodes set $\mathcal{E}_1^T = \{\mathcal{E}_\tau(\mathcal{V}_\tau, \mathcal{X}_\tau)\}_{\tau=1}^T$
Parameter: GClust, GCoar, GL $\leftarrow$ Clustering, Coarsening and Learning Module
Output: Graph $G_T(X_T, A_T)$

Train clustering module *train*($\mathcal{M}_{clust}$, GClust, G_0)
for each $E_t(V_t, X_t)$ in $\mathcal{E}_1^T$ **do**
  $C_t = \textit{infer}(\mathcal{M}_{clust}, X_t)$, $C_t \in \mathbb{R}^{N_t}$ denotes the communities of N_t nodes at time t.
  $I_t = \textit{unique}(C_t)$.
  for each I_t^i in I_t **do**
    G_{t-1}^i = subgraph(G_{t-1}, I_t^i)
    $\{S_{t-1}^i, P_{t-1}^i\} = \mathcal{M}_{coar}(G_{t-1}^i)$, $S_{t-1}^i \in \mathbb{R}^{k\times d}$ are features of k supernodes, $P_{t-1}^i \in \mathbb{R}^{k\times N_t^i}$ is the partition matrix.
    $Gc_{t-1}^i(S_{t-1}^i, A_{t-1}^i) = \mathcal{M}_{gl}(S_{t-1}^i, X_t^i)$, Gc_{t-1}^i is the learned graph on super-nodes S_{t-1}^i and new node X_t^i.
    $\omega_t^i \leftarrow$ [ ]
    for $x \in X_t^i$ **do**
      $\omega_t^i.\textit{append}(x)$
      $n_p = \{n \mid A_{t-1}^i[n] > 0\}$
      $\omega_t^i.\textit{append}(n_p)$
    end for
    $G_{t-1} = \textit{update}(G_{t-1}, \mathcal{M}_{gl}(\omega_t^i))$
  end for
  $G_t = G_{t-1}$
end for
return $G_T(X_T, A_T)$

Then, with probability $\Pi_{\{c\in\phi\}}p(c)$, it holds that $\mathcal{N}_k(\mathcal{E}_i) \subseteq \omega_{V_\tau^i}$ where

$$p(c) \leq 1 - \frac{2}{\sqrt{2\pi}}\frac{c}{r}\left[1 - e^{-r^2/(2c^2)}\right],$$

and ϕ is a set containing all pairwise distance values ($c = \|v - u\|$) between every node $v \in \mathcal{E}_i$ and the nodes $u \in \omega_{V_\tau^i}$. Here, $\pi^{-1}(s)$ denotes the set of nodes mapped to supernode s, r is the bin-width hyperparameter of $\mathcal{M}_{coar}$.

Proof: The probability that LSH random projection (Kataria et al., 2024; Datar et al., 2004) preserves the distance between two nodes v and u i.e., $d(u, v) = c$, is given by:

$$p(c) = \int_0^r \frac{1}{c} f_2\left(\frac{t}{c}\right)\left(1 - \frac{t}{r}\right) dt,$$

where $f_2(x) = \frac{2}{\sqrt{2\pi}}e^{-x^2/2}$ represents the Gaussian kernel when the projection matrix is randomly sampled from p-stable($p = 2$) distribution (Datar et al., 2004).
The probability $p(c)$ can be decomposed into two terms:

$$p(c) = S_1(c) - S_2(c),$$

$S_1(c)$ and $S_2(c)$ are defined as follows:

$$S_1(c) = \frac{2}{\sqrt{2\pi}}\int_0^r e^{-(t/c)^2/2}dt \leq 1,$$

$$S_2(c) = \frac{2}{\sqrt{2\pi}}\int_0^r e^{-(t/c)^2/2}\frac{t}{r}dt.$$

$$S_2(c) = \frac{2}{\sqrt{2\pi}} \cdot \frac{c}{r}\int_0^r e^{-(t/c)^2/2}\frac{t}{c^2}dt$$

