DyG2Vec: Representation Learning for Dynamic Graphs with Self-Supervision

Mohammad Ali Alomrani*
Huawei Noah's Ark Lab
Toronto, Canada

Mahdi Biparva*†
Huawei Noah's Ark Lab
Toronto, Canada

Yingxue Zhang
Huawei Noah's Ark Lab
Toronto, Canada

Mark Coates
McGill University
Montreal, Canada

ABSTRACT

Temporal graph neural networks have shown promising results in learning inductive representations by automatically extracting temporal patterns. However, previous works often rely on complex memory modules or inefficient random walk methods to construct temporal representations. In addition, the existing dynamic graph encoders are non-trivial to adapt to self-supervised paradigms, which prevents them from utilizing unlabeled data. To address these limitations, we present an efficient yet effective attention-based encoder that leverages temporal edge encodings and window-based subgraph sampling to generate task-agnostic embeddings. Moreover, we propose a joint-embedding architecture using non-contrastive SSL to learn rich temporal embeddings without labels. Experimental results on 7 benchmark datasets indicate that on average, our model outperforms SoTA baselines on the future link prediction task by 4.23% for the transductive setting and 3.30% for the inductive setting while only requiring 5-10x less training/inference time. Additionally, we empirically validate the SSL pre-training significance under two probings commonly used in language and vision modalities. Lastly, different aspects of the proposed framework are investigated through experimental analysis and ablation studies.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; *Learning latent representations*; • **Theory of computation → Semi-supervised learning**.

KEYWORDS

dynamic graphs, graph neural networks, self-supervised learning

*Both authors contributed equally to the paper.
†Correspondence to mahdi.biparva@huawei.com

MLG 2023, August 9, 2023, Long Beach, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Mohammad Ali Alomrani, Mahdi Biparva, Yingxue Zhang, and Mark Coates. 2018. DyG2Vec: Representation Learning for Dynamic Graphs with Self-Supervision. In *Proceedings of 19TH International Workshop on Mining and Learning With Graphs (MLG 2023).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recently, temporal graph neural nets [12, 20, 26, 37] have emerged as promising representation learning approaches that are able to extract temporal patterns from an ever-evolving dynamic graph in order to make accurate future predictions. However, such models have several shortcomings. First, they heavily rely on chronological training and/or complex memory modules to construct predictions [18, 26, 37, 40]. Consequently, encoding any dynamic graph requires sequentially iterating through all edges, which is intractable for large graphs due to the high computational overhead. Second, the encoding modules either use inefficient message-passing procedures [40] that enforce temporal causality, or expensive random walk-based algorithms [12, 37] with heuristic feature encoding strategies that are engineered for edge-level tasks only. Finally, as opposed to other temporal domains [6, 32], most works on dynamic graphs have focused on pushing downstream task performance rather than learning general pre-trained models.

Self-Supervised Representation Learning (SSL) has shown promise in achieving competitive performance for different data modalities on multiple predictive tasks [19]. Given a large corpus of unlabelled data, SSL postulates that unsupervised pre-training is sufficient to learn robust representations that are predictive for downstream tasks with minimal fine-tuning. Contrastive SSL methods, despite their early success, rely heavily on negative samples, extensive data augmentation, and large batch sizes [8, 13]. Non-contrastive methods address these shortcomings, incorporating information theoretic principles through architectural innovations or regularization methods [1]. The success of such SSL methods on sequential data [6, 24, 32] suggests that one can learn rich temporal node embeddings from dynamic graphs without direct supervision. While there are some recent attempts at using SSL for dynamic graphs such as DDGCL [30] and DySubC [11], they tend to require high memory and computation due to negative sampling and focus more on pushing downstream performance rather than learning rich general representations.

In this work, we propose DyG2Vec, a novel efficient encoder-decoder model for continuous-time dynamic graphs that benefits

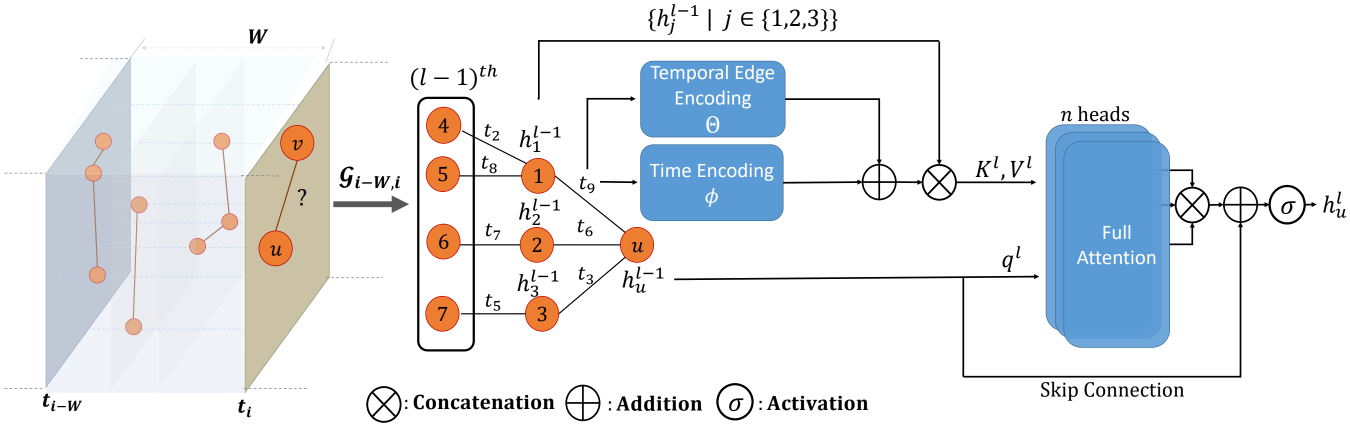

Figure 1: Using DyG2Vec Window Framework to encode the target node u. Every slice of the dynamic graph $\mathcal{G}$ contains edges that arrived at the same continuous timestamp. The blue interval represents the history graph $\mathcal{G}_{i-W,i}$ that is encoded to make a prediction on the target edge (u, v). Note that both u and v share the same sampled history graph. For simplicity, we omit edge features m_p from the attention encoder.

from a window-based architecture that acts as a regularizer to avoid over-fitting. DyG2Vec is an efficient attention-based graph neural network that performs message-passing across structure and time to output task-agnostic node embeddings, without the need for expensive random-walk anonymyzation procedures [12, 37] or memory modules [26, 28]. We equip DyG2Vec with the ability to perform non-contrastive SSL, which allows the model to learn rich representations without labels. Our results on 7 benchmark datasets indicate that on average, DyG2Vec outperforms the SoTA baseline CaW [37] on the future link prediction task by 4.23% for the transductive setting and 3.30% for the inductive setting. In addition, DyG2Vec addresses the efficiency bottleneck often experienced with other dynamic graph encoding alternatives. It reduces the training/inference time by 5-10x compared to the state-of-the-art models, thereby providing superior model performance with a significantly reduced computational demand. This efficiency gain significantly enhances the model's scalability potential for large graphs.

Our main contributions can be summarized as follows:

- We propose an effective message-passing encoder that leverages temporal edge encoding to increase its expressive power.
- We eliminate the need for memory modules or expensive causal random-walk extraction methods through efficient window-based subgraph encoding, making it easier to extract temporal motifs involving target nodes.
- We propose a non-contrastive joint-embedding pre-training method that is capable of learning rich representations while avoiding the time-consuming negative sampling procedures.

