DyFormer : A Scalable Dynamic Graph Transformer with Provable Benefits on Generalization Ability

Weilin Cong* Yanhong Wu† Yuandong Tian† Mengting Gu† Yinglong Xia†
Chun-cheng Jason Chen† Mehrdad Mahdavi*

Abstract

Transformers have achieved great success in several domains, including Natural Language Processing and Computer Vision. However, its application to real-world graphs is less explored, mainly due to its high computation cost and its poor generalizability caused by the lack of enough training data in the graph domain. To fill in this gap, we propose a scalable Transformer-like dynamic graph learning method named **Dy**namic Graph Trans**former** (DyFormer) with *spatial-temporal encoding* to effectively learn graph topology and capture implicit links. To achieve efficient and scalable training, we propose *temporal-union graph* structure and its associated *subgraph-based node sampling strategy*. To improve the generalization ability, we introduce two complementary *self-supervised pre-training tasks* and show that jointly optimizing the two pre-training tasks results in a smaller Bayesian error rate via an information-theoretic analysis. Extensive experiments on the real-world datasets illustrate that DyFormer achieves a consistent $1\% \sim 3\%$ AUC gain (averaged over all time steps) compared with baselines on all benchmarks. [Code]

1 Introduction

In recent years, graph representation learning has been recognized as a fundamental learning problem and has received much attention due to its widespread use in various domains, including social network analysis [14, 12], traffic prediction [19], knowledge graphs [26, 28], drug discovery [6], and recommender systems [2, 32]. Most existing graph representation learning works focus on static graphs. However, real-world graphs are intrinsically dynamic where nodes and edges can appear and disappear over time. For example, the Facebook social network can be considered as a giant dynamic graph, where a new node is created when a user registers the account, and an edge between two nodes is created when a user connects to another one as a friend. The dynamic nature of real-world graphs motivates graph learning methods that can model temporal evolutionary patterns and predict node properties or future links.

Although dynamic graph is important and has wide application, solving real-world dynamic graph learning problems is more challenging than traditional static graph learning problem due to the following reasons: ***(1) missing or spurious links in dynamic graph***: Real-world static graphs are potentially affected by missing/spurious links, applying Graph Neural Networks (GNNs) on real-world graphs could result in ineffective message aggregation over unrelated neighbors from missing/spurious connections. The issue is more severe on dynamic graphs because GNNs cannot distinguish whether it is missing/spurious links or is the temporal evolutionary pattern of the dynamic graph, which could potentially lead to poor generalization. Although several attempts [22, 18, 10, 29] have been made to generalize the static graph algorithm to dynamic graphs by first learning node representations on each static graph snapshot then aggregating these representations from the temporal dimension, these methods still suffer from the aforementioned missing/spurious links issue. Furthermore, aggregating information on the temporal dimension could further carry such error over time, which can significantly affect downstream task accuracy; ***(2) scalability issue at the temporal dimension***: Unlike a fixed-size static graph, the size of a dynamic graph can increase over time. The complexity of most static graph GNNs is dependent on graph sizes, which makes these algorithms not scalable on large graphs [34, 4]. Dynamic graphs introduce an additional level of complexity dependency on the number of time steps, which makes the computation issue more severe. Motivated by the importance and wide applications of dynamic graphs, we propose DyFormer to solve the aforementioned challenges.

To overcome ***the missing or spurious links issue***, DyFormer leverages the Transformer [25] as the backbone to model all pair-wise node relations using the fully-connected self-attention mechanism. By

*Penn State University.
†Meta AI.

doing so, DyFormer can model the relation between node pairs that have no links existed in the original graph, thus becoming robust to graphs with missing and spurious links. Meanwhile, to fully take advantage of the existing spatial and temporal information from the given dynamic graphs, we generalize the positional encoding to the graph domain using spatial-temporal encoding (Section 3.3) by injecting both spatial and temporal graph evolutionary information as inductive biases into DyFormer, which can help our model better utilize the existing graph structure and learn a graph's evolutionary patterns over time. Furthermore, to alleviate the potential poor generalization ability caused by missing/spurious links, we introduce two complementary dynamic graph pre-training tasks that help DyFormer present a better performance on the downstream tasks (Section 4.1) and a provable benefit on generalization ability using information theory. To improve the ***scalability issue***, we propose to make the complexity independent on both the graph size and the number of time-steps. To achieve this, we first introduce the *temporal-union graph* structure that aggregates graph information from multiple time-steps into a unified meta-graph (Section 3.1). Then, we develop a two-tower architecture (Section 3.4) with a novel subgraph-based node sampling strategy (Section 3.2) to model a subset of nodes with their contextual information. These approaches improve DyFormer's training efficiency and scalability from temporal and spatial perspectives.

To this end, we summarize our contributions as follows: (1) a two-tower Transformer-based method named DyFormer with the spatial-temporal encoding that can capture implicit edge connections in addition to the input graph topology; (2) two complementary pre-training tasks to improve generalization ability and robustness to missing/spurious links, which are proven beneficial using information theory; (3) a *temporal-union graph* data structure that efficiently summarizes the spatial-temporal information of dynamic graphs and a novel sampling strategy that makes DyFormer has complexity independent on graph size and the number of time steps; and (4) a comprehensive evaluation on real-world datasets with ablation studies to validate the effectiveness of DyFormer.

2 Preliminaries and related works

We first define dynamic graphs, then review related works on dynamic graph and graph Transformers.

Dynamic graph definition. The nodes and edges in a dynamic graph may appear and disappear over time. We consider a dynamic graph as a sequence of static graph snapshots with a temporal order $\mathbb{G} := \{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$, where the t-th snapshot graph $\mathcal{G}_t(\mathcal{V}, \mathcal{E}_t)$ is an undirected graph with a shared node set $\mathcal{V}$ of all time steps and an edge set $\mathcal{E}_t$. We also denote its adjacency matrix as $\mathbf{A}_t$. Our goal is to learn the node representation at each time-step t, which can be used for any specific downstream task such as link prediction or node classification. Please notice that our setting is the same as the dynamic graph learning setting in [22, 18], where dynamic graph is defined as a set of temporal ordered snapshot graphs, in which the shared node set $\mathcal{V}$ are updated when new snapshot graph arrives.

Dynamic graph learning. Previous dynamic graph representation learning methods usually extend static graph algorithms by further taking the temporal information into consideration. They can mainly be classified into three categories: (1) *smoothness-based methods* learn a graph autoencoder to generate node embeddings on each graph snapshot and ensure the temporal smoothness of the node embeddings across consecutive time-steps. For example, DyGEM [10] uses the learned embeddings from the previous time-step to initialize the embeddings in the next time-step. DynAERNN applies RNN to smooth node embeddings at different time-steps; (2) *Recurrent-based methods* capture the temporal dependency using RNN. For example, GCRN [23] first computes node embeddings on each snapshot using GCN [3], then feeds the node embeddings into an RNN to learn their temporal dependency. EvolveGCN [18] uses RNN to estimate the GCN weight parameters at different time-steps; (3) *Attention-based methods* use self-attention mechanism for both spatial and temporal message aggregation. For example, DySAT [22] propose to use the self-attention mechanism for both temporal and spatial information aggregation. TGAT [29] encodes the temporal information into the node feature, then applies self-attention on the temporal augmented node features. However, *smoothness-based methods* heavily rely on temporal smoothness and are inadequate when nodes exhibit vastly different evolutionary behaviors, *recurrent-based methods* scale poorly when the number of time-steps increases due to RNN's recurrent nature, *attention-based methods* only consider the self-attention on existing edges and are sensitive to missing/spurious links in graphs. In contrast, DyFormer leverages Transformer to capture the spatial-temporal dependency between all nodes pairs, does not over-relying on the given graph structures, and is less sensitive to missing/spurious links.

Graph Transformers. Recently, several attempts have been made to leverage Transformer for graph representation learning. For example, Graphormer [31] and GraphTransformer [7] use scaled dot-product attention [25] for message aggregation and generalizes the idea of positional encoding to graph domains. GraphBert [35] first samples an egocentric network for each

node, then order all nodes into a sequence based on node importance, and feed into the Transformer. However, GRAPHORMER [31] is only feasible to small molecule graphs and cannot scale to large graphs due to the significant computation cost of full attention; GRAPH-TRANSFORMER [7] only considers the first-hop neighbor aggregation, which makes it sensitive to noisy graphs; GRAPHBERT [35] does not leverage the graph topology and can perform poorly when graph topology is important. In contrast, DYFORMER encodes the input graph structures as an inductive bias to guide the full-attention optimization, which balances the trade-offs between noisy input robustness and efficiently learning an underlying graph structure.

3 Method

In this section, we first introduce the temporal union-graph (Section 3.1) and our sampling strategy (Section 3.2) that can reduce the overall complexity from the temporal and spatial perspectives. Then, we introduce spatial-temporal encoding technique (Section 3.3), describe the two-tower transformer architecture design, and explain how to integrate the spatial-temporal encoding to DYFORMER (Section 3.4). Figure 1 illustrates the overall DYFORMER design.