Expanding $S_2(c)$:

$$S_2(c) = \frac{2}{\sqrt{2\pi}} \cdot \frac{c}{r} \int_0^{r^2/(2c^2)} e^{-y} dy$$

$$S_2(c) = \frac{2}{\sqrt{2\pi}} \cdot \frac{c}{r} \left[1 - e^{-r^2/(2c^2)}\right]$$

Thus, the probability $p(c)$ can be bounded as:

$$p(c) \leq 1 - \frac{2}{\sqrt{2\pi}} \frac{c}{r} \left[1 - e^{-r^2/(2c^2)}\right].$$

Now, let ϕ be the set of all pairwise distances $d(u, v)$, where $v \in \mathcal{E}_i$ and node$\omega_{V_\tau^i}$. The probability that all nodes in $\mathcal{N}_k(\mathcal{E}_i)$ are preserved within $\omega_{V_\tau^i}$, requires that all distances $c \in \phi$ are also preserved. The probability is then given by:

$$\prod_{c \in \phi} p(c).$$

$$\prod_{c \in \phi} p(c) \leq \prod_{c \in \phi} \left(1 - \frac{2}{\sqrt{2\pi}} \frac{c}{r} \left[1 - e^{-r^2/(2c^2)}\right]\right).$$

E Continual Learning and Dynamic Graph Learning

In this subsection, we highlight the key distinctions between Graph Structure Learning (GSL) and related fields to justify our specific selection of related works in Section 2.2. GSL is often confused with topics such as Continual Learning (CL) and Dynamic Graph Learning (DGL).

CL (Van de Ven & Tolias, 2019; Zhang et al., 2022; Parisi et al., 2019) addresses the issue of catastrophic forgetting, where a model's performance on previously learned tasks degrades significantly after training on new tasks. In CL, the model has access only to the current task's data and cannot utilize data from prior tasks. Conversely, DGL (Kim et al., 2022; Wu et al., 2023; You et al., 2022) focuses on capturing the evolving structure of graphs and maintaining updated graph representations, with access to all prior information.

While both *CL and DGL* aim to *enhance model adaptability* to dynamic data, GSL is primarily concerned with generating *high-quality graph structures* that can be leveraged for downstream tasks such as node classification (Kipf & Welling, 2016), link prediction (Lü & Zhou, 2011), and graph classification (Vogelstein et al., 2012). Moreover, in CL and DGL, different tasks typically involve distinct data distributions, whereas GSL assumes a consistent data distribution throughout.

F Related Work

Table 8 presents the formulations and associated time complexities of various unsupervised Graph Structure Learning methods.

Table 8: Unsupervised Graph Structure Learning Methods

Method	**Time Complexity**	**Formulation**
GLasso	$O(N^3)$	$\max_\Theta \log\det\Theta - \mathrm{tr}(\hat{\Sigma}\Theta) - \rho\lVert\Theta\rVert_1$
log-model	$O(N^2)$	$\min_{W\in\mathcal{W}} \lVert W \circ Z\rVert_{1,1} - \alpha \mathbf{1}^T \log(W\mathbf{1}) + \frac{\beta}{2}\lVert W\rVert_F^2$
$l2$-model	$O(N^2)$	$\min_{W\in\mathcal{W}} \lVert W \circ Z\rVert_{1,1} + \alpha\lVert W\mathbf{1}\rVert^2 + \alpha\lVert W\rVert_F^2 + \mathbf{1}\{\lVert W\rVert_{1,1} = n\}$
large-model	$O(N\log(N))$	$\min_{W\in\tilde{W}} \lVert W \circ Z\rVert_{1,1} - \alpha \mathbf{1}^T \log(W\mathbf{1}) + \frac{\beta}{2}\lVert W\rVert_F^2$