2 RELATED WORK

We review the most relevant literature on dynamic graph and self-supervised representation learning.

Representation learning for dynamic graphs: Early works on representation learning for continuous-time dynamic graphs typically divide the graph into snapshots that are encoded by a static GNN and then processed by an RNN module [15, 22, 27]. Such methods fail to learn fine-grained temporal patterns at smaller timescales within each snapshot. Therefore, several RNN-based methods were introduced that sequentially update node embeddings as new edges arrive. JODIE [18] employs two RNN modules to update the source and destination embeddings of an arriving edge. DyRep [33] adds a temporal attention layer to take into account multi-hop interactions when updating node embeddings. TGAT [40] includes an Attention-based Message-Passing (AMP) architecture to aggregate messages from a historical neighborhood. TGN [26] alleviates the expensive neighborhood aggregation of TGAT by using an RNN memory module to encode the history of each node. CaW [37] extracts temporal patterns through an expensive procedure that samples temporal random walks and encodes them with an LSTM. This procedure must be performed for every prediction. PINT [28] is a memory-based method that leverages injective message-passing and relative positional encodings to overcome the theoretical weakness of both memory-based methods (e.g., TGN) and walk-based methods (e.g., CaW). Jin et al. [12] adapt CaW to include spatio-temporal bias and exploitation-exploration trade-off sampling biases, employing differential equations (ODE) to effectively model the irregularly sampled temporal interactions of a node. NAT [20] abandons the commonly used message-passing and walk-based paradigms and instead adopts dictionary-based learning by caching a fixed number of interactions for each node. Node representations are then built by aggregating temporal and structural features within the cache.

Self-supervised representation learning: Multiple works explore learning visual representations without labels [19]. The more recent contrastive methods generate random views of images through data augmentations, and then force representations of positive pairs to be similar while pushing apart representations of negative pairs [4, 10]. With the goal of attaining hard negative samples, such methods typically use large batch sizes [4] or memory banks [5, 10]. Non-contrastive methods such as BYOL [9] and VICReg [2] eliminate the need for negative samples through various techniques such as regularization or architecture tricks that avoid representation collapse [13]. Recently, several SSL methods have been adapted to pre-train GNNs [38]. Deep Graph Infomax (DGI) [36] and InfoGCL [39]) rely on mutual information maximization or information bottle-necking between patch-level and graph-level summaries. BGRL [29] adapts BYOL to graphs to eliminate the need for negative samples, which are often memory-heavy in the graph setting. The experiments demonstrate the high degree of scalability of non-contrastive methods and their effectiveness in leveraging both labeled and unlabeled data. In this work, we follow a principled approach for SSL pre-training based on VICReg [1] compared to other methods such as BGRL that rely on architecture tricks and heuristics.

Most adaptations of SSL for dynamic graphs have focused on improving downstream task performance via auxiliary losses rather than learning general pre-trained models. Previous works [11] either use contrastive learning methods, which require high memory and computation due to negative sampling [29], or incorporate weak encoders [30], which leads to performance deterioration, particularly for large-scale graphs. Furthermore, readily adapting prior SSL methods to temporal domains is non-trivial as dynamic graphs can involve heavy distribution shifts. For example, new nodes arrive and others depart, and these arrival patterns occur at different timescales. As a result, there has been limited success in adapting SSL pre-training to dynamic graphs.

Position of our work: DyG2Vec relies on efficient message-passing GNNs without requiring the computationally expensive temporal causality on subgraph sampling [40]. *Our architecture does not use complex memory-based architectures which require designing memory update schemes and can suffer from obsolete node memory for large batch sizes* [42]. While random-walk-based works [12, 37] alleviate these issues with online feature construction through causal walks, such methods are orders of magnitude slower and difficult to parallelize on GPUs [20]. In contrast to prior works, *our method neither maintains a cache or memory for each node nor requires the full history to make predictions.* Instead, it operates on a fixed-size window of the past relations to generate node embeddings. Furthermore, we fall under the message-passing paradigm which can leverage GPU parallelism using cutting-edge frameworks [7]. Last, we propose a joint-embedding architecture that is compatible with recent SSL methods. In our experiments, we show how this allows the model to learn temporal patterns even without direct training on downstream tasks.

3 PROBLEM FORMULATION

A Continuous-Time Dynamic Graph (CTDG) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ is a sequence of $E = |\mathcal{E}|$ interactions, where $\mathcal{X} = (X^V, X^E)$ is the set of input features containing the *node features* $X^V \in \mathbb{R}^{N\times D^V}$ and the *edge features* $X^E \in \mathbb{R}^{E\times D^E}$. $\mathcal{E} = \{e_1, e_2, \ldots, e_E\}$ is the set of interactions. There are $N = |\mathcal{V}|$ nodes, and D^V and D^E are the dimensions of the node and edge feature vectors, respectively. An edge $e_i = (u_i, v_i, t_i, m_i)$ is an interaction between any two nodes $u_i, v_i \in \mathcal{V}$, with $t_i \in \mathbb{R}$ being a continuous timestamp, and $m_i \in X^E$ an edge feature vector. For simplicity, we assume that the edges are undirected and ordered by time (i.e., $t_i \leq t_{i+1}$). A temporal sub-graph $\mathcal{G}_{i,j}$ is defined as a set consisting of all the edges in the interval $[t_i, t_j]$, such that $\mathcal{E}_{ij} = \{e_k \mid t_i \leq t_k < t_j\}$. Any two nodes can interact multiple times throughout the time horizon; therefore, $\mathcal{G}$ is a multi-graph.

Our goal is to learn a model f that maps the input graph to a representation space. The model is a pre-trainable encoder-decoder architecture, $f = (g_\theta, d_\gamma)$. The encoder g_θ maps a dynamic graph to node embeddings $\boldsymbol{H} \in \mathbb{R}^{N\times D^H}$; the decoder d_γ performs a task-specific prediction given the embeddings. The model is parameterized by the encoder/decoder parameters (θ, γ). More concretely,

$$\boldsymbol{H} = g_\theta(\mathcal{G}), \qquad \boldsymbol{Z} = d_\gamma(\boldsymbol{H}; \bar{e}), \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^{D^Y}$ is the prediction of task-specific labels (e.g., edge prediction or source node classification labels) of the target (future) edge $\bar{e}$. The node embeddings $\boldsymbol{H}$ must capture the temporal and structural dynamics of each node such that the future can be accurately predicted from the past, e.g., future edge prediction given past edges. The main distinction of this design is that, unlike previous dynamic graph models [26, 37, 40], the encoder must produce embeddings independent of the downstream task specifications. This special trait can allow the model to be compatible with the SSL paradigm where an encoder is pre-trained separately and then fine-tuned together with a task-specific decoder to predict the labels.

To this end, we present a novel DyG2Vec framework, that can learn rich node embeddings at any timestamp t independent of the downstream task. DyG2Vec is formulated as a two-stage framework. In the first stage, we use a non-contrastive SSL method to learn the model $f^{SSL} = (g_\theta, d_\psi)$ over various sampled dynamic sub-graphs with self-supervision. d_ψ is an SSL decoder that is only used in the SSL pre-training stage. In the second stage, a task-specific decoder d_γ is trained on top of the pre-trained encoder g_θ to compute the outputs for the downstream tasks, e.g., future edge prediction or dynamic node classification [37, 40].