3.1 Temporal-union graph generation One major challenge of applying Transformers on graph representation learning is its significant computation and memory overhead. In Transformers, the computation cost of self-attention is $\mathcal{O}(|\mathcal{E}|\,d)$ and its memory cost is $\mathcal{O}(|\mathcal{E}|+|\mathcal{V}|\,d)$. When using full attention, the computation graph is fully connected with $|\mathcal{E}| = |\mathcal{V}|^2$, where the overall complexity is quadratic in the graph size. Although LINFORMER [27] and BIGBIRD [33] reduce its complexity to $\mathcal{O}(|\mathcal{V}|d)$ by using sparse self-attention, it is still computationally prohibitive as the size of real-world graph are easily sized to the billion level. On dynamic graphs, this problem can be even more severe if one naively extends the static graph algorithm to a dynamic graph, e.g., first extracting the spatial information of each snapshot graph separately, then jointly reasoning the temporal information on all snapshot graphs [22, 18]. By doing so, the overall complexity grows linearly with the number of time-steps T, i.e., with $\mathcal{O}(|\mathcal{V}|^2Td)$ computation and $\mathcal{O}(|\mathcal{V}|^2T+|\mathcal{V}|Td)$ memory cost. To reduce the dependency of the overall complexity on the number of time-steps, we propose to first aggregate dynamic graphs $\mathbb{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ into a *temporal-union graph* $\mathcal{G}^{\text{union}}(\mathcal{V}, \mathcal{E}')$ then employ DYFORMER on the generated temporal-union graph, where $\mathcal{E}' = \mathit{Unique}\{(i,j) : (i,j) \in \mathcal{E}_t,\ t \in [T]\}$ is the set of all possible unique edges in $\mathbb{G}$. As a result, the overall complexity of DYFORMER does not grow with the number of time-steps. Details on how to leverage spatial-temporal encoding to recover the temporal information of edges are described in Section 3.3.

3.2 Target node driven context node sampling Although the temporal-union graph can alleviate the computation burden from the temporal dimension, due to the overall quadratic complexity of self-attention with respect to the input graph size, scaling the training of Transformer to real-world graphs is still non-trivial. Therefore, a properly designed sampling strategy that makes the overall complexity independent with graph sizes is necessary. Our goal is to design a sub-graph sampling strategy that ensures a fixed number of well-connected nodes and a lower computational complexity. To this end, we propose to first sample a subset of nodes that we are interested in as *target nodes*, then sample their common neighbors as *context nodes*.

Let *target nodes* $\mathcal{V}_{\text{tgt}} \subseteq \mathcal{V}$ be the nodes that we are interested in and want to compute its node representation. For example, for the link prediction task, $\mathcal{V}_{\text{tgt}}$ are the set of nodes that we aim to predict whether they are connected. Then, the *context nodes* $\mathcal{V}_{\text{ctx}} \subseteq \{\mathcal{N}(i) \mid \forall i \in \mathcal{V}_{\text{tgt}}\}$ are sampled as the common neighbors of the target nodes. Notice that since context nodes $\mathcal{V}_{\text{ctx}}$ are sampled as the common neighbors of the target nodes, they can provide local structure information for nodes in the target node set. Besides, since two different nodes in the target node set can be far apart with a disconnected neighborhood, the neighborhood of two nodes can provide an approximation of the global view of the full graph. During sampling, to control the randomness involved in the sampling process, $\mathcal{V}_{\text{ctx}}$ are chosen as the subset of nodes with the top-K joint Personalized PageRank (PPR) score [1] to nodes in $\mathcal{V}_{\text{tgt}}$, where PPR score is a node proximity measure that captures the importance of two nodes in the graph. More specifically, our joint PPR sampler proceeds as follows: First, we compute the approximated PPR vector $\boldsymbol{\pi}(i) \in \mathbb{R}^N$ for all node $i \in \mathcal{V}_{\text{tgt}}$, where the j-th element in $\boldsymbol{\pi}(i)$ can be interpreted as the probability of a random walk to start at node i and end at node j. We then compute the approximated joint PPR vector $\hat{\boldsymbol{\pi}}(\mathcal{V}_{\text{tgt}}) = \sum_{i \in \mathcal{V}_{\text{tgt}}} \boldsymbol{\pi}(i) \in \mathbb{R}^N$. Finally, we select K context nodes where each node $j \in \mathcal{V}_{\text{ctx}}$ has the top-K joint PPR score in $\hat{\boldsymbol{\pi}}(\mathcal{V}_{\text{tgt}})$. In practice, the context node size K is the same as the target node size $|\mathcal{V}_{\text{tgt}}|$.

3.3 Spatial-temporal encoding Given a temporal-union graph, our next step is to translate the spatial-temporal information from snapshot graphs to the temporal-union graph $\mathcal{G}_{\text{union}}$, which can be recognized and leveraged by Transformers. Most classical GNNs

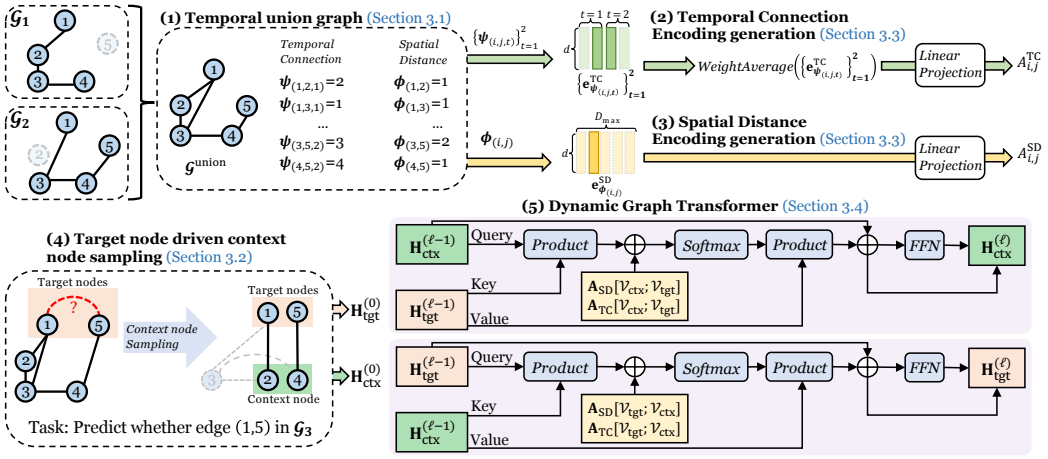

Figure 1: Overview of using DyFormer for link prediction. Given snapshot graphs $\{\mathcal{G}_1, \mathcal{G}_2\}$ as input, **(1)** we first generate the temporal union graph with the considered max shortest path distance $D_{\max} = 5$, and its associated **(2)** temporal connection encoding and **(3)** spatial distance encoding. Then, the encodings are mapped into $A^{\text{TC}}_{i,j}, A^{\text{SD}}_{i,j}$ for each node pairs (i, j) using a fully connected layer. To predict whether an edge exists in $\mathcal{G}_3$, we first **(4)** sample target and context nodes, then apply **(5)** DyFormer to encode target nodes and context nodes separately.

either over-rely on the given graph structure by only considering the first- or higher-order neighbors for feature aggregation [31] (which could make the model fails to capture the inter-relation between nodes that are not connected in the labeled graph) or directly learn graph adjacency without using the given graph structure [5] (which makes the optimization problem challenging because the model has to iteratively learn model parameters and estimate the graph structure). To avoid the above two extremes, we present two simple but effective encoding designs, i.e., *temporal connection encoding* and *spatial distance encoding*, and introduce how to integrate them into DyFormer.

Temporal connection encoding. Temporal connection (TC) encoding is designed to inform DyFormer if an edge (i, j) exists in the t-th snapshot graph. We denote $\mathbf{E}^{\text{TC}} = [\mathbf{e}^{\text{TC}}_{2t-1}, \mathbf{e}^{\text{TC}}_{2t}]^{T}_{t=1} \in \mathbb{R}^{2T \times d}$ as the temporal connection encoding look-up-table where d represents the hidden dimension size, which is indexed by a function $\psi(i, j, t)$ indicating whether an edge (i, j) exists at time-step t. More specifically, we have $\psi(i, j, t) = 2t$ if $(i, j) \in \mathcal{G}_t$, $\psi(i, j, t) = 2t - 1$ if $(i, j) \notin \mathcal{G}_t$ and use this value as an index to extract the corresponding temporal connection embedding from the look-up table for next-step processing. Note that during pre-training or the training on first few time-steps, we need to mask-out certain time-steps to avoid leaking information related to the predicted items (e.g., the temporal reconstruction task in Section. 4.1). In these cases, we set $\psi(i, j, t') = \emptyset$ where t' denotes the time-step we mask-out, and skip the embedding extraction at time t'.