G Run Time Analysis

In the context of clustering module, $k - NN$ is the fastest algorithm, while Spectral Clustering is the slowest. Suppose we aim to learn the structure of a graph with N nodes. The clustering module, however, is only applied to a randomly sampled, smaller, static subgraph with k nodes, where $k \ll N$. In the worst-case scenario, spectral clustering requires $\mathcal{O}(k^3)$ time, whereas in the best case, $k - NN$ requires $\mathcal{O}(k^2)$ time. For coarsening module, LSH-based coarsening framework (Kataria et al., 2023),

has the best time complexity of $\mathcal{O}(\frac{k_\tau}{c})$ while FGC denotes the worst case with a time-complexity of $\mathcal{O}((\frac{k_\tau}{c})^2\|S_\tau^i\|)$ where c is the number of communities detected by clustering module $\mathcal{M}_{\text{clust}}$, $\|S_\tau^i\|$ is the number of coarsened node in the relevant community at τ timestamp and k_τ denotes number of nodes at τ timestamp. For learning module, $A-NN$ is the most efficient algorithm with time complexity as $\mathcal{O}(NlogN)$, while $GLasso$ has the worst computational cost of $\mathcal{O}(N^3)$. So, the effective time complexity of GraphFLEx is upper bounded by $\mathcal{O}(k^3 + (\frac{k_\tau}{c})^2\|S_\tau^i\| + \alpha^3)$ and lower bounded by $\mathcal{O}(k^2 + \frac{k_\tau}{c} + \alpha log\alpha)$ where $\alpha = \|S_\tau^i\| + \|\mathcal{E}_\tau^i\|$. GraphFLEx's efficiency in term of computational time is evident in Figure 2 and further quantified in Table 2.
Out of the three modules of GraphFLEx first module($\mathcal{M}_{\text{clust}}$) is trained once, and hence its run time is always bounded; computational time for second module($\mathcal{M}_{\text{coar}}$) can also be controlled because some of the methods either needs training once (Jin et al., 2021) or have linear time complexity (Kataria et al., 2023). Consequently, both the clustering and coarsening modules contribute linearly to the overall time complexity, denoted as $\mathcal{O}(N)$. Thus, the effective time complexity of GraphFLEx is given by $\mathcal{O}(N + \mathcal{O}(\mathcal{M}_{gl}(\|S_i, X_\tau^i\|))$. The overall complexity scales either linearly or sub-linearly, depending on α and the $\mathcal{M}_{gl}$ module. For instance, when $\mathcal{M}_{gl}$ is A-NN the complexity remains linear, if $\alpha\log(\alpha) \approx N$, whereas for $GLasso$, a linear behavior is observed when $\alpha^3 \approx N$.

H Node classification with GNNs and Graph Transformers.

We used four GNN models, namely GCN, GraphSage, GIN, and GAT. These GNN backbones (GCN, GraphSAGE, GIN, GAT) use two hidden layers of 64 units each and are trained for 500 epochs with Adam (learning rate 0.003, weight decay 5×10^{-4}).

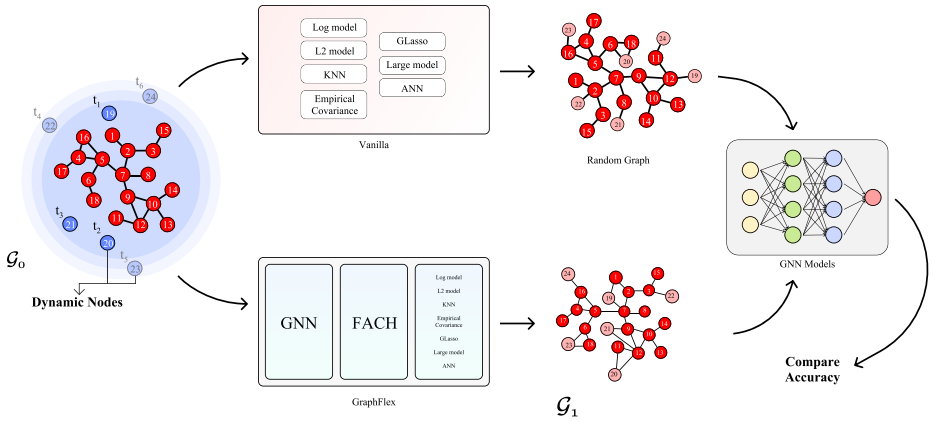


Figure 7: GNN training pipeline.

