HOT: Higher-Order Dynamic Graph Representation Learning with Efficient Transformers

Maciej Besta[1*] **Afonso Claudino Catarino**[1*] **Lukas Gianinazzi**[1] **Nils Blach**[1]
Piotr Nyczyk[2] **Hubert Niewiadomski**[2] **Torsten Hoefler**[1]

[1]Department of Computer Science, ETH Zurich; [2]Cledar

Abstract

Many graph representation learning (GRL) problems are dynamic, with millions of edges added or removed per second. A fundamental workload in this setting is dynamic link prediction: using a history of graph updates to predict whether a given pair of vertices will become connected. Recent schemes for link prediction in such dynamic settings employ Transformers, modeling individual graph updates as single tokens. In this work, we propose HOT: a model that enhances this line of works by harnessing higher-order (HO) graph structures; specifically, k-hop neighbors and more general subgraphs containing a given pair of vertices. Harnessing such HO structures by encoding them into the attention matrix of the underlying Transformer results in higher accuracy of link prediction outcomes, but at the expense of increased memory pressure. To alleviate this, we resort to a recent class of schemes that impose hierarchy on the attention matrix, significantly reducing memory footprint. The final design offers a sweetspot between high accuracy and low memory utilization. HOT outperforms other dynamic GRL schemes, for example achieving 9%, 7%, and 15% higher accuracy than – respectively – DyGFormer, TGN, and GraphMixer, for the MOOC dataset. Our design can be seamlessly extended towards other dynamic GRL workloads.

1 Introduction

Analyzing graphs in a dynamic setting, where edges and vertices can be arbitrarily modified, has become an important task. For example, the Twitter social network may experience even 500 million new tweets in a single day, while retail transaction graphs consisting of billions of transactions are generated every year [3]. Other domains where dynamic networks are often used are transportation [8, 51, 126, 131, 132], physical systems [59, 87, 96], scientific collaboration [22, 24, 65, 100, 129], and others [2, 42, 71, 74, 101, 133, 135, 138]. A fundamental task in such a dynamic graph setting is predicting links that would appear in the future, based on the history of previous graph modifications. This task is crucial for understanding the future of the graph datasets, which enables more accurate graph analytics in real-time in production settings such as recommendation systems in online stores. Moreover, it enables better performance decisions, for example by adjusting load balancing strategies with the knowledge of future events [15, 16, 95].

Recent years brought intense developments into harnessing graph representation learning (GRL) techniques for the above-described tasks [65], resulting in a broad domain called dynamic graph representation learning (DGRL) [9, 55, 66, 99, 100, 129]. Initially, various approaches have been proposed based on Temporal Random Walks [63, 120], Sequential Models [36, 116], Memory Networks [27, 103], or the Dynamic Graph Neural Networks (GNNs) paradigm, in which node embeddings are iteratively updated based on the information passed by their neighbors [34, 81, 92, 110, 118, 127], while considering temporal data from the past [92, 127]. However, the most recent and powerful schemes such as DyGFormer [134] instead directly harness the Transformer model [111] for the DGRL tasks. The intuition is that dynamic graphs can be modeled as a sequence of updates over time [134], and they could hence benefit from Transformers by treating these updates as individual tokens. This enhanced predictions for dynamic graphs [134].

[*]First-author contribution

In parallel to the developments in DGRL, many GRL schemes have harnessed higher-order (HO) graph structure [1, 11, 25, 25, 45, 46, 77, 108, 128]. Fundamentally, they harness relationships between vertices that go beyond simple edges, for example triangles. Incorporating HO graph structures results in fundamentally more powerful predictions for many workloads [83]. This line of works, however, has primarily focused on static GRL.

In this work, we embrace the HO graph structures for higher accuracy in DGRL (**contribution #1**). For this, we harness two powerful HO structures: k-hop neighbors and more general subgraphs containing a given pair of vertices. We harness these structures by encoding them appropriately into the attention matrix of the underlying Transformer. This results in higher accuracy of link prediction. However, harnessing HO leads to substantially larger memory utilization, which limits its applicability. This is because, intuitively, to make a prediction in the temporal setting, one needs to consider the history of past updates. As HO increases the number of graph elements to be considered in such a history, the number of entities to be used is substantially inflated. For example, when relying on the Transformer architecture (as in DyGFormer), the number of tokens that must be used is significantly increased when incorporating HO. For example, when incorporating x 2-hop neighbors for each vertex, the token count increases by a factor of x. This becomes even higher for $k > 2$ and more complex HO structures such as triangles. Hence, one needs to decrease the considered length of the history of updates to make the computation feasible. But then, considering shorter history entails lower accuracy, effectively annihilating benefits gained from using HO in the first place.

We tackle this by harnessing the state of the art outcomes from the world of Transformers (**contribution #2**), where *one imposes hierarchy into the traditionally "flat" attention matrix*. Examples of such recent *hierarchical Transformer models* are RWKV [86], Swin Transformer [78], hierarchical BERT [84], Nested Hierarchical Transformer [140], Hift [32], or Block-Recurrent Transformer [60]. We employ these designs to alleviate the memory requirements of the attention matrix by dividing it into parts, computing attention locally within each part, and then using such blocks to obtain the final outcomes. For concreteness, we pick Block-Recurrent Transformer [60], but our approach can be used with others. Our final outcome, a model called HOT, supported by a theoretical analysis (**contribution #3**), ensures long-range and high-accuracy dynamic link prediction. It outperforms all other dynamic GRL schemes (**contribution #4**), for example achieving 9%, 7%, and 15% higher accuracy than – respectively – DyGFormer, TGN, and GraphMixer, on the MOOC dataset. Our work illustrates the importance of HO structures in the temporal dimension.

2 Background

Static graphs are usually modeled as a tuple $G = (V, E, f, w)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $f : V \to \mathbb{R}^{d_N}$ are the node features and $w : E \to \mathbb{R}^{d_E}$ are the edge features. **Dynamic graphs** are more complex to represent, as it is necessary to capture their evolution over time. In this work, we focus on the commonly used *Continuous-Time Dynamic Graph (CTDG)* representation [34, 36, 63, 71, 79, 81, 92, 110, 116, 118, 120, 127]. A CTDG is a tuple $(G^{(0)}, T)$, where $G^{(0)} = (V^{(0)}, E^{(0)}, f^{(0)}, w^{(0)})$ represents the initial state of the graph and T is a set of tuples of the form (*timestamp*, *event*) representing events to be applied to the graph at given timestamps. These events could be node additions/deletions, edge additions/deletions, or feature updates.