Spatial distance encoding. Spatial distance (SD) encoding is designed to provide DyFormer a global view of the graph structure. The success of Transformer is largely attributed to its global receptive field due to its full attention, i.e., each token in the sequence can attend independently to other tokens and process its representations. Computing full attention requires the model to explicitly capturing the positions dependency between tokens, which can be achieved by either assigning each position an absolute positional encoding or encode the relative distance using relative positional encoding. However, for graphs, the design of unique node positions is not mandatory because a graph is not changed by the permutation of its nodes. To encode the global structural information of a graph in the model, inspired by [31], we adopt a spatial distance encoding that measures the relative spatial relationship between any two nodes in the graph, which is a generalization of the classical Transformer's positional encoding to the graph domain. Let $D_{\max}$ be the maximum shortest path distance (SPD) we considered, where $D_{\max}$ is a hyper-parameter that can be smaller than the graph diameter. Specifically, given any node i and node j, we define $\phi(i, j) = \min\{\text{SPD}(i, j),\ D_{\max}\}$ as the SPD between the two nodes if $\text{SPD}(i, j) < D_{\max}$ and otherwise as $D_{\max}$. Let $\mathbf{E}^{\text{SD}} = [\mathbf{e}^{\text{SD}}_1, \ldots, \mathbf{e}^{\text{SD}}_{D_{\max}}] \in \mathbb{R}^{D_{\max} \times d}$ as the spatial distance lookup-table which is indexed by the $\phi(i, j)$, where $\phi(i, j)$ is used to select the spatial distance encoding $\mathbf{e}^{\text{SD}}_{\phi(i,j)}$ that provides the spatial distance information of two nodes.

Integrate spatial-temporal encoding. We integrate temporal connection encoding and spatial dis-

tance encoding by projecting them as a bias term in the self-attention module. Specifically, to integrate the *spatial-temporal encoding* of node pair (i, j) to DyFormer, we first gather all its associated temporal connection encodings on different time-steps as $\{\mathbf{e}^{\text{TC}}_{\phi(i,j,t)}\}_{t=1}^{T}$. Then, we apply weight average on all encodings over the temporal axis and projected the temporal averaged encoding as a scalar by $A^{\text{TC}}_{i,j} = \mathit{Linear}\big(\mathit{WeightAverage}(\{\mathbf{e}^{\text{TC}}_{\phi(i,j,t)}\}_{t=1}^{T})\big) \in \mathbb{R}$, where the aggregation weight is learned during training. Similarly, to integrate the *spatial distance encoding*, we project the spatial distance encoding of node pair (i, j) as a scalar by $A^{\text{SD}}_{i,j} = \mathit{Linear}(\mathbf{e}^{\text{SD}}_{\phi(i,j)}) \in \mathbb{R}$. Then, $A^{\text{TC}}_{i,j}$ and $A^{\text{SD}}_{i,j}$ are used as the bias term to the self-attention, which we describe in detail in Section 3.4.

3.4 Graph Transformer architecture As shown in Figure 1, each layer in DyFormer consists of two towers (i.e., the target node tower and the context node tower) to encode the target nodes and the context nodes separately. The same set of parameters are shared between two towers. The two-tower structure is motivated by the fact that nodes within each group are sampled independently but there exist neighborhood relationships between inter-group nodes. Only attending inter-group nodes help DyFormer better capture this context information without fusing representations from irrelevant nodes. In the following, we provide details on the context node tower and the detailed formulation for the target node tower can be obtained by switching "ctx" with "tgt".

- First, we compute the self-attentions to aggregate information from target nodes to context nodes (denote as "ctx") and from context nodes to target nodes (denote as "tgt"). Let define $\mathbf{H}^{(\ell)}_{\text{ctx}} \in \mathbb{R}^{|\mathcal{V}_{\text{ctx}}|\times d}$ as the ℓ-th layer output of the context-node tower and $\mathbf{H}^{(\ell)}_{\text{tgt}} \in \mathbb{R}^{|\mathcal{V}_{\text{tgt}}|\times d}$ as the ℓ-th layer output of the target-node tower. Then, the ℓth layer self-attention is
$$\mathbf{A}^{(\ell)}_{\text{ctx}} = \frac{(\texttt{LN}(\mathbf{H}^{(\ell-1)}_{\text{ctx}})\mathbf{W}^{(\ell)}_{Q})(\texttt{LN}(\mathbf{H}^{(\ell-1)}_{\text{tgt}})\mathbf{W}^{(\ell)}_{K})^{\top}}{\sqrt{d}},$$
where $\texttt{LN}(\mathbf{H})$ stands for applying layer normalization on $\mathbf{H}$ and $\mathbf{W}^{(\ell)}_{Q}, \mathbf{W}^{(\ell)}_{K}$ are weight matrices.
- Then, we integrate spatial-temporal encoding as a bias term to self-attention as
$$\mathbf{P}^{(\ell)}_{\text{ctx}} = \mathbf{A}^{(\ell)}_{\text{ctx}} + \mathbf{A}_{\text{TC}}[\mathcal{V}_{\text{ctx}}; \mathcal{V}_{\text{tgt}}] + \mathbf{A}_{\text{SD}}[\mathcal{V}_{\text{ctx}}; \mathcal{V}_{\text{tgt}}],$$
where $\mathbf{A}_{\text{TC}}[\mathcal{V}_A; \mathcal{V}_B]$, $\mathbf{A}_{\text{SD}}[\mathcal{V}_A; \mathcal{V}_B]$ denote the the matrix form of the projected temporal connection and spatial distance self-attention bias with row and column indexed by $\mathcal{V}_A$ and $\mathcal{V}_B$.[1]
- After that, we use the normalized $\mathbf{P}^{(\ell)}_{\text{ctx}}$ and $\mathbf{P}^{(\ell)}_{\text{tgt}}$ to propagate information between two towers, i.e.,
$$\mathbf{Z}^{(\ell)}_{\text{ctx}} = \mathit{Softmax}(\mathbf{P}^{(\ell)}_{\text{ctx}})\texttt{LN}(\mathbf{H}^{(\ell-1)}_{\text{tgt}})\mathbf{W}^{(\ell)}_{V} + \mathbf{H}^{(\ell-1)}_{\text{ctx}},$$
- Finally, a residual connected feed-forward network is applied to the aggregated message to produce the final output $\mathbf{H}^{(\ell)}_{\text{ctx}} = \mathit{FFN}(\texttt{LN}(\mathbf{Z}^{(\ell)}_{\text{ctx}})) + \mathbf{Z}^{(\ell)}_{\text{ctx}}$ where $\mathit{FFN}(\cdot)$ denotes the multi-layer feed-forward network. The final layer output of the target node tower $\mathbf{H}^{(L)}_{\text{tgt}}$ will be used to compute the loss defined in Section 4.

[1] Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the element at the i-th row and j-th column is denoted as $A_{i,j}$, the submatrix formed from row $\mathcal{I}_{\text{row}} = \{a_1, \ldots, a_r\}$ and columns $\mathcal{I}_{\text{col}} = \{b_1, \ldots, b_s\}$ is denoted as $\mathbf{A}[\mathcal{I}_{\text{row}}; \mathcal{I}_{\text{col}}]$.

4 DyFormer training

Transformers usually require a significant amount of supervised data to guarantee their generalization ability on unseen data. However, existing dynamic graph datasets are relatively small and may not be sufficient to train a powerful Transformer. To overcome this challenge, we propose to first pre-train DyFormer with two complementary self-supervised objective functions (in Section 4.1). Then, we fine-tune DyFormer using the supervised objective function (in Section 4.2). Notice that the same set of snapshot graphs but different objective functions are used for pre-training and fine-tuning. Finally, via an information-theoretic analysis, we show that the representation can enjoy a better generalization ability on downstream tasks by optimizing our pre-training losses (in Section 4.3).

4.1 Pre-training We introduce a *temporal reconstruction loss* $\mathcal{L}_{\text{recon}}(\mathbf{\Theta})$ and a *multi-view contrastive loss* $\mathcal{L}_{\text{view}}(\mathbf{\Theta})$ as self-supervised object functions. Then, our overall pre-taining loss is $\mathcal{L}_{\text{pre-train}}(\mathbf{\Theta}) = \mathcal{L}_{\text{recon}}(\mathbf{\Theta}) + \gamma\mathcal{L}_{\text{view}}(\mathbf{\Theta})$, where γ is a hyper-parameter that balances the importance of two pre-taining tasks as in Figure 2.

