DYNAMIC GNN ADAPTATION VIA CACHED PROPAGATION AND SVD-GUIDED SELECTIVE OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks offer a potent mechanism for learning on graph data across domains and tasks. A vast majority of these mechanisms assume that the underlying graph and the corresponding node/edge attributes are static, and do not change over time. However, graphs corresponding to most real-world phenomena evolve via the addition or deletion of nodes and edges, or changes to node/edge attributes.

This paper proposes efficient solutions for maintaining the GNN over such evolving graphs without recourse to full retraining. Our solution use cached propagation for the forward passes and a spectral value decomposition based selective optimization for the backward passes.

We adapt our solution to two representative GNN architectures, Graph Convolution Networks (GCNs) and Graph Sample and Aggregate (SAGE) over two downstream tasks: node classification and link prediction. Extensive experimental evaluation indicates that our solutions offer competitive accuracy and speedup over existing dynamic GNN methods while keeping the spectral drift small [1,2].

1 INTRODUCTION

Graph neural networks (GNNs) have emerged as a powerful solution for learning on graph data and have produced remarkable success across domains over various tasks such as node classification, link prediction, recommendation systems, fraud detection, biological network analysis, and scientific knowledge discovery (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Wu et al., 2020). Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) and GraphSAGE (SAGE) (SAmple and aggreGatE) (Hamilton et al., 2017) have become particularly popular due to their ability to effectively aggregate neighborhood information through message passing operations and learn expressive graph representations. However, most of the existing GNN methods assume that the underlying graph is static, where the topology of the graph and node/edge attributes remain fixed during the training/inference phases.

In most of the real-world networks, graphs are inherently dynamic and evolve through the addition or deletion of nodes and edges over time (Kazemi et al., 2020). For example, citation networks evolve as new papers are published. When the underlying graph evolves, the GNN model trained on the original graph must be updated or retrained to adapt to new structural changes. In such scenarios, retraining the GNN models from scratch after every graph update becomes computationally expensive and often impractical, especially for large-scale networks, and may introduce instability in representations.

Recent research on dynamic and incremental graph learning focuses on addressing these limitations through incremental updates, temporal frameworks, or effective subgraph adaptations (Pareja et al., 2020; Rossi et al., 2020; Xu et al., 2020; Di & et al., 2022). However, these do not explicitly exploit the structural continuity over graph updates (Ou et al., 2016; Levie et al., 2021; NT & Maehara, 2019). In many dynamic networks, consecutive graphs with batch updates maintain strong structural similarity, as the small batch updates do not strongly affect the overall graph structure. This implies that the representations learned on them evolve smoothly over minor graph updates rather than large spectral drifts (Gama et al., 2020; Levie et al., 2021). Large changes in the singular values of the

[1] Visit https://anonymous.4open.science/r/DASS-GNN-692C for the source code and datasets for this paper.
[2] **Disclosure:** LLMs were used for grammar checking and paraphrasing of author-written text only.

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.

weight matrix in GNNs may destabilize the latent representation geometry, leading to performance degradation on downstream tasks (Kipf & Welling, 2016; Zhang & Chen, 2018), particularly in link prediction, where pairwise structural relations are important.

These challenges motivate the development of an efficient dynamic graph learning framework capable of controlled spectral adaptation while reducing redundant computations during graph evolution. A key observation underlying our work is that graph updates and the resulting propagation changes are often localized. Consequently, in the forward pass, recomputing propagation for the entire graph after every update is unnecessary, as most of the previous propagation remains valid. At the same time, in the backpropagation, adapting all weight parameters of a GNN to every update can also be excessive, since adapting only a subset of the model parameters can be sufficient to accommodate the new graph state while reducing optimization cost. These two redundant computational bottlenecks arise in GNN training: repeated propagation in the forward pass and unnecessary parameter optimization in the backward pass. Unlike existing works that focus mainly on temporal graph propagation, our proposed framework, DASS-GNN (Dynamic Adaptation through Selective Spectral Optimization for GNNs), investigates the dynamic graph learning problem from a controlled spectral adaptation perspective. DASS-GNN combines two key components: (1) reuse graph propagation through incremental updates and (2) SVD-guided selective parameter optimization for preserving important spectral directions. We instantiate the propagation caching mechanism for two representative message-passing GNNs: cached-AX propagation for GCN and cached neighborhood aggregation for SAGE.

Alongside this, the proposed work analyzes spectral deviations using drift in singular value spectra and weight drift under graph updates and demonstrates different spectral behaviors across downstream tasks such as node classification and link prediction.

The main contributions of this work are as follows:

- We propose an efficient dynamic graph learning framework, DASS-GNN, based on propagation reuse and SVD-guided selective optimization for evolving graphs.
- We demonstrate DASS-GNN framework using two popular message-passing GNNs, viz., GCN and SAGE.
- We conduct extensive experiments on five benchmark datasets across scales and domains for node classification and link prediction, demonstrating substantial computational reductions while maintaining predictive performance, scalability, and spectral stability over retraining.

2 Related Work

Popular GNN models, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), and graph attention networks (Veličković et al., 2018), are designed under the assumption that the underlying graph is static and require retraining from scratch on graph updates. To address this limitation, Dynamic Graph Neural Networks (DGNNs) (Kazemi et al., 2020; Xie & et al., 2024) were introduced to model topological and temporal evolution through discrete-time and continuous-time architectures.

Dynamic GNNs Recent representative DGNN models include EvolveGCN (Pareja et al., 2020), DYSAT (Sankar et al., 2020), TGN (Rossi et al., 2020), and IncreGNN (Di & et al., 2022). EvolveGCN (Pareja et al., 2020) trains RNNs using the GNN weight parameters over time to predict future parameters; DySAT (Sankar et al., 2020) combines topological and temporal information through self-attention mechanisms; TGN (Rossi et al., 2020) introduces memory modules and event-driven aggregation methods for continuous-time graph learning. IncreGNN (Di & et al., 2022) fine-tunes the original model using a k-hop subgraph around the affected nodes to reduce computations. However, these models either repeatedly recompute after every graph update or fine-tune using localized subgraphs, resulting in computational overhead or loss of global context.

Efficient Message Passing To improve the scalability of static graph learning, FastGCN (Chen et al., 2018) and GraphSAINT (Zeng et al., 2020) introduce stochastic neighborhood sampling strategies, whereas SGC (Wu et al., 2019) and SIGN (Frasca et al., 2020) utilize precomputations and cached propagation. Although these models are effective for static graphs, precomputations and cached propagation become outdated and invalid as the graph updates.

Parameter-Efficient Optimization Another interesting research direction is parameter-efficient optimization for reducing training cost. Singular Value Decomposition (SVD) is a widely used low-rank matrix approximation for model reduction and optimization (Eckart & Young, 1936). Recent methods like LoRA (Hu et al., 2022) demonstrate that updating only a small subset of learnable model parameters can preserve predictive performance. Further, adaptive neural computation (Han et al., 2021) selectively optimizes crucial model components to improve training efficiency.

3 Proposed Approach

3.1 Problem Formulation

Let $G_t = (V_t, E_t)$ denote a graph at time t, where V_t is the set of nodes and E_t is the set of edges at time $t \geq 0$, adjacency matrix $A_t \in \mathbb{R}^{n \times n}$ where $n = |V_t|$, and feature matrix $X_t \in \mathbb{R}^{n \times d}$, where d is the feature vector dimension of each node.

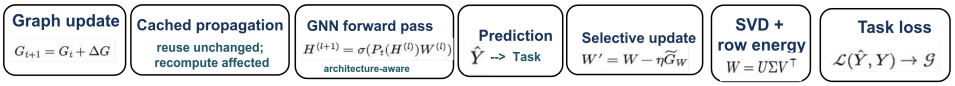


Figure 1: DASS-GNN Architecture

We consider a discrete-time dynamic graph evolution over batches of edges inserted or deleted. Let ΔG_t denote the graph update from time t to $t+1$. The updated graph at time $t+1$ is given as $G_{t+1} = G_t + \Delta G_t$, where $\Delta G_t = (V_t, \Delta E_t)$ with $\Delta E_t = \Delta E_t^+ \cup \Delta E_t^-$. Further, we set $E_{t+1} := \left(E_t \cup \Delta E_t^+\right) \setminus \Delta E_t^-$. The adjacency matrix at time $t+1$ is defined as $A_{t+1} = A_t + \Delta A_t$, where ΔA_t denotes the affected adjacency due to ΔG_t and is defined as $(\Delta A_t)_{ij} = +1$ if edge (i, j) is inserted, -1 if edge (i, j) is deleted, and 0 otherwise.

A message-passing GNN can generally be expressed as a sequence of propagation and transformation operations. For layer ℓ, we write $H_t^{(\ell+1)} = \sigma(\mathcal{P}_t(H_t^{(\ell)}) \times W^\ell)$, where $H_t^{(\ell)}$ denote node representations at layer ℓ, $\mathcal{P}_t(.)$ denote the graph propagation at time t, W^ℓ denote the learnable weight matrix, and $\sigma(.)$ denote the activation function. The propagation operator $\mathcal{P}$ depends on the underlying GNN model. For example, $\mathcal{P}$ corresponds to adjacency propagation in GCN, whereas it corresponds to mean neighborhood aggregation in SAGE.

