Trimming the Fat: Redundancy-Aware Acceleration Framework for DGNNs

Renhong Huang[1, 2], Yuxuan Cao[1], Yi Li[1], Junwei Hu[1], Zihua Xiong[2], Shuai Fang[2], Sheng Guo[2], Bo Zheng[2], Yang Yang[1]*

[1]Zhejiang University,
[2]MyBank, AntGroup
{renh2, caoyx, ly_future, graf_spee, yangya}@zju.edu.cn
{xiongzihua.xzh, fangshuai.fangshua, guosheng.guosheng, guangyuan}@mybank.cn

Abstract

Temporal graphs are essential for modeling complex real-world systems, such as social interactions, financial transactions, and recommendation systems, but the high computational cost and model complexity of dynamic graph neural networks (DGNNs) pose significant challenges for practical deployment. Although various pruning and sampling techniques have proven effective in accelerating static GNNs, they fall short in dynamic settings due to temporal dependencies in evolving graph structures. To address these challenges, we propose TrimDG, a general framework that accelerates DGNNs by eliminating both static and runtime redundancies. For static redundancy, we introduce a novel node influence metric, Temporal Personalized PageRank (TPP), to prune less informative nodes, and employ temporal binning to remove redundant events. For runtime redundancy during training, we develop an adaptive sampling strategy guided by graph information bottleneck and further reduce sampling frequency through temporal batch selector and sampling cache. Theoretical analysis supports our design, and experiments on real-world datasets show that TrimDG reduces runtime by an average of 83.49% across diverse DGNN backbones while maintaining strong predictive performance, demonstrating both its efficiency and generalizability.

1 Introduction

Graphs are fundamental data structures, and dynamic graphs are prevalent in real-world scenarios, where both graph topology and node or edge attributes evolve over time (Yang, Chatelain, and Adam 2024). Owing to their ability to capture temporal dynamics, dynamic graphs are widely applied across domains, including social networks (Zhou et al. 2025), transportation systems (Wang et al. 2025), and finance (Khodabandehlou and Golpayegani 2024). To model these dynamics, Dynamic Graph Neural Networks (DGNNs) have been proposed to update node embeddings online via time-stamped events (Rossi et al. 2020; Kumar, Zhang, and Leskovec 2019; Trivedi et al. 2022), capturing both structural and temporal patterns using mechanisms such as temporal attention and memory modules (Rossi et al. 2020; Xu et al. 2020).

Despite their effectiveness, DGNNs face significant scalability challenges when applied to large scale dynamic graphs (Wang et al. 2021), which often involve millions of nodes and billions of daily timestamped edges. For instance, Alipay's platform serves 1.43 billion users and processes approximately 49 billion transactions per day. We attribute these challenges to two key factors. Firstly, the *neighbor explosion problem*: as events accumulate, a node's temporal neighborhood grows unbounded and becomes increasingly redundant, making naïve sampling computationally heavy and memory intensive (Xu et al. 2020; Rossi et al. 2020). Secondly, most existing DGNNs reprocess the full historical event trace for each node to sample and aggregate relevant events (Rossi et al. 2020), which is highly inefficient at scale. Together, these issues hinder the deployment of DGNNs on real-world, industrial-scale dynamic graphs. Addressing them is critical to enabling scalable and practical DGNN systems.

Prior studies have explored various strategies to improve the efficiency of GNNs on large scale graphs. In static settings, techniques such as graph pruning (Rong et al. 2019; Zheng et al. 2020) and graph sampling (Chen, Ma, and Xiao 2018; Zeng et al. 2019; Sun et al. 2022) have been proposed to alleviate the computational burden by reducing the size of the input graph. While these methods mitigate issues like neighbor explosion and redundancy, they often ignore and fail to handle the temporal order of events. Furthermore, in a dynamic context, such time-agnostic repeated sampling can introduce temporal biases or sample historically irrelevant nodes, thereby diluting the crucial information from noise.

To tackle these challenges in temporal settings, recent efforts (Zhou et al. 2022; Li et al. 2023a,b) explored incorporating time-awareness into sampling strategies for dynamic graph learning. These methods typically aim to capture temporal dynamics via heuristic sampling, yet they largely overlook redundancy in temporal graphs, both in terms of static structure and dynamic interactions. In particular, the matter of how to systematically identify and eliminate redundant information, while ensuring that the selected subgraphs remain informative and causally consistent, remains a largely open problem. Effectively addressing this challenge requires a principled framework that balances computational efficiency and temporal fidelity, which existing approaches have yet to achieve.

To this end, we present TrimDG, a modular framework designed to accelerate DGNNs by systematically removing both static and runtime redundancy. For static redundancy,

*Corresponding author.

we propose a novel dynamic node influence metric, Temporal Personalized PageRank (TPP), to identify and prune less informative nodes, and introduce temporal binning to eliminate repetitive or uninformative events. For runtime redundancy during training, we develop an adaptive sampling strategy guided by graph information bottleneck, and further reduce sampling frequency through a temporal batch selector and a lightweight sampling cache mechanism. Our framework is compatible with a wide range of DGNN architectures and preserves the integrity of the input data. We also provide theoretical analysis to support the design of each component. Extensive experiments on real-world datasets demonstrate that TrimDG reduces average training time by 83.49% across diverse DGNN backbones, without sacrificing predictive performance. These results highlight the efficiency, scalability, and broad applicability of our approach.

2 Preliminary

Continuous Time Dynamic Graph. A continuous time dynamic graph (CTDG) models events in a streaming setting, where events arrive sequentially with timestamps. Each event is denoted as $\delta(t) = (v_i, v_j, \mathbf{x}_{ij}, t)$, representing an event from source node v_i to node v_j at time t, with $\mathbf{x}_{ij} \in \mathbb{R}^d$ being the edge feature vector of dimension d. The set of all n events in chronological order is:

$$\mathcal{E} = \{\delta(t_1), \delta(t_2), \ldots, \delta(t_n)\},$$

where $0 \le t_1 \le t_2 \le \cdots \le t_n \le T$. A CTDG is thus defined as $G = (V, \mathcal{E})$, where V is the set of nodes involved in the temporal events.

The objective of representation learning on CTDGs is to compute time-aware node embeddings based on historical events. Given a node v at time t, let $\mathcal{H}_v(t) = \{(u, v, \mathbf{x}_{uv}, t') \mid t' < t\}$ denote the history of all events involving v prior to t. The objective is to learn the temporal encoding function:

$$f_\theta : (v, t, \mathcal{H}_v(t)) \mapsto h_v(t),$$

where $h_v(t) \in \mathbb{R}^d$ is the embedding of node v at t. The learned embedding are evaluated on two downstream tasks: *(i) Dynamic link prediction:* predicting the existence or likelihood of an edge between u and v at time t; *(ii) Dynamic node classification:* inferring the label or state of a node $w \in \{u, v\}$ at time t. In this work, we primarily focus on dynamic node classification.