Assume a CTDG $(G^{(0)}, T)$, a timestamp $t \in \mathbb{N}$, and an edge (u, v). In **dynamic link prediction**, the task is to decide, whether there is some event e pertaining to (u, v), such that $(t, e) \in \{ (t', e) \in T \mid t' = t \}$, while only considering the CTDG $(G^{(0)}, \{ (t', e) \in T \mid t' < t \})$. This problem is usually considered in two settings, the *transductive* setting (predicting links between nodes that were seen during training) and the *inductive* setting (predicting links between nodes not seen during training).

Transformer [111] harnesses the multi-head attention mechanism in order to overcome the inherent sequential design of recurrent neural networks (RNNs) [7, 29, 105]. Transformer has been effectively applied to different ML tasks, including but not limited to image recognition and time-series forecasting [41, 76]. It has also laid the ground for many recent advances in generative AI [31, 89]. We focus on the encoder-only architecture, following past DGRL works [134]. Let $x = (x_1, \dots x_n)$ be a sequence of n d-dimensional inputs $x_i \in \mathbb{R}^d$. We consider the input matrix $X \in R^{n \times d}$, a single input token is represented as an individual row in X. Transformer details are in Appendix A.

The runtime of Transformer increases quadratically with the length of the input sequence. As such, many efforts have been made to make it more efficient [12, 37, 38, 90, 122]. Some of these efforts culminated in the **Block-Recurrent Transformer** [60], a model which tries to bring together the advantages of Transformers and LSTMs [57], a special kind of RNN. As this is one of the principal building blocks of our model, we will briefly describe its architecture in the following.

The Block-Recurrent Transformer is essentially an extension of the Transformer-XL model [37] and the sliding-window attention [12] mechanism. Long input sequences are divided into segments of size S, and further divided into blocks of size B. Following the sliding-window attention mechanism, instead of applying attention to the whole sequence, the Block-Recurrent Transformer applies it on each block individually. The elements of each block may attend to recurrently computed state vectors as well. With B state vectors, we get attention matrices of size $B \times 2B$. As B is constant, the cost of applying attention on each block is linear with respect to the segment size. This improves on the aforementioned quadratic runtime of the vanilla Transformer. More BRT details are in Appendix A.

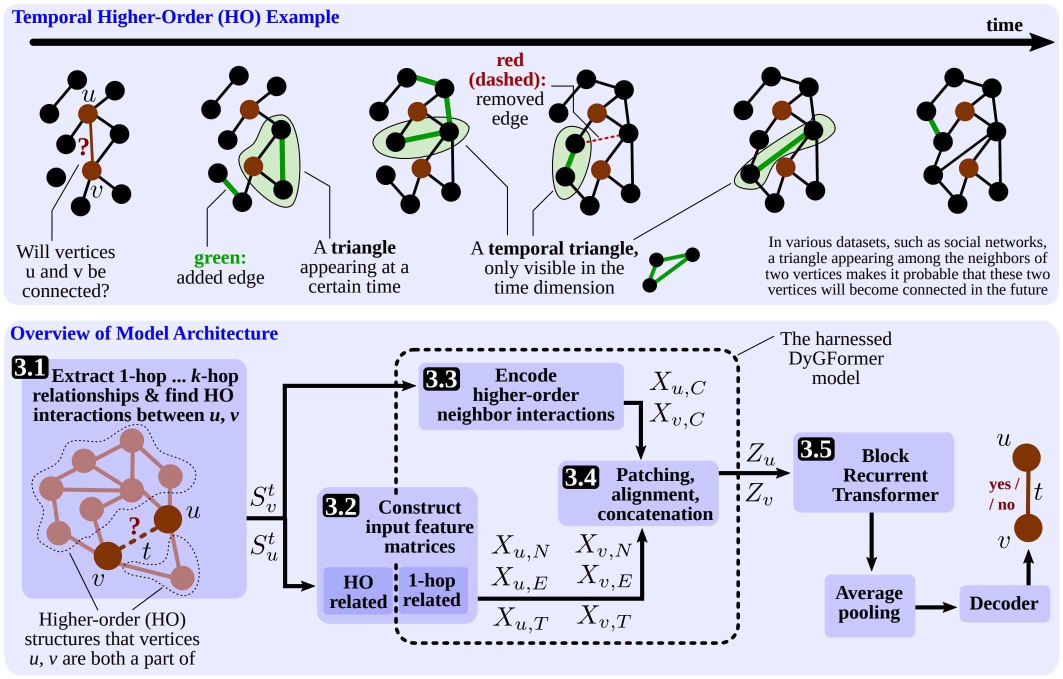
Temporal Higher-Order (HO) Example
time
u
?
v
Will vertices u and v be connected?
green: added edge
A triangle appearing at a certain time
red (dashed): removed edge
A temporal triangle, only visible in the time dimension
In various datasets, such as social networks, a triangle appearing among the neighbors of two vertices makes it probable that these two vertices will become connected in the future
Overview of Model Architecture
3.1 Extract 1-hop ... k-hop relationships & find HO interactions between u, v
u
?
t
v
Higher-order (HO) structures that vertices u, v are both a part of
S_v^t
S_u^t
3.2 Construct input feature matrices
HO related
1-hop related
3.3 Encode higher-order neighbor interactions
$X_{u,C}$
$X_{v,C}$
The harnessed DyGFormer model
3.4 Patching, alignment, concatenation
$X_{u,N}$ $X_{v,N}$
$X_{u,E}$ $X_{v,E}$
$X_{u,T}$ $X_{v,T}$
Z_u
Z_v
3.5 Block Recurrent Transformer
Average pooling
Decoder
u
yes / / no
t
v

Figure 1: Illustration of a temporal higher-order example and an overview of the HOT model.