3.1.1 Problem Statement

Given a dynamic graph $G_t(V_t, E_t)$ at time t, a trained GNN model $\mathcal{M}_t$, and a batch update ΔE_t, the objective is to efficiently obtain an adapted model $\mathcal{M}_{t+1}$ for the graph G_{t+1}.

A naive solution to the above problem is to retrain the model on the updated graph. For a sparse message-passing layer, this retraining is proportional to the graph size and can become computationally expensive for large graphs. Moreover, in many real-world settings, graph changes are very small compared to the original graph $|\Delta E_t| \ll |E_t|$. Therefore, it is expected that the change in the propagation is localized to the endpoints of edges in ΔE_t, and in the back-propagation, one can localize the change to the dominant spectral directions. We use the above observations to propose a solution in the following.

3.2 DASS-GNN

In this section, we propose a dynamic graph representation learning framework, DASS-GNN, that introduces mechanisms for forward-pass (Section 3.2.1), and backward pass (Section 3.2.2). Figure 1 shows the architecture of our solution.

3.2.1 Propagation Caching

The first component of DASS-GNN reduces redundant computation in the forward pass by caching graph-dependent propagation results. On an update ΔG_t at time t, a naive DGNN recomputes the propagation over the entire graph. This involves repeating computations for nodes whose propagation dependencies have not changed. To characterize the portion of the propagation that must be recomputed, we consider defining an affected-node set V_{aff}. For nodes outside this set, i.e., $v \notin V_{\text{aff}}$, the cached propagation can be reused: $\mathcal{P}_{t+1}(H)_v = \mathcal{P}_t(H)_v$. The actual composition of the set V_{aff}depends on the GNN architecture. Thus, propagation caching avoids redundant computations by restricting recomputation only to the rows of $\mathcal{P}_t$ whose neighborhood dependencies have changed after a graph update, as given in Algorithm 1.

Theorem 1 (Localized Propagation). *Let $\mathcal{P}_t$ and $\mathcal{P}_{t+1}$ denote the propagation operators before and after the graph update ΔG_t, and let V_{aff} denote a set of endpoints and their one-hop neighbors from ΔE_t. For any node $v \notin V_{\text{aff}}$, updated propagation can be reused from cache as $\mathcal{P}_{t+1}(i,:) = \mathcal{P}_t(i,:)$.*

The proof for Theorem 1 is provided in the Appendix A.

Theorem 2 (Propagation Perturbation Bound). *Let the propagation matrices at time t and $t+1$ be, $P_t = \hat{A}_t X$, $P_{t+1} = \hat{A}_{t+1} X$ respectively, where the normalized adjacency matrix is $\hat{A}_t = D_t^{-1/2}(A_t + I)D_t^{-1/2}$, and the fixed feature matrix is $X \in \mathbb{R}^{n \times d}$. Then the propagated feature perturbation satisfies the following: $\|P_{t+1} - P_t\|_F \leq \|\hat{A}_{t+1} - \hat{A}_t\|_F \, \|X\|_F$.*

The proof for Theorem 2 is given in Appendix A.

Theorems 1,2 provide the theoretical basis for using Algorithm 1. By Theorem 1, recomputation due to batch update is required only for the rows indexed by V_{aff}, while propagation of remaining rows can be safely reused. Theorem 2 quantifies the effect of graph-induced perturbations on propagated features.

We note that the caching mechanism is architecture-aware. Different GNN architectures can use appropriate architecture-aware mechanisms at Line 3 in Algorithm 1 to implement propagation. In this work, we show this for two representative message-passing architectures: GCN (Kipf & Welling, 2017) and SAGE (Hamilton et al., 2017).

Cached-AX Propagation for GCN For GCN, the cached quantity is the normalized adjacency-feature product (Cached-AX). Let $\hat{A}_t = D^{\frac{-1}{2}}(A_t + I)D^{\frac{-1}{2}}$ denote the normalized adjacency matrix, $\mathcal{P}_t = \hat{A}_t X_t$ denote the cached propagation matrix, and $\mathcal{N}_t(v)$ denote its neighborhood in G_t. Further, we define V_{aff} as the endpoints of the edges in ΔG_t along with their one-hop neighbors.

For GCN, we specify Line 3 of Algorithm 1 as follows. After a graph update ΔG_t, a standard GCN model recomputes $\mathcal{P}_{t+1} = \hat{A}_{t+1} X$ over the entire updated graph. Instead, the proposed Cached-AX propagation calculates updated propagation for a row $i \in V_{\text{aff}}$ using $\mathcal{P}_{t+1}(i,:) = \sum_{j \in \mathcal{N}_{t+1}(i) \cup \{i\}} \hat{A}_{t+1,ij} X_j$, whereas for an unaffected row we set $\mathcal{P}_{t+1}(i,:) = \mathcal{P}_t(i,:)$.

Cached-SAGE Neighborhood Aggregation for SAGE For SAGE, the cached quantity is the first-hop neighborhood aggregation (Cached-SAGE) computed over node features $h^{(0)} = X$ used by the message-passing layer. For a node v at layer ℓ, let $\mathcal{N}_t(v)$ denote its neighborhood in G_t. The SAGE aggregation can be written as $m_t^{(\ell=0)}(v) = \frac{1}{|\mathcal{N}_t(v)|} \sum_{u \in \mathcal{N}_t(v)} h_t^{(0)}(u)$. The resulting node representation can be computed as $h_t^{(\ell+1)}(v) = \sigma\left(W^{(\ell)} \times \left[h_t^{(\ell)}(v) || m_t^{(\ell)}\right]\right)$, where $||$ denotes concatenation. We define V_{aff} as the edge endpoints in ΔG_t along with their 1-hop neighbors.

For SAGE, we specify Line 3 of Algorithm 1 as follows. After a graph update ΔG_t, standard SAGE recomputes these neighborhood aggregations over the entire updated graph, yielding $\mathcal{P}_{t+1}$. Instead, the proposed Cached Neighborhood Aggregation recomputes $\mathcal{P}_{t+1}(i,:) = \frac{1}{|\mathcal{N}_{t+1}(i)|} \sum_{j \in \mathcal{N}_{t+1}(i)} h_{t+1}^{(\ell)}(j)$ only for nodes $i \in V_{\text{aff}}$ whose neighborhoods are affected by ΔG_t, while retaining the cached values for unaffected nodes ($i \notin V_{\text{aff}}$) as $\mathcal{P}_{t+1}(i,:) = \mathcal{P}_t(i,:)$.

While propagation caching reduces the computational cost of the forward pass, backpropagation still updates the entire weight matrix, leading to unnecessary optimization overhead under small graph updates. To address this, we introduce a selective parameter-update mechanism that prioritizes rows associated with the left singular subspace of the weight matrix.

3.2.2 SVD-Guided Selective Backward Updates

Let the learnable weight matrix at layer ℓ be $W^{(\ell)}$. For a given training loss $\mathcal{L}$, the gradient with respect to $W^{(\ell)}$ is $\nabla W^{(\ell)} = \frac{\partial \mathcal{L}}{\partial W^{(\ell)}}$. Using the gradient ∇, the standard gradient descent algorithm updates the entire weight matrix using $W^{(\ell)} \leftarrow W^{(\ell)} - \eta \nabla W^{(\ell)}$, where η denotes the learning rate.

However, not all rows of the weight matrix $W^{(\ell)}$ contribute equally to the training process. The truncated singular value decomposition of the weight matrix is $W^{(\ell)} \approx U_r^{(\ell)} \Sigma_r^{(\ell)} (V_r^{(\ell)})^\top$, where r denotes the number of retained singular components, $\Sigma_r^{(\ell)} \in \mathbb{R}^{r \times r}$ denote the singular matrix containing $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$, $U_r^{(\ell)} \in \mathbb{R}^{d \times r}$ and $V_r^{(\ell)} \in \mathbb{R}^{r \times d'}$ denotes left and right singular

matrices, respectively. We compute the row energy score for a row i as $s_i = \sum_{j=1}^{r} \left(U_r^{(\ell)}\right)_{ij}^2$. The row energy score measures how much a weight-matrix row participates in the retained subspace of $U_r^{(\ell)}$. A larger row energy score s_i indicates that row i aligns more closely with the retained subspace, has stronger representation, and is therefore prioritized for selective parameter updates.

For a given update budget of k rows, we select the top-k dominant row indices $\mathcal{I} = \text{TopK}(\{s_i\}, k)$ in descending order of row energies. Then, we construct a binary mask $M \in \{0,1\}^{d \times d'}$ based on the selected rows as follows: $M_{ij} = 1^\top$ if $i \in \mathcal{I}, 0^\top$ otherwise. The masked gradient is defined as $\widehat{\nabla W}^{(l)} = M \odot \nabla W^{(l)}$, where $\odot$ denotes element-wise multiplication. Finally, the model parameters are updated via $W^{(l)} \leftarrow W^{(l)} - \eta \widehat{\nabla W}^{(l)}$. We define the parameter update fraction as $\rho_{\text{param}} = \frac{k}{d_{\text{out}}} \in (0, 1]$. Thus, only the selected (dominant) rows are updated while preserving the remaining parameter space. Algorithm 2 presents SVD-guided selective backpropagation.