Dynamic Graph Neural Networks. DGNNs are tailored to process CTDGs by modeling both temporal dynamics and structural relationships. For node v at time t, DGNNs typically generate node embeddings through the following steps:

(1) Neighbor Sampling. At each timestamp t, the model samples a subset of historical neighbors $\mathcal{N}_v(t) \subseteq \mathcal{H}_v(t)$. Sampling strategies select neighbors based on temporal criteria, typically using recent sampling (most recent events) or uniform sampling (events sampled uniformly over time). By repeatedly sampling neighbors across timestamps, the model constructs a subgraph for each target node, which is then used for subsequent embedding updates.

(2) Time Encoding. To capture temporal dynamics, DGNNs encode time difference $\Delta t = t - t_u$ for each neighbor $u \in \mathcal{N}_v(t)$ into a temporal embedding. Common encoding schemes include positional encoding, *e.g.*, $[\cos(\omega\Delta t), \sin(\omega\Delta t)]$ with learnable frequency ω, or learnable time embeddings that directly map Δt to a trainable vector.

(3) Aggregating and Updating. To capture temporal and structural dependencies, node embeddings are updated by aggregating information from the sampled subgraph formed. A common approach is attention based aggregation where each node in the sampled graph is assigned an attention weight based on its relevance to the target node v at time t:

$$\alpha_{uv} = \text{softmax}\left(\text{score}(h_u(t_u), h_v(t), \Delta t)\right),$$

where score$(\cdot)$ is a relevance function (*e.g.*, dot product or MLP), and $\Delta t = t - t_u$. The updated embedding is $h_v(t) = \sum_{u \in \mathcal{N}_v(t)} \alpha_{uv} m_u(t)$, with $m_u(t)$ being the encoded message from subgraph. Alternatively, RNN-based methods treat events as sequences and update node states recurrently:

$$h_v(t) = \text{RNN}(\hat{h}_v(t), \text{input}(t)),$$

with input(t) summarizing neighbor, edge and time features, and $\hat{h}_v(t)$ denoting the previous hidden embedding of node.

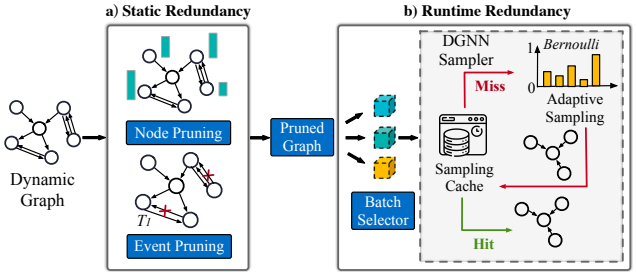


Figure 1: The overall framework of TrimDG. For static redundancy, TrimDG uses the Temporal Personalized PageRank (TPP) metric to prune less informative nodes and applies temporal binning to remove redundant events. For runtime redundancy, it employs an adaptive sampling strategy to reduce sampled subgraph sizes and further decreases sampling frequency via temporal batch selector and sampling cache.

3 Methodology

To improve the efficiency of dynamic graph training, we propose TrimDG to systematically mitigate redundancies across different stages. As illustrated in Figure 1, we identify and categorize two major types of redundancy in dynamic graphs: (1) Static redundancy (§ 3.1), and (2) Runtime redundancy (§ 3.2). Further, we analyze the complexity of the TrimDG framework (§ 3.3).

3.1 Static Redundancy

Graphs often contain redundant information, such as duplicate edges and noisy signals (Jin et al. 2020; Liu et al. 2022). This phenomenon has been extensively studied in the context of static graphs. Several works (Jin et al. 2022; Zhang et al. 2021; Rong et al. 2019; Xu et al. 2023; Cao et al. 2025; Huang et al. 2024b) demonstrate that training on a carefully selected subset of the graph can even outperform full graph training,

which implies that a small subset of key nodes accounts for the majority of the learning, suggesting the presence of substantial structural redundancy.

To investigate whether similar phenomena exist in dynamic graphs, we conduct an empirical study on the Wikipedia dataset. As shown in Figure 2, as the proportion of dropped events increases, model performance shows notable improvements. This finding empirically confirms the presence of redundancy in dynamic graphs. We define this phenomenon as static redundancy, referring to superfluous information inherent in dynamic graphs.

This observation raises the critical question of how to systematically identify and eliminate such redundancy. To address this, we propose a metric to quantify node influence (Figure 1(a)), which serves as an effective tool to filter redundant nodes in dynamic graphs.

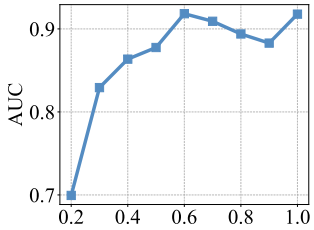


Figure 2: Performance of TGAT on the Wikipedia dataset under different event drop rates. Redundant events are removed by random dropping. The results indicate that removing redundant events can improve model performance, revealing the existence of static redundancy in dynamic graphs.

Dynamic Node Influence for Pruning. In this part, we build on (Gasteiger, Qian, and Günnemann 2022; Xu et al. 2018), which relates node influence to Personalized PageRank (PPR). PPR could capture both importance and proximity of nodes, making it suitable for measuring influence in message passing. Extending this to dynamic graphs, we incorporate temporal information and propose Temporal Personalized PageRank (TPP), which reflects how much a node affects other node's representations or predictions over time. Specifically, the influence score for node u is defined as:

$$\mathrm{TPP}(u) = \frac{1-\xi}{N} + \xi \sum_{(u,v,t)\in\mathcal{H}_v(T)} \frac{\mathrm{TPP}(v)\cdot e^{-\lambda(T-t)}}{\deg^+(v)}, \quad (1)$$

where ξ is the damping factor, λ controls temporal decay, $\deg^+(v)$ denotes the out-degree of node v, and T is the maximum timestamp. TPP score quantifies the expected probability of information propagation along time-respecting edges in dynamic graph. Using this metric, we retain the top-β fraction of nodes to filter out statically redundant nodes, where β is the pruning ratio. The theoretical foundation of this metric is provided in § 4, demonstrating its effectiveness in capturing temporal dependencies.

Temporal Binning for Event Merging. Beyond node-level filtering, we further explore redundancy in event sequences. Such redundancy is usually caused by high-frequency interactions within short time windows, which often carry limited new information and waste sampling budget. For example, a user may repeatedly interact with another user over short spans on the same topic, generating redundant events that do not meaningfully change node states.