3 The HOT Model

We now present the design of HOT, see Figure 1. The key idea behind HOT is to harness the HO graph structures within the temporal dimension, and combine them with an efficient Transformer design applied to the temporal dimension modeled with tokens. For this, we extend the template design proposed by the DyGLib library [134] that was provided as a setting where DyGFormer is implemented; it offers structured training and evaluation pipelines for dynamic link prediction. We extend DyGLib and its DyGFormer design with the ability to harness HO and the hierarchical Transformers; we pick the Block-Recurrent Transformer (BRT) as a specific design of choice.

Overall, HOT computes node representations within its encoder module; these representations are then leveraged to solve the downstream link prediction tasks using a suitable decoder. Let u, v be nodes in some CTDG and t a timestamp. Given this, the model first extracts and appropriately encodes the higher-order neighbors of each vertex (Section 3.1), followed by constructing input feature matrices (Section 3.2). Then, the input matrices are augmented by encoding the selected HO interactions (Section 3.3). After that, certain adjustments are made, such as passing the matrices via MLPs (Section 3.4), followed by plugging in BRT (Section 3.5).

3.1 Extracting Higher-Order Neighbors

The model relies on the historical 1-hop and 2-hop interactions of the nodes u and v before t to make its predictions. The set of **1-hop** interactions of a vertex u contains all tuples (u, u', t') for which there is an interaction between nodes u and u' at timestamp $t' < t$. To enable a trade-off between memory consumption and accuracy, we only add the s_1 *most recent* interactions from the set of 1-hop interactions to the list of considered interactions S_u for u. Then, we form the set of **2-hop** interactions. For each 1-hop interaction tuple (u, u', t') in S_u, we consider the interactions (u', u'', t'') with $t'' < t$. We add (u, u'', t'') for the s_2 most recent such interactions to the list of interactions S_u. This scheme is further generalized to arbitrary k-hop neighbourhoods by iterating the construction processes. In our experiments, we focus on 1-hop and 2-hop neighborhoods.

The parameters s_1 and s_2 (and any other s_k if one uses $k > 2$) introduce a tradeoff between accuracy vs. preprocessing time and memory overhead. Specifically, larger values of any s_i result in a more complete view of the temporal graph structure that is encoded into the HOT model. This usually increases the accuracy of predictions. On the other hand, they also increase the preprocessing as well as the memory overhead. This is because both the size $|S_u|$ and the preprocessing time are $O(s_1 s_2)$ (or $O(\prod_{i\in\{1,\ldots,k\}} s_i)$ for higher k).

3.2 Constructing Input Feature Matrices

In the next step, the model constructs neighbor and link encodings based on the raw node and link features of the CTDG. For node u, it constructs the matrices $X_{u,N} \in \mathbb{R}^{|S_u|\times d_N}$ and $X_{u,E} \in \mathbb{R}^{|S_u|\times d_E}$, where d_N and d_E are the dimensions of the node and edge feature vectors, respectively. The matrix $X_{u,N}$ is further extended to include a one-hot encoding to differentiate 1-hop neighbours from 2-hop neighbours. Specifically, two values b_1 and b_2 are appended to every row in $X_{u,N}$. We set $b_1 = 1$ and $b_2 = 0$ if the corresponding node is a 1-hop neighbour, and $b_1 = 0$ and $b_2 = 1$ if the corresponding node is a 2-hop neighbour; d_N is updated accordingly.

For positional encodings, we rely on the scheme introduced in the TGAT model, reused in DyGLib. Here, for some timestamp t', the time interval encoding for the time interval $\Delta t' = t - t'$ is given by

$$\sqrt{1/d_T}\left[\cos\left(w_1\Delta t'\right), \sin\left(w_1\Delta t'\right), \ldots, \cos\left(w_{d_T}\Delta t'\right), \sin\left(w_{d_T}\Delta t'\right)\right], \tag{1}$$

where w_1, …, w_{d_T} are trainable weights. The model computes this for every interaction in S_u, forming $X_{u,T}$. All these matrices are constructed for v analogously.

3.3 Encoding Higher-Order Neighbor Interactions

We next extend the DyGFormer's neighborhood encoding into the HO interactions. For this, we introduce matrices C_u, which – for each vertex u – determine the count of neighbors shared by u and any other vertex v interacting with u, enabling us to encode temporal triangles containing u and v (in the following description, we focus on triangles for concreteness; other HO structures are enabled by considering neighbors beyond 1 hop). Specifically, for any vertex u, the i-th row of the matrix $C_u \in \mathbb{R}^{|S_u|\times 2}$ contains two numbers. The first one is the number of occurrences of a neighbor w of u (w is identified by the i-row of S_u) within S_u. The second number is the count of occurrences of w within S_v, i.e., the temporal neighbourhood of v. Then, we project these vectors of occurrences onto a d_C-dimensional feature space using MLPs with one ReLU-activated hidden layer. The output are the two matrices $X_{u,C} \in \mathbb{R}^{|S_u|\times d_C}$, $X_{v,C} \in \mathbb{R}^{|S_v|\times d_C}$. $X_{u,C}$ is then computed from C_u as follows ($X_{v,C}$ is computed analogously):

$$X_{u,C} = \text{MLP}_0(C_u[:,0]) + \text{MLP}_1(C_u[:,1]) \in \mathbb{R}^{|S_u|\times d_C}. \tag{2}$$

If u and v have a common 1-hop neighbour, then the subgraph induced by those three nodes is a triangle. Thus, this encodes the information about u and v being a part of this triangle into the harnessed feature matrix. More generally, by considering the common k-hop neighbors the model can encode any cycle containing nodes u and v of length up to $2k + 1$. By extension, it can also encode any structure that is a conjunction of such cycles. This further increases the scope of the HO structures taken into account.