We randomly split data in 60%, 20%, 20% for training-validation-test. The results for these models on Cora, Citeseer, and some synthetic datasets are presented in Table 9. Figure 7 illustrates the pipeline for training our GNN models. Graph structures were learned using both existing methods and Graph-Flex, and GNN models were subsequently trained on both structures. Node classification accuracy results across all datasets are presented in Table 9 and Table 3. We further include node-classification accuracies of recent Graph Transformer models like GOAT (Kong et al., 2023), Nodeformer (Wu et al., 2022a), and Exoformer (Shirzad et al., 2023) when trained on graph structures learned by GraphFLEx (GLex) (see Table 10). Figure 8 shows the accuracies when structure is learned or given with 3 different scenarios (Vanilla, GraphFlex, Original) across different datasets.

I Computational Efficiency

Table 11 illustrates the remaining computational time for learning graph structures using GraphFLEx with existing Vanilla methods on Synthetic datasets. While traditional methods may be efficient for small graphs, GraphFLEx scales significantly better, excelling on large datasets like *Pubmed* and *Syn 5*, where most methods fail.

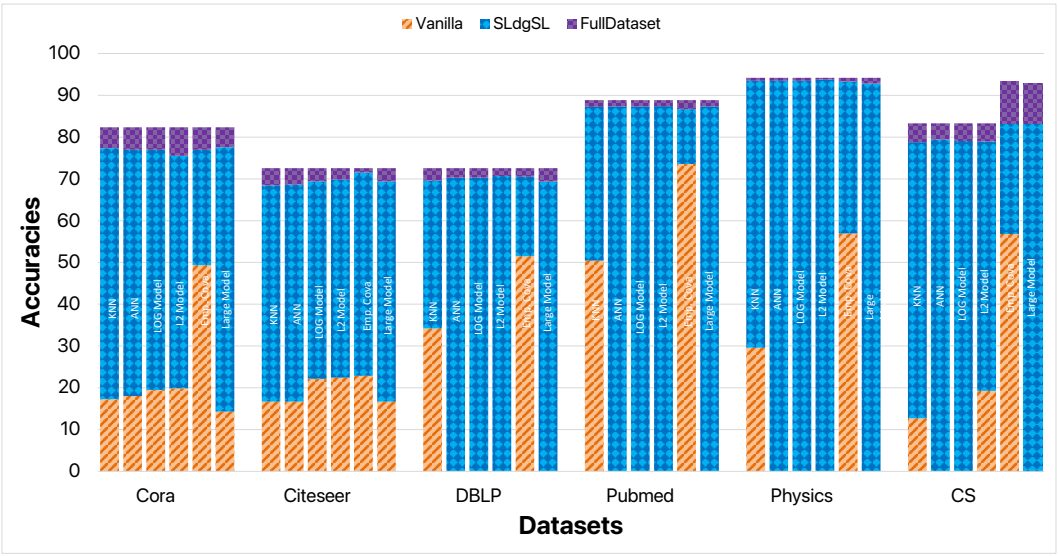


Figure 8: GraphSage accuracies when structure is learned or given with 3 different scenarios(Vanilla, GraphFlex, Original) across different datasets, highlighting performance with 30% node growth over 25 timestamps.

Table 9: Node classification accuracies on different GNN models using GraphFLEx (GFlex) with existing Vanilla (Van.) methods. The experimental setup involves treating 70% of the data as static, while the remaining 30% of nodes are treated as new nodes coming in 25 different timestamps. The best and the second-best accuracies in each row are highlighted by dark and lighter shades of Green, respectively. GraphFLEx's structure beats all of the vanilla structures for every dataset. OOM and OOT denotes out-of-memory and out-of-time respectively.