To mitigate this, we propose temporal binning, which discretizes continuous time into intervals of length T_{merge}, merging multiple events between the same nodes occurring within the same interval. Formally, for node u with historical events $\mathcal{H}_u(t)$, two events (u,v,t_0) and (u,v,t_1) are merged if $|t_1 - t_0| \le T_{\text{merge}}$. The merged event history $\mathcal{H}_{\text{merge}}(t)$ is defined as: $\mathcal{H}_{\text{merge}}(t) = \{(u,v,t_i') \mid v \in \mathcal{N}_u(t)$ with $t_i' = \max\{t_j \mid t_j \in [t_i - T_{\text{merge}}, t_i], (u,v,t_j) \in \mathcal{H}_u(t)\}\}$. Importantly, this procedure preserves temporal consistency, as each merged timestamp aligns with the most recent event in its bin, maintaining the chronological order of interactions.

While fixed-interval binning reduces redundancy in dense regions, it struggles under varying temporal patterns. For example, online recommender systems often benefit from fine-grained recent data due to rapidly evolving user preferences. To address this, we propose the adaptive binning strategy, where the merging window T_{merge} is adjusted based on event density, enabling more effective filtering of static redundancy across diverse temporal distributions. Specifically, for each node pair (u,v), we compute the mean of consecutive event intervals Δt_{uv} after discarding outliers above the third quartile: $\mathrm{Mean}(\{\Delta t_{uv} \mid Q_1 \le \Delta t_{uv} \le Q_3\})$, which serves as the adaptive merge window T_{merge}.

3.2 Runtime Redundancy

In this section, we focus on runtime redundancy, which refers to redundant information persisting during model training that causes unnecessary computational overhead. Unlike static redundancy inherent in dynamic graphs, runtime redundancy arises from data that gradually becomes less informative relative to the model's learning needs (*e.g.,* dynamically generated redundant edges in evolving temporal graphs). To address this issue, we target one of the most frequent and time-consuming parts in DGNNs, namely neighbor sampling. As illustrated in Figure 1(b), we propose an adaptive sampling strategy on temporal graphs based on graph bottleneck information to reduce redundancy during training.

Adaptive Sampling for Runtime Redundancy. We perform adaptive edge sampling during training via a learnable neighbor sampler. The goal is to obtain a sampled subgraph G_{sub}^t that (1) preserves task-relevant patterns and (2) minimizes redundancy relative to G^t, thereby improving training efficiency. This follows the information bottleneck principle (Seo, Kim, and Park 2023; Wu et al. 2020; Tishby and Zaslavsky 2015), formalized as:

$$\min_{G_{\text{sub}}^t} -I(Y; G_{\text{sub}}^t) + \eta I(G^t; G_{\text{sub}}^t), \quad (2)$$

where Y denotes the task labels, $I(\cdot;\cdot)$ is mutual information, and Lagrange multiplier η balances task relevance and compression. Further, given a node u, we treat each edge in $N_u(t)$ as candidate and sample via Bernoulli process with inclusion probability $P(e_{uv})$ determined by the neighbor sampler:

$$P(e_{uv}) \sim \mathrm{Bernoulli}\left(\mathrm{Sigmoid}\left(\mathrm{MLP}_\phi\left(h(e_{uv})\right)\right)\right), \quad (3)$$

where MLP_ϕ is a multi-layer perceptron parameterized by ϕ and the edge representation $h(e_{uv})$ is defined as $h(e_{uv}) = \text{Concat}(h_u(t)|h_v(t)|x_{ij}^e|\Phi(t))$, with $\Phi(t)$ being the time encoding function.

To address the non-differentiability of Bernoulli sampling, we adopt the Gumbel-softmax reparameterization trick (Jang, Gu, and Poole 2017), allowing gradient based optimization by relaxing the binary sampling into a continuous approximation. Finally, we collect the selected edges to construct the subgraph G_{sub}^t centered at node u at time t, which is then used to update the node representation $h_u(t)$.

With the optimization variables defined, we now describe the optimization process. The first term $I(Y; G_{\text{sub}}^t)$ encourages the sampled subgraph to retain label relevant information and is optimized via the standard dynamic node classification loss $\mathcal{L}_{\text{node}}$. The second term, $I(G^t; G_{\text{sub}}^t)$ penalizes redundancy by minimizing the mutual information between the original and the sampled subgraphs, promoting compactness. Inspired by InfoMax estimators (Veličković et al. 2019; You et al. 2020), we adopt the following contrastive loss:

$$\mathcal{L} = -\log \frac{\exp\left(h_u(G_{\text{sub}}^t) \cdot h_u(G^t)\right)}{\sum_{v \neq u} \exp\left(h_u(G_{\text{sub}}^t) \cdot h_v(G^t)\right)}. \tag{4}$$

Here, subgraph pairs corresponding to the same node (original and sampled) are treated as positive examples, while pairs from different nodes are negatives.

Sampling Frequency Reduction. In addition to reducing subgraph size via adaptive sampling, lowering sampling frequency is another key strategy for reducing training overhead. Thus, we introduce two complementary techniques as follows.

First, to avoid unnecessary sampling on uninformative instances, we introduce batch selector to filter training samples before creating subgraphs. Motivated by curriculum learning and uncertainty aware training (Zhou et al. 2020; Shannon 1948), uncertain samples quantified by the entropy of the predicted label distribution are known to provide richer gradients and accelerate convergence. For each node u, predictive entropy is computed as $-\sum_i \hat{y}_i \log(\hat{y}_i)$, and the top $\mu\%$ most uncertain nodes are retained to form the next mini-batch, with μ being the batch selection ratio.

Second, to reduce redundant neighbor sampling, we maintain an LRU (Least Recently Used) cache that stores recently sampled neighbors. Since cached neighborhoods may become outdated as the graph evolves, a cache bypass probability γ triggers re-sampling with probability γ even on a cache hit, balancing reuse efficiency and temporal adaptivity.

By combining the above sampling techniques, TrimDG significantly reduces sampling overhead, improving scalability to large temporal graphs.

3.3 Complexity Analysis

Here we analyze the computational complexity of TrimDG, which is compatible with any DGNNs architecture. Without loss of generality, we take TGAT, a widely used attention-based DGNN as an example. As mentioned in (Vaswani et al. 2017), the per-batch time complexity of a TGAT layer with k attention heads and l layers is $O\left((k\tilde{N})^l\right)$, and the total training time over the dataset is $O\left(|\mathcal{E}|(k\tilde{N})^l/B\right)$, where $\tilde{N}$ is the average neighborhood size and B is the batch size.

For TrimDG, we first consider the impact of static redundancy removal. Let the pruning ratio be β; then the number of batches is reduced to $O\left(\beta|\mathcal{E}|/B\right)$. During training, to further reduce runtime redundancy, we optimize edge-level sampling probabilities via information bottleneck principle, typically computed with MLPs. Assuming each probability computation has complexity $O(\tilde{N})$, and that a fraction ζ of edges are retained after sampling, the per-batch complexity becomes $O\left(\tilde{N}d + (k\zeta\tilde{N})^l\right)$. Consequently, the overall time complexity of TrimDG is $O\left(\beta|\mathcal{E}|\left(\tilde{N} + (k\zeta\tilde{N})^l\right)/B\right)$ when coupled with TGAT. This deduction process can be similarly applied to other DGNN backbones.