3.4 Patching, Alignment, Concatenation

Before feeding $X_{\cdot,\cdot}$ into the selected efficient Transformer, we harness several optional transformations from DyGLib, which further improve the model performance and memory utilization. First, the patching technique bundles multiple rows of a matrix together into one row, so as to reduce the size of the input sequence. Then, alignment reduces the feature dimension of this input by projecting each row onto a smaller dimension. Let us examine the scheme on the matrix $X_{u,N}$. The model constructs $M_{u,N} \in \mathbb{R}^{l_u\times P\cdot d_N}$ by dividing $X_{u,N}$ into $l_u = \left\lceil \frac{|S_u|}{P} \right\rceil$ patches and flattening P temporally adjacent encodings. The same procedure is applied to the other matrices $X_{\cdot,\cdot}$. The encodings then need to be aligned to a common dimension d. For the matrix $M_{u,N}$, we have

$$Z_{u,N} = M_{u,N}W_N + b_N \in \mathbb{R}^{l_u\times d}, \tag{3}$$

where $W_N \in \mathbb{R}^{P\cdot d_N\times d}$ and $b_N \in \mathbb{R}^d$ are trainable parameters. The matrices $Z_{u,E}$, $Z_{u,T}$, and $Z_{u,C}$ are extracted identically from the matrices $M_{u,E}$, $M_{u,T}$, and $M_{u,C}$. The same applies to the matrices belonging to node v. Finally, the extracted matrices are concatenated horizontally into

$$Z_u = Z_{u,N}\|Z_{u,E}\|Z_{u,T}\|Z_{u,C} \in \mathbb{R}^{l_u\times 4d}, \quad Z_v = Z_{v,N}\|Z_{v,E}\|Z_{v,T}\|Z_{v,C} \in \mathbb{R}^{l_v\times 4d}. \tag{4}$$

3.5 Harnessing Temporal Hierarchy with Block-Recurrent Transformer

Finally, we harness a hierarchical Transformer to minimize memory required for keeping the HO temporal structures. The BRT divides Z into B blocks and applies attention locally on each individual block. Additionally, each block is cross-attended with a recurrent state, which allows the element in one block to attend to a summary of the elements in the previous blocks. The outputs of each block are then concatenated into a matrix H, which shares its dimensions with the input Z. Finally, temporal node representations for nodes u and v are computed with the average pooling layer.

3.6 Computational Cost

Let $s = \prod_i s_i$, such that $|S_u| \in O(s)$ and $|S_v| \in O(s)$ and let Δ be the largest number of interactions among vertices u and v. Then, the cost to compute S_u and S_v is $O(s\Delta \log \Delta)$. The cost to construct the Z_u and Z_v is dominated by the alignment process, which costs $O(sd_N d)$ for the node matrix $Z_{u,N}$. The cost is analogous for the other matrices constituting Z_u and Z_v. The remaining costs are that of the BRT model for feature dimension 8d, which is, in particular, linear in the number of interactions s. The attention mechanism in a block-recurrent Transformer costs $O(sdB/P)$ [60], compared to $\Theta(s^2 d/P)$ in the vanilla Transformer. As the block size B approaches the number of interactions s, the cost of the BRT attention approaches that of the vanilla Transformer.

4 Evaluation

We now illustrate the advantages of HOT over the state of the art.

4.1 Experimental Setup

We follow recent established methodologies for evaluating DGRL [88]. We list the most relevant information here, and include details of model parameters and related information in the appendix. We use the standard evaluation metrics, namely the **Average Precision (AP)** (using the scikit implementation [85]) and the **Area Under the ROC (AUC)**.

We use all major state-of-the-art baselines for DGRL on CTDGs; these are TGN [92], CAWN [120], TCL [116], GraphMixer [36], DyGFormer [134]), as well as a purely memorisation-based approach (EdgeBank [88]). Note that our analyses confirm recent findings that illustrate the superiority of DyGFormer among these baselines [134].

We consider both **transductive** and **inductive** setting. In the former, the whole graph structure is visible during training. In the latter, we test the dynamic link prediction and the dynamic node classification on the graph structure that was not visible during training.

Recent work on benchmarking dynamic GRL [134] illustrates the importance of evaluating different sampling schemes beyond plain random sampling. We follow this approach and use all three discussed sampling strategies, showing that HOT achieves competitive results over the state of the art *regardless of how links are sampled*. First, we use **random sampling** (RNES), which is widely used in most dynamic link prediction evaluations. In RNES, when observing some positive edge sample, we generate a negative one by changing its destination node to some random vertex. Second, we use a recent scheme called **historical sampling** (HNES) [88]. In many datasets, it is common to observe repeated interactions (i.e., edges that appear and disappear several times). With random sampling, most negative edge queries will be new to the model (i.e., unobsered in the past). As such, it becomes easy to discard these edges, instead of discardig the repeated edges. In historical sampling, we sample negative edges from the set of previously observed edges, which do *not* appear in the current timestamp (if we cannot sample enough edges this way, the rest is sampled using random sampling). Third, we also use **inductive sampling** (INES) [88], in which we sample edges from those that were not seen during training and do not appear in the current timestamp, i.e., new edges. As before, if we cannot sample enough edges this way, the rest is sampled using the random sampling technique. This scheme produces a metric to better evaluate the model's ability to induce new relations.

All three above sampling schemes can be considered for both transductive or inductive evaluation setting. Note that we do not consider the historical and the inductive sampling techniques in the inductive evaluation setting. This is because the sampled set becomes quite small and unlikely to produce meaningful results. Additionally, consider that both techniques are equivalent in the inductive setting. This is clearly visible in the results obtained by DyGFormer [134].

4.2 Analysis of Performance

We illustrate the comparison of the performance of different models in Figure 2. In the MOOC graph dataset, the model successfully leverages 2-hop interactions to make its predictions more accurate.