We consider two example downstream tasks: future link prediction (FLP), and dynamic node classification (DNC). In each task, we make a prediction on a set of target (positive) edges $\bar{\mathcal{E}}$. For FLP, this is augmented by a set of negative edges. Each negative edge (u_j, v'_j, t_j, m_j) differs from its corresponding positive edge only in the destination node, $v'_j \neq v_j$, which is selected at random from all nodes. The FLP task is then binary classification for the test set of $2|\bar{\mathcal{E}}|$ edges. In the DNC task, a dynamic label is associated with each node that participates in an interaction. We are provided with $\{(u_j, t_j)\}$, i.e., the source node and interaction time. The goal is to predict the source node labels for the test interactions. *It is important to note that each prediction must be made given only access to the past, i.e., edges before time t_j.* The performance metrics are detailed in Appendix A.2.

4 METHODOLOGY

We now introduce our novel dynamic graph learning framework DyG2Vec, which can achieve downstream task-agnostic representation. We first outline the encoder architecture. We then introduce the window-based downstream training approach. Finally, we present the SSL pre-training approach with a non-contrastive loss function for dynamic graphs.

4.1 DyG2Vec Encoding Model

Our encoder combines a self-attention mechanism for message-passing with a learnable time-encoding module that provides relative time encoding. We also introduce a novel temporal edge encoding that efficiently captures the temporal structural relationship between nodes. The full architecture is outlined in Figure 1.

Temporal Attention Embedding: Given a dynamic graph $\mathcal{G}$, the encoder g_θ computes the embedding $\boldsymbol{h}_i^L \in \mathbb{R}^{D^H}$ of node i through a series of L multi-head attention (MHA) layers [34] that aggregate messages from its L-hop neighborhood [35, 40].

Given a node embedding $\boldsymbol{h}_i^{l-1}$ at layer $l-1$, we uniformly sample N 1-hop neighborhood interactions of node i, $\mathcal{N}(i) = \{e_p, \dots, e_k\} \subseteq \mathcal{E}$. The embedding $\mathbf{h}_i^l$ at layer l is calculated by:

$$\mathbf{h}_i^l = \mathbf{W}_1 \mathbf{h}_i^{l-1} + \mathsf{MHA}^l(\mathbf{q}^l, \mathbf{K}^l, \mathbf{V}^l), \tag{2}$$

$$\mathbf{q}^l = \mathbf{h}_i^{l-1}, \tag{3}$$

$$\mathbf{K}^l = \mathbf{V}^l = [\Phi_p(t_p), \dots, \Phi_k(t_k)] \,. \tag{4}$$

Here, $\mathbf{W}_1$ is a learnable mapping matrix, $\mathsf{MHA}^l(\cdot)$ is a multi-head dot-product attention layer, and $\Phi_p(t_p)$ represents the edge feature vector of edge $e_p = (u_p, v_p, t_p, \boldsymbol{m}_p) \in \mathcal{N}(i)$ at time t_p:

$$\Phi_p(t_p) = [\boldsymbol{h}_{u_p}^{l-1} \mid\mid f_p(t_p) \mid\mid \boldsymbol{m}_p], \tag{5}$$

$$f_p(t_p) = \phi(\bar{t}_i - t_p) + \Theta_p(t_p)\,, \tag{6}$$

$$\bar{t}_i = \max\{t_l \mid e_l \in \mathcal{N}(i)\}\,, \tag{7}$$

where $||$ denotes concatenation and $\phi(t) = [\cos\omega_1 t, \dots, \cos\omega_{D^H} t]$ is a learnable Time2Vec module that helps the model be aware of the relative timespan between a sampled interaction and the most recent interaction of node i in the input graph. $\Theta_p(.)$ is a temporal edge encoding function, described in more detail below. In contrast to TGAT's recursive message passing procedure [40], the message passing in our encoder is 'flat': at every iteration, the same set of node embeddings is used to propagate messages to neighbors. That is, we do not restrict messages to flow towards the source node only but rather treat the *sampled temporal graph as undirected.* This allows the encoder to better capture the multi-hop common neighbors between the target nodes, which are vital to learning the temporal motifs and predicting future interactions. Moreover, unlike CaW [37], *we do not restrict the neighbor sampling to go backwards in time* (i.e. causal sampling) as we found this to be too restrictive and degrade the overall performance on downstream tasks (See Section 6). Lastly, note that the relative time encoding is with respect to the latest timestamp, $\bar{t}_i$, incident to the source and not with respect to the target edge timestamp; hence, allowing the encoding step to be independent of the prediction (decoding) step and making the generated embeddings task-agnostic.

Temporal Edge Encoding: Dynamic graphs often follow evolutionary patterns that reflect how nodes interact over time [17]. For example, in social networks, two people who share many friends are likely to interact in the future. Therefore, we incorporate two simple yet effective temporal encoding methods that provide inductive biases to capture common structural and temporal evolutionary behaviour of dynamic graphs. The temporal edge encoding function is then:

$$\Theta_p(t_p) = \mathbf{W}_2[z_p(t_p) || c_p(t_p)]\,, \tag{8}$$

where we incorporate (i) *Temporal Degree Centrality* $z_p(t_p) \in \mathbb{R}^2$: the concatenated current degrees of nodes u_p and v_p at time t_p; and (ii) *Common Neighbors* $c_p(t_p) \in \mathbb{R}$: the number of common 1-hop neighbors between nodes u_p and v_p at time t_p.

By using the degree centrality as an edge feature, the model is able to learn any bias towards more frequent interactions with high-degree nodes. The number of common neighbors helps capture temporal motifs, and it is known to often have a strong positive correlation with the likelihood of a future interaction [41].

4.2 DyG2Vec Downstream Training

In the downstream training stage, the DyG2Vec model $f = (g_\theta, d_\psi)$ consists of the encoder g_θ and a task-specific decoder d_ψ which is trained using a similar window-based training strategy. The model is trained to make predictions depending on the downstream tasks (e.g., link prediction or node classification). It is important to note that all tasks considered for dynamic graphs involve predicting a (future) target edge given access to the past interactions. However, rather than having access to all past edges, we limit the model to a fixed window of W interactions. That is, to predict a target edge $\bar{e} = (u_j, v_j, t_j, m_j)$, we sample an input (history) graph $\mathcal{G}_{j-W,j}$ from the time interval $\{t_{j-W}, t_j\}$, centered at u_j and v_j, and make a prediction as follows: $\boldsymbol{H} = g_\theta(\mathcal{G}_{j-W,j})$ is the matrix of node embeddings returned by the encoder, and $\boldsymbol{z} = d_\psi(\boldsymbol{H}; \bar{e})$ is the prediction output of the decoder. The model parameters are optimized by training with a loss function $\mathcal{L}_D(\boldsymbol{z}, \boldsymbol{o})$, where $\mathcal{L}_D$ is defined depending on the downstream task and $\boldsymbol{o}$ contains task-specific labels (See Section 3). It is important to note that, unlike previous methods [37, 40], the *embeddings of u_j and v_j are generated through message passing on the same sampled graph.* Consequently, the encoder can better recognize similar historical patterns between the target nodes without the need for costly motif-correlation through counting that is performed in walk-based methods [12, 37].