Temporal reconstruction loss. To ensure that the spatial-temporal encoding is effective and can inform DyFormer the temporal dependency between multiple snapshot graphs, we introduce a temporal reconstruction loss as our first pre-training objective. Our goal is to reconstruct the t-th graph snapshot $\mathcal{G}_t$'s structure using all graph snapshot $\mathbb{G}$. Let $\mathbf{H}^{(L)}_{\text{tgt}}(t)$ denote the target-node tower's final layer output computed on $\mathbb{G}$. To decode the graph structure of graph snapshot $\mathcal{G}_t$, we use a fully connected layer as the temporal structure decoder that takes $\mathbf{H}^{(L)}_{\text{tgt}}(t)$ as input and output $\mathbf{E}(t) = \mathit{Linear}(\mathbf{H}^{(L)}_{\text{tgt}}(t)) \in \mathbb{R}^{|\mathcal{V}_{\text{tgt}}|\times d}$ with $\mathbf{e}_i(t) \in \mathbb{R}^d$ denotes the i-th row of $\mathbf{E}(t)$. Then, the temporal reconstruction loss is $\mathcal{L}_{\text{recon}}(\mathbf{\Theta}) = \sum_{t=1}^{T} \mathit{LinkPredLoss}(\{\mathbf{e}_i(t)\}_{i\in\mathcal{V}_{\text{tgt}}}, \mathcal{V}_{\text{tgt}}, \mathcal{E}_t)$, where $\sigma(\cdot)$ is Sigmoid function and $\mathit{LinkPredLoss}(\{\mathbf{x_i}\}_{i\in\mathcal{S}}, \mathcal{S}, \mathcal{E}) :=$
$$\sum_{i,j\in\mathcal{S}} \Big(- \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{x}_i^{\top}\mathbf{x}_j)) - \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{x}_i^{\top}\mathbf{x}_j)) \Big).$$

Multi-view contrastive loss. Recall that $\mathcal{V}_{\text{ctx}}$ is constructed by *deterministically* selecting the common

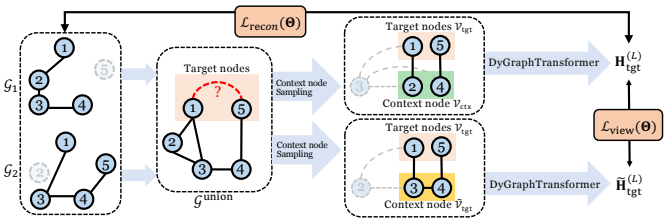

Figure 2: Pre-training: Given snapshot graphs $\{\mathcal{G}_1, \mathcal{G}_2\}$ as input, we first generate the temporal union graph. Then, we sample the target node $\mathcal{V}_{\text{tgt}}$ and two different set of context nodes $\mathcal{V}_{\text{ctx}}, \widetilde{\mathcal{V}}_{\text{ctx}}$. After that, we apply DyFormer on $\{\mathcal{V}_{\text{tgt}}, \mathcal{V}_{\text{ctx}}\}$ and $\{\mathcal{V}_{\text{tgt}}, \widetilde{\mathcal{V}}_{\text{ctx}}\}$ to output $\mathbf{H}_{\text{tgt}}^{(L)}$ and $\widetilde{\mathbf{H}}_{\text{tgt}}^{(L)}$. We optimize $\mathcal{L}_{\text{view}}(\mathbf{\Theta})$ by maximizing the similarity between $\mathbf{H}_{\text{tgt}}^{(L)}$ and $\widetilde{\mathbf{H}}_{\text{tgt}}^{(L)}$, and optimize $\mathcal{L}_{\text{recon}}(\mathbf{\Theta})$ by recovering snapshot graphs using $\mathbf{H}_{\text{tgt}}^{(L)}$.

neighbors of $\mathcal{V}_{\text{tgt}}$ with the top-K PPR score. Then, we introduce $\widetilde{\mathcal{V}}_{\text{ctx}}$ as the subset of the common neighbors of $\mathcal{V}_{\text{tgt}}$ *randomly* sampled with sampling probability of each node proportional to its PPR score. Since a different set of context nodes are provided for the same set of target nodes, $\{\mathcal{V}_{\text{tgt}}, \widetilde{\mathcal{V}}_{\text{ctx}}\}$ provides an alternative view of $\{\mathcal{V}_{\text{tgt}}, \mathcal{V}_{\text{ctx}}\}$ when computing the representation for nodes in $\mathcal{V}_{\text{tgt}}$. Notice that although the provided context nodes are different, since they have the same target nodes, it is natural to expect the calculated representation have high similarity. We denote $\mathbf{H}_{\text{tgt}}^{(L)}$ and $\widetilde{\mathbf{H}}_{\text{tgt}}^{(L)}$ as the final layer model output that are computed on $\{\mathcal{V}_{\text{tgt}}, \mathcal{V}_{\text{ctx}}\}$ and $\{\mathcal{V}_{\text{tgt}}, \widetilde{\mathcal{V}}_{\text{ctx}}\}$. To this end, we introduce our second self-supervised objective function as $\mathcal{L}_{\text{view}}(\mathbf{\Theta}) = \|\mathbf{H}_{\text{tgt}}^{(L)} - SG(\widetilde{\mathbf{H}}_{\text{tgt}}^{(L)})\|_{\text{F}}^2 + \|SG(\mathbf{H}_{\text{tgt}}^{(L)}) - \widetilde{\mathbf{H}}_{\text{tgt}}^{(L)}\|_{\text{F}}^2$, where SG denotes stop gradient.

4.2 Fine-tuning To apply the pre-trained model on downstream tasks, we choose to fine-tune the pre-trained model with downstream task objective functions. Here, we take link prediction as an example. Our goal is to predict the existence of a link at time $T+1$ using information up to time T. Let $\mathbf{H}_{\text{tgt}}^{(L)}(\{\mathcal{G}_j\}_{j=1}^t)$ denote the final output of DyFormer using snapshot graphs $\{\mathcal{G}_j\}_{j=1}^t$. Then, the link prediction loss is $\mathcal{L}_{\text{LinkPred}}(\mathbf{\Theta}) = \sum_{t=1}^{T-1} \mathit{LinkPredLoss}(\mathbf{H}_{\text{tgt}}^{(L)}(\{\mathcal{G}_j\}_{j=1}^t), \mathcal{V}_{\text{tgt}}, \mathcal{E}_{t+1})$.

4.3 Importance of pre-training In this section, we show that our pre-training objectives can improve the generalization error under mild assumptions and results in a better performance on downstream tasks. Let X denote the input random variable, S as the self-supervised signal (also known as a different view of input X), and $Z_X = f(X), Z_S = f(S)$ as the representations that are generated by a deterministic mapping function f. In our setting, we have the sampled sub-graph of temporal-union graph $\mathcal{G}^{\text{union}}$ induced by node $\{\mathcal{V}_{\text{tgt}}, \mathcal{V}_{\text{ctx}}\}$ as input X, the sampled subgraph of $\mathcal{G}^{\text{union}}$ induced by node $\{\mathcal{V}_{\text{tgt}}, \widetilde{\mathcal{V}}_{\text{ctx}}\}$ as self-supervised signal S, and DyFormer as f that computes the representation of X, S by $Z_X = f(X), Z_S = f(S)$. Besides, we introduce the task-relevant information as Y, which refers to the information that is required for downstream tasks. For example, when the downstream task is link prediction, Y can be the ground truth graph structure about which we want to reason. Notice that in practice we have no access to Y during pre-training and it is only introduced as the notation for analysis. Furthermore, let $H(A)$ denote entropy, $H(A|B)$ denote conditional entropy, $I(A;B)$ denote mutual information, and $I(A;B|C)$ denote conditional mutual information. More details and preliminaries on information theory are deferred to Appendix D.

In the following, we study the generalization error of the learned representation Z_X under the binary classification setting. We choose *Bayes error rate* (i.e., the lowest possible test error rate a binary classifier can achieve) as our evaluation metric, which can be formally defined as $P_e = 1 - \mathbb{E}[\max_y \mathrm{P}(Y = y|Z_X)]$. Before proceeding to our result, we make the following assumption on input X, self-supervised signal S, and task-relevant information Y.

Assumption 1. *We assume the task-relevant information is shared between the input random variable X, self-supervised signal S, i.e., we have $I(X;Y|S) = 0$ and $I(S;Y|X) = 0$.*

We argue the above assumption is mild because input X and self-supervised signal S are two different views of the data, and are expected to contain task-relevant information Y. In Proposition 4.1, we make connections between the Bayes error rate and pre-training losses, which explains why the proposed pre-training losses are helpful for downstream tasks. Proof in Appendix D.

Proposition 4.1. *We can upper bound Bayes error rate by $P_e \leq 1 - \exp(-H(Y) + I(Z_X;X) - I(Z_X;X|Y))$, and reduce the upper bound of P_e by (1) maximizing the mutual information $I(Z_X;X)$ between the learned representation Z_X and input X, which can be achieved by minimizing temporal reconstruction loss $\mathcal{L}_{recon}(\mathbf{\Theta})$, and (2) minimizing the task-irrelevant information between the learned representation Z_X and input X, which can be achieved by minimizing our multi-view loss $\mathcal{L}_{view}(\mathbf{\Theta})$.*

The Proposition 4.1 suggests that if we can create a different views S of our input data X such that both X and S contain the task-relevant information Y, then by jointly optimizing two pre-training losses can result in the representation Z_X with a lower Bayes error rate P_e.