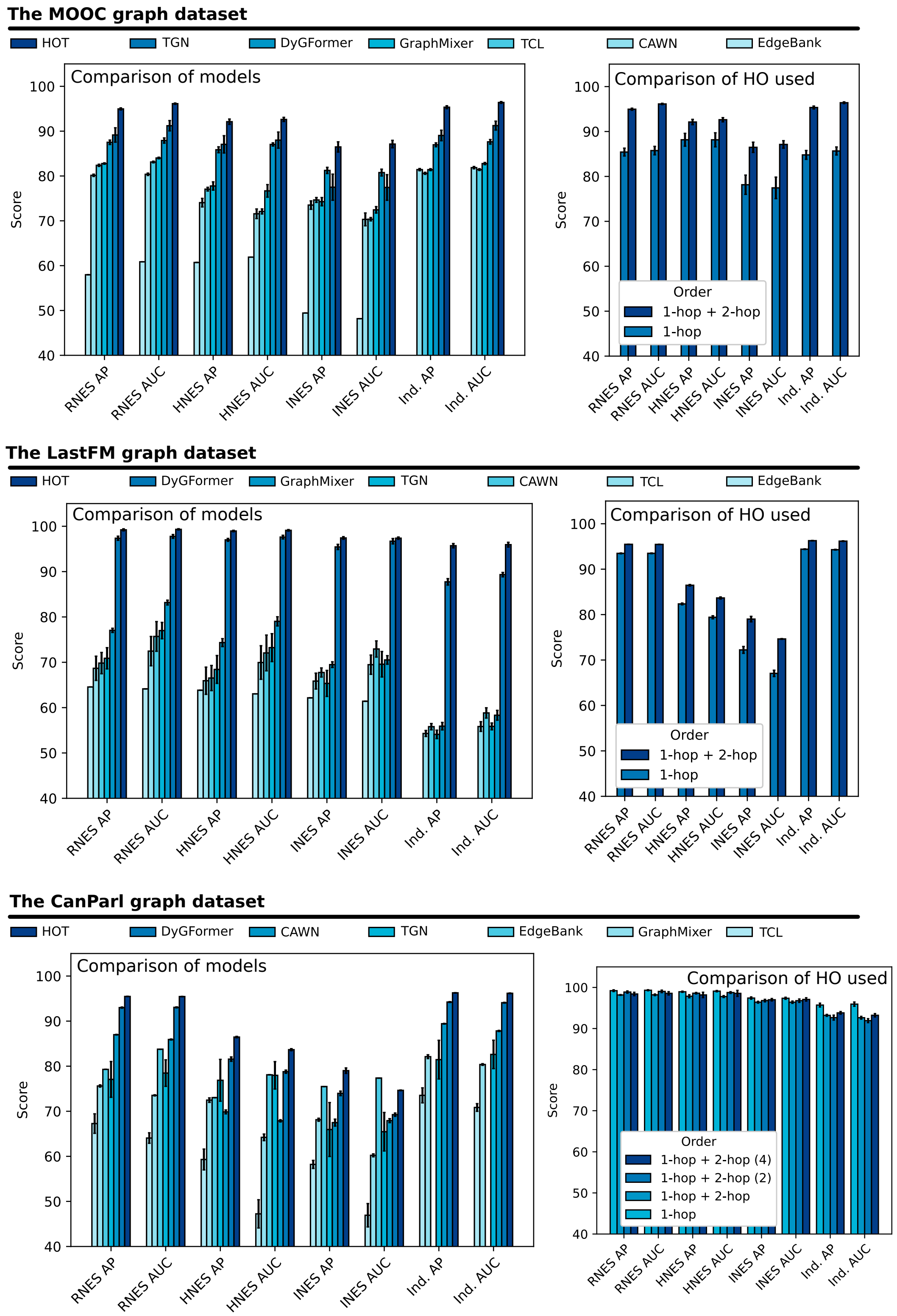

Figure 2: AP (%) and AUC (%) scores on the MOOC, LastFM and CanParl datasets using the various negative edge sampling techniques (RNES, HNES, INES) in the transductive setting, and using the random negative edge sampling technique in the inductive setting (Ind). Baseline results are the best ones provided by [134].

This leads to the best results over all the evaluation metrics and negative edge sampling strategies. These advantages hold for both inductive and inductive settings. Similar performance patterns can be seen for the LastFM dataset, where higher-order neighbors also enable obtaining more powerful predictions. As for the Can. Parl. dataset, the results obtained without the higher-order structure are already quite high, and the additional graph structural information does not seem to be adding significant amount of value to the model in this case. Furthermore, it is possible that the decay in performance with higher s_2 values originates from the added noise that comes with considering more information. In any case, HOT still ensures the highest scores for both the AP and the AUC metric.

Another factor that benefits HOT, particularly in comparison with the DyGFormer, is the horizontal concatenation of the matrices Z_u and Z_v, as opposed to the vertical concatenation in the DyGFormer case. This concatenation strategy forces the attention modules to consider the elements in the same position (in either sequences) together, as one element. We conjecture that this additionally allows the model to better infer the chronological position of each element in the sequence.

4.3 Analysis of Higher-Order (HO) Characteristics

The benefits of harnessing HO structures are clearly visible in the rightmost plots of Figure 2. There, we fix s_1 and vary s_2. For the MOOC and LastFM, we consider $s_2 \in \{0, 1\}$; for CanParl, we consider $s_2 \in \{0, 1, 2, 4\}$. By setting $s_2 = 0$, we exclude any HO structures beyond triangles from the model's consideration (the results in this setting correspond to the bars labeled "1-hop"). Overall, CanParl does not benefit from higher values of s_2. Contrarily, for the other two datasets, the model clearly benefits from the HO structures. Setting $s_2 = 1$, i.e., considering not only triangles, but any cycles of up to 5 nodes, further improves performance.

4.4 Analysis of Memory Consumption

We also investigate in more detail the impact of the block size B (see Section 3.5) and the patch size P (see Section 3.4) on the memory consumption. The results are in Figure 3 and they show the size needed for the attention matrix in Vanilla Transformer, compared to HOT. First, assuming that blocks are processed sequentially as in RNNs, reducing B also decreases the needed memory (for a fixed P). While large values of B inflate the total memory beyond what is needed for the Vanilla Transformer, smaller blocks result in much less memory needed. This comes with a tradeoff, and the latency to process the model increases linearly with the number of blocks. This however can be alleviated with schemes designed to specifically speed up the RNN processing. The patch size P has a similar effect: the smaller it is, the more memory is required for Vanilla Transformer (for a fixed B), due to the patching flattening (Section 3.4). Hence, selecting B and P is a design choice that would impact both the needed memory and performance. In our implementation, we offer a set of scripts that offer a quick assessment of the required memory, facilitating the design of schemes based on HOT. Overall, based on the given input sequence length S, the patch size P, and the block size B, the total amount of elements in attention matrices of the vanilla Transformer can be assessed as $(3S/P) \cdot (8d)$. Furthermore, considering just one block, the attention matrix based on BRT needs about $(9 \cdot 2B) \cdot (8d)$ elements.