In practice, both β and ζ are typically small, leading to significant training time reductions, often quadratic (*e.g.*, when $l = 2$) compared to the original DGNN backbone.

4 Theoretical Analysis

In this section, we establish the connection between node influence in dynamic graph neural networks and the proposed Temporal Personalized PageRank (TPP) score, demonstrating the validity of the metric.

Theorem 4.1 (Rationality for temporal personalized pagerank)**.** *Consider the message passing phase in a dynamic graph neural network. Let* $\text{Inf}(v_i, v_j)$ *denote the influence of node* v_j *on node* v_i *as defined by the influence based formulation in (Xu et al. 2018). This influence value can be measured by the Temporal Personalized PageRank (TPP) score and can be computed as:*

$$\text{Inf}(v_i, v_j) = \left\{\xi\left(\mathbf{I}_n - (1-\xi)\hat{\mathbf{A}}\right)^{-1} r_{v_i}\right\}_{v_j},$$

where $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ *is the identity matrix,* $\xi \in (0, 1]$ *is the damping factor in TPP,* $r_{v_i} \in \mathbb{R}^n$ *is a one-hot vector indicating the starting node* v_i*,* $\{\cdot\}_{v_j}$ *denotes the* v_j*-th entry of the resulting vector,* $\hat{\mathbf{A}} \in \mathbb{R}^{n\times n}$ *is a time-decayed adjacency matrix with each entry defined as* $\hat{\mathbf{A}}_{ij} = e^{-\lambda(T-t_{ij})}\mathbf{A}_{ij}$*,* $\mathbf{A}_{ij}$ *denoting the original adjacency entry and* t_{ij} *being the timestamp of the interaction from* v_j *to* v_i*.*

This formulation highlights the rationality of TPP in capturing influence propagation in dynamic graphs. The detailed proof is provided in Appendix A.6.

5 Experiments

In this section, we conduct a comprehensive evaluation of the efficiency and performance of TrimDG across diverse datasets from various domains and scales. We also perform component-level analysis to demonstrate the contribution of each module. Detailed datasets and experimental setup information can be found in Appendix A.3, while additional results, such as convergence stability analysis, dynamic link prediction performance, and sampling cache analysis, are provided in Appendix A.4. Our code is available at https://github.com/zjunet/TrimDG.

5.1 Experimental Setup

Datasets. We evaluate our framework on six dynamic graph datasets spanning diverse domains for node classification. A summary is shown in Table 1 and detailed statistics are provided in Appendix A.3.

Wikipedia (Xu et al. 2020): A bipartite graph recording edits to Wikipedia pages over month. Nodes represent users and pages, and edges denote editing behaviors with timestamps, encoded with the Linguistic Inquiry and Word Count feature. The task is to predict dynamic labels indicating whether users are temporarily banned from editing.

Reddit (Xu et al. 2020): A bipartite graph capturing user posts in subreddits over one month. Nodes correspond to users and subreddits, and timestamped edges represent posting actions enriched with 172-dimensional LIWC features. Dynamic labels indicate whether users are banned from posting.

BitOtc , **BitAlpha** (Kumar et al. 2016): Bipartite graphs from Bitcoin OTC trading platform and Alpha trading platform, respectively. Users are categorized as raters or ratees. Edges represent timestamped trust ratings between users, including features derived from transaction metadata. Node labels classify users as benign (users rated ≥ 5 by trusted users) or fraudulent (users rated ≤ 5 by trusted users).

MOOC (Kumar, Zhang, and Leskovec 2019): A bipartite graph capturing student interactions on an online course platform. Nodes represent users and items (*e.g.*, videos, answers), and timestamped edges indicate interaction events such as video views or answer submissions. Edge features encode interaction types, and labels indicate dropout events.

DGraph (Huang et al. 2022): A large-scale, real-world dynamic graph from the financial domain, provided by Finvolution Group. Nodes represent platform users, and edges denote timestamped emergency contact relationships.

Datasets	Domains	#Nodes	#Links
Wikipedia	Social	9,227	157,414
Reddit	Social	10,984	672.447
BitAlpha	Finance	3,783	24,186
BitOtc	Finance	5,881	35,592
MOOC	Interaction	7,144	411,749
DGraph	Finance	3,700,550	4,300,999

Table 1: Brief statistics of datasets from different domains. #Nodes and #Links denote the numbers of nodes and edges.

Backbones and Baselines. To showcase the broad applicability of TrimDG across various DGNNs, we evaluate it on seven widely used and effective continuous-time dynamic graph learning backbones, covering four major paradigms: graph convolutions, memory networks, random walks, and sequential models. The selected models include JODIE (Kumar, Zhang, and Leskovec 2019), DyRep (Trivedi et al. 2022), TGAT (Xu et al. 2020), TGN (Rossi et al. 2020), TCL (Wang et al. 2021), GraphMixer (Cong et al. 2023), and DyGFormer (Yu et al. 2023). We follow the implementations from (Yu et al. 2023), and detailed descriptions of these backbones are provided in Appendix A.3.

As for baselines, we compare our method with both static and dynamic graph acceleration approaches. Under the experimental setting, TrimDG is generally applicable to other DGNNs. Therefore, we evaluate it against the following baselines adapted from prior works: (1) DropEdge (Rong et al. 2019), a heuristic method for accelerating static GNNs by randomly dropping edges from the graph. The drop ratio is set to 50%. (2) TASER (Deng et al. 2024), a dynamic graph acceleration method, from which we adopt the temporal adaptive mini-batch selection module for comparison. (3) STEP (Li et al. 2023a), an effective pruning framework that prunes the evolving graph structure in dynamic graphs.

It is worth noting that our work focuses on algorithmic-level acceleration for DGNNs, rather than system-level optimizations such as GPU/CPU scheduling or low-level runtime improvements, which are orthogonal to our method and beyond the scope of this work. Detailed discussion and comparison with system-level methods are included in Appendix A.4.

Implementation Details. To ensure fair comparison, we adopt standardized yet scenario adaptive configurations for baseline models based on (Yu et al. 2023). All models are trained for 50 iterations using mini-batch training. For smaller-scale datasets like Wikipedia, BitAlpha and BitOtc, a batch size of 200 is employed, while larger datasets such as MOOC, Reddit and DGraph utilize a batch size of 400 to accommodate their larger graph scale. Additional implementation details can be found in Appendix A.3.

For TrimDG, hyperparameters are set as $\beta = 0.6$, $\xi = 0.85$, $\lambda = 0.1$ and $\mu = 0.5$ to control sampling granularity and feature aggregation strategy. To regulate the optimization goal during sampling, the hyperparameter η is set to $1e$-3, balancing model complexity and prediction accuracy. Details hardware configurations are presented in Appendix A.3.