Dataset	Model	ANN		KNN		log-model		l2-model		COVAR		large-model		Base Struc.
		Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	
Cora	GAT	18.73	73.84	20.96	73.65	16.14	72.36	18.74	73.10	49.72	77.55	14.28	76.43	79.77
	SAGE	17.25	77.37	18.00	76.99	19.48	77.40	19.85	75.51	49.35	76.99	14.28	77.55	82.37
	GCN	17.99	78.11	17.81	77.92	18.55	77.74	20.41	79.22	47.31	80.52	14.28	79.03	84.60
	GIN	16.69	76.44	18.74	80.52	17.44	76.25	19.29	76.62	48.79	78.85	14.28	76.06	81.63
Citeseer	GAT	16.51	61.82	25.00	62.27	19.24	64.70	18.18	63.48	20.91	62.73	16.67	62.27	66.42
	SAGE	16.66	68.48	16.67	68.64	22.12	69.39	22.42	69.85	22.88	71.52	16.67	69.39	72.57
	GCN	28.18	60.00	16.67	61.97	20.45	65.45	19.70	64.24	21.06	64.70	16.67	63.18	68.03
	GIN	16.66	64.39	16.67	63.94	20.15	59.85	18.64	63.64	22.12	60.30	16.67	61.81	67.38
Syn 4	GAT	29.55	92.07	OOM	90.86	OOT	91.64	OOT	91.64	35.79	92.52	OOT	93.74	89.49
	SAGE	26.75	87.89	OOM	91.05	OOT	86.64	OOT	86.64	32.92	90.44	OOT	86.01	90.03
	GCN	28.85	51.97	OOM	19.58	OOT	18.29	OOT	18.92	33.80	26.60	OOT	36.85	21.43
	GIN	28.50	65.61	OOM	31.06	OOT	26.51	OOT	26.56	34.03	46.40	OOT	47.10	29.35
Syn 6	GAT	44.00	86.80	43.60	86.60	30.00	78.75	55.40	92.80	36.20	93.60	31.80	92.80	97.20
	SAGE	41.00	93.80	41.40	93.60	33.75	88.75	57.60	94.00	35.20	94.80	28.20	95.60	97.40
	GCN	43.60	88.80	42.20	87.40	26.25	81.25	55.60	92.40	31.40	94.40	25.20	94.00	99.40
	GIN	39.60	89.00	40.40	86.60	21.25	82.50	55.20	91.80	30.00	94.60	30.40	92.00	98.80
Syn 8	GAT	29.55	99.75	33.75	88.75	88.25	99.25	88.25	99.25	26.00	85.50	94.00	96.00	98.50
	SAGE	26.75	100.0	32.50	100.0	88.75	99.50	88.75	99.50	26.75	100.0	92.50	100.0	100.0
	GCN	28.85	98.75	31.75	99.75	88.75	99.00	88.75	99.00	28.50	99.25	95.00	100.0	100.0
	GIN	28.50	50.00	30.50	91.00	82.25	91.50	82.25	91.50	27.25	81.75	91.75	92.25	78.25

J VISUALIZATION OF GROWING GRAPHS

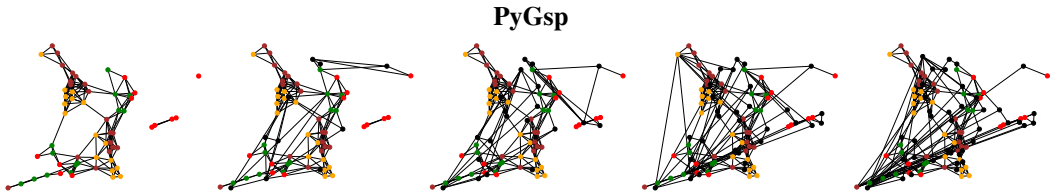
This section helps us visualize the phases of our growing graphs. We have generated a synthetic graph of 60 nodes using PyGSP-Sensor and HE methods mentioned in Appendix A. We then added 40 new nodes denoted using black color in these existing graphs at four different timestamps. Figure 9 and Figure 10 shows the learned graph structure after each timestamp for two different Synthetic graphs.

Table 10: Node classification accuracies on different recent Graph Transformer models when trained on structure learned using GraphFLEx (GFlex). The experimental setup involves treating 70% of the data as static, while the remaining 30% of nodes are treated as new nodes coming in 25 different timestamps.