Algorithm 1: Propagation Caching

Input: Graph $G_t = (V_t, E_t)$, graph update ΔE_t, node features X_t
Output: Updated cached propagation $\mathcal{P}_{t+1}$

Identify the affected node set V_{aff} from the changed propagation dependencies induced by ΔE_t ;
foreach *node* $i \in V_{\text{aff}}$ **do**
    $\mathcal{P}_{t+1}(i,:) \leftarrow$ recomputed_propagation(i) ;
    // depends on GNN architecture
foreach *node* $i \notin V_{\text{aff}}$ **do**
    $\mathcal{P}_{t+1}(i,:) = \mathcal{P}_t(i,:)$;
return $\mathcal{P}_{t+1}$

Algorithm 2: SVD-Guided Selective Backpropagation

Input: Weight matrix W, rank r, top k rows, gradients ∇W
Output: Selectively updated weight matrix

Compute truncated SVD: $W \approx U_r \Sigma_r V_r^\top$;
Compute row energy scores: $s_i = \sum_{j=1}^{r} \left(U_r^{(\ell)}\right)_{ij}^2$;
Select top-k important rows: $\mathcal{I} = \text{TopK}(s_i)$;
Initialize binary mask $M \leftarrow 0$;
foreach *row index* $i \in \mathcal{I}$ **do**
    Set: $M_{i,:} = 1$;
Apply masked gradient update: $\nabla W \leftarrow M \odot \nabla W$;
Update parameters using optimizer ;

Theorem 3 (Gradient Energy Preservation under SVD Masking)**.** *Let $W^{(\ell)} \in \mathbb{R}^{d_{out} \times d_{in}}$ denote the layer ℓ weight matrix, $\nabla W^{(\ell)}$ denote its parameter gradient, $W^{(\ell)} \approx U_r \Sigma_r V_r^\top$ denote its rank-r truncated SVD, and $\mathcal{I} = \text{TopK}(\{s_i\}, k)$ denote the set of selected top-k row indices based on row energies $s_i = \sum_{j=1}^{r} (U_r)_{ij}^2$. For the binary mask $M \in \{0,1\}^{d_{out} \times d_{in}}$, the masked gradient update $\widetilde{\nabla W}^{(\ell)} = M \odot \nabla W^{(\ell)}$ satisfies $\|\nabla W^{(\ell)} - \widetilde{\nabla W}^{(\ell)}\|_F^2 \leq (1 - \rho_{spec}) \|\nabla W^{(\ell)}\|_F^2$, where $\rho_{spec} = \frac{1}{r} \sum_{i \in \mathcal{I}} s_i \in (0, 1]$ represents the fraction of preserved spectral energy across d_{out} rows.*

Theorem 4 (Bounded Parameter Drift Guarantee)**.** *Assume the loss function is L-smooth and gradients are bounded such that $\|\nabla W_T^{(l)}\|_F \leq G$ across fine-tuning iterations T. Under SVD-guided masked backpropagation over T_e fine-tuning epochs with learning rate η and parameter update ratio $\rho_{param} = k/d_{out}$, the parameter drift at dynamic update step t relative to the previously adapted model $W_{t-1}^{(\ell)}$ satisfies $\|W_t^{(\ell)} - W_{t-1}^{(\ell)}\|_F \leq \eta \cdot T_e \cdot \sqrt{\rho_{param}} \cdot G$ while full retraining from random initialization satisfies $\|W_t^{(\ell)retrain} - W_{\text{rand}}^{(\ell)}\|_F \leq \eta T_{\text{retrain}} \cdot G$.*

Theorem 3 explicitly proves energy preservation of the truncated left singular subspace by the selected row optimization. Building on this, Theorem 4 guarantees that selective updates strictly bound parameter drift, ensuring optimization stability across dynamic steps.

The effectiveness of this selection strategy is therefore evaluated empirically through ablation and sensitivity experiments. As these selected rows correspond to dominant singular directions justified by Theorem 3, most of the spectral energy is preserved while reducing computations.

Algorithm 3 provides a complete dynamic adaptation procedure proposed in DASS-GNN. We denote the GCN and SAGE implementations of the proposed framework as **DASS-GNN-C** and **DASS-GNN-S**, respectively. In Line 4 of Algorithm 3, DASS-GNN-C (*resp.* DASS-GNN-S) uses the modified Algorithm 1 as mentioned in Section 3.2.1 (3.2.1).

Algorithm 3: Complete DASS-GNN Dynamic Adaptation

Input: Graph G_t, node features X, model $\mathcal{M}_t$, graph updates ΔG_t
Output: Adapted model $\mathcal{M}_{t+1}$

Initialize propagation cache using G_t and $\mathcal{M}_t$;
Receive graph update ΔG_t and obtain updated graph G_{t+1};
Identify propagation dependencies affected by ΔG_t;
Update propagation cache using Algorithm 1;
Perform forward propagation using the updated cache;
Compute task-specific loss;
Perform SVD-guided backpropagation using Algorithm 2;
Obtain adapted model $\mathcal{M}_{t+1}$;
return *adapted model $\mathcal{M}_{t+1}$ and updated propagation;*

Table 1: Characteristics of the datasets

Dataset	Nodes	Edges	Features	Classes
Cora (G1)	2,708	5,429	1,433	7
Citeseer (G2)	3,327	4,732	3,703	6
PubMed (G3)	19,717	44,338	500	3
Ogbn-arxiv (G4)	169,343	1,166,243	128	40
Ogbn-products (G5)	2,449,029	61,859,140	100	47

4 Experimental Setup

Our experimental objectives include predictive performance, computational efficiency, and spectral drift, under dynamic graph updates. We also study the sensitivity of our algorithms.

4.1 Datasets, Baseline Methods and Model Configurations

We evaluate DASS-GNN on five benchmark datasets from (Yang et al., 2016; Hu et al., 2020b) spanning graph scales and domains. The datasets exhibit diverse homophily, sparsity, scale, and update sensitivity, providing a comprehensive evaluation under different graph properties. Dataset statistics are reported in Table 1. Dynamic updates are introduced through batch edge insertions and deletions. To simulate realistic graph evolution, dynamic updates are introduced through batch edge insertions and deletions. Given an initial graph $G_0 = (V_0, E_0)$ constructed using an initial edge ratio $\alpha = 0.7$, the graph undergoes dynamic perturbations across update steps. ΔE_t^+ is sampled uniformly from the unobserved edge stream, while ΔE_t^- accounts for structural decay by removing active edges uniformly at random.

We compare the proposed methods, DASS-GNN-C and DASS-GNN-S, against the baselines given in Table 2. The retrain method initializes a new GNN model after every update, whereas Cached-AX, IncreGNN, and DASS-GNN models adapt on existing tuned model parameters. Therefore, spectral drift comparisons reflect parameter evolution under graph updates. For all the baselines and proposed models, the learning rate is set to 5×10^{-4} except for retraining (0.01), optimizer is Adam; dropout rate to 0.3, hidden dimensions to 128, initial edge ratio to 0.7, and the number of batches to 10. For DASS-GNN-C/S models, top-k rows is set to 10%, for IncreGNN, the subgraph hops as 2.

Table 2: Baseline methods and computational complexity. GCN and SAGE indicate the GNN backbones used in our experiments. Here, $|E|$ is the number of edges, $|\Delta E|$ is the number of edges in the graph update, d and d_{out} are hidden and output dimensions, k is the number of rows updated by selective optimization, r is LoRA rank, and $|E_s|$ is the number of edges in the affected subgraph.

Method	Arch	Description	Forward	Backward
Retrain	GCN, SAGE	Full retraining from scratch on the updated graph	$\mathcal{O}(\|E\|d)$	$\mathcal{O}(\|E\|d)$
Original Model	GCN, SAGE	Inference using the original model without adaptation	$\mathcal{O}(\|E\|d)$	–
IncreGNN (Di & et al., 2022)	GCN, SAGE	k-hop subgraph-based fine-tuning around affected nodes	$\mathcal{O}(\|E_s\|d)$	$\mathcal{O}(\|E_s\|d)$
EvolveGCN-O (Pareja et al., 2020)	GCN	LSTM-based evolution of GCN weight parameters	$\mathcal{O}(\|E\|d)$	$O(\|E\|d) + C_{\text{LSTM}}$
LoRA (Hu et al., 2022)	GCN, SAGE	Frozen weights with low-rank trainable updates	$\mathcal{O}(\|E\|d) + \mathcal{O}(\|V\|dr)$	$\mathcal{O}(\|E\|d) + \mathcal{O}(\|V\|dr)$
Cached Propagation (Ablation)	GCN, SAGE	Only cached propagation without selective optimization	$\mathcal{O}(\|V_{\text{aff}}\|d + \epsilon d)$	Full backward
DASS-GNN-C	GCN	Cached propagation with selective optimization	$\mathcal{O}(\|V_{\text{aff}}\|d + \epsilon d)$	$T_{\text{SVD}} + O(kd_{\text{out}})$
DASS-GNN-S	SAGE	Cached neighborhood aggregation with selective optimization	$\mathcal{O}(\|V_{\text{aff}}\|d + \epsilon d)$	$T_{\text{SVD}} + O(kd_{\text{out}})$

5 Results of DASS-GNN-C

5.1 Node Classification

Given a graph $G(V, E)$ with node features X, the objective of the node classification is to predict a class label for each node in the graph. The standard cross-entropy loss from the node classification literature (Kipf & Welling, 2017; Feng et al., 2020; Yu et al., 2023) is used.