5 Discussion

5.1 Applications of HOT Beyond Link Prediction

The generic structure of HOT ensures that it can be straightforwardly extended towards other dynamic graph ML tasks, including edge, node, or graph classification or regression.

To illustrate this on a concrete example, we implemented *dynamic node classification* within HOT. Here, we are given a CTDG $(G^{(0)}, T)$, a timestamp $t \in \mathbb{N}$, a node $v \in V$, and a set of $N \in \mathbb{N}$ classes $\{S_i\}_{i=1}^{N}$. The goal is to decide, for which $1 \leq i \leq N$, $v \in S_i$. Naturally, we assume $v \in \bigcup_{i=1}^{N} S_i$ $\forall v \in V$. To solve this task in HOT, we employ transfer learning, i.e., we harness the parameters from the dynamic link prediction task to compute suitable dynamic node representations, and then train a suitable MLP decoder on those representations. The preliminary evaluation indicate performance comparable to, or better than DyGFormer on – respectively – the Wikipedia and the Reddit datasets. A more detailed investigation into the model design for dynamic node classification, as well as more extensive evaluations, are future work. However, our preliminary results indicate potential in harnessing the HO structures in the general dynamic GRL setting.

Other dynamic GRL tasks could be achieved using established GRL methods. For example, graph classification could be used by applying pooling on top of the edge and node classification [9, 55, 66, 99, 100, 129].

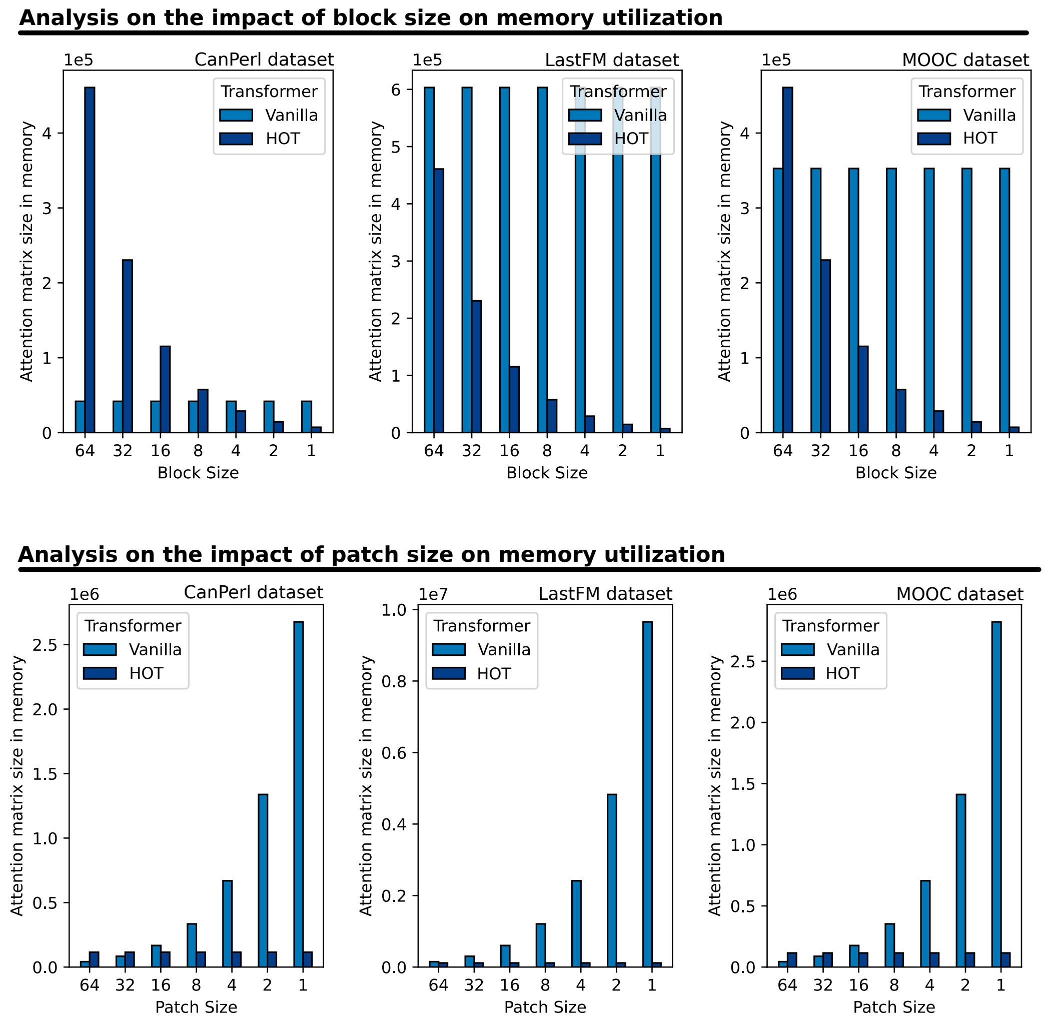

Figure 3: The analysis of the impact of the block and patch size on memory utilization.

5.2 Limitations of HOT

The limitations of HOT are largely analogous to those of any HO scheme. First, the time complexity required to search for HO structures may limit the size of the graphs or the length of the temporal history to be considered. Here, HOT's harnessing of efficient Transformers alleviates the time and storage complexity needed for the on-the-fly predictions. Still, HOT needs to search for HO structures to construct the input feature matrices; however, it is a one-time preprocessing overhead for a given dataset. Second, utilizing "too much" of HO may introduce excessive amounts of noise and ultimately lower the accuracy. This limitation is also generic to HO learning, it is commonly alleviated by appropriately limiting the size of the harnessed HO structures [137].

6 Related Work

Our work touches on many areas. We now briefly discuss related works.

Graph Neural Networks and Graph Representation Learning Graph neural networks (GNNs) emerged as a highly successful part of the graph representation learning (GRL) field [52]. Numerous GNN models have been developed [19, 20, 30, 33, 49, 52, 97, 125, 139, 142], including convolutional [53, 70, 102, 123, 128], attentional [23, 82, 106, 112], message-passing [10, 28, 50, 96, 121], or – more recently – higher-order (HO) ones [1, 1, 13, 26, 83, 93, 94]. Moreover, a large number of software frameworks [44, 58, 62, 73, 80, 109, 113–115, 117, 119, 124, 136, 141, 143], and even hardware accelerators [48, 68, 69, 75, 130] for processing GNNs have been introduced over the last years. All these schemes target static graphs, with no updates, while in this work, we target the dynamic GRL, where graphs evolve over time. However, our model can be seamlessly used for a temporal static setting, where the graph comes with the available history of past updates.