5 Experiments

We evaluate DyFormer using dynamic graph link prediction, which has been widely used in [22, 10] to compare its performance with a variety of static and dynamic graph representation learning baselines. Results on node classification is deferred to Appendix B.

Datasets. The detailed data statistics are summarized in Table 1, where the dynamic graph is defined as a set of temporal ordered snapshot graph. Following the procedure as described in [22], the graph snapshots are created by splitting the data using suitable time windows such that each snapshot has an equitable number of interactions. In each snapshot, the edge weights are determined by the number of interactions.

Table 1: Dataset statistics.

	RDS	**UCI**	**Yelp**	**ML-10M**	**SNAP-Wikipedia**	**SNAP-Reddit**
$\|\mathcal{V}\|$	167	1,809	6,569	20,537	9,227	11,000
$\|\mathcal{E}\|$	1,521	16,822	95,361	43,760	157,474	672,447
T	100	13	16	13	11	11

Link prediction. To compare DyFormer with baselines, we follow the evaluation strategy in [10, 37, 22] by training a logistic regression classifier taking two node embeddings as input for dynamic graph link prediction. Specifically, we learn the dynamic node representations on snapshot graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ and evaluate DyFormer by predicting links at $\mathcal{G}_{T+1}$. For evaluation, we consider all links in $\mathcal{G}_{T+1}$ as positive examples and an equal number of sampled unconnected node pairs as negative examples. We split 20% of the edge examples for training the classifier, 20% of examples for hyper-parameters tuning, and the rest 60% of examples for model performance evaluation following the practice of existing studies (e.g., [22]). We evaluate the link prediction performance using Micro and Macro scores, where the Micro is calculated across the link instances from all the time-steps while the Macro is computed by averaging the AUC at each time-step. During inference, all nodes in the testing set (from 60% edge samples in $\mathcal{G}_{T+1}$) are selected as the target nodes. To scale the inference of the testing sets of any sizes, we compute the full-attention by first splitting all self-attentions into multiple chunks then iteratively compute the self-attention in each chunk. Since only a fixed number of self-attention is computed at each iteration, we significantly reduce DyFormer's inference memory consumption. We also repeat all experiments three times with different random seeds.

5.1 Experiment results

The effectiveness of DyFormer. Table 2 indicates the state-of-the-art performance of our approach on link prediction tasks, where DyFormer achieves a consistent $1\% \sim 3\%$ Macro gain on all datasets. Besides, DyFormer is more stable when using different random seeds observed from a smaller standard deviation of the AUC score. To better understand the behaviors of different methods from a finer granularity, we compare the model performance at each time-step in Figure 4 and observe that the performance of DyFormer is relatively more stable than other methods over time. Besides, we additionally report the results of dynamic link prediction evaluated only on unseen links at each time-step. Here, we define unseen links as the ones that first appear at the prediction time-step but are not in the previous graph snapshots. From Table 3, we find that although all methods achieve a lower AUC score, which may be due to the new link prediction is more challenging, DyFormer still achieves a consistent $1\% \sim 3\%$ Macro AUC-score gain.

Table 2: Comparing DyFormer with baselines using *Micro-* and *Macro-AUC* on real-world datasets.

Method	**AUC**	**RDS**	**UCI**	**Yelp**	**ML-10M**
Node2Vec	Micro	81.10 ± 0.87	81.41 ± 0.60	68.93 ± 0.33	90.50 ± 0.83
	Macro	82.85 ± 0.86	81.39 ± 0.76	67.38 ± 0.49	89.48 ± 0.62
GraphSAGE	Micro	85.49 ± 0.96	79.85 ± 2.62	62.36 ± 1.01	86.31 ± 0.97
	Macro	86.64 ± 0.89	78.45 ± 2.01	58.36 ± 0.91	90.23 ± 0.90
DynAERNN	Micro	80.56 ± 0.77	79.29 ± 1.90	71.54 ± 0.83	87.01 ± 0.88
	Macro	80.16 ± 0.91	83.81 ± 1.25	72.29 ± 0.58	89.04 ± 0.67
DynGEM	Micro	79.29 ± 1.01	76.36 ± 0.83	69.43 ± 1.09	79.80 ± 0.88
	Macro	81.94 ± 1.97	78.22 ± 0.99	69.93 ± 0.78	84.86 ± 0.49
DySAT	Micro	83.89 ± 0.92	83.10 ± 0.99	69.00 ± 0.22	88.91 ± 0.87
	Macro	83.60 ± 0.68	86.32 ± 1.46	69.42 ± 0.25	90.63 ± 0.91
EvolveGCN	Micro	85.35 ± 0.87	85.81 ± 0.50	68.99 ± 0.67	92.79 ± 0.21
	Macro	86.53 ± 0.76	84.18 ± 0.72	69.41 ± 0.26	93.45 ± 0.19
DyFormer	Micro	$\mathbf{88.77 \pm 0.50}$	$\mathbf{87.91 \pm 0.32}$	$\mathbf{73.39 \pm 0.21}$	$\mathbf{95.30 \pm 0.36}$
	Macro	$\mathbf{89.77 \pm 0.46}$	$\mathbf{88.49 \pm 0.43}$	$\mathbf{74.31 \pm 0.23}$	$\mathbf{96.16 \pm 0.22}$

Table 3: Comparison of *Micro-* and *Macro-AUC* on real-world datasets restricted to new edges.

Method	**AUC**	**RDS**	**UCI**	**Yelp**	**ML-10M**
Node2Vec	Micro	75.62 ± 1.42	75.31 ± 0.83	68.83 ± 0.29	88.92 ± 0.79
	Macro	76.25 ± 0.85	75.82 ± 0.96	68.00 ± 0.51	88.01 ± 0.50
GraphSAGE	Micro	80.21 ± 0.87	76.56 ± 1.91	61.97 ± 1.00	85.18 ± 0.89
	Macro	79.99 ± 0.78	75.94 ± 1.88	58.49 ± 0.89	89.31 ± 0.93
DynAERNN	Micro	68.43 ± 1.13	77.39 ± 2.10	70.82 ± 0.93	86.89 ± 0.75
	Macro	68.18 ± 1.23	81.82 ± 1.71	71.56 ± 0.77	89.45 ± 0.53
DynGEM	Micro	72.43 ± 1.62	74.72 ± 0.73	69.23 ± 1.76	77.18 ± 1.96
	Macro	74.49 ± 2.21	76.34 ± 0.78	70.67 ± 1.32	82.62 ± 0.49
DySAT	Micro	76.28 ± 1.34	81.18 ± 1.09	69.12 ± 0.21	88.21 ± 0.64
	Macro	76.87 ± 1.21	83.43 ± 1.57	69.20 ± 0.20	88.98 ± 0.87
EvolveGCN	Micro	78.36 ± 0.91	81.99 ± 0.73	68.73 ± 0.64	90.91 ± 0.32
	Macro	79.18 ± 1.01	82.18 ± 0.76	68.63 ± 0.30	91.45 ± 0.29
DyFormer	Micro	$\mathbf{82.78 \pm 0.56}$	$\mathbf{85.78 \pm 0.99}$	$\mathbf{73.32 \pm 0.22}$	$\mathbf{93.01 \pm 0.23}$
	Macro	$\mathbf{82.89 \pm 0.52}$	$\mathbf{86.21 \pm 0.56}$	$\mathbf{73.88 \pm 0.22}$	$\mathbf{93.56 \pm 0.21}$

The effectiveness of pre-training. We compare the performance of DyFormer with/without pre-training. As shown in Figure 3, DyFormer 's performance is significantly improved if we first pre-train it with the self-supervised loss then fine-tuning on downstream tasks. When comparing the AUC scores at each time-step, we observe that DyFormer without pre-training has a lower performance but a larger variance. This may be due to the vast number of training parameters in DyFormer , which potentially requires more data to be trained well. The self-supervised pre-training

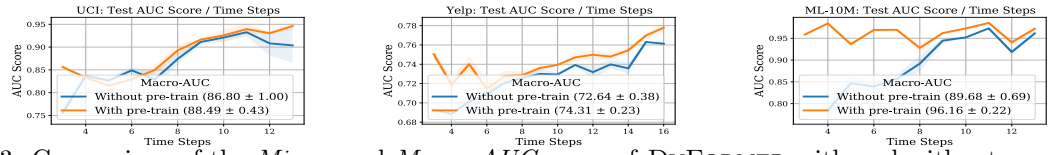

Figure 3: Comparison of the *Micro-* and *Macro-AUC* score of DyFormer with and without pre-training.

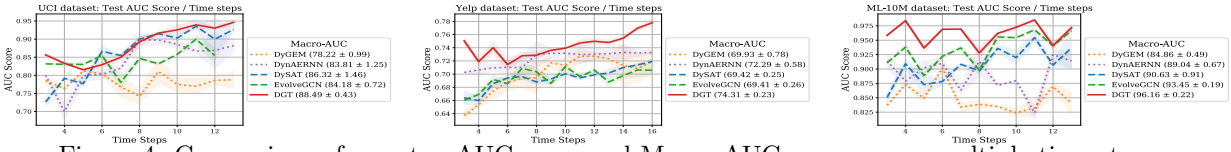

Figure 4: Comparison of per-step AUC-score and Macro AUC-score across multiple time steps.