Table 3: DASS-GNN-C results across 10 batches. R (Retrain), DC (DASS-GNN-C), I (IncreGNN (Di & et al., 2022)), L (LoRA (Hu et al., 2022)), and EO (EvolveGCN-O (Pareja et al., 2020)). Speedup (S) (×) and Metric Gain (AG%) are reported relative to Retrain (R).

Node Classification								Link Prediction							
D	**M**	**Acc (%)**	**AG (%)**	**Time (s)**	**S (×)**	**Epochs**	$\mathcal{D}_{\text{SVD}}$	**D**	**M**	**AUC (%)**	**AG (%)**	**Time (s)**	**S (×)**	**Epochs**	$\mathcal{D}_{\text{SVD}}$
	R	82.06 ± 1.36	–	0.12	1.00×	69.3	7.40		R	95.72 ± 0.66	–	0.38	1.00×	34.1	5.27
	I	82.73 ± 0.62	+0.67%	0.03	4.00×	**22.5**	**3.82**		I	95.84 ± 0.72	+0.12%	0.17	2.24×	15.6	0.53
G1	L	67.44 ± 1.59	-14.62%	0.06	2.00×	42.8	2.43	**G1**	L	96.43 ± 0.54	+0.71%	0.34	1.12×	29.2	0.46
	EO	50.81 ± 21.02	-31.25%	0.10	1.20×	19.6	13.5		EO	96.26 ± 1.30	+0.54%	0.16	2.38×	12.1	5.08
	DC	$\mathbf{82.50 \pm 0.89}$	**+0.44%**	**0.04**	**3.00×**	**17.5**	**2.11**		**DC**	$\mathbf{96.48 \pm 0.32}$	**+0.76%**	**0.28**	**1.36×**	**24.8**	**0.34**
	R	71.69 ± 1.03	–	0.11	1.00×	21.2	8.02		R	94.35 ± 0.87	–	0.41	1.00×	31.9	0.57
	I	68.25 ± 0.91	-3.44%	0.03	3.67×	20.5	5.71		I	96.12 ± 1.09	+1.77%	0.23	1.78×	17.6	0.55
G2	L	52.05 ± 1.88	-19.64%	0.03	3.67×	14.3	3.83	**G2**	L	96.61 ± 0.41	+2.26%	0.31	1.32×	23.3	0.13
	EO	28.49 ± 7.96	-43.20%	0.34	0.32×	14.5	29.99		EO	82.84 ± 5.93	-11.51%	0.60	0.68×	29.3	132.6
	DC	$\mathbf{70.08 \pm 0.79}$	**-1.61%**	**0.05**	**2.20×**	**19.6**	**3.78**		**DC**	96.36 ± 0.58	+2.01%	0.34	1.21×	25.9	0.27
	R	86.01 ± 0.26	–	0.21	1.00×	56.8	0.94		R	99.10 ± 0.27	–	4.35	1.00×	66.7	1.53
	I	86.08 ± 0.14	+0.07%	0.04	5.25×	13	0.62		I	99.47 ± 0.25	+0.37%	1.12	3.88×	17.2	0.57
G3	L	65.60 ± 1.26	-20.41%	0.11	1.91×	50.0	1.38	**G3**	L	99.26 ± 0.27	+0.16%	1.83	2.38×	27.6	0.56
	EO	82.62 ± 1.22	-3.39%	0.13	1.62×	16.3	10.9		EO	97.97 ± 0.40	-1.13%	1.56	2.79×	22.7	1.66
	DC	$\mathbf{86.20 \pm 0.11}$	**+0.19%**	**0.07**	**3.00×**	**11.0**	**0.54**		**DC**	$\mathbf{99.13 \pm 0.29}$	**+0.03%**	**2.28**	**1.91×**	**34.0**	**0.23**
	R	69.59 ± 0.52	–	3.14	1.00×	300.0	0.67		R	99.32 ± 0.03	–	95.12	1.00×	175.9	0.74
	I	69.55 ± 0.59	-0.04%	0.61	5.15×	50.0	9.23		I	99.33 ± 0.05	+0.01%	7.85	12.12×	14.5	0.39
G4	L	55.77 ± 1.25	-13.82%	0.66	4.76×	50.0	2.47	**G4**	L	99.30 ± 0.02	-0.02%	6.04	15.75×	11.0	0.02
	EO	63.43 ± 0.95	-6.16%	0.93	3.38×	37.6	23.56		EO	97.71 ± 0.33	-1.61%	12.43	7.65×	21.9	0.83
	DC	$\mathbf{69.69 \pm 0.54}$	**+0.10%**	**0.89**	**3.53×**	**43.8**	**0.61**		**DC**	$\mathbf{99.30 \pm 0.02}$	**-0.02%**	**6.34**	**15.00×**	**11.0**	**0.01**
	R	84.38 ± 0.02	–	349	1.00×	300.0	0.79		R	78.12 ± 0.31	–	2443	1.00×	436.0	0.48
	I	84.12 ± 0.59	-0.26%	30.0	11.63×	42.7	0.82		I	76.33 ± 0.05	-1.79%	462	5.29×	24.5	0.39
G5	L	75.77 ± 0.25	-8.61%	35.2	9.91×	50.0	1.02	**G5**	L	77.30 ± 0.02	-0.82%	824	2.96×	23.0	0.41
	EO	78.67 ± 0.09	-5.71%	37.3	9.36×	47.4	12.62		EO	76.71 ± 0.33	-1.41%	789	3.10×	21.9	0.78
	DC	$\mathbf{84.69 \pm 0.04}$	**+0.31%**	**31.0**	**11.26×**	**39.8**	**0.60**		**DC**	$\mathbf{78.20 \pm 0.03}$	**+0.08%**	**361**	**6.77×**	**21.0**	**0.29**

5.1.1 Evaluation Metrics

We evaluate predictive performance on node classification using accuracy, computational efficiency, and spectral deviation $\mathcal{D}_{\text{SVD(W)}} = \sum_{\ell=1}^{L} \left\| \sigma(W^{(\ell)}_{\mathcal{M}_{\text{original}}}) - \sigma(W^{(\ell)}_{\mathcal{M}_{\text{updated}}}) \right\|_2$, where $\mathcal{M}_{\text{original}}$ denotes the model trained on G_t, and $\mathcal{M}_{\text{updated}}$ denotes the model trained on G_{t+1}, which measures the L_2 drift of singular values between original and updated models.

5.1.2 Analysis of Results

The results given in Table. 3 indicate that DASS-GNN-C achieves the highest or competitive predictive performance across datasets while reducing adaptation time over retraining. Overall, the node classification results indicate that DASS-GNN-C can adapt to successive graph updates without requiring full retraining while achieving comparable accuracies and substantial speedups. As Fig. 2 (see Appendix B) shows, in most instances, the 3-tuple of (accuracy gain, speedup, spectral drift) for DASS-GNN-C is not dominated by the 3-tuples of the other baseline methods. This indicates that DASS-GNN-C offers a good trade-off with respect to accuracy, speedup, and spectral drift.

Table 4: DASS-GNN-C results over 10 batches. Accuracy/AUC-ROC is reported as mean ± standard deviation, $\mathcal{D}_{\text{SVD}}$ and Time are reported as averages. D: Dataset, R: Retrain, CA: Cached-AX, DC: DASS-GNN-C.