Dynamic Graph Representation Learning There have been many approaches to solving dynamic GRL in the past years. JODIE [71] is an RNN-based approach for bipartite graphs. The model constructs and maintains embeddings of source and target nodes and updates these recurrently using different RNNs. DyRep [110] is an RNN-based approach, which keeps node representations and updates them using a self-attention mechanism, which dynamically weights the importance of different structures in the graph. TGAT [127] computes node representations based on the static GAT model [112], i.e., by aggregating messages from the temporal neighborhood of each node using self-attention. The main difference between TGAT and GAT is that timestamps are taken as input as well and taken into consideration after a suitable time encoding is found. TGN [92] divides the model architecture into the message function, the message aggregator, the memory updater, and the embedding module. Given two nodes and a timestamp, the model fetches node representations from memory, aggregates them, and updates its memory. It then builds node embeddings for downstream tasks using a TGAT-like layer. CAWN [120] builds on the concept of anonymous walks [61] and extends them to consider the time dimension. Given two nodes and a timestamp, the model collects causal anonymous walks starting from both nodes. They are then encoded using RNNs and finally aggregated for predictions. TCL [116] first orders previous interactions of each node in some order that reflects both temporal and structural positioning (for any two pairs of nodes). The model then runs a Transformer on each of the nodes' previous interactions. These Transformers are coupled through cross-attention. GraphMixer [36] is a simple architecture consisting of three modules: the link encoder, the node encoder, and the link classifier. The link encoder summarizes temporal link information using an MLP-mixer [107], and the node encoder captures node information using neighbour mean-pooling. The link classifier applies a 2-layer MLP on the output of the two other modules to formulate a prediction. Finally, DyGFormer [134] is an approach based on the Transformer model. The model proposed in this work, HOT, extends DyGFormer and outperforms all other baselines for different datasets thanks to harnessing the HO graph structures.

Dynamic and Streaming Graph Computing There also exist systems for processing dynamic and streaming graphs [14, 15, 35, 95] beyond GRL. *Graph streaming frameworks* such as STINGER [43] or Aspen [40] emerged to enable processing and analyzing dynamically evolving graphs. *Graph databases* are systems used to manage, process, analyze, and store vast amounts of rich and complex graph datasets. Graph databases have a long history of development and focus in both academia and in the industry, and there has been significant work on them [4, 5, 16–18, 39, 47, 54, 64, 72]. Such systems often execute graph analytics algorithms (e.g., PageRank) concurrently with graph updates (e.g., edge insertions). Thus, these frameworks must tackle unique challenges, for example effective modeling and storage of dynamic datasets, efficient ingestion of a stream of graph updates concurrently with graph queries, or support for effective programming model. Here, recently introduced *Neural graph databases* focus on integrating Graph Databases with GRL capabilities [21, 91]. Our model could be used to extend these systems with dynamic GRL workloads.

7 Conclusion

Dynamic graph representation learning (DGRL), where graph datasets may ingest millions of edge updates per second, is an area of growing importance. In this work, we enhance one of the most recent and powerful works in DGRL, the Transformer-based DGRL, by harnessing higher-order (HO) graph structures: k-hop neighbors and more general subgraphs. As this approach enables harnessing more information from the temporal dimension, it results in the higher accuracy of prediction outcomes. Simultaneously, it comes at the expense of increased memory pressure through the larger underlying attention matrix. For this, we employ a recent class of Transformer models that impose hierarchy on the attention matrix, picking Block-Recurrent Transformer for concreteness. This reduces memory footprint while ensuring – for example – 9%, 7%, and 15% higher accuracy in dynamic link prediction than – respectively – DyGFormer, TGN, and GraphMixer, for the MOOC dataset.

Our design illustrates that a careful combination of models and paradigms used in different settings, such as the Higher-Order graph structures and the Block-Recurrent Transformer, results in advantages in both performance and memory footprint in DGRL. This approach could be extended by considering other state-of-the-art Transformer schemes or HO GNN models.

Acknowledgements

We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Daint machines, and for their excellent technical support. We thank Timo Schneider for help with infrastructure at SPCL. This project received funding from the European Research Council (Project PSAP, No. 101002047), and the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955513 (MAELSTROM). This project was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. This project received funding from the European Union's HE research and innovation programme under the grant agreement No. 101070141 (Project GLACIATION).

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29. 2, 8

[2] Unai Alvarez-Rodriguez, Federico Battiston, Guilherme Ferraz de Arruda, Yamir Moreno, Matjaž Perc, and Vito Latora. 2021. Evolutionary dynamics of higher-order interactions in social networks. *Nature Human Behaviour* 5, 5 (2021), 586–595. 1

[3] Khaled Ammar. 2016. Techniques and Systems for Large Dynamic Graphs. In *SIGMOD'16 PhD Symposium*. ACM, 7–11. 1

[4] Renzo Angles and Claudio Gutierrez. 2008. Survey of Graph Database Models. *in ACM Comput. Surv.* 40, 1, Article 1 (2008), 39 pages. https://doi.org/10.1145/1322432.1322433 9

[5] Renzo Angles and Claudio Gutierrez. 2018. An Introduction to Graph Data Management. In *Graph Data Management, Fundamental Issues and Recent Developments*. 1–32. 9

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016). 17

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014). 2

[8] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in neural information processing systems* 33 (2020), 17804–17815. 1

[9] Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. 2021. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–37. 1, 7

[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018). 8

[11] Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. 2020. Networks beyond pairwise interactions: structure and dynamics. *Physics Reports* 874 (2020), 1–92. 2

[12] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020). 2, 3

[13] Austin R Benson et al. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230. 8

[14] Maciej Besta. 2021. *Enabling High-Performance Large-Scale Irregular Computations*. Ph. D. Dissertation. ETH Zurich. 9

[15] Maciej Besta et al. 2022. Practice of Streaming Processing of Dynamic Graphs: Concepts, Models, and Systems. *IEEE TPDS* (2022). 1, 9

[16] Maciej Besta et al. 2023. The Graph Database Interface: Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores. In *ACM/IEEE Supercomputing*. 1, 9

[17] Maciej Besta, Robert Gerstenberger, Nils Blach, Marc Fischer, and Torsten Hoefler. 2023. *GDI: A Graph Database Interface Standard*. Technical Report. Available at https://spcl.inf.ethz.ch/Research/Parallel_Programming/GDI/.