Table 4: Comparison of DyFormer and its variants with input graph with different noisy level.

	Method	AUC	10%	20%	50%
UCI	1-hop attention	Micro	82.97 ± 0.56	81.23 ± 0.78	77.85 ± 0.66
		Macro	83.01 ± 0.61	82.10 ± 0.60	78.43 ± 0.67
	Full attention	Micro	86.98 ± 0.51	86.10 ± 0.57	84.36 ± 0.49
		Macro	86.12 ± 0.57	85.93 ± 0.59	85.51 ± 0.51
Yelp	1-hop attention	Micro	70.00 ± 0.20	68.55 ± 0.21	65.32 ± 0.22
		Macro	69.94 ± 0.20	68.45 ± 0.23	65.61 ± 0.15
	Full attention	Micro	70.99 ± 0.20	71.74 ± 0.19	70.93 ± 0.21
		Macro	71.64 ± 0.18	71.67 ± 0.21	69.93 ± 0.21

alleviates this by utilizing additional unlabeled data.

Results on dataset with missing/spurious links. We study the effect of noisy input on the performance of DyFormer using *UCI*[16, 17] and *Yelp* datasets. We achieve this by randomly selecting 10%, 20%, 50% of the node pairs and changing their connection status either from connected to not-connected or from not-connected to connected. As shown in Table 4, although the performance of both using full-attention and 1-hop attention decreases as the noisy level increases, the performance of using full-attention aggregation is more stable and robust as the noisy level changes. This is because 1-hop attention relies more on the given structure, while full-attention only take the give structure as a reference and learns the "ground truth" graph structure by gradient descent update.

More results. Due to the space limit, more ablation study results are deferred to Appendix A, which includes ablation study on the effectiveness of spatial-temporal encoding, the number of layers in DyFormer , two-tower and single-tower model architecture, self-attention mechanism, and computation cost.

6 Conclusion

In this paper, we introduce DyFormer for dynamic graph representation learning, which can efficiently leverage the graph topology and capture implicit edge connections. To further improve the generalization ability, two complementary pre-training tasks are introduced. To handle large-scale dynamic graphs, a temporal-union graph structure and a target-context node sampling strategy are designed for an efficient and scalable training. Extensive experiments on real-world dynamic graphs show that DyFormer presents significant performance gains over several state-of-the-art baselines. Potential future directions include exploring GNNs on continuous dynamic graphs and studying its expressive power.

Acknowledgements

This work was supported in part by NSF grant 2008398.

References

[1] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science*, 2006.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. In *International Conference on Knowledge Discovery & Data Mining*, 2017.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 2016.

[4] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations (ICLR)*, 2020.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[6] Kien Do, Truyen Tran, and Svetha Venkatesh. Graph transformation policy network for chemical reaction prediction. In *International Conference on Knowledge Discovery & Data Mining*, 2019.

[7] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[8] Meir Feder and Neri Merhav. Relations between entropy and error probability. *IEEE Transactions on Information theory*, 1994.

[9] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 2020.

[10] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *CoRR*, 2018.

[11] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, 2016.

[12] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[13] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 2015.

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

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

[16] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*, 2013.

[17] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social networks*, 2009.

[18] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Conference on Artificial Intelligence*, 2020.

[19] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. Semi-supervised user geolocation via graph convolutional networks. In *Proceedings of the Association for Computational Linguistics*, 2018.

[20] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[21] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[22] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*, 2018.

[23] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, 2018.

[24] Yao-Hung Hubert Tsai, Yue Wu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Self-supervised learning from a multi-view perspective. *arXiv preprint arXiv:2006.05576*, 2020.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[26] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, 2019.

[27] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[28] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. In *International Conference on Knowledge Discovery & Data Mining*, 2019.

[29] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*, 2020.

[30] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. 2021.

[31] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.

[32] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, 2018.

[33] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020.

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

[35] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[36] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

[37] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang. Dynamic Network Embedding by Modelling Triadic Closure Process. In *Conference on Artificial Intelligence*, 2018.

A More experiment results

The effectiveness of spatial-temporal encoding. In Table 5, we conduct an ablation study by independently removing two encodings to validate the effectiveness of spatial-temporal encoding. We observe that even without any encoding (i.e., ignoring the spatial-temporal graph topologies), due to full-attention, DyFormer is still very competitive comparing with the state-of-the-art baselines in Table 2. However, we also observe a $0.6\% \sim 4.6\%$ performance gain when adding the spatial connection and temporal distance encoding, which empirically shows their effectiveness.

Table 5: Comparison of the *Micro-* and *Macro-AUC* of with and without temporal-connection (TC) and spatial-distance (SD) encoding on the real-world datasets.

Method	AUC	UCI	Yelp	ML-10M
Both encoding	Micro	**87.91 ± 0.32**	**73.39 ± 0.21**	**95.30 ± 0.36**
	Macro	**88.49 ± 0.43**	**74.31 ± 0.23**	**96.16 ± 0.22**
Without encoding	Micro	83.27 ± 0.29	72.82 ± 0.37	91.81 ± 0.43
	Macro	83.87 ± 0.47	73.80 ± 0.38	92.59 ± 0.35
Only TC encoding	Micro	84.78 ± 0.31	73.36 ± 0.26	94.51 ± 0.37
	Macro	84.60 ± 0.42	74.31 ± 0.25	95.43 ± 0.29
Only SD encoding	Micro	87.01 ± 0.46	72.98 ± 0.32	92.34 ± 0.40
	Macro	87.99 ± 0.47	73.90 ± 0.36	93.13 ± 0.33

The effectiveness of stacking more layers. When stacking more layers, traditional GNNs usually suffer from the over-smoothing [36, 30] and result in a degenerated performance. We study the effect of applying more DyFormer layers and show results in Table 6. In contrast to previous studies, DyFormer has a relatively stable performance and does not suffer much from performance degradation when the number of layers increases. This is potentially due to that DyFormer only requires a shallow architecture since each individual layer is capable of modeling longer-range dependencies due to full-attention. Besides, the self-attention mechanism can automatically attend importance neighbors, therefore alleviate the over-smoothing and bottleneck effect.

Table 6: Comparison of the *Micro-* and *Macro-AUC* score of DyFormer with different number of layers.

Method	AUC	UCI	Yelp	ML-10M
2 layers	Micro-AUC	87.89 ± 0.43	74.30 ± 0.21	94.99 ± 0.21
	Macro-AUC	88.31 ± 0.53	74.29 ± 0.23	96.08 ± 0.15
4 layers	Micro-AUC	87.42 ± 0.36	**73.39 ± 0.21**	95.30 ± 0.36
	Micro-AUC	88.35 ± 0.37	**74.31 ± 0.23**	**96.16 ± 0.22**
6 layers	Micro-AUC	**87.91 ± 0.32**	74.30 ± 0.20	**95.35 ± 0.28**
	Micro-AUC	**88.49 ± 0.43**	74.28 ± 0.22	96.11 ± 0.18

Comparing two-tower to single-tower architecture. In Table 7, we compare the performance of DyFormer with single- and two-tower design where a single-tower means a full-attention of over all pairs of target and context nodes. We observe that the two-tower DyFormer has a consistent performance gain (0.5% Micro- and Macro) over the single-tower on Yelp and ML-10M [13]. This may be due to that the nodes within the target or context node set are sampled independently while inter-group nodes are likely to be connected. Only attending inter-group nodes helps DyFormer better capturing these contextual information without fusing representations from irrelevant nodes.

Table 7: Comparison of the *Micro-* and *Macro-AUC* of DyFormer using single-tower and two-tower model architecture on the real-world datasets.

Method	AUC	UCI	Yelp	ML-10M
Single-tower	Micro	87.86 ± 0.60	72.95 ± 0.20	94.80 ± 0.81
	Macro	88.27 ± 0.68	73.81 ± 0.21	95.49 ± 0.57
Two-tower	Micro	**87.91 ± 0.32**	**73.39 ± 0.21**	**95.30 ± 0.36**
	Macro	**88.49 ± 0.43**	**74.31 ± 0.23**	**96.16 ± 0.22**

Comparing K-hop attention with full-attention. To better understand full-attention, we compare it with 1-hop and 3-hop attention. These variants are evaluated based on the single-tower DyFormer to include all node pairs into consideration. Table 8 shows the results where we observe that the full-attention presents a consistent performance gain around $1\% \sim 3\%$ over the other two variants. This demonstrates the benefits of full-attention when modeling implicit edge connections in graphs with a larger receptive fields comparing to its K-hop counterparts.

Table 8: Comparison of the *Micro-* and *Macro-AUC* of *full attention* and *K-hop attention* using the single-tower architecture on the real-world datasets.