D	Node Classification									Link Prediction								
	Accuracy			$\mathcal{D}_{\text{SVD}}$			**Time (s)**			**AUC-ROC**			$\mathcal{D}_{\text{SVD}}$			**Time (s)**		
	R	**CA**	**DC**	**R**	**CA**	**DC**	**R**	**CA**	**DC**	**R**	**CA**	**DC**	**R**	**CA**	**DC**	**R**	**CA**	**DC**
G1	0.82 ± 0.02	0.82 ± 0.01	$\mathbf{0.83 \pm 0.01}$	7.40	3.01	**2.11**	0.12	**0.03**	**0.03**	0.96 ± 0.03	$\mathbf{0.96 \pm 0.01}$	$\mathbf{0.96 \pm 0.01}$	5.27	1.21	**0.34**	0.38	**0.18**	0.28
G2	$\mathbf{0.72 \pm 0.02}$	0.69 ± 0.03	$\mathbf{0.70 \pm 0.02}$	8.02	3.98	**3.78**	0.11	**0.03**	0.05	0.94 ± 0.01	0.96 ± 0.01	$\mathbf{0.96 \pm 0.01}$	0.57	0.54	**0.27**	0.41	**0.22**	0.34
G3	0.86 ± 0.26	0.86 ± 0.15	$\mathbf{0.86 \pm 0.11}$	0.94	0.76	**0.54**	0.21	**0.05**	0.07	0.99 ± 0.02	$\mathbf{0.99 \pm 0.01}$	0.99 ± 0.02	1.53	1.11	**0.23**	4.35	**1.12**	2.28
G4	0.70 ± 0.12	0.70 ± 0.13	$\mathbf{0.70 \pm 0.01}$	0.67	**0.48**	0.61	3.14	**0.36**	0.89	0.99 ± 0.03	0.99 ± 0.02	$\mathbf{0.99 \pm 0.01}$	0.74	0.33	**0.01**	95.12	7.54	**6.34**
G5	0.84 ± 0.00	0.84 ± 0.00	$\mathbf{0.85 \pm 0.01}$	0.79	0.68	**0.60**	349	**30**	31	0.78 ± 0.31	0.78 ± 0.12	$\mathbf{0.78 \pm 0.09}$	0.48	0.30	**0.29**	2443	364	**361**

5.1.3 Ablation Study

The results in Table 4 demonstrate that applying cached-AX propagation and fine-tuning the model produces better predictive accuracy with maximum speedups against retraining. Further, adding selective optimization to the cached-AX yields the best predictive accuracy with slightly lower speedups (due to SVD computations in the backward pass) compared to Cached-AX. Retraining possesses fluctuating spectral deviations ($\mathcal{D}_{\text{SVD}}$), as it starts with random initial parameters every time, whereas the proposed Cached-AX and DASS-GNN-C possess consistent spectral deviations, with DASS-GNN-C method being best in most cases.

5.1.4 Sensitivity to Batch Sizes and k

We vary batch size as $\{1\%, 2\%, 3\%, 5\%, 10\%\}$ of the total edges. From our experiments, we note that DASS-GNN-C maintains relatively stable accuracy across the evaluated batch sizes while keeping $\mathcal{D}_{\text{SVD}}$ controlled, demonstrating that the selective update mechanism can accommodate increasingly larger graph changes without requiring full retraining. Fig 6 (see Appendix C) shows that

Table 5: DASS-GNN-S results across 10 batches. R (Retrain), DS (DASS-GNN-S), I (IncreGNN), and L (LoRA). Speedup (S) (×) and Metric Gain (AG%) are reported relative to Retrain (R).

Node Classification								Link Prediction							
D	**M**	**Acc (%)**	**AG (%)**	**Time (s)**	**S (×)**	**Epochs**	$\mathcal{D}_{\text{SVD}}$	**D**	**M**	**AUC (%)**	**AG (%)**	**Time (s)**	**S (×)**	**Epochs**	$\mathcal{D}_{\text{SVD}}$
G1	R	83.55 ± 0.67	–	0.06	1.00×	33	35.85	**G1**	R	91.80 ± 1.20	–	2.14	1.00×	120.0	0.45
	I	83.86 ± 1.07	+0.30%	0.04	1.55×	22.3	11.21		I	92.15 ± 1.10	+0.38%	1.23	1.74×	65.0	0.32
	L	84.27 ± 1.46	+0.72%	0.05	1.26×	21.8	16.80		L	90.50 ± 1.45	−1.42%	1.52	1.41×	80.0	0.35
	DS	$\mathbf{85.12 \pm 1.45}$	**+1.57%**	**0.05**	**1.2×**	**30.3**	**19.47**		**DS**	$\mathbf{93.42 \pm 0.95}$	**+1.76%**	**0.85**	**2.52×**	**45.0**	**0.28**
G2	R	72.79 ± 0.92	–	0.06	1.00×	31.0	30.89	**G2**	R	88.40 ± 2.10	–	4.52	1.00×	200.0	0.62
	I	77.50 ± 1.63	+4.71%	0.03	2×	12.9	6.33		I	89.95 ± 1.80	+1.75%	2.21	2.05×	95.0	0.41
	L	72.14 ± 0.65	-0.65%	0.05	1.2×	21.3	5.01		L	87.10 ± 2.30	−1.47%	3.05	1.48×	130.0	0.64
	DS	72.91 ± 0.56	+0.12%	0.05	1.2×	14.2	10.74		**DS**	$\mathbf{89.50 \pm 1.65}$	**+1.24%**	**1.64**	**2.76×**	**70.0**	**0.38**
G3	R	88.14 ± 0.29	–	0.93	1.00×	44.1	63.67	**G3**	R	85.60 ± 1.85	–	18.90	1.00×	110.0	0.58
	I	88.35 ± 0.31	+0.20%	0.43	2.16×	19.4	32.24		I	86.20 ± 1.60	+0.70%	9.10	2.08×	50.0	0.35
	L	88.01 ± 0.19	-0.13%	0.56	1.66×	19.7	6.56		L	84.10 ± 2.10	−1.75%	12.80	1.48×	70.0	0.63
	DS	$\mathbf{88.37 \pm 0.25}$	**+0.23%**	**0.56**	**1.66×**	**18.5**	**21.69**		**DS**	$\mathbf{87.35 \pm 1.20}$	**+2.04%**	**9.80**	**1.93×**	**42.0**	**0.31**
G4	R	69.02 ± 0.65	–	38.87	1.00×	299.3	52.18	**G4**	R	79.20 ± 2.40	–	112.50	1.00×	350.0	0.85
	I	71.74 ± 0.35	+2.72%	5.41	7.18×	30.0	8.68		I	81.50 ± 2.10	+2.90%	48.20	2.33×	140.0	0.52
	L	68.87 ± 0.57	-0.15%	5.16	7.53×	21.0	5.09		L	78.00 ± 2.60	−1.52%	65.10	1.73×	190.0	0.67
	DS	$\mathbf{71.78 \pm 0.28}$	**+2.76%**	**2.44**	**15.94×**	**19.7**	**5.03**		**DS**	$\mathbf{81.10 \pm 1.90}$	**+2.40%**	**28.60**	**3.93×**	**85.0**	**0.44**
G5	R	85.07 ± 0.35	–	1002	1.00×	343	42.18	**G5**	R	74.32 ± 0.40	–	3256.0	1.00×	378.2	1.63
	I	84.74 ± 0.23	-0.33%	95.41	10.50×	42.5	18.82		I	71.50 ± 1.02	−3.79%	386.1	8.43×	128.0	0.65
	L	83.87 ± 0.73	-1.20%	102.6	9.77×	37.2	14.26		L	70.26 ± 2.45	−5.46%	678.2	4.80×	104.0	0.46
	DS	$\mathbf{85.81 \pm 0.36}$	**+0.74%**	**86.21**	**11.62×**	**26.5**	**15.20**		**DS**	$\mathbf{76.21 \pm 0.49}$	**+2.54%**	**314.9**	**10.34×**	**65.0**	**0.25**

Table 6: DASS-GNN-S results over 10 batches. Accuracy/AUC is reported as mean ± standard deviation , $\mathcal{D}_{\text{SVD}}$ and Time are reported as averages . R: Retrain, CS: Cached-SAGE, DS: DASS-GNN-S.

D	**Node Classification**									**Link Prediction**								
	Accuracy			$\mathcal{D}_{\text{SVD}}$			**Time (s)**			**AUC**			$\mathcal{D}_{\text{SVD}}$			**Time (s)**		
	R	**CS**	**DS**	**R**	**CS**	**DS**	**R**	**CS**	**DS**	**R**	**CS**	**DS**	**R**	**CS**	**DS**	**R**	**CS**	**DS**
G1	0.84 ± 0.06	0.85 ± 0.07	$\mathbf{0.85 \pm 0.02}$	35.85	25.36	19.47	0.06	0.05	**0.05**	0.91 ± 0.05	0.92 ± 0.22	$\mathbf{0.93 \pm 0.31}$	0.45	0.47	**0.28**	2.14	**0.76**	0.85
G2	0.72 ± 0.01	0.73 ± 0.01	$\mathbf{0.73 \pm 0.01}$	30.89	17.84	**10.74**	0.06	**0.04**	0.05	0.88 ± 0.01	0.89 ± 0.03	$\mathbf{0.89 \pm 0.01}$	0.62	0.52	**0.38**	4.52	**1.24**	1.64
G3	0.88 ± 0.01	0.88 ± 0.00	$\mathbf{0.88 \pm 0.00}$	63.67	38.64	21.69	0.95	**0.43**	0.56	0.85 ± 0.04	0.86 ± 0.01	$\mathbf{0.87 \pm 0.01}$	0.58	0.47	0.31	18.9	**9.12**	9.80
G4	0.69 ± 0.01	0.70 ± 0.01	$\mathbf{0.71 \pm 0.01}$	52.18	0.92	**5.03**	38.87	**2.04**	2.44	0.79 ± 0.02	0.81 ± 0.05	$\mathbf{0.81 \pm 0.02}$	0.85	0.67	**0.44**	112.5	**27.4**	28.6
G5	0.85 ± 0.03	0.85 ± 0.01	$\mathbf{0.86 \pm 0.01}$	42.18	32.5	**15.2**	1002	95.3	**86.21**	0.74 ± 0.03	0.75 ± 0.05	$\mathbf{0.76 \pm 0.02}$	1.63	0.93	**0.25**	3256	377.2	**314.9**

as the batch size increases, larger structural changes are introduced into the graph, resulting in a gradual change in predictive performance and spectral deviation.