5.2 Experimental Results

Main Results. Table 2 presents the dynamic node classification results. Compared with vanilla DGNN backbone models, TrimDG significantly reduces runtime across multiple DGNNs, achieving the largest average speedup over seven backbone models among all baselines. Among the backbones, TGAT benefits the most, while JODIE shows weaker acceleration. This is because TGAT relies heavily on attention-based aggregation and neighbor sampling, which align well with our optimization strategy, whereas JODIE lacks such components. Furthermore, we observe that TrimDG not only reduces runtime by 83.49%, but also improves predictive performance by an average of 2.66% on Wikipedia. This indicates that removing redundant computations not only accelerates training but also filters out noisy information, yielding better representations.

Across different datasets, we observe that as the graph grows in scale, TrimDG achieves increasingly significant advantages over the baselines. This phenomenon can be attributed to the scaling effect of TrimDG's acceleration mechanism. Furthermore, we also evaluate TrimDG on dynamic link prediction

Model	Method	Wikipedia		Reddit		Mooc		BitAlpha		BitOtc		DGraph		Avg. rank	
		AUC	Runtime	AUC	Runtime	AUC	Runtime	AUC	Runtime	AUC	Runtime	AUC	Runtime	AUC	Runtime
JODIE	Vanilla	86.38 ±1.00	1246.90	52.76 ±3.79	3809.66	67.14 ±1.66	2943.68	62.10 ±0.94	45.73	65.46 ±4.03	54.05	/	/	3.2	4.5
	DropEdge	85.55 ±5.27	530.09	52.37 ±2.38	2128.52	65.31 ±1.60	1154.91	66.83 ±4.20	31.35	74.76 ±4.92	37.55	/	/	3.6	3.2
	STEP	85.07 ±3.37	226.53	54.94 ±5.04	1358.21	60.31 ±0.95	693.34	76.53 ±4.63	31.74	76.41 ±7.30	25.83	/	/	3.2	2.3
	TASER	87.35 ±0.57	274.49	57.28 ±2.44	924.10	70.27 ±1.38	525.17	71.47 ±1.84	29.95	85.04 ±1.08	43.75	/	/	1.7	2.3
	TrimDG	86.64 ±1.30	266.35	52.39 ±2.28	830.75	60.61 ±1.61	473.92	76.78 ±9.14	38.99	87.63 ±1.73	24.78	55.54 ±0.27	6774.04	2.7	1.7
DyRep	Vanilla	78.74 ±1.49	9747.83	57.83 ±5.19	43543.27	64.24 ±1.34	27619.97	63.14 ±10.61	432.17	73.40 ±9.36	537.43	/	/	3.0	4.3
	DropEdge	77.55 ±4.28	5336.76	53.65 ±5.99	25979.80	66.80 ±0.59	15999.00	66.30 ±5.57	286.74	69.83 ±12.51	304.89	/	/	3.3	3.0
	STEP	86.13 ±1.29	5863.08	56.67 ±5.84	31605.80	57.71 ±6.75	9146.31	76.41 ±1.25	149.33	75.42 ±10.85	139.23	/	/	2.3	2.8
	TASER	85.05 ±2.10	2116.25	45.72 ±2.81	29040.80	70.26 ±0.51	5623.65	69.16 ±8.29	134.12	86.57 ±0.30	189.58	/	/	2.5	2.2
	TrimDG	87.51 ±1.78	687.43	50.83 ±0.94	2184.14	58.22 ±7.16	905.72	71.42 ±18.01	103.72	89.71 ±5.53	47.44	/	/	2.2	1.0
TGAT	Vanilla	79.10 ±1.50	10393.35	66.93 ±2.27	42104.05	64.76 ±0.84	24575.41	60.26 ±4.94	451.21	73.64 ±1.54	526.61	76.98 ±0.74	9877.62	3.0	4.5
	DropEdge	81.89 ±1.68	6205.99	72.04 ±1.67	24006.56	65.75 ±0.90	14584.96	58.29 ±4.76	275.89	74.49 ±1.41	300.95	73.18 ±1.03	4318.14	2.5	3.8
	STEP	78.79 ±2.15	3675.39	59.47 ±1.43	22979.36	65.29 ±0.39	12428.34	75.62 ±1.78	218.81	64.81 ±9.64	165.81	/	/	3.0	2.8
	TASER	77.51 ±3.73	1853.94	54.09 ±3.24	6302.52	59.94 ±2.15	4059.26	69.28 ±1.34	126.10	86.04 ±5.59	183.15	/	/	3.5	2.3
	TrimDG	81.90 ±1.02	410.66	55.80 ±4.07	1381.25	63.34 ±1.78	653.02	60.61 ±10.59	101.13	81.87 ±2.57	73.62	77.41 ±1.08	1294.77	2.5	1.0
TGN	Vanilla	83.70 ±4.98	11636.33	61.92 ±6.90	48513.78	70.16 ±1.83	29464.63	57.56 ±4.36	527.35	70.65 ±1.74	629.25	/	/	2.7	4.3
	DropEdge	76.89 ±4.49	6440.36	60.17 ±1.71	28451.14	69.78 ±1.49	15919.83	44.83 ±10.99	302.34	80.97 ±6.76	361.53	/	/	3.3	2.7
	STEP	85.32 ±3.65	6886.55	63.89 ±2.59	35614.41	68.39 ±0.78	14033.92	78.25 ±4.35	222.42	82.08 ±4.09	217.46	/	/	1.8	2.7
	TASER	84.89 ±1.74	6725.71	52.98 ±5.14	35256.75	67.91 ±1.71	20479.04	68.49 ±8.71	161.18	88.19 ±3.11	229.72	/	/	2.8	2.7
	TrimDG	84.87 ±2.22	849.87	55.04 ±2.46	2088.81	68.65 ±0.20	1506.23	68.30 ±12.45	112.98	90.60 ±1.79	79.16	/	/	2.5	1.0
TCL	Vanilla	78.05 ±1.91	1468.49	66.47 ±5.94	4548.92	67.78 ±1.91	2723.49	63.43 ±2.29	80.93	80.68 ±0.78	103.43	80.54 ±0.50	2727.75	2.2	4.8
	DropEdge	78.66 ±4.76	796.02	58.53 ±10.69	2455.45	66.84 ±1.44	1536.48	62.17 ±1.17	50.45	79.86 ±0.08	65.14	77.31 ±1.66	1372.50	3.3	2.7
	STEP	83.50 ±0.72	573.98	60.93 ±0.18	4145.90	66.68 ±0.35	2032.12	72.63 ±5.27	55.17	69.54 ±5.19	33.97	/	/	3.5	3.3
	TASER	77.71 ±3.58	290.97	60.98 ±4.25	3833.45	67.35 ±2.17	2397.90	61.33 ±2.27	43.83	81.44 ±0.56	68.74	75.82 ±0.58	837.17	3.5	2.7
	TrimDG	84.23 ±0.66	272.96	56.47 ±4.23	877.27	70.06 ±0.52	267.70	61.47 ±3.11	56.29	82.32 ±13.93	30.14	77.04 ±1.42	697.64	2.5	1.5
GraphMixer	Vanilla	80.43 ±2.55	1467.81	53.97 ±4.71	5515.32	63.17 ±0.73	3111.31	59.24 ±4.45	80.88	62.75 ±8.06	104.37	79.22 ±0.30	2694.73	3.0	3.7
	DropEdge	81.46 ±3.46	898.47	49.52 ±4.50	3273.49	61.56 ±0.28	1836.32	61.91 ±1.04	51.23	44.80 ±5.55	58.64	78.13 ±0.36	1226.66	3.8	2.0
	STEP	86.62 ±1.69	3268.87	58.06 ±0.83	15111.90	66.66 ±0.16	3837.71	66.13 ±4.24	84.45	60.77 ±3.88	107.85	/	/	2.3	4.7
	TASER	76.52 ±1.25	813.54	50.98 ±5.25	4175.90	55.86 ±0.92	2553.30	66.17 ±0.90	90.10	68.14 ±1.46	120.42	79.24 ±1.68	1450.37	2.7	3.5
	TrimDG	85.85 ±0.51	358.03	57.13 ±1.25	965.89	54.66 ±2.41	470.14	54.51 ±2.93	51.79	66.27 ±0.55	38.78	78.84 ±1.06	1182.97	3.2	1.2
DyGFormer	Vanilla	89.67 ±1.14	3062.51	62.24 ±9.83	11962.62	69.46 ±0.63	6676.24	79.34 ±5.29	149.49	87.44 ±4.31	203.05	75.34 ±0.97	6147.51	1.8	4.0
	DropEdge	82.93 ±2.40	1918.15	64.75 ±7.89	6280.22	66.84 ±0.82	3791.50	72.38 ±11.56	96.28	81.74 ±4.74	104.00	71.46 ±0.51	3217.65	3.7	2.2
	STEP	85.31 ±1.72	3415.87	65.06 ±2.04	16385.72	69.62 ±0.42	12914.37	74.33 ±4.30	150.68	72.21 ±8.38	152.52	/	/	3.0	4.7
	TASER	88.13 ±0.98	3311.40	58.54 ±6.02	4175.90	65.10 ±1.04	4975.74	73.37 ±3.84	87.36	84.88 ±0.93	163.94	74.21 ±0.48	2957.43	3.3	2.7
	TrimDG	78.51 ±10.01	1007.35	61.54 ±0.93	1839.51	71.66 ±0.91	884.23	73.22 ±3.98	147.56	74.66 ±5.28	105.92	76.31 ±0.72	2143.75	3.2	1.5