Data	Model	Base	GFlex(KNN)	GFlex(ANN)	GFlex(Sub.)	GFlex(covar)	GFlex(l2)	GFlex(log)
Cora	Goat	68.01	65.90	70.20	70.10	71.20	69.60	68.30
	Nodeformer	65.80	65.80	65.80	64.50	65.91	63.12	64.27
	Exoformer	71.80	68.16	69.80	70.30	67.70	69.80	70.30
Citeseer	Goat	65.02	65.10	62.40	66.13	61.20	63.03	66.10
	Nodeformer	61.00	61.00	61.71	63.48	62.60	61.34	61.54
	Exoformer	58.40	60.70	59.80	60.70	59.40	60.21	59.80
DBLP	Goat	81.92	80.68	80.31	82.25	82.29	79.86	80.10
	Nodeformer	73.76	73.76	75.34	74.39	72.94	73.29	74.96
	Exoformer	72.12	72.12	72.71	73.61	72.92	69.84	71.62
CS	Goat	93.05	93.24	93.32	90.84	92.39	93.16	92.03
	Nodeformer	94.60	95.12	95.12	95.36	94.91	95.12	93.29
	Exoformer	95.53	95.33	94.63	93.26	95.33	95.09	94.32

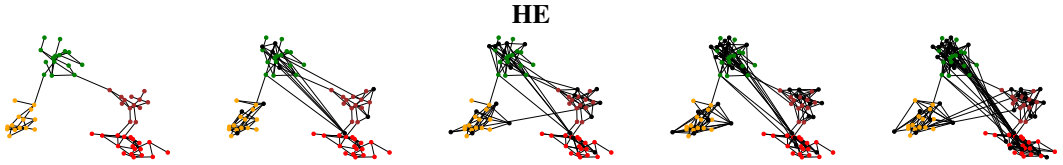
Table 11: Computational time for learning graph structures using GraphFLEx (GFlex) with existing methods (Vanilla referred to as Van.). The experimental setup involves treating 50% of the data as static, while the remaining 50% of nodes are treated as incoming nodes arriving in 25 different timestamps. The best times are highlighted by color Green. OOM and OOT denote out-of-memory and out-of-time, respectively.

Data	ANN		KNN		log-model		l2-model		COVAR		large-model	
	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex	Van.	GFlex
Syn 1	19.4	9.8	2.5	10.5	2418	56.4	37.2	8.8	3.5	8.3	205	9.4
Syn 2	47.3	16.9	6.6	18.3	14000	144	214	22.6	20.3	18.6	1259	16.4
Syn 5	5.1	11.5	0.8	7.3	57.4	28	1.1	5.8	0.2	4.8	3.2	5.3
Syn 6	16.6	9.9	2.8	11.4	1766	96.3	193	101	5.3	8.9	324	9.6
Syn 7	10.6	7.4	1.4	8.9	704	85.2	10.3	7.9	0.9	6.4	36.5	8.2
Syn 8	19.6	11.2	2.5	11.7	2416	457	37.2	17.0	3.4	10.9	204	11.7



(a) Initial graph G_0 (b) α= 10, G_1 (c) α= 20, G_2 (d) α = 30, G_3 (e) α = 40, G_4

Figure 9: This figure illustrates the growing structure learned using GraphFlex for dynamic nodes. New nodes are denoted using black color, and α denotes number of new nodes. *PyGsp* denotes type synthetic graph.



(a) Initial graph G_0 (b) α = 10, G_1 (c) α = 20, G_2 (d) α= 30, G_3 (e) α = 40, G_4

Figure 10: This figure illustrates the growing structure learned using GraphFlex for dynamic nodes. New nodes are denoted using black color, and α denotes the number of new nodes. *HE* denotes the type of synthetic graph.

K Ablation Study

In this section, we present an ablation study to analyze the role of individual modules within GraphFLEx and their influence on the final graph structure. Specifically, we focus on two aspects: (i) the significance of the clustering module, and (ii) the effect of varying module configurations on the learned graph topology.