The window-based training strategy has several major advantages. First, the window acts as a regularizer by providing a natural inductive bias towards recent edges, which are often more predictive of the immediate future. Second, it avoids costly time-based neighborhood sampling [37]. Third, relying on a fixed window-size for message-passing allows for constant memory and computational complexity, which is well-suited to the practical *online streaming* data scenario.

4.3 Self-supervised Pre-training for Dynamic Graphs

Previous work [21] has shown that temporal motifs develop at different timescales throughout a dynamic graph. For example, question-answer patterns on StackOverflow typically take 30 min to develop while messaging patterns on social media platforms can take less than 20 minutes to form. Inspired by such observations, we outline a window-based pre-training strategy where the encoder is trained on a sliding window of the dynamic graph in an effort to learn the fine-grained temporal patterns throughout the time horizon.

Given the full input dynamic graph $\mathcal{G}_{0,E}$, a set of intervals I is generated by dividing the entire time-span $\{t_0, t_E\}$ into $M = \lceil E/S \rceil - 1$ intervals with stride S and interval length W (See Appendix A.2 for details). Let $B \subset I$ be a mini-batch (randomly sampled subset) of intervals. Given B, the sub-graph sampler $m(\mathcal{G}, B; W)$ constructs the mini-batch of input graphs: $\hat{\mathcal{G}} = \{\mathcal{G}_{i,j} \mid [i, j) \in B\}$. In principle, $\mathcal{G}_{i,j} \in \hat{\mathcal{G}}$ is an input graph to the SSL pre-training. The parameter W controls the size of the window while S controls the stride between intervals. In practice, we found that setting $S = 200$ and $W = 32K$ gives a reasonable trade-off to learn both the long-range and short-range patterns within the dynamic graph.

We formulate a joint-embedding architecture [3] for DyG2Vec in which two views of a mini-batch of sub-graphs are generated through random transformations. The transformations are randomly sampled from a distribution defined by a distortion pipeline. The encoder maps the views to node embeddings which are processed by the predictor to generate node representations. We minimize an SSL objective (Eq. 9, described below) to optimize the model parameters end-to-end in the pre-training stage.

Views: The temporal distortion module generates two views of the input graphs $\hat{\mathcal{G}}' = t'(\hat{\mathcal{G}})$ and $\hat{\mathcal{G}}'' = t''(\hat{\mathcal{G}})$ where the transformations t' and t'' are sampled from a distribution $\mathcal{T}$ over a pre-defined set of candidate graph transformations. In this work, we use edge dropout and edge feature masking [29] in the transformation pipeline. See Appendix A.2 for more details.

Embedding: The encoding model g_θ is an Attention-based Message-Passing (AMP) neural network presented in Sec. 4.1. It produces node embeddings $\boldsymbol{H}'$ and $\boldsymbol{H}''$ for the views $\hat{\mathcal{G}}'$ and $\hat{\mathcal{G}}''$ of the input graphs $\mathcal{G}_{i,j}$. We elaborate on the details of the encoder in Sec. 4.1.

Prediction: The decoding head d_γ for our self-supervised learning design consists of a node-level predictor p_ϕ that outputs the final representations $\boldsymbol{Z}'$ and $\boldsymbol{Z}''$, where $\boldsymbol{Z} = p_\phi(\boldsymbol{H})$.

SSL Objective: In order to learn useful representations, we minimize a regularization-based SSL loss function [2]:

$$\mathcal{L}^{SSL} = \lambda s(Z', Z'') + \mu[v(Z') + v(Z'')] + \nu[c(Z') + c(Z'')]\,. \quad (9)$$

In this loss function, the weights λ, μ, and ν control the emphasis placed on each of three regularization terms. The *invariance* term s encourages representations of the two views to be similar. The *variance* term v is included to prevent the well-known collapse problem [13]. The covariance term c promotes maximization of the information content of the representations.

Unlike previous regularization-based SSL approaches [2, 4] in computer vision, we do not use a projector network because the embedding dimensions are relatively small in the graph domain. Following the pre-training stage, we replace the SSL decoder with a task-specific downstream decoder d_ψ that is trained on top of the *frozen* pre-trained encoder.

Table 1: Dynamic Graph Datasets. % Repetitive Edges: % of edges which appear more than once in the dynamic graph.

Dataset	# Nodes	# Edges	# Unique Edges	Edge Features	Node Labels	Bipartite	% Repetitive Edges
Reddit	11,000	672,447	78,516	✓	✓	✓	54%
Wikipedia	9,227	157,474	18,257	✓	✓	✓	48%
MOOC	7,144	411,749	178,443	✓	✓	✓	53%
LastFM	1980	1,293,103	154,993			✓	68%
UCI	1899	59,835	13838			✓	62%
Enron	184	125,235	2215				92%
SocialEvolution	74	2,099,519	2506				97%

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup

Baselines: We compare DyG2Vec to five state-of-the-art baseline models: DyRep [33], JODIE [18], TGAT [40], TGN [26], CaW [37], and NAT [20]. DyRep, JODIE, and TGN sequentially update node embeddings using an RNN. TGAT applies message passing via attention on a sampled temporal subgraph. CaW samples temporal random walks and learns temporal motifs by counting node occurrences in each walk. NAT builds temporal node representations using a cache that stores a limited set of historical interactions for each node.

Downstream Tasks: We evaluate all models on two temporal tasks: future link prediction (FLP), and dynamic node classification (DNC). In FLP, the goal is to predict the probability of future edges occurring given the source, destination, and timestamp. For each positive edge, we sample a negative edge that the model is trained to predict as negative. The DNC task involves predicting the label of the source node of a future interaction. Both tasks are trained using binary cross entropy loss. For FLP, we evaluate all models on the transductive and inductive settings. The latter is a more challenging setting where a model makes a prediction on unseen nodes. See Appendix A.2 for details.

For the FLP task, we report the Average Precision (AP) metric. For the DNC task, we report the area under the curve (AUC) metric due to the prevailing issue of class imbalance in dynamic graphs.

Datasets: We use 7 real-world datasets: Wikipedia, Reddit, MOOC, and LastFM [18]; SocialEvolution, Enron, and UCI [37]. These datasets span a wide range in terms of number of nodes and interactions, time range, and repetition ratio. The dataset statistics are presented in Table 1. We perform the same 70%-15%-15% chronological split for all datasets as in [37]. The datasets are split differently under two settings: Transductive and Inductive. Under the transductive setting, a dataset is split normally by time, i.e., the model is trained on the first 70% of links and tested on the rest. In the inductive setting, we strive to test the model's prediction performance on edges with unseen nodes. Therefore, following [37], we randomly assign 10% of the nodes to the validation and test sets and remove any interactions involving them in the training set. Additionally, to ensure an inductive setting, we remove any interactions not involving these nodes from the test set. All our datasets are publicly available. The code will be publicly available upon publication.

Training Protocols and Hyperparameters: We train and test DyG2Vec under three different evaluation protocols commonly adapted in the SSL community [2, 9]. In the supervised setting, DyG2Vec is initialized with random parameters and trained directly on the downstream tasks and compared to all supervised baselines. In the self-supervised setting, the encoder is pre-trained using

Table 2: Future link Prediction Performance in AP (Mean ± Std). Bold font and ul font represent first- and second-best performance respectively.