We consider $k \in \{10\%, 20\%, 40\%, 60\%, 80\%\}$ rows based on row energy scores. The results in Fig 7 (in Appendix C) show that accuracy remains relatively stable across the different selection ratios, whereas computation time and spectral drift are more sensitive to k. Increasing k allows a larger portion of the model parameters to adapt, leading to higher $\mathcal{D}_{\text{SVD}}$ and computation. Thus, a smaller k provides stronger parameter selectivity and spectral stability, while larger k enables broader adaptation at higher spectral drift and additional computational cost.

5.2 Link Prediction

Given a graph $G(V, E)$ and node embeddings h_u and h_v, the link prediction objective is to predict edge formation between nodes u and v. The edge probability is computed as $p(u, v) = \sigma(h_u^T h_v)$, where $\sigma(\cdot)$ denotes the sigmoid function. The standard binary cross-entropy loss from the literature (Zhang & Chen, 2018; Rossi et al., 2020; Hamilton et al., 2017) is used in training/adaptation. We evaluate link prediction using *AUC-ROC*, *time*, and $\mathcal{D}_{\text{SVD}}$ as defined in Section 5.1.1.

5.2.1 Analysis of Results

Table 3 demonstrates that DASS-GNN-C maintains close AUC-ROC, good speedup, and low spectral drift across datasets. Fig 3 (see Appendix B) shows that DASS-GNN-C exhibits favorable trade-off among 3-tuple (accuracy gain, speedup, and spectral drift) compared to other baselines Overall, the results support the effectiveness of combining propagation caching with selective optimization for efficient and stable dynamic link prediction.

5.2.2 Ablation Study

Table 4 shows that cached-AX provides the main computational speedup over retraining across datasets. Adding SVD-guided selective optimization further reduces spectral drift in most cases while maintaining comparable AUC-ROC, with a small runtime overhead. These results indicate that propagation caching primarily improves efficiency, whereas selective optimization provides control over parameter adaptation and spectral stability additionally.

5.2.3 Sensitivity to batch sizes and k

Our experiments over varying batch sizes (in Section 5.1.4) indicate that across the evaluated batch sizes, DASS-GNN-C maintains competitive predictive performance with substantially lower adap-

tation time than retraining. Figure 8 (see Appendix C) shows that AUC generally decreases as the graph-update batch size increases, reflecting the larger structural perturbations introduced by larger updates. Varying k as in Section 5.1.4, the results shown in Fig. 7 indicate that increasing the selected top-k fraction reduces parameter selectivity and generally increases spectral drift and predictive performance marginally except for Cora dataset. Across the top $k\%$ rows, a marginal computational overhead is introduced due to SVD computation in DASS-GNN.

6 Results of DASS-GNN-S

6.1 Node classification (Similar to Section 5.1)

6.1.1 Analysis of Results

Table 6 shows that DASS-GNN-S maintains competitive accuracy while reducing adaptation cost across datasets. This indicates that the main computational benefit comes from propagation caching, whereas SVD-guided selective adaptation mainly provides additional control over parameter drift. Further, Fig 4 (see Appendix B) illustrates the trade-off among 3-tuple (accuracy gain, speedup, and spectral drift), where DASS-GNN-S achieves favorable trade-off among them over other baselines.

6.1.2 Ablation Study

Table 6 demonstrates that Cached neighborhood aggregation provides the main reduction in adaptation time, while adding SVD-guided optimization generally reduces spectral drift with small change in accuracy. DASS-GNN-S therefore combines the computational benefit of cached neighborhood aggregation with additional control over parameter drift.

6.1.3 Sensitivity to Batch Sizes and k

Varying batch sizes as in Section 5.1.4, from our experiments we note that DASS-GNN-S remains competitive while achieving a substantial computational speedup over retraining. Fig. 9 (See Appendix C) demonstrates that an increase in the batch size generally reduces predictive accuracy as the graph undergoes larger topological changes. Varying k as in Section 5.1.4, Fig. 10 (Appendix C) demonstrates an increase in the selected fraction of rows reduces the selectivity of DASS-GNN-S and generally increases spectral drift. Therefore, a smaller dominant subspace provides more controlled adaptation, while larger k offers more adaptation flexibility but increased spectral drift.

6.2 Link prediction (Similar to Section 5.2)

6.2.1 Analysis of Results

Table 6 shows that DASS-GNN-S maintains competitive AUC while substantially reducing adaptation time. Further, Fig 5(see Appendix B) demonstrates the trade-off among 3-tuple (AUC gain, speedup and spectral drift) favorably achieved by DASS-GNN-S over other baselines.

6.2.2 Ablation Study

Table 6 shows that cached neighborhood aggregation provides the main computational speedups, while SVD-guided optimization further controls spectral drift. Overall, the predictive performance of DASS-GNN-S is maintained close to Cached-SAGE, with a small improvement on the Citeseer dataset. Thus, SVD-based selective optimization improves stability without sacrificing the efficiency obtained through propagation caching.

6.2.3 Sensitivity to Batch Sizes and k

Over varying batch sizes as in Section 5.1.4, our experiments indicate that DASS-GNN-S remains competitive while retaining substantially lower runtime than retraining. Fig. 11 (see Appendix C) shows that AUC generally decreases as the update batch size increases, reflecting the greater difficulty of adapting to larger structural changes. Varying k as in Section 5.1.4, Fig. 10 shows that increasing k reduces selectivity and increases spectral deviation, while AUC remains relatively stable over the evaluated range. This suggests that a relatively small dominant subspace is sufficient for effective SAGE adaptation.

7 Conclusion

In this paper, we proposed a dynamic GNN framework, DASS-GNN, that addresses two computational bottlenecks in evolving graphs: repeated feature propagation in the forward pass and unnecessary optimization of parameters in the backward pass. In future, we plan to extend this framework for evolving heterogeneous graphs.

REFERENCES

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, 2018.

Shimin Di and et al. Incregnn: Incremental graph neural networks for dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 22092–22103, 2020.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Federico Monti, and Michael Bronstein. Sign: Scalable inception graph neural networks. In *ICML Workshop on Graph Representation Learning*, 2020.

Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

Will Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1024–1034, 2017.

Yizeng Han, Yibing Shen, Zhihang Liu, Yifan Wang, Yue Dong, Shiji Song, and Gao Huang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Weihua Hu, Matthias Fey, Jure Leskovec, and et al. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:22118–22133, 2020a.

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. In *Advances in Neural Information Processing Systems*, volume 33, 2020b.

Seyed Mehran Kazemi, Rishab Goel, Krikamani Jain, Parisa Kordjamshidi, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research (JMLR)*, 21(70):1–73, 2020.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. NIPS Workshop on Bayesian Deep Learning.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Ron Levie, Elvin Isufi, Gitta Kutyniok, and Andreas Loukas. Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research (JMLR)*, 22(272):1–59, 2021.

Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1105–1114, 2016.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Hiroki Toyoda, Robert Salz, Shinya Joshi, and Vivek Patel. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 5363–5370, 2020.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML Workshop on Graph Representation Learning*, 2020.

Aravind Sankar, Yanhua Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*, pp. 519–527, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Felix Wu, Tianyi Zhang, Amauri H. Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pp. 6861–6871, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Yuan Xie and et al. Dynamic graph neural networks: A survey. *Frontiers of Computer Science*, 2024.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhilin Yang, William W. Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 40–48. PMLR, 2016.

Le Yu, Leilei Sun, Bowen Du, Tongyu Zhu, and Weifeng Lv. Label-enhanced graph neural network for semi-supervised node classification. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11802–11815, 2023.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5165–5175, 2018.

A Omitted Proofs

A.1 Proof of Theorem 1

Proof. Given the normalized adjacency matrix $\hat{A}_t = D_t^{-1/2}(A_t + I)D_t^{-1/2}$, adding or removing an edge changes only the corresponding entries of A and the degrees of the endpoints.

Therefore, only the corresponding rows and columns of affected nodes V_{aff} will have modified normalization coefficients after the batch update. For all other nodes whose degree and neighborhood remain unaffected, the corresponding entries of the normalized adjacency matrix remain the same, i.e., $\hat{A}_{i,:}^{t+1} = \hat{A}_{i,:}^{t}$.

Therefore, only rows and columns associated with the affected nodes have modified normalization coefficients. For every node whose neighborhood and degree remain unchanged,$\hat{A}_{i,:}^{t+1} = \hat{A}_{i,:}^{t}$. Hence, $(\hat{A}_{t+1}X)_i = (\hat{A}_t X)_i$.

Thus, all the propagated features outside the affected local neighborhood remain the same and can be reused from the cache while recomputing propagated features for the locally affected rows. □

A.2 Proof of Theorem 2

Proof. Given propagation matrices P_{t+1} and P_t, we can observe that the difference in the propagation matrices is $P_{t+1} - P_t = (\hat{A}_{t+1} - \hat{A}_t)X$.