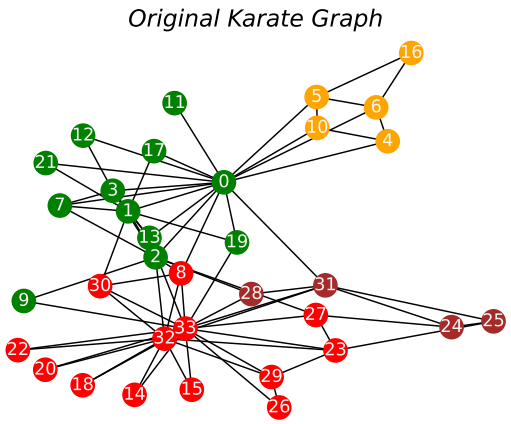
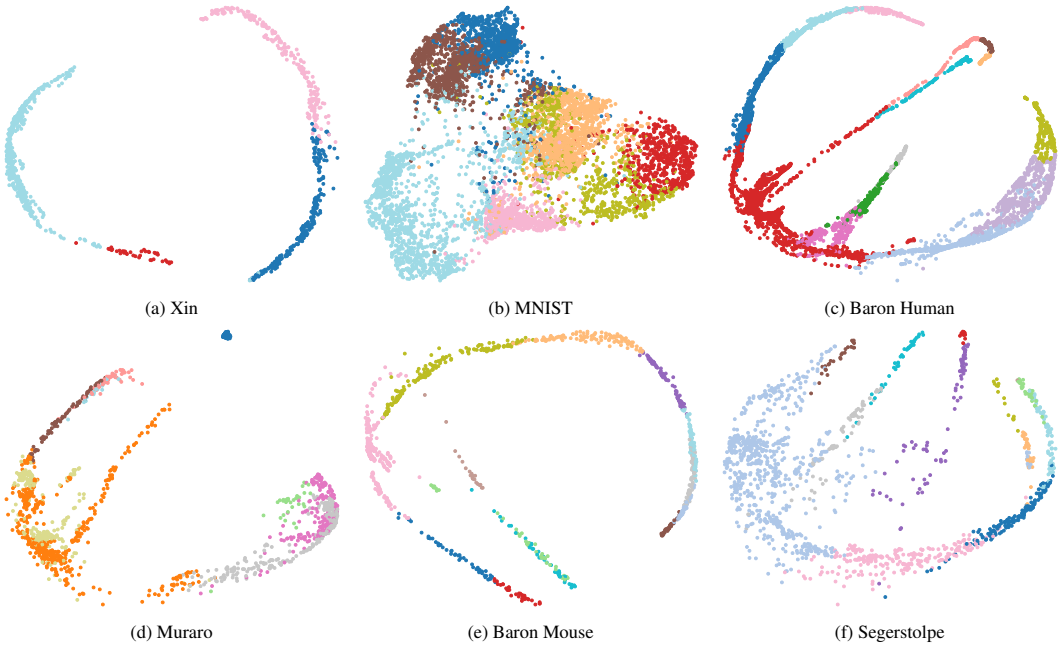


Figure 11: Original Karate Graph



(a) Xin (b) MNIST (c) Baron Human

(d) Muraro (e) Baron Mouse (f) Segerstolpe

Figure 12: PHATE visualization of clusters learnt using GraphFlex clustering module for scRNA-seq datasets.

K.1 Clustering Quality

Figure 12 shows the PHATE (Moon et al., 2019) visualization of clusters learned using GraphFLEx's clustering module $\mathcal{M}_{clust}$ for 6 single-cell RNA datasets, namely Xin, $MNIST$, $Baron-Human$, $Muraro$, $BaronMouse$, and $Segerstolpe$ datasets.

K.2 Clustering Module Evaluation

To evaluate the effectiveness of the clustering module, we compute standard metrics such as Normalized Mutual Information (NMI), Conductance (C), and Modularity (Q) across various datasets (see Table 6 in Section 4.5). These metrics collectively validate the quality of the discovered clusters, thereby justifying the use of a clustering module as a foundational step in GraphFLEx. Since clustering in GraphFLEx is applied only once on a randomly sampled small set of nodes, selecting the right method can be considered as part of hyperparameter tuning, where these clustering measures can guide the optimal choice based on dataset characteristics.