Setting	Model	Wikipedia	Reddit	MOOC	LastFM	Enron	UCI	SocialEvol.
Transductive	JODIE	0.956 ± 0.002	0.979 ± 0.001	0.797 ± 0.01	0.691 ± 0.010	0.785 ± 0.020	0.869 ± 0.010	0.847 ± 0.014
	DyRep	0.955 ± 0.004	0.981 ± 1*e*-4	0.840 ± 0.004	0.683 ± 0.033	0.795 ± 0.042	0.524 ± 0.076	0.885 ± 0.004
	TGAT	0.968 ± 0.001	0.986 ± 3*e*-4	0.793 ± 0.006	0.633 ± 0.002	0.637 ± 0.002	0.835 ± 0.003	0.631 ± 0.001
	TGN	0.986 ± 0.001	0.985 ± 0.001	0.911 ± 0.010	0.743 ± 0.030	0.866 ± 0.006	0.843 ± 0.090	0.966 ± 0.001
	CaW	0.976 ± 0.007	0.988 ± 2*e*-4	0.940 ± 0.014	0.903 ± 1*e*-4	0.970 ± 0.001	0.939 ± 0.008	0.947 ± 1*e*-4
	NAT	0.987 ± 0.001	0.991 ± 0.001	0.874 ± 0.004	0.859 ± 1*e*-4	0.924 ± 0.001	0.944 ± 0.002	0.944 ± 0.010
	DyG2Vec	**0.995 ± 0.003**	**0.996 ± 2e-4**	**0.980 ± 0.002**	**0.960 ± 1e-4**	**0.991 ± 0.001**	**0.988 ± 0.007**	**0.987 ± 2e-4**
Inductive	JODIE	0.891 ± 0.014	0.865 ± 0.021	0.707 ± 0.029	0.865 ± 0.03	0.747 ± 0.041	0.753 ± 0.011	0.791 ± 0.031
	DyRep	0.890 ± 0.002	0.921 ± 0.003	0.723 ± 0.009	0.869 ± 0.015	0.666 ± 0.059	0.437 ± 0.021	0.904 ± 3*e*-4
	TGAT	0.954 ± 0.001	0.979 ± 0.001	0.805 ± 0.006	0.644 ± 0.002	0.693 ± 0.004	0.820 ± 0.005	0.632 ± 0.005
	TGN	0.974 ± 0.001	0.954 ± 0.002	0.855 ± 0.014	0.789 ± 0.050	0.746 ± 0.013	0.791 ± 0.057	0.904 ± 0.023
	CaW	0.977 ± 0.006	0.984 ± 2*e*-4	0.933 ± 0.014	0.890 ± 0.001	0.962 ± 0.001	0.931 ± 0.002	0.950 ± 1*e*-4
	NAT	0.986 ± 0.001	0.986 ± 0.002	0.832 ± 1*e*-4	0.878 ± 0.003	0.949 ± 0.010	0.926 ± 0.010	0.952 ± 0.006
	DyG2Vec	**0.992 ± 0.001**	**0.991 ± 0.002**	**0.938 ± 0.010**	**0.979 ± 0.006**	**0.987 ± 0.004**	**0.976 ± 0.002**	**0.978 ± 0.010**

our SSL framework, and the performance is measured under two evaluation protocols: Linear and Semi-supervised Probing. In the linear evaluation setting, the decoder is trained on top of the frozen pre-trained encoder and compared to the supervised counterpart. In the semi-supervised evaluation setting, the decoder is trained on top of the frozen pre-trained encoder on a random portion of the dataset (i.e., a fraction of the target edges). The DyG2Vec encoder performs $L = 3$ layers of message passing. We sample $N = 64$ temporal neighbors at the first hop and 1 neighbor at the second and third hops. All neighbors are sampled uniformly at random. We found that uniform sampling within a window works better than only looking at the latest N neighbors of a node [26, 40]. Other hyperparameters are discussed in Appendix A.2. For the DNC task, following prior work [26], the decoder is trained on top of the frozen encoder that is pre-trained on the future link prediction task unless otherwise explicitly stated.

5.2 Experimental Results

Future Link Prediction: We report the test AP scores for future link prediction in Table 2. Our model outperforms all sequential and message-passing baselines on 7/7 of the datasets in the transductive setting. The gap is particularly large on the UCI and LastFM datasets, where DyG2Vec outperforms the second-best methods (NAT and CaW) by over 4% and 6% respectively. Interestingly, while SocialEvol. is the largest dataset with $\sim 2M$ edges, our model is able to achieve SoTA performance while only using the last 8000 edges to predict any future edge. This further cements the findings in [40] that capturing recent interactions may be more important for certain tasks. Our window-based framework offers a good trade-off between capturing recent interactions and recurrent patterns which both have a major influence on future interactions. In the inductive settings, most methods drop in performance due to difficult nature of predicting over unseen nodes. However, DyG2Vec still outperforms the best methods significantly (e.g., 8% gap for LastFM) which demonstrates its ability to learn temporal motifs rather than overfitting to node identities.

Table 3: Transductive Dynamic Node Classification Performance in AUC (Mean ± Std). Avg. Rank reports the mean rank of a method across all datasets.

Model	Wikipedia	Reddit	MOOC	Avg. Rank ↓
TGAT	0.800 ± 0.010	**0.664 ± 0.009**	0.673 ± 0.006	3.0
JODIE	0.843 ± 0.003	0.566 ± 0.016	0.672 ± 0.002	3.7
Dyrep	**0.873 ± 0.002**	0.633 ± 0.008	0.661 ± 0.012	3.3
TGN	0.828 ± 0.004	0.655 ± 0.009	0.674 ± 0.007	2.3
DyG2Vec	0.824 ± 0.050	0.649 ± 0.020	**0.785 ± 0.005**	2.6

Dynamic Node classification: We evaluate DyG2Vec on 3 datasets for node classification where the labels indicate whether a user will be banned from editing/posting after an interaction. This task is challenging both due to its dynamic nature (i.e., nodes can change labels) and the high class imbalance (only 217 of 157K interactions result in a ban). We measure performance using the AUC metric to deal with the class imbalance. Table 3 shows that DyG2Vec outperforms all baselines on the MOOC dataset significantly by over 10%. For Wikipedia and Reddit, DyG2Vec is within 2 − 5% of the best performance. Overall, none of the methods display the best performance consistently across all 3 datasets. We believe this is due to the high class imbalance problem which makes it a better fit for anomaly detection methods [25].

Training/Inference Speed: Relying on a fixed window of history to produce task-agnostic node embeddings gives DyG2Vec a significant advantage in speed and memory. Figure 2 shows the performance and runtime per epoch of all methods on the three large datasets: LastFM, SocialEvolution and MOOC. DyG2Vec is many orders of magnitude faster than CaW due to the latter's expensive random walk sampling procedure. RNN-based methods such as TGN have a good runtime on LastFM and MOOC; however, they are significantly slower on SocialEvol. which has a small number of nodes (74) but a large number of interactions ($\sim 2M$). This suggests that memory-based methods are slower for settings where a node's memory is updated frequently. Furthermore, while TGAT has a