The Frobenius norm of the difference can be written as, $\|P_{t+1} - P_t\|_F = \|(\hat{A}_{t+1} - \hat{A}_t)X\|_F$.

Then, application of the sub-multiplicative matrix norm inequality (i.e., $\|AB\|_F \leq \|A\|_F \|B\|_F$) on that difference results in the following:

$$\|P_{t+1} - P_t\|_F \leq \|\hat{A}_{t+1} - \hat{A}_t\|_F \, \|X\|_F$$

Hence, the following is proved. $\boxed{\|P_{t+1} - P_t\|_F \leq \|\hat{A}_{t+1} - \hat{A}_t\|_F \, \|X\|_F}$ □

A.3 Proof of Theorem 3

Proof. Let $g_i = \nabla W_{i,:}^{(\ell)}$ denote the i-th row vector of the weight gradient matrix $\nabla W^{(\ell)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. The total squared Frobenius norm of the gradient matrix is given by $\|\nabla W^{(\ell)}\|_F^2 = \sum_{i=1}^{d_{\text{out}}} \|g_i\|_2^2$.

Under SVD-guided row selection, rows $i \notin \mathcal{I}$ are masked out to zero, yielding the residual norm $\|\nabla W^{(\ell)} - \hat{\nabla} W^{(\ell)}\|_F^2 = \|\nabla W^{(\ell)}\|_F^2 - \sum_{i \in \mathcal{I}} \|g_i\|_2^2$.

The row energy scores $s_i = \sum_{j=1}^{r} (U_r)_{ij}^2$ represent the row-wise projection density onto the top-r left singular subspace U_r, satisfying $\sum_{i=1}^{d_{\text{out}}} s_i = r$. By selecting indices $\mathcal{I} = \text{TopK}(\{s_i\}, k)$, the selection preserves a fraction $\rho_{\text{spec}} = \frac{1}{r} \sum_{i \in \mathcal{I}} s_i \in (0, 1]$ of the dominant spectral energy.

Assuming the gradient distribution aligns with the principal directional energy of $W^{(\ell)}$, the retained gradient energy satisfies $\sum_{i \in \mathcal{I}} \|g_i\|_2^2 \geq \rho \|\nabla W^{(\ell)}\|_F^2$. Substituting this lower bound into above mentioned residual norm yields $\|\nabla W^{(\ell)} - \hat{\nabla} W^{(\ell)}\|_F^2 \leq (1 - \rho)\|\nabla W^{(\ell)}\|_F^2$, which establishes the bounded energy preservation guarantee of the truncated singular subspace for row-wise gradient masking. □

A.4 Proof of Theorem 4

Proof. Consider the parameter update step at iteration T under Algorithm 2 as $W_{T+1}^{(\ell)} = W_T^{(\ell)} - \eta \widetilde{\nabla W}_T^{(\ell)}$ which is equal to $W_T^{(\ell)} - \eta(M_T \odot \nabla W_T^{(\ell)})$. Accumulating updates ($\left\| \sum_{T=1}^{T_e} \eta \widetilde{\nabla W}_T^{(\ell)} \right\|_F$) across T_e fine-tuning iterations within dynamic step t $\|W_t^{(\ell)} - W_{t-1}^{(\ell)}\|_F \leq \eta \sum_{T=1}^{T_e} \|\widetilde{\nabla W}_T^{(\ell)}\|_F$.

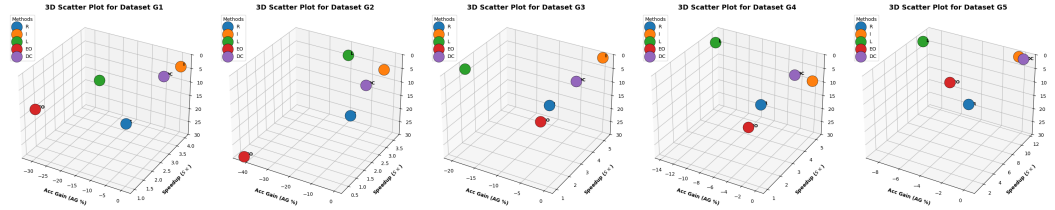


Figure 2: DASS-GNN-C Node classification

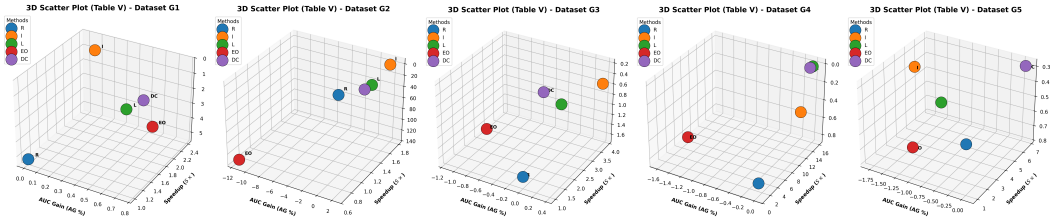


Figure 3: DASS-GNN-C Link Prediction

Applying Theorem 3, the masked gradient satisfies $\|\widetilde{\nabla W}_T^{(\ell)}\|_F \leq \sqrt{\rho_{\text{param}}}\|\nabla W_T^{(\ell)}\|_F$. Substituting $\|\nabla W_T^{(\ell)}\|_F \leq G$ gives $\|W_t^{(\ell)} - W_{t-1}^{(\ell)}\|_F \leq \eta \cdot T_e \cdot \sqrt{\rho_{\text{param}}} \cdot G$. Applying the triangle inequality relative to initial model $W_0^{(\ell)}$ yields $\|W_t^{(\ell)} - W_0^{(\ell)}\|_F \leq \|W_{t-1}^{(\ell)} - W_0^{(\ell)}\|_F + \eta \cdot T_e \cdot \sqrt{\rho_{\text{param}}} \cdot G$. Since $T_e \ll T_{\text{retrain}}$, the cumulative parameter drift remains strictly bounded compared to full retraining.

□

A.5 Datasets diversity

Cora and Citeseer are relatively small citation networks with sparse neighborhood structures and high update sensitivity, where localized edge modifications can substantially affect message propagation and learned representations. In particular, Citeseer exhibits higher sparsity and weaker structural connectivity, making it highly sensitive to incremental graph changes. PubMed represents a moderately larger and more stable homophilic graph, where perturbations tend to have smoother effects on neighborhood aggregation. The OGB datasets are obtained from the *Open Graph Benchmark (OGB)* (Hu et al., 2020a). The ogbn-products dataset is used to evaluate scalability on a large product co-purchasing network with dynamic graph perturbations. In contrast, ogbn-arxiv is a large-scale evolving citation network with comparatively lower homophily and more complex propagation dynamics, making dynamic adaptation more challenging. These variations motivate the need for computationally efficient yet stable dynamic GNN update mechanisms capable of handling diverse graph characteristics across different scales and structural regimes.