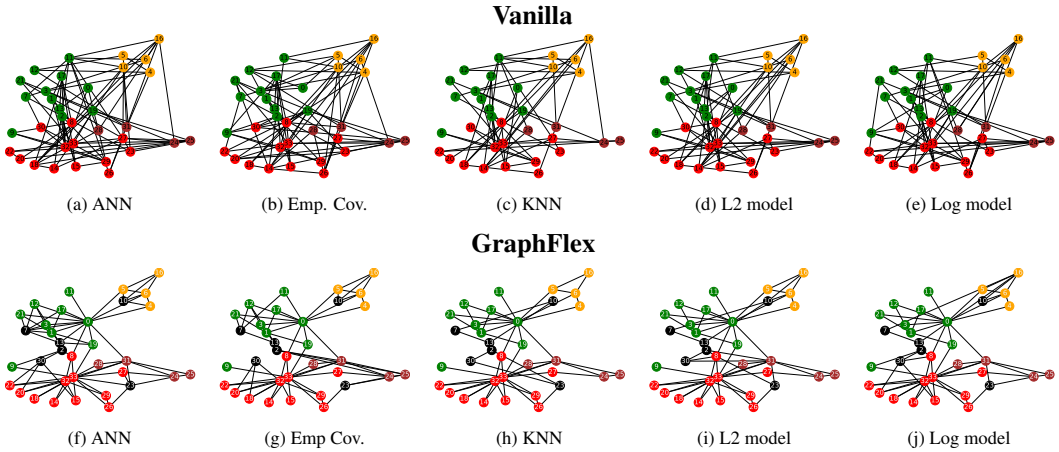


Figure 13: This figure compares the structures learned on Zachary's karate dataset when existing methods are employed with GraphFlex and when existing methods are used individually. We consider six nodes, denoted in black, as dynamic nodes.

K.3 Impact of Module Choices on Learned Graph Structure

This section involves a comparison of the graph structure learned from GraphFlex with existing methods. Six nodes were randomly selected and considered as new nodes. Figure 13 visually depicts the structures learned using GraphFlex compared to other methods. It is evident from the figure that the structure known with GraphFlex closely resembles the original graph structure. Figure 11 shows the original structure of Zachary's karate club network (Zachary, 1977). We assumed six random nodes to be dynamic nodes, and the structure learned using GraphFlex compared to existing methods is shown in Figure 13.

L Module Selection

The Experiments section provides analyses to inform the selection of clustering and coarsening components. In particular, Section 4.5 evaluates clustering quality using quantitative metrics, which help choose a clustering method suited to a graph's community structure and distribution. Moreover, Figs. 13 illustrate how different learning-module choices impact the final learned structure. Additional practical guidance is summarized below:

Clustering Methods. *K-means* is computationally efficient and performs well when clusters are roughly spherical; it is a strong choice for large datasets where speed is critical. *Spectral clustering* leverages eigenvalue decompositions to capture complex community structure, even when groups are not easily separated by simple distance metrics; however, it can be expensive at scale. *Deep learning–based clustering* adapts to non-linear, high-dimensional patterns and is effective for complex, feature-rich graph data, though it typically requires more compute.

Coarsening Methods. *UGC* (used in our main experiments) is a scalable, LSH-based approach that merges nodes with similar features and supports arbitrary coarsening ratios. Spectral approaches such as *LVE* and *LVN* preserve local eigenstructure but can limit coarsening flexibility. *Heavy Edge Matching (HE)* prioritizes the contraction of strong edges and is efficient for edge-weighted graphs. *Algebraic-distance* and *affinity-based* schemes rely on node-proximity metrics and can be more costly. *Kron reduction* maintains spectral fidelity but is typically too slow for very large graphs. *FGC* jointly integrates graph structure and node attributes, optimizing clustering and coarsening together, but is computationally demanding.