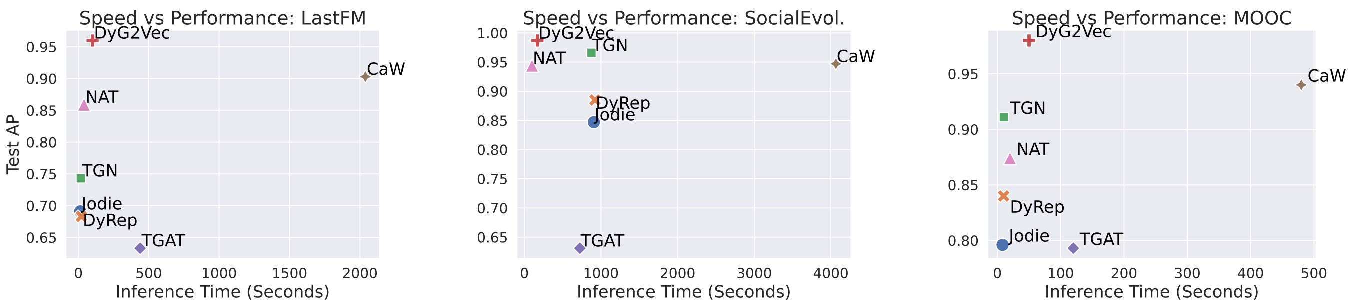

Figure 2: Transductive FLP Performance (Test AP) vs Inference runtime (s) on 3 datasets. Inference time represents the time it takes to predict the whole test set. The test sets are approximately of size 400K, 600K, and 100K edges respectively.

similar AMP encoder, DyG2Vec improves the efficiency and performance significantly. This reveals the significance of the window-based mechanism and the encoder architecture. Overall, DyG2Vec presents the best trade-off between speed and performance. A more detailed complexity analysis is included in Appendix A.1.

Table 4: Linear probing AP results (Mean ± Std) on Transductive Future Link Prediction.

Setting	UCI	Enron	MOOC	LastFM
Random-init	0.865 ± 0.004	0.913 ± 0.007	0.863 ± 0.001	0.817 ± 0.002
Supervised	0.988 ± 0.007	0.991 ± 0.001	0.980 ± 0.002	0.960 ± 1*e*-4
SSL-init	0.954 ± 0.002	0.966 ± 0.001	0.931 ± 0.001	0.930 ± 2*e*-4

SSL for Future Link Prediction: Table 4 reports the transductive AP results for DyG2Vec under 3 different settings. Namely, we compare a random frozen encoder (Random-init) and an SSL pre-trained encoder (SSL-init) with the supervised baseline. The results reveal that our SSL pre-training learns informative node embeddings that are almost on par with the fully supervised baseline. This supports the capability of the non-contrastive methods to learn generic representations across unlabelled large-scale dynamic graphs, which is in line with the findings for other data modalities [2]. The Random-init baseline is surprisingly good, as observed by recent works [29], but is outperformed by the SSL pre-trained encoder. Additional comparisons with weaker dynamic graph SSL methods are discussed in Appendix A.1.

Semi-supervised Learning on Dynamic Node Classification: The DNC task is challenging due to its highly imbalanced labels. In Figure 3, we show that SSL is an effective pre-training strategy for the DNC task, particularly in the low-label data regime where each model is trained on a portion of the target edges. This highlights the potential of SSL to effectively use unlabeled data for representation learning and prevent representations from overfitting to such imbalanced classification tasks.

5.3 Ablation and Sensitivity Analysis

We perform a detailed study on different instances of our framework with 3 datasets. All ablation results are reported in Figure 4.

Window Size: We observe that a large window size works best for most datasets. However, we see a minor drop in performance

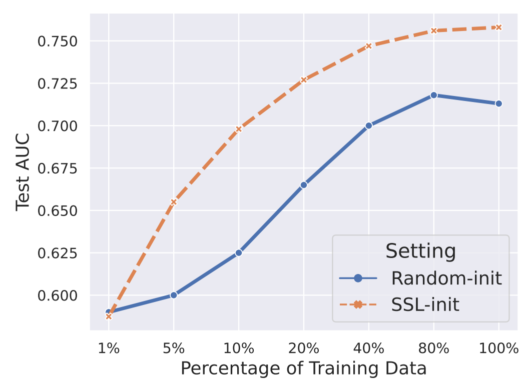

Figure 3: Semi-Supervised Learning on Dynamic Node Classification. For each setting, DyG2Vec was trained on a varying random portion of the training data.

($\sim$ 1%) for MOOC due to the inherently different recurring temporal patterns. As observed by [40], recent and/or recurrent interactions are often the most predictive of future interactions. Therefore, datasets with long range dependencies favor larger window sizes to capture the recurrent patterns while some datasets benefit from an increased bias towards recent interactions. Our window-based framework coupled with uniform neighbor sampling strikes a balance between the two. This shows that the fixed window size also contributes to the performance as it helps limit irrelevant information that is not highly predictive of future interactions. Nonetheless, as we show in Section 6, the attention-based encoder coupled with the time encoding function is able to learn the innate temporal dependencies regardless of the window size.

Number of Layers: Increasing the number of embedding layers improves performance for most datasets benefit from more embedding layers, and this effect is more noticeable for some (e.g., MOOC). This suggests that these datasets contain higher order temporal correlations among the nodes that must be learned using long-range message passing. Overall, the results show that one can choose to sacrifice some performance to further improve the speed of DyG2Vec by decreasing the window size and the number of layers.

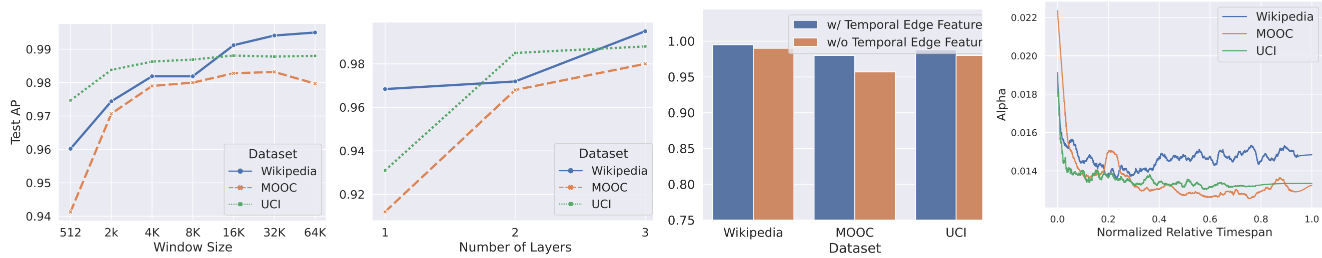

Figure 4: Ablation, sensitivity, and attention analysis on 3 datasets for the FLP transductive task. Last figure plots the Average Attention Weight versus Relative Timespan for DyG2Vec trained with $W = 64K$. The relative timespan is normalized with the maximum timespan across all interactions. A higher timespan means a farther interaction.

Temporal Edge Features: The results show a substantial decrease in performance for MOOC when temporal edge features are removed (i.e., 1-4% drop). This indicates that such temporal edge features provide useful multi-hop information about the evolution of the dynamic graph [41].

6 ANALYSIS

Attention Weight Analysis: A particular advantage of an attention-based architecture is that attention weights allow for easy interpretability of the learned temporal dependencies. In Figure 4 (rightmost plot), we plot the attention weights α_{ij} with respect to the relative timespan. That is, for each test target edge $e_i = (u_i, v_i, t_i)$, we plot the attention weights to the one-hop neighbors of both target nodes, $\{\alpha_{u_i v_p} | v_p \in \mathcal{N}(u_i)\} \cup \{\alpha_{v_i v_k} | v_k \in \mathcal{N}(v_i)\}$, versus the relative timespans, $\{\bar{t}_{u_i} - t_p\} \cup \{\bar{t}_{v_i} - t_k\}$. Here, $\bar{t}_{u_i}$ represents the maximum timestamp incident to node u_i. Therefore, the attention weights indicate how much the model is attending to old and recent interactions. Unsurprisingly, higher importance is given to the most recent interactions. However, both Wikipedia and UCI display higher weights for larger timespans compared to MOOC, indicating that they have long-range dependencies. This explains why performance monotonically increases for Wikipedia as W increases while slightly degrades for the MOOC dataset, as seen in Figure 4.