Table 2: Dynamic node classification performance under AUC score (%) and Runtime (s) across various DGNNs backbone models. Underline indicates the best result per backbone per dataset and results are averaged over five runs with standard deviations. The notation "/" means out of memory or no convergence for more than one day of training.

tasks, and have recorded the RAM and GPU memory consumption of our experiments. These additional statistics and analysis are provided in the Appendix A.4.

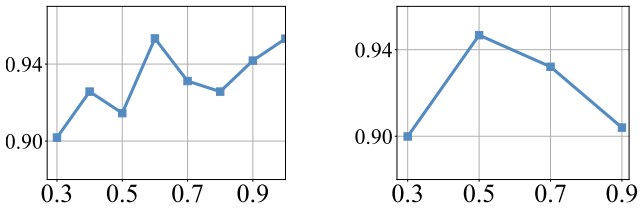
Ablation Studies. To assess the contribution of each component in TrimDG, we perform ablation studies with the following variants: (1) TrimDG-SR, without static redundancy removal; (2) TrimDG-RR, without runtime redundancy removal; (3) TrimDG-BS, without the batch selector; and (4) TrimDG-C, without the sampling cache. We decompose the total training process into data processing, model training and sampling. Performance metrics are reported under these breakdowns. The experiments are conducted on Wikipedia, training for 30 epochs, and use TGAT as the backbone. Experimental settings are detailed in Appendix A.3.

Method	ROC-AUC	Time (s)		
		Data preprocess	Training	Sampling
TGAT	75.12 ±1.00	**1.72**	962.38	732.78
TrimDG-SR	76.36 ±7.53	1.81	315.60	264.61
TrimDG-RR	75.78 ±1.50	2.09	377.83	321.70
TrimDG-BS	**81.71** ±1.77	2.01	279.91	226.61
TrimDG-C	80.85 ±2.65	2.05	349.15	299.64
TrimDG	80.76 ±5.90	2.07	**237.14**	**197.60**

Table 3: Performance comparison regarding ablation studies.

The results, depicted in Table 3, clearly demonstrate the contributions of each component to enhancing the efficiency of TrimDG. In terms of the ROC-AUC score, ablation variants show improvements compared to TGAT, which indirectly indicates the presence of substantial redundancy in dynamic graph data. TrimDG-BS performs slightly better than TrimDG , but at the cost of taking longer time. As for data preprocessing, TrimDG has a relatively higher time consumption than that of TGAT, since we need to remove static redundancy before training. TrimDG-SR has a similar time cost in data preprocessing compared to TGAT, further confirming that the extra time cost is caused by pre-sampling. Regarding sampling time and training time, it can be observed that models with sampling strategies all reduce these components to a certain extent, with TrimDG benefiting the most. These results reveal that TrimDG ultimately reduces the total time consumption through intricate cooperation between different components, and that each component is an indispensable part of TrimDG.

Hyper-Parameters Analysis. We evaluate the effects of the pruning ratio β and the batch selection ratio μ on the performance of TrimDG, training for 30 epochs on Wikipedia and with TGAT as the backbone. Results are shown in Figure 3. We observe that selecting data within a moderate ratio range yields better performance than using the full dataset, which again confirms the substantial static and runtime redundancy in dynamic graphs. Additionally, when μ and β exceed this optimal range, performance begins to degrade, suggesting that overly aggressive sampling may lead to information loss. However, the performance remains comparable to that of using all the data, even with sub-optimal hyper-parameter settings, demonstrating the hyper-parameter robustness.



(a) Pre-sampling ratio β. (b) Batch selection ratio μ.

Figure 3: Performance of TrimDG under varying pruning ratio β and batch selection ratio μ.