Method	AUC	UCI	Yelp	ML-10M
Full attention	Micro	**87.86 ± 0.60**	**72.95 ± 0.20**	**94.80 ± 0.81**
	Macro	**88.27 ± 0.68**	**73.81 ± 0.21**	**95.49 ± 0.57**
1-hop neighbor	Micro	84.62 ± 0.31	71.33 ± 0.43	91.88 ± 0.73
	Macro	85.10 ± 0.15	71.45 ± 0.45	92.18 ± 0.44
3-hop neighbor	Micro	87.01 ± 0.89	71.19 ± 0.22	91.83 ± 0.92
	Macro	87.48 ± 0.88	72.31 ± 0.22	92.33 ± 0.82

Computation time and memory consumption. In Table 9, we compare the memory consumption and epoch time on the last time step of ML-10M and Yelp dataset. We chose the last time step of these two datasets because its graph size is relatively larger than others, which can provide a more accurate time and memory estimation. The memory consumption is record by `nvidia-smi` and the time is recorded by function `time.time()`. During pre-training, DyFormer samples 256 context node and 256 context node at each iteration. During fine-tuning, DyFormer first 256 positive links (links in the graph) and sample $2,560$ negative links (node pairs that do not exist in the graph), then treat all nodes in the sampled node pairs at target nodes and sample the same amount of context nodes. Notice that although the same sampling size hyper-parameter

is used, since the graph size and the graph density are different, the actual memory consumption and time are also different. For example, since the Yelp dataset has more edges with more associated nodes for evaluation than ML-10M, the memory consumption and time are required on Yelp than on ML-10M dataset.

Table 9: Comparison of the epoch time and memory consumption of DyFormer with baseline methods.

Dataset	Method	Memory	Epoch / Total time
ML-10M	DySAT	9.2GB	97.2s/4276.8s (45 epochs)
	EvolveGCN	13.6GB	6.9s/821.1s (120 epochs)
	DyFormer (Pretrain)	6.5GB	38.9s/986.5s (89 epochs)
	DyFormer (Finetune)	10.1GB	2.98s/62.2s (22 epochs)
Yelp	DySAT	5.4GB	29.4s/4706.4s (160 epochs)
	EvolveGCN	7.5GB	19.14s/1091.2s (57 epochs)
	DyFormer (Pretrain)	21.3GB	11.8s/413.5s (34 epochs)
	DyFormer (Finetune)	21.3GB	21.41s/521.6s (23 epochs)

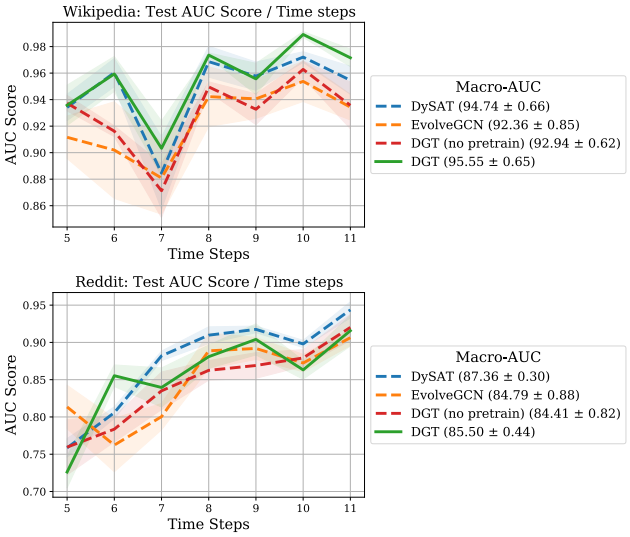

Figure 5: Comparison of DyFormer with baselines across multiple time steps, where the Macro score is reported in the box next to the curves

B Node classification results

In this section, we show that although DyFormer is orginally designed for the link prediction task, the learned representation of DyFormer can be also applied to binary node classification. We evaluate DyFormer on Wikipedia and SNAP-Reddit dataset, where dataset statistic is summarized in Table 1. The snapshot is created in a similar manner as the link prediction task. As shown in Table 10 and Figure 5, DyFormer performs around 0.7% better than all baselines on the Wikipedia dataset and around 0.7% better than EvolveGCN on SNAP-Reddit dataset. However, the results DyFormer on the SNAP-Reddit dataset[2] is slightly lower than DySAT. This is potentially due to DyFormer is less in favor of a dense graph, e.g., SNAP-Reddit dataset, with very dense graph structure information encoded by spatial-temporal encodings.

Table 10: Comparison of the *Micro-* and *Macro-AUC* on the real-world datasets for binary node classification task.

Method	AUC	SNAP-Wikipedia	SNAP-Reddit
DySAT	Micro	94.69 ± 0.46	$\mathbf{87.35 \pm 0.28}$
	Macro	94.74 ± 0.66	$\mathbf{87.36 \pm 0.30}$
EvolveGCN	Micro	92.31 ± 0.68	84.72 ± 0.89
	Macro	92.36 ± 0.85	84.79 ± 0.88
DyFormer (w/o pre-training)	Micro	92.90 ± 0.84	82.37 ± 0.78
	Macro	92.94 ± 0.62	84.41 ± 0.82
DyFormer (w/ pre-training)	Micro	$\mathbf{95.49 \pm 0.66}$	85.48 ± 0.43
	Macro	$\mathbf{95.55 \pm 0.65}$	85.50 ± 0.44

[2]The Reddit dataset are collected and released by SNAP at Stanford University. Please refer to `http://snap.stanford.edu/jodie/` for details.

C Experiment configuration

C.1 Hardware specification and environment

We run our experiments on a single machine with Intel i9-10850K, Nvidia RTX 3090 GPU, and 32GB RAM memory. The code is written in Python 3.7 and we use PyTorch 1.4 on CUDA 10.1 to train the model on the GPU.

C.2 Baseline hypter-parameters tuning

Baselines. [3] We compare with several state-of-the-art methods as baselines including both static and dynamic graph learning algorithms. For static graph learning algorithms, we compare against Node2Vec[11] and GraphSAGE[12]. To make the comparison fair, we feed these static graph algorithms the same temporal-union graph used in DyFormer rather than any single graph snapshots. For dynamic graph learning algorithms, we compare against DynAERNN[9], DynGEM[10], DySAT[22], and EvolveGCN [18]. We use the official implementations for all baselines and select the best hyper-parameters for both baselines and DyFormer.

Model setups. The hyper-parameters for each dataset is selected using grid search on the validation set. More specifically, we select the negative sampling ratio (i.e., number of positive edge/number of negative edge)

[3]We only compare with dynamic graph algorithms that takes a set of temporal ordered snapshot graph as input, and leave the study on other dynamic graph structure (e.g., continuous time-step algorithms [15, 29, 20] and datasets) as a future direction.

in the range $\{0.01, 0.1, 1\}$, number of the self-attention head is selected in the range $\{8, 16\}$, feature dimension is selected in the range $\{128, 256\}$, number of layers in the range $\{2, 4, 6\}$, maximum shortest path distance $D_{\max}$ in the range $\{2, 3, 5\}$ on the validation set, mini-batch size as 512, and the pre-training loss weight $\gamma = 1$.

We tune the hyper-parameters of baselines following their recommended guidelines.

NODE2VEC[4]: We use the default setting as introduced in [11]. More specifically, for each node we use 10 random walks of length 80, context window size as 10. The in-out hyper-parameter p and return hyper-parameter q are selected by grid-search in range $\{0.25, 0.5, 1, 2, 5\}$ on the validation set.

GRAPHSAGE[5]: We use the default setting in [12]. More specifically, we train two layer GNN with neighbor sampling size 25 and 10. The neighbor aggregation is selected by grid-search from "mean-based aggregation", "LSTM-based aggregation", "max-pooling aggregation", and "GCN-based aggregation" on the validation set. In practice, GCN aggregator performs best on RDS [21], and UCI, and max-pooling aggregator performs best on Yelp and ML-10M.

DYNGEM and DYNAERNN[6]: We use the default setting as introduced in [10] and [9]. The scaling and regularization hyper-parameters is selected by grid-search in range $\alpha \in \{10^{-6}, 10^{-5}\}$, $\beta \in \{0.1, 1, 2, 5\}$, and $\nu_1, \nu_2 \in \{10^{-6}, 10^{-4}\}$ on the validation set.

DYSAT[7]: We use the default setting and model architecture as introduced in [22]. The co-occurring positive node pairs are sampled by running 10 random walks of length 40 for each node. The negative sampling ratio is selected by grid-search in the range $\{0.01, 0.1, 1\}$, number of the self-attention head is selected in the range $\{8, 16\}$, and the feature dimension is selected in the range $\{128, 256\}$ on the validation set.

EVOLVEGCN[8]: We use the default setting and model architecture as introduced in [18]. We train both EvolveGCN-O and EvolveGCN-H and report the architecture with the best performance on the validation set. In practice, EvolveGCN-O performs best on UCI, Yelp, and ML-10M, EvolveGCN-H performs best on Enron and RDS.