Neighbor Sampling and Temporal Edge Encoding: In Table 5, we study the effect of neighbor sampling and temporal edge encoding on the downstream FLP task. Although Fig. 4 shows the importance of multi-hop sampling for discovering high-order temporal motifs, we found that DyG2Vec gives more importance to one-hop neighbors for most datasets. In fact, sampling a single neighbor for higher hops gives SoTA performance compared to sampling 20 neighbors per hop (see 1^{st} and 5^{th} rows in Table 5). This suggests that the 1-hop recent interactions within a window are the most representative interactions for future prediction tasks. Moreover, unlike prior random-walk and AMP methods [12, 37, 40], which argue for causal sampling (i.e. sampling backwards in time) to discover evolving temporal motifs, we have found this form of sampling to have little effect on the performance (See 2^{nd} row). Lastly, removing edge encodings almost always hurts performance. In fact, performing causal sampling with 20 neighbors at each hop, as done in TGAT, and removing temporal edge encodings causes up to 8% drop in performance (See last row).

Table 5: Effect of neighbor sampling and temporal edge encoding on performance. The first row is the default setting where we sample 64,1,1 neighbors at the first, second and third hops respectively.

Temporal Edge Encoding	Causal Sampling	Num Neighbors	Wikipedia	MOOC	UCI
✓		64,1,1	**0.995**	0.982	**0.988**
✓	✓	64,1,1	0.993	**0.984**	0.986
		64,1,1	0.990	0.957	0.980
	✓	64,1,1	0.989	0.965	0.976
✓		20,20,20	0.992	0.949	0.981
✓	✓	20,20,20	0.984	0.955	0.971
		20,20,20	0.990	0.927	0.958
	✓	20,20,20	0.982	0.906	0.946

Window-based Pre-training: In Table 6, we show the importance of window-based pre-training to learn the fine-grained temporal motifs of dynamic graphs. The "Full-graph" SSL setting represents applying the SSL loss on the full dynamic graph at once for a total of 300 epochs. Note that this is similar to the pre-training strategy used on static graphs and is difficult to scale for large scale graphs that do not fit to memory. The window-based strategy outperforms the full-graph mode for most datasets, particularly for large graphs (e.g. MOOC and LastFM) where we observe up to a 10% gap.

Table 6: Effect of Window-based pre-training on Linear probing AP results on Transductive Future Link Prediction.

SSL Setting	UCI	Enron	MOOC	LastFM
Window-based	**0.956**	0.965	**0.931**	**0.930**
Full-graph	0.954	**0.966**	0.912	0.838

7 CONCLUSION

We introduce DyG2Vec, a novel window-based encoder-decoder model for dynamic graphs. It is an efficient attention-based message-passing model that utilizes multi-head attention modules to encode node embeddings across time. Furthermore, we present a joint-embedding architecture for dynamic graphs in which two views of temporal sub-graphs are encoded to minimize a non-contrastive loss function. We evaluate the SSL pre-training of DyG2Vec under both linear and semi-supervised protocols and demonstrate the effectiveness of such pre-training on benchmark datasets. Our window-based architecture allows for efficient message-passing and robust prediction abilities. We aim to further explore ways to improve the capacity of the dynamic graph models to learn long-range dependencies. Additionally, it seems promising to investigate other SSL paradigms aligned with temporal graphs.

REFERENCES

[1] Randall Balestriero and Yann LeCun. 2022. Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods. *arXiv preprint arXiv:2205.11508* (2022).

[2] Adrien Bardes, Jean Ponce, and Yann LeCun. 2022. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. In *Proc. Int. Conf. on Learning Representations*.

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In *Proc. Int. Conf. on Neural Information Processing Systems*.

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proc. Int. Conf. on Machine Learning*.

[5] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. 2020. Improved Baselines with Momentum Contrastive Learning. *arXiv preprint arXiv:2003.04297* (2020).

[6] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, and et al. 2021. Time-Series Representation Learning via Temporal and Contextual Contrasting. In *Proc. Int. Joint Conf. on Artificial Intelligence*.

[7] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[8] Quentin Garrido, Yubei Chen, Adrien Bardes, Laurent Najman, and Yann Lecun. 2022. On the duality between contrastive and non-contrastive self-supervised learning. *arXiv preprint arXiv:2206.02574* (2022).

[9] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, and et al. 2020. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Proc. Advances in Neural Information Processing Systems*.

[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2019. Momentum Contrast for Unsupervised Visual Representation Learning. *arXiv preprint arXiv:1911.05722* (2019).

[11] Linpu Jiang, Ke-Jia Chen, and Jingqiang Chen. 2021. Self-Supervised Dynamic Graph Representation Learning via Temporal Subgraph Contrast. *arXiv preprint arXiv:2112.08733* (2021).

[12] Ming Jin, Yuan-Fang Li, and Shirui Pan. 2022. Neural Temporal Walks: Motif-Aware Representation Learning on Continuous-Time Dynamic Graphs. In *Thirty-Sixth Conference on Neural Information Processing Systems*.

[13] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. 2022. Understanding Dimensional Collapse in Contrastive Self-supervised Learning. In *Proc. Int. Conf on Learning Representations*.

[14] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. 2019. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321* (2019).

[15] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* (2020).

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. 2011. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment* (2011).

[18] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proc. Int. Conf. on Knowledge Discovery & Data Mining*.

[19] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[20] Yuhong Luo and Pan Li. 2022. Neighborhood-aware Scalable Temporal Network Representation Learning. In *The First Learning on Graphs Conference*.

[21] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. 2017. Motifs in Temporal Networks. In *ACM Int. Conf. on Web Search and Data Mining*.

[22] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, and et al. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. *Proc. of the AAAI Conference on Artificial Intelligence* (2020).

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, and et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. Advances in Neural Information Processing Systems*.

[24] Mandela Patrick, Yuki M. Asano, Polina Kuznetsova, Ruth Fong, João F. Henriques, Geoffrey Zweig, and Andrea Vedaldi. 2021. Multi-modal Self-Supervision from Generalized Data Transformations. In *Proc. Int. Conf. on Computer Vision*.

[25] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F. Samatova. 2015. Anomaly detection in dynamic networks: a survey. *WIREs Computational Statistics* 7, 3 (2015), 223–247.

[26] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML Workshop on Graph Representation Learning*.

[27] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proc. Int. Conf. on Web Search and Data Mining*.

[28] A. H. Souza, D. Mesquita, S. Kaski, and V. Garg. 2022. Provably expressive temporal graph networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[29] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *Proc. Int. Conf. on Learning Representations*.

[30] Sheng Tian, Ruofan Wu, Leilei Shi, Liang Zhu, and Tao Xiong. 2021. Self-supervised Representation Learning on Dynamic Graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1814–1823.