6 Related Work

Dynamic Graph Representation Learning. Recent advances in dynamic graph learning have led to a variety of models aimed at capturing temporal dependencies in evolving graphs. Early works on discrete time dynamic graphs (DTDGs) treat graph evolution as a sequence of snapshots and apply static GNNs per timestep, often overlooking fine-grained temporal order. In contrast, continuous-time dynamic graph (CTDG) models operate directly on event streams to capture real-time dynamics. CTDG models can be broadly categorized into three types: (1) RNN-based models (*e.g.*, JODIE (Kumar, Zhang, and Leskovec 2019), DyRep (Trivedi et al. 2022) and TGN (Rossi et al. 2020)) maintain evolving node states through sequential updates; (2) self-attention models (*e.g.*, TGAT (Xu et al. 2020) and DyGFormer (Yu et al. 2023)) use temporal encodings and attention to capture long-range dependencies; and (3) temporal random walk models (*e.g.*, CAWN) incorporate time into walk sampling to preserve temporal order. While these methods achieve strong performance, they often suffer from redundant computation, inefficient memory usage, and limited ability to model long-term interactions or high-degree nodes, motivating the need for more efficient architectures.

Accelerators for Static GNNs. To address the scalability challenges of traditional full-batch GNNs on large-scale graphs, prior works have introduced several acceleration strategies, which can be categorized into sampling-based methods, graph pruning techniques, and model-level optimizations. Sampling-based methods, such as FastGCN (Chen, Ma, and Xiao 2018) and GraphSAINT (Zeng et al. 2019), reduce computation by training on sampled subgraphs, though they may incur redundancy from overlapping samples. Graph pruning techniques, like DropEdge (Rong et al. 2019), GraphAlign (Huang et al. 2024a) and NeuralSparse (Zheng et al. 2020), remove task-irrelevant edges to lower memory usage and enhance generalization. Model-level approaches, including SGC (Wu et al. 2019) and PPNP (Gasteiger, Bojchevski, and Günnemann 2018), eliminate inter-layer nonlinearity and decouple propagation, enabling efficient inference via precomputed diffusion matrices such as Personalized PageRank.

Accelerators for Dynamic GNNs. Recent advances in accelerating DGNNs have explored various strategies across both algorithm-level and system-level optimizations.

For algorithm-level strategies, TGN (Rossi et al. 2020) enhances efficiency by encoding temporal information and maintaining memory states. Zebra (Li et al. 2023b) replaces recursive temporal message passing with top-k Temporal Personalized PageRank queries to efficiently aggregate influential neighbors. STEP (Li et al. 2023a) introduces the pruning network to eliminate noisy edges.

For system-level optimizations, frameworks such as TGL (Zhou et al. 2022) and DistTGL (Zhou et al. 2023) employ techniques like the Temporal-CSR layout, parallel sampling, and distributed memory management to enable scalable training. SPEED (Chen et al. 2023) leverages advanced graph partitioning and hierarchical pipeline parallelism to reduce computational overhead. Bottleneck analysis by (Chen et al. 2022) further identifies four major hardware bottlenecks that limit the efficiency of DGNN training.

Despite these advances, most existing approaches do not reduce redundancy from a data-centric perspective on dynamic graphs. Such a perspective can help eliminate redundant information while preserving model performance. Our proposed framework fills this gap by identifying and removing both static and runtime redundancy, thereby improving the efficiency of DGNNs. This method is model-agnostic and can be effectively integrated into various mainstream frameworks.

7 Conclusion

In this work, we address the scalability challenges faced by Dynamic Graph Neural Networks (DGNNs) in real-world industrial-scale applications. We attribute the inefficiency of DGNNs to static and dynamic redundancies and propose a unified optimization framework, TrimDG, that systematically reduces computational overhead. Through a combination of temporal-aware pre-sampling via influence-based pruning, adaptive sampling with graph bottleneck guidance, and efficient batch selection, TrimDG effectively mitigates neighbor explosion and redundant computation without compromising temporal integrity. Extensive experiments on public benchmarks demonstrate the effectiveness and scalability of our approach, offering a practical and efficient solution for deploying DGNNs on large-scale, real-world dynamic graphs.

Acknowledgements

This work is supported by NSFC (No. 62322606, No. 62441605), Zhejiang NSF (LR22F020005), and a collaboration funding by MYbank, Ant Group.

References

Baig, M. B.; and Akoglu, L. 2015. Correlation of node importance measures: An empirical study through graph robustness. In *WWW*, 275–281.

Benczúr, A. A.; and Karger, D. R. 1996. Approximating st minimum cuts in Õ (n 2) time. In *STOC*, 47–55.

Cai, C.; Wang, D.; and Wang, Y. 2021. Graph coarsening with neural networks. *ICLR*.

Cao, Y.; Xu, J.; Zhao, C.; Wang, J.; Yang, C.; Wang, C.; and Yang, Y. 2025. How to Use Graph Data in the Wild to Help Graph Anomaly Detection? *KDD*.

Chakaravarthy, V. T.; Pandian, S. S.; Raje, S.; Sabharwal, Y.; Suzumura, T.; and Ubaru, S. 2021. Efficient scaling of dynamic graph neural networks. In *SC*, 1–15.

Chen, H.; Alhinai, Y.; Jiang, Y.; Na, E.; and Hao, C. 2022. Bottleneck analysis of dynamic graph neural network inference on cpu and gpu. In *IISWC*, 130–145. IEEE.

Chen, H.; and Hao, C. 2023. Dgnn-booster: A generic fpga accelerator framework for dynamic graph neural network inference. In *FCCM*, 195–201. IEEE.

Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *ICLR*.

Chen, X.; Liao, Y.; Xiong, Y.; Zhang, Y.; Zhang, S.; Zhang, J.; and Sun, Y. 2023. Speed: Streaming partition and parallel acceleration for temporal interaction graph embedding. *arXiv preprint arXiv:2308.14129*.

Cong, W.; Zhang, S.; Kang, J.; Yuan, B.; Wu, H.; Zhou, X.; Tong, H.; and Mahdavi, M. 2023. Do we really need complicated model architectures for temporal networks? *ICLR*.

Deng, G.; Zhou, H.; Zeng, H.; Xia, Y.; Leung, C.; Li, J.; Kannan, R.; and Prasanna, V. 2024. TASER: Temporal adaptive sampling for fast and accurate dynamic graph representation learning. In *IPDPS*, 926–937. IEEE.

Gasteiger, J.; Bojchevski, A.; and Günnemann, S. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *ICLR*.

Gasteiger, J.; Qian, C.; and Günnemann, S. 2022. Influence-based mini-batching for graph neural networks. In *LoG*, 9–1. PMLR.

Huang, R.; Xu, J.; Jiang, X.; An, R.; and Yang, Y. 2024a. Can Modifying Data Address Graph Domain Adaptation? In *SIGKDD*, 1131–1142.

Huang, R.; Xu, J.; Jiang, X.; Pan, C.; Yang, Z.; Wang, C.; and Yang, Y. 2024b. Measuring task similarity and its implication in fine-tuning graph neural networks. In *AAAI*, 12617–12625.