B Trade-off Scatter Plots

C Sensitivity Analysis

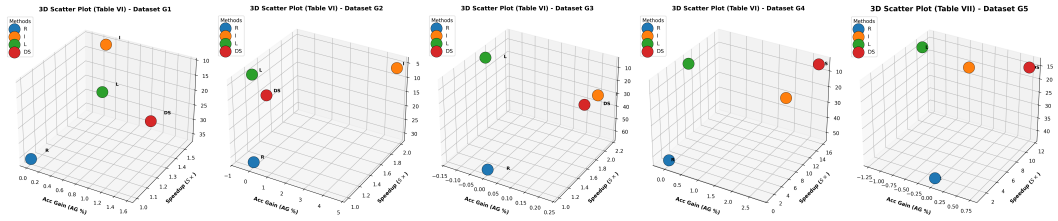


Figure 4: DASS-GNN-S Node Classification

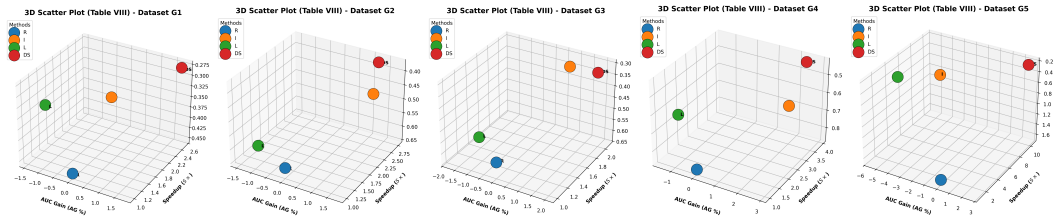


Figure 5: DASS-GNN-S Link Prediction

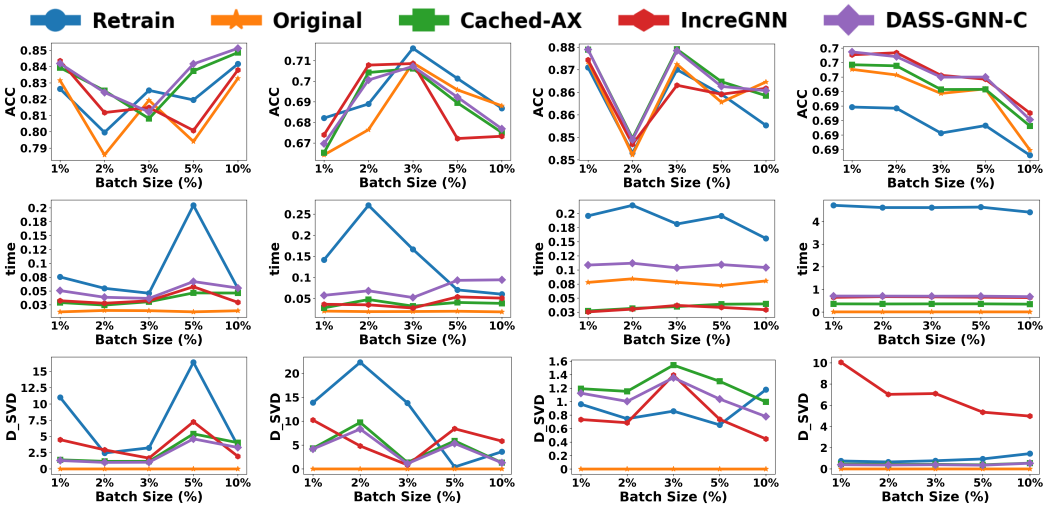
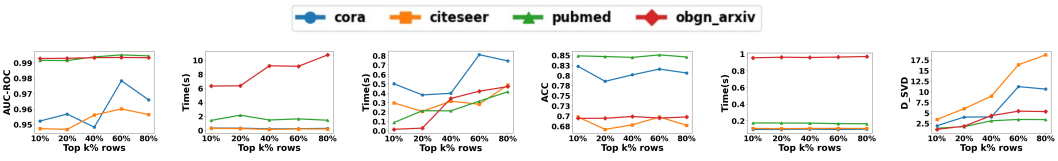


Figure 6: Node classification results over different batch sizes across datasets (left-to-right: Cora, Citeseer, Pubmed, and Ogbn-arxiv).



(a) LP AUC (b) LP Time (c) LP $\mathcal{D}_{\text{SVD}}$ (d) NC Acc (e) NC Time (f) NC $\mathcal{D}_{\text{SVD}}$

Figure 7: DASS-GNN-C sensitivity to $k\%$ for link prediction (a–c) and node classification (d–f).

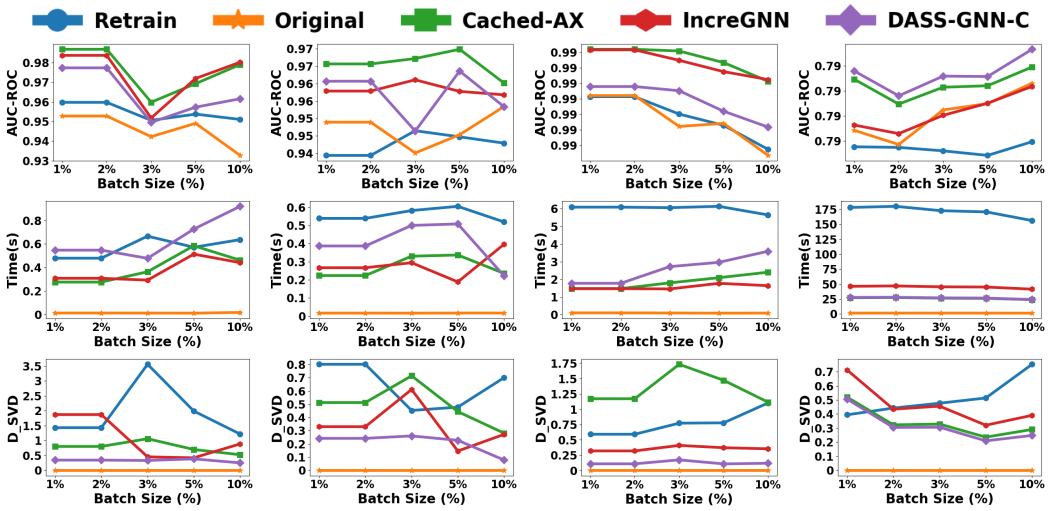


Figure 8: Link prediction results over different batch sizes on four datasets (left-to-right: Cora, Citeseer, Pubmed, and Ogbn-arxiv).

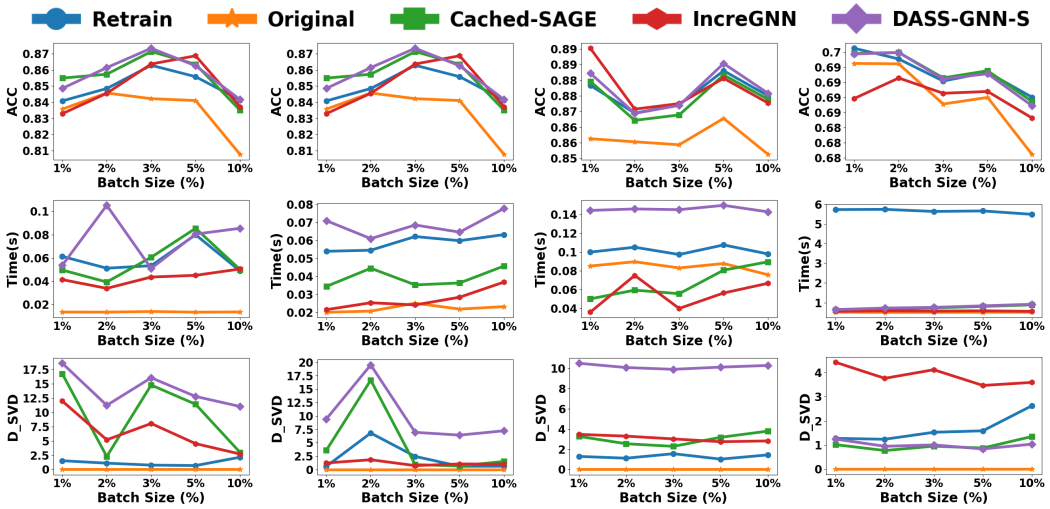
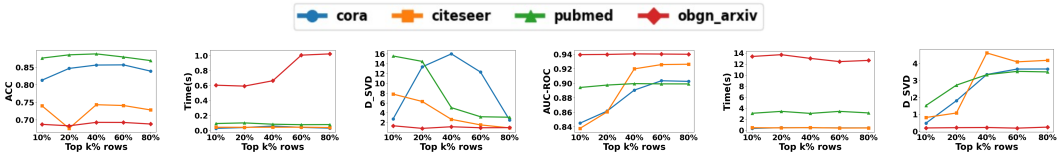


Figure 9: SAGE node classification results across batch sizes and datasets (left-to-right: Cora, Citeseer, Pubmed, and Ogbn-arxiv).



(a) NC Acc (b) NC Time (c) NC $\mathcal{D}_{\text{SVD}}$ (d) LP AUC (e) LP Time (f) LP $\mathcal{D}_{\text{SVD}}$

Figure 10: DASS-GNN-S sensitivity to $k\%$ across node classification (a–c) and link prediction (d–f) tasks.

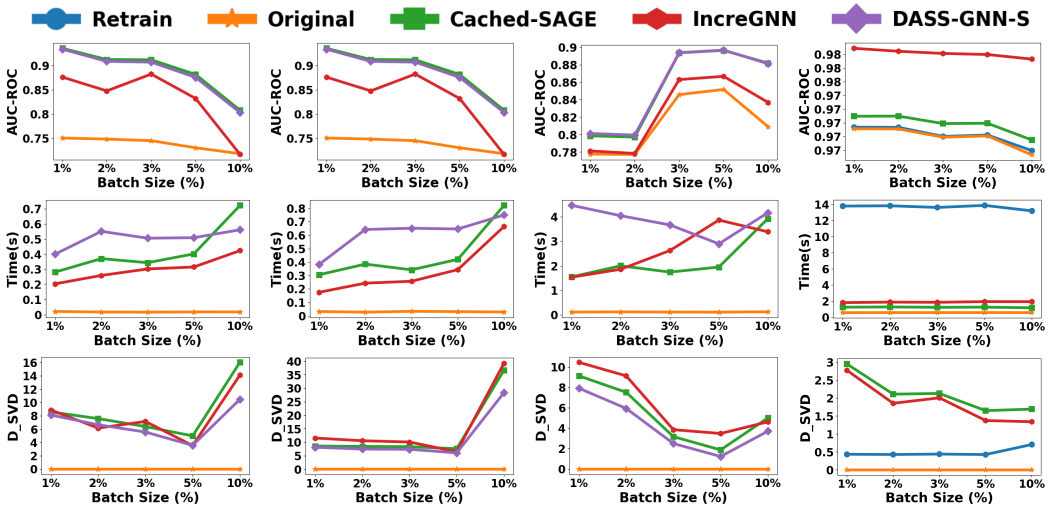


Figure 11: SAGE link prediction results over across batch sizes and datasets (left-to-right: Cora, Citeseer, Pubmed, and Ogbn-arxiv).