[31] Sheng Tian, Ruofan Wu, Leilei Shi, Liang Zhu, and Tao Xiong. 2021. Self-Supervised Representation Learning on Dynamic Graphs *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 1814–1823. https://doi.org/10.1145/3459637.3482389

[32] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. 2022. VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training. *arXiv preprint arXiv:2203.12602* (2022).

[33] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *Proc. Int. Conf. on Learning Representations*.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*.

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. Int. Conf. Learning Representations*.

[36] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Lio, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *Proc. Int. Conf. on Learning Representations*.

[37] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *Proc. Int. Conf. on Learning Representations*.

[38] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. 2022. Self-Supervised Learning of Graph Neural Networks: A Unified Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), 1–1.

[39] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Haifeng Chen, and Xiang Zhang. 2021. InfoGCL: Information-Aware Graph Contrastive Learning. In *Proc. Advances in Neural Information Processing Systems*.

[40] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *Proc. Int. Conf. on Representation Learning* (2020).

[41] Lin Yao, Luning Wang, Lv Pan, and Kai Yao. 2016. Link Prediction Based on Common-Neighbors for Dynamic Social Network. *Procedia Computer Science* (2016).

[42] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proc. VLDB Endow.* 15, 8 (jun 2022), 1572–1580. https://doi.org/10.14778/3529337.3529342

A APPENDIX

A.1 Additional Results

A.1.1 Runtime and Computational Complexity. The main runtime overhead lies in how each of the baselines processes the input graph to predict a target edge. CaW samples M L-hop random walks for each target edge. This is followed by an expensive set-based anonymization scheme. To achieve good performance, CaW can require relatively long walks (e.g., for Enron, $L = 5$). On the other hand, memory-based methods and TGAT sample a different L-hop subgraph for each target edge. DyG2Vec samples similar to

TGAT but does so within a constant window size W and without enforcing temporal causality.

Thus, assuming we use sparse operations in Pytorch Geometric [7] for message-passing, the encoding computational complexities are: DyG2Vec = $O(LW)$; CaW = $O(LMN_s)$ and TGN and variants = $O(LN_s)$. Here, N_s represents the maximum number of sampled nodes in an L-hop subgraph. We can see that the main difference is the factor M. The factor N_s comes from the complexity of message passing at each hop (assuming sparse operations). Note that DyG2Vec is limited to $O(W)$ nodes so it does not have this factor.

Table 7: Downstream Freeze test AP Results (after pre-training). DDGCL pre-training and downstream training were run with default parameters described in the work.

Model	MOOC	Enron	UCI	LastFM
DyG2Vec	**93.1**	**96.6**	**95.4**	**93.0**
DDGCL	84.3	83.0	85.3	78.8

A.1.2 Comparison to other dynamic Graph SSL Methods. As mentioned in Section 2, DDGCL [31] proposed a contrastive SSL method for dynamic graphs that learns rich representations by contrasting node embeddings across time. Though experiments show improved performance on the future link prediction and dynamic node classification task, we believe the approach comes with several shortcomings that limit it's advantages in real world graphs. First, it is built on the TGAT [40] encoder which, as seen in Table 2, is a weak encoder; particularly, for large datasets such as LastFM. Second, experiments for the FLP task are limited to the Reddit and Wikipedia datasets which are relatively easy. Lastly, the authors do not experiment under the standard settings in graph SSL literature such as the freeze and semi-supervised settings. Table 7 shows the results for downstream future link prediction under the freeze setting. The results show up to 10% gap compared to DyG2Vec, particularly for datasets where the TGAT encoder under-performs (e.g. Enron, UCI).

A.2 Implementation Details

We train our model using the Pytorch framework [23]. The dynamic graph data and GNN encoder architecture are implemented using Pytorch Geometric [7]. The ReLU activation function is used for all models. The code and datasets will be made publicly available upon acceptance.

Window-based framework: As mentioned in Section 4, during SSl pre-training, the full dynamic graph $G_{0,E}$ is divided into a set of intervals I that is generated by dividing the entire time-span into $M = \lceil E/S \rceil - 1$ intervals with stride S and interval length W:

$$I = \left\{ \left[\max(0, jS - W),\, \min(jS, E) \right) \mid j \in \{1, 2, \ldots, M\} \right\}. \quad (10)$$

Here, W defines the number of edges in an interval and S defines the stride. Note that we include all intervals up to but not including $[E - W, E)$ so that the target interval contains at least one edge.

Decoder Architecture: Denote by t^{max} the timestamp of the latest interaction, within the provided history, incident to node u. For future link prediction, to predict a target interaction (u, v, t), our decoder maps the sum of the two node embeddings of u and v and a time embedding of $t - t^{max}$ to an edge probability. Following [40], the FLP decoder is a 2-layer MLP.

For dynamic node classification, to predict the label of node u for interaction (u, v, t), the decoder maps the source node embedding and time embedding of $t - t^{max}$ to class probabilities. Following [40], the DNC decoder is a 3-layer MLP with a dropout layer with $p = 0.1$.

The time embedding is calculated using a trainable Time2Vec module [14]. The time embedding allows the decoder to be time-aware; hence, possibly output different predictions for the same nodes/edges at different timestamps.

For SSL pre-training, the predictor p_ϕ is a simple 2-layer MLP that maps node embeddings $\boldsymbol{H}$ to node representations $\boldsymbol{Z}$.

Distortion Pipeline: We use the common edge dropout and edge feature dropout distortions. Both distortions are applied with dropout probability $p_d = 0.3$ which we have found to work best in a validation experiment exploring the values $p_d \in \{0.1, 0.15, 0.2, 0.3\}$. The edge feature dropout is applied on the temporal edge encodings introduced in Section 4.1, i.e., $z_p(t_p)$ and $c_p(t_p)$.

Hyper-parameters: We use a constant learning rate of 0.0001 for all datasets and tasks. DyG2Vec is trained for 100 epochs for both downstream and SSL pre-training. The model from the last epoch of pre-training is used for downstream training. For downstream evaluation, we pick the model with the best validation AP performance. Overall, we found that DyG2Vec converges within ~ 50 epochs.

For downstream training, We use a constant window size of $64K$ for all datasets except for MOOC, SocialEvolve, and Enron where we found a smaller window size of $8K$ works best. The batch size is set to 200 target edges. However, the model could be sped up by increasing batch size at the cost of higher memory. During SSL pre-training, we use a constant window size of 32K with stride 200.

Following previous work [26, 40], all dynamic node classification training experiments are performed with L2-decay parameter $\lambda = 0.00001$ to alleviate over-fitting.

A.3 Baselines

Baselines: Following prior work [26, 40], all baselines are trained with a constant learning rate of 0.0001 using the Adam optimizer [16] on batch-size 200 for a total of 50 epochs. The early stopping strategy is used to stop training if validation AP does not improve for 5 epochs. For JODIE [18], DyRep [33], and TGN [26], we use the general framework implemented by [26]. The node memory dimension is set to 172. For the NAT baseline [20], we utilize the results in the paper for the common datasets since the setup is the same. We generate results for the missing datasets with the default hyperparameters.

For TGAT, we use the default hyperparameters of 2 layer neighbor sampling with 20 neighbors sampled at each hop. For the CaW method, we tune the time decay parameter $\alpha \in S$ where $S =$, and length of the walks $m \in \{2, 3, 4, 5\}$ on the validation set. The number of heads for the walking-based attention is fixed to 8.