Huang, X.; Yang, Y.; Wang, Y.; Wang, C.; Zhang, Z.; Xu, J.; Chen, L.; and Vazirgiannis, M. 2022. Dgraph: A large-scale financial dataset for graph anomaly detection. *NeurIPS*, 35: 22765–22777.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax.

Jin, W.; Ma, Y.; Liu, X.; Tang, X.; Wang, S.; and Tang, J. 2020. Graph structure learning for robust graph neural networks. In *SIGKDD*, 66–74.

Jin, W.; Zhao, L.; Zhang, S.; Liu, Y.; Tang, J.; and Shah, N. 2022. Graph condensation for graph neural networks. *ICLR*.

Khodabandehlou, S.; and Golpayegani, A. H. 2024. FiFrauD: Unsupervised Financial Fraud Detection in Dynamic Graph Streams. *ACM TKDD*, 18(5).

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *ICLR*.

Kumar, S.; Spezzano, F.; Subrahmanian, V.; and Faloutsos, C. 2016. Edge weight prediction in weighted signed networks. In *ICDM*, 221–230. IEEE.

Kumar, S.; Zhang, X.; and Leskovec, J. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *SIGKDD*, 1269–1278.

Li, J.; Tian, S.; Wu, R.; Zhu, L.; Zhao, W.; Meng, C.; Chen, L.; Zheng, Z.; and Yin, H. 2023a. Less can be more: Unsupervised graph pruning for large-scale dynamic graphs. *arXiv preprint arXiv:2305.10673*.

Li, W.; Wu, L.; Zhao, P.; Wang, J.; and Karypis, G. 2023b. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *Proc. VLDB Endow.*, 16(6): 1332–1345.

Liu, Y.; Zheng, Y.; Zhang, D.; Chen, H.; Peng, H.; and Pan, S. 2022. Towards unsupervised deep graph structure learning. In *WWW*, 1392–1403.

Loukas, A. 2019. Graph reduction with spectral and cut guarantees. *JMLR*, 20(116): 1–42.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

Por, E.; van Kooten, M.; and Sarkovic, V. 2019. Nyquist–Shannon sampling theorem. *Leiden University*, 1(1): 1–2.

Poursafaei, F.; Huang, S.; Pelrine, K.; and Rabbany, R. 2022. Towards better evaluation for dynamic link prediction. *NeurIPS*, 35: 32928–32941.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2019. Dropedge: Towards deep graph convolutional networks on node classification. *ICLR*.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *ICML Workshop*.

Seo, S.; Kim, S.; and Park, C. 2023. Interpretable prototype-based graph information bottleneck. *NeurIPS*, 36: 76737–76748.

Shannon, C. E. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3): 379–423.

Sun, Y.; Deng, H.; Yang, Y.; Wang, C.; Xu, J.; Huang, R.; Cao, L.; Wang, Y.; and Chen, L. 2022. Beyond Homophily: Structure-aware Path Aggregation Graph Neural Network. In *IJCAI*.

Sun, Y.; Liu, Z.; Hooi, B.; Yang, Y.; Fathony, R.; Chen, J.; and He, B. 2025. Multi-Label Node Classification with Label Influence Propagation. In *ICLR*.

Tishby, N.; and Zaslavsky, N. 2015. Deep learning and the information bottleneck principle. In *ITW*, 1–5. IEEE.

Tolstikhin, I. O.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *NeurIPS*, 34: 24261–24272.

Trivedi, R.; Farajtabar, M.; Biswal, P.; and Zha, H. 2022. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *NeurIPS*, 30.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *ICLR*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep graph infomax. *ICLR*.

Wang, C.; Sun, D.; and Bai, Y. 2023. PiPAD: pipelined and parallel dynamic GNN training on GPUs. In *PPoPP*, 405–418.

Wang, L.; Chang, X.; Li, S.; Chu, Y.; Li, H.; Zhang, W.; He, X.; Song, L.; Zhou, J.; and Yang, H. 2021. TCL: Transformer-based Dynamic Graph Modelling via Contrastive Learning. *CoRR*.

Wang, L.; Zhuang, J.; Ma, S.; and Lin, H. 2025. Frequency Enhanced Dynamic Graph Convolutional Networks for Traffic Flow Forecasting. *IEEE Access*, 13: 103451–103462.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *ICML*, 6861–6871. Pmlr.

Wu, T.; Ren, H.; Li, P.; and Leskovec, J. 2020. Graph information bottleneck. *NeurIPS*, 33: 20437–20448.

Xu, D.; Ruan, C.; Korpeoglu, E.; Kumar, S.; and Achan, K. 2020. Inductive Representation Learning on Temporal Graphs. In *ICLR*.

Xu, J.; Huang, R.; JIANG, X.; Cao, Y.; Yang, C.; Wang, C.; and YANG, Y. 2023. Better with Less: A Data-Active Perspective on Pre-Training Graph Neural Networks. In *NeurIPS*, volume 36, 56946–56978. Curran Associates, Inc.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*, 5453–5462. PMLR.

Yang, L.; Chatelain, C.; and Adam, S. 2024. Dynamic graph representation learning with neural networks: A survey. *IEEE Access*, 12: 43460–43484.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph contrastive learning with augmentations. *NeurIPS*, 33: 5812–5823.

Yu, L.; Sun, L.; Du, B.; and Lv, W. 2023. Towards better dynamic graph learning: New architecture and unified library. *NeurIPS*, 36: 67686–67700.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2019. Graphsaint: Graph sampling based inductive learning method. *ICLR*.

Zhang, W.; Wang, Y.; You, Z.; Cao, M.; Huang, P.; Shan, J.; Yang, Z.; and Cui, B. 2021. Rim: Reliable influence-based active learning on graphs. *NeurIPS*, 34: 27978–27990.

Zheng, C.; Zong, B.; Cheng, W.; Song, D.; Ni, J.; Yu, W.; Chen, H.; and Wang, W. 2020. Robust graph representation learning via neural sparsification. In *ICML*, 11458–11468. PMLR.

Zhou, H.; Zheng, D.; Nisa, I.; Ioannidis, V.; Song, X.; and Karypis, G. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proc. VLDB Endow.*, 15(8).

Zhou, H.; Zheng, D.; Song, X.; Karypis, G.; and Prasanna, V. 2023. Disttgl: Distributed memory-based temporal graph neural network training. In *SC*, 1–12.

Zhou, W.; Wang, C.; Luo, F.; Wang, Y.; Gao, M.; and Wen, J. 2025. Community-Enhanced Dynamic Graph Convolutional Networks for Rumor Detection on Social Networks. *IEEE TCSS*, 12(2): 818–831.

Zhou, Y.; Yang, B.; Wong, D. F.; Wan, Y.; and Chao, L. S. 2020. Uncertainty-aware curriculum learning for neural machine translation. In *ACL*, 6934–6944.