D Pre-training can reduce the irreducible error

D.1 Preliminary Data processing inequality. Random variables X, Y, Z are said to form a *Markov chain* $X \to Y \to Z$ if the joint probability mass function can be written as $P(x, y, z) = p(x)p(y|x)p(z|y)$. Suppose random variable X, Y, Z forms a Markov chain $X \to Y \to Z$, then we have $I(X; Y) \geq I(X; Z)$.

Bayes error and entropy. In the binary classification setting, *Bayes error rate* is the lowest possible test error rate (i.e., irreducible error), which can be formally defined as

$$P_e = \mathbb{E}\left[1 - \max_y p(Y = y|X)\right], \tag{D.1}$$

where Y denotes label and X denotes input. [8] derives an upper bound showing the relation between *Bayes error rate* with entropy:

$$-\log(1 - P_e) \leq H(Y|X). \tag{D.2}$$

The above inequality is used as the foundation of our following analysis.

D.2 Proof of Proposition 4.1 In the following, we utilize the analysis framework developed in [24] to show the importance of two pre-training loss functions. By using Eq. D.2, we have $-\log(1 - P_e) \leq H(Y|Z_X)$. By rearanging the above inequality, we have the following upper bound on the Bayes error rate

(D.3)

$$\begin{aligned} P_e &\leq 1 - \frac{1}{\exp\big(H(Y|Z_X)\big)} \\ &\underset{(a)}{=} 1 - \frac{1}{\exp\big(H(Y) - I(Z_X; Y)\big)} \\ &\underset{(b)}{=} 1 - \frac{1}{\exp\big(H(Y) - I(Z_X; X) + I(Z_X; X|Y)\big)}, \end{aligned}$$

where equality (a) is due to $I(Z_X; Y) = H(Y) - H(Y|Z_X)$, equality (b) is due to $I(Z_X; Y) = I(Z_X; X) - I(Z_X; X|Y) + I(Z_X; Y|X)$ and $I(Z_X; Y|X) = 0$ because $Z_X = f(X)$ is a deterministic mapping given input X. Our goal is to find the deterministic mapping function f to generate Z_X that can maximize $I(Z_X; X) - I(Z_X; X|Y)$, such that the upper bound on the right hand side of Eq. D.3 is minimized. We can achieve this by:

- Maximizing the mutual information $I(Z_X; X)$ between the representation Z_X to the input X.
- Minimizing the task-irrelevant information $I(Z_X; X|Y)$, i.e., the mutual information between the representation Z_X to the input X given task-relevant information Y.

In the following, we first show that minimizing $\mathcal{L}_{\text{recon}}(\mathbf{\Theta})$ can maximize the mutual information $I(Z_X; X)$, then we show that minimizing $\mathcal{L}_{\text{view}}(\mathbf{\Theta})$ can minimize the task irrelevant information $I(Z_X; X|Y)$.

Maximize mutual information $I(Z_X; X)$. By the relation between mutual information and entropy

[4] https://github.com/aditya-grover/node2vec
[5] https://github.com/williamleif/GraphSAGE
[6] https://github.com/palash1992/DynamicGEM
[7] https://github.com/aravindsankar28/DySAT
[8] https://github.com/IBM/EvolveGCN

$I(Z_X;X) = H(X) - H(X|Z_X)$, we know that maximizing the mutual information $I(Z_X;X)$ is equivalent to minimizing the conditional entropy $H(X|Z_X)$. Notice that we ignore $H(X)$ because it is only dependent on the raw feature and is irrelevant to feature representation Z_X. By the definition of conditional entropy, we have

(D.4)
$$\begin{aligned}
H(X|Z_X) &= \sum_{z_x \in \mathcal{Z}_\mathcal{X}} p(z_x) H(X|Z_X = z_x) \\
&= \sum_{z_x \in \mathcal{Z}_\mathcal{X}} p(z_x) \sum_{x \in \mathcal{X}} -p(x|z_x) \log p(x|z_x) \\
&= \sum_{z_x \in \mathcal{Z}_\mathcal{X}} \sum_{x \in \mathcal{X}} -p(x, z_x) \log p(x|z_x) \\
&= \mathbb{E}_{\mathrm{P}(X,Z_X)}\Big[-\log \mathrm{P}(X|Z_X)\Big] \\
&= \min_{Q_{\boldsymbol{\theta}}} \mathbb{E}_{\mathrm{P}(X,Z_X)}\Big[-\log Q_{\boldsymbol{\theta}}(X|Z_X)\Big] \\
&\quad - \mathrm{KL}\Big(\mathrm{P}(X|Z_X) \| Q_{\boldsymbol{\theta}}(X|Z_X)\Big) \\
&\leq \min_{Q_{\boldsymbol{\theta}}} \mathbb{E}_{\mathrm{P}(X,Z_X)}\Big[-\log Q_{\boldsymbol{\theta}}(X|Z_X)\Big]
\end{aligned}$$

where $Q_{\boldsymbol{\theta}}(\cdot|\cdot)$ is a variational distribution with $\boldsymbol{\theta}$ represent the parameters in $Q_{\boldsymbol{\theta}}$ and KL denotes KL-divergence.

Therefore, maximizing mutual information $I(Z_X;X)$ can be achieved by minimizing $\mathbb{E}_{\mathrm{P}_{X,Z_X}}[-\log Q_{\boldsymbol{\theta}}(X|Z_X)]$. By assuming $Q_{\boldsymbol{\theta}}$ as the categorical distribution and $\boldsymbol{\theta}$ as a neural network, minimizing $\mathbb{E}_{\mathrm{P}_{X,Z_X}}[-\log Q_{\boldsymbol{\theta}}(X|Z_X)]$ can be think of as introducing a neural network parameterized by $\boldsymbol{\theta}$ to predict the input X from the learned representation Z_X by minimizing the binary cross entropy loss.

Minimize the task irrelevant information $I(Z_X;X|Y)$. Recall that in our setting, input X is the node features of $\{\mathcal{V}_{\text{target}}, \mathcal{V}_{\text{context}}\}$ and the subgraph induced by $\{\mathcal{V}_{\text{target}}, \mathcal{V}_{\text{context}}\}$. The self-supervised signal S is node features of $\{\mathcal{V}_{\text{target}}, \widetilde{\mathcal{V}}_{\text{context}}\}$ and the subgraph induced by $\{\mathcal{V}_{\text{target}}, \widetilde{\mathcal{V}}_{\text{context}}\}$. Therefore, it is natural to make the following mild assumption on the input random variable X, self-supervised signal S, and task relevant information Y.

Assumption 2. *We assume tall task-relevant information is shared between the input random variable X, self-supervised signal S, i.e., we have $I(X;Y|S) = 0$ and $I(S;Y|X) = 0$.*

In the following, we show that minimizing $I(Z_X;X|Y)$ can be achieved by minimizing $H(Z_X|S)$. From data processing inequality, we have $I(X;Y|S) \geq I(Z_X;Y|S) \geq 0$. From Assumption 2, we have $I(X;Y|S) = 0$, therefore we know $I(Z_X;Y|S) = 0$. By the relation between mutual information and entropy, we have

(D.5)
$$\begin{aligned}
I(Z_X;X|Y) &= H(Z_X|Y) - H(Z_X|X,Y) \\
&\underset{(a)}{=} H(Z_X|Y) \\
&= H(Z_X|S,Y) + I(Z_X;S|Y) \\
&= H(Z_X|S) - I(Z_X;Y|S) + I(Z_X;S|Y) \\
&\underset{(b)}{=} H(Z_X|S) + I(Z_X;S|Y) \\
&\underset{(c)}{\leq} H(Z_X|S) + I(X;S|Y),
\end{aligned}$$

where equality (a) is due to $H(Z_X|X,Y) = 0$ since $Z_X = f(X)$ and f is a deterministic mapping, equality (b) is due to $I(Z_X,Y|S) = 0$, and inequality (c) is due to data processing inequality.

From Eq. D.4, we know that

(D.6)
$$\begin{aligned}
H(Z_X|S) &= \mathbb{E}_{\mathrm{P}(S,Z_X)}[-\log \mathrm{P}(Z_X|S)] \\
&\leq \min_{Q'_{\boldsymbol{\phi}}} \mathbb{E}_{\mathrm{P}(S,Z_X)}\Big[-\log Q'_{\boldsymbol{\phi}}(Z_X|S)\Big].
\end{aligned}$$

By assuming $Q'_{\boldsymbol{\phi}}$ as the Gaussian distribution and $\boldsymbol{\phi}$ as a neural network, minimizing $\mathbb{E}_{\mathrm{P}_{S,Z_X}}[-\log Q_{\boldsymbol{\phi}}(Z_X|S)]$ can be think of as introducing a neural network parameterized by $\boldsymbol{\phi}$ that take S as input and output $Z_S = \text{NeuralNetwork}_{\boldsymbol{\phi}}(S)$, then minimize the mean-square error between Z_X and Z_S.