[18] Maciej Besta, Robert Gerstenberger, Peter Emanuel, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2023. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *ACM CSUR* (2023). 9

[19] Maciej Besta, Raphael Grob, Cesare Miglioli, Nicola Bernold, Grzegorz Kwasniewski, Gabriel Gjini, Raghavendra Kanakagiri, Saleh Ashkboos, Lukas Gianinazzi, Nikoli Dryden, et al. 2022. Motif Prediction with Graph Neural Networks. In *ACM KDD*. 8

[20] Maciej Besta and Torsten Hoefler. 2023. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. *IEEE TPAMI* (2023). 8

[21] Maciej Besta, Patrick Iff, Florian Scheidl, Kazuki Osawa, Nikoli Dryden, Michal Podstawski, Tiancheng Chen, and Torsten Hoefler. 2022. Neural Graph Databases. In *LOG*. 9

[22] Maciej Besta, Cesare Miglioli, Paolo Sylos Labini, Jakub Tětek, Patrick Iff, Raghavendra Kanakagiri, Saleh Ashkboos, Kacper Janda, Michał Podstawski, Grzegorz Kwaśniewski, et al. 2022. Probgraph: High-performance and high-accuracy graph mining with probabilistic set representations. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–17. 1

[23] Maciej Besta, Pawel Renc, Robert Gerstenberger, Paolo Sylos Labini, Alexandros Ziogas, Tiancheng Chen, Lukas Gianinazzi, Florian Scheidl, Kalman Szenes, Armon Carigiet, et al. 2023. High-Performance and Programmable Attentional Graph Neural Networks with Global Tensor Formulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–16. 8

[24] Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, et al. 2021. Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra. *arXiv preprint arXiv:2103.03653* (2021). 1

[25] Christian Bick, Elizabeth Gross, Heather A Harrington, and Michael T Schaub. 2021. What are higher-order networks? *arXiv preprint arXiv:2104.11329* (2021). 2

[26] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. 2021. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems* 34 (2021), 2625–2640. 8

[27] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. arXiv:1506.02075 [cs.LG] 1

[28] Xavier Bresson and Thomas Laurent. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017). 8

[29] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906* (2017). 2

[30] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42. 8

[31] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901. 2

[32] Ziang Cao, Changhong Fu, Junjie Ye, Bowen Li, and Yiming Li. 2021. Hift: Hierarchical feature transformer for aerial tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15457–15466. 2

[33] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2020. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675* (2020). 8

[34] Xiaofu Chang, Xuqin Liu, Jianfeng Wen, Shuang Li, Yanming Fang, Le Song, and Yuan Qi. 2020. Continuous-time dynamic graph learning via neural interaction processes. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 145–154. 1, 2

[35] Sutanay Choudhury, Khushbu Agarwal, Sumit Purohit, Baichuan Zhang, Meg Pirrung, Will Smith, and Mathew Thomas. 2017. Nous: Construction and querying of dynamic knowledge graphs. In *IEEE ICDE*. 1563–1565. 9

[36] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2023. Do We Really Need Complicated Model Architectures For Temporal Networks? *arXiv preprint arXiv:2302.11636* (2023). 1, 2, 5, 9

[37] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019). 2, 3, 17

[38] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359. 2

[39] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A survey on NoSQL stores. *ACM Computing Surveys (CSUR)* 51, 2, Article 40 (2018), 43 pages. `https://doi.org/10.1145/3158661` 9

[40] Laxman Dhulipala et al. 2019. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. *arXiv:1904.08380* (2019). 9

[41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020). 2

[42] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng, and Philip S Yu. 2021. Continuous-time sequential recommendation with temporal graph collaborative transformer. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 433–442. 1

[43] Guoyao Feng et al. 2015. DISTINGER: A distributed graph data structure for massive dynamic graph processing. In *IEEE Big Data*. 1814–1822. 9

[44] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019). 8

[45] Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. 2022. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems* 35 (2022), 31376–31390. 2

[46] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020). 2

[47] Santhosh Kumar Gajendran. 2012. A survey on NoSQL databases. *University of Illinois* (2012). 9

[48] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *IEEE/ACM MICRO*. 8

[49] Lukas Gianinazzi, Maximilian Fries, Nikoli Dryden, Tal Ben-Nun, and Torsten Hoefler. 2021. Learning Combinatorial Node Labeling Algorithms. *arXiv preprint arXiv:2106.03594* (2021). 8

[50] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272. 8

[51] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 922–929. 1

[52] William L Hamilton et al. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017). 8

[53] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 8

[54] Jing Han, E Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*. IEEE, 363–366. 9

[55] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 11 (2021), 7436–7456. 1, 7

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778. 17

[57] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. 9, 8 (1997), 46 pages. `https://doi.org/10.1162/neco.1997.9.8.1735` 2

[58] Yuwei Hu et al. 2020. Featgraph: A flexible and efficient backend for graph neural network systems. *arXiv preprint arXiv:2008.11359* (2020). 8

[59] Zijie Huang, Yizhou Sun, and Wei Wang. 2020. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems* 33 (2020), 16177–16187. 1

[60] DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. *arXiv preprint arXiv:2203.07852* (2022). 2, 5

[61] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous walk embeddings. In *International conference on machine learning*. PMLR, 2186–2195. 9

[62] Zhihao Jia et al. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *MLSys* (2020). 8

[63] Ming Jin, Yuan-Fang Li, and Shirui Pan. 2022. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. *Advances in Neural Information Processing Systems* 35 (2022), 19874–19886. 1, 2

[64] R. Kumar Kaliyar. 2015. Graph databases: A survey. In *ICCCA*. 785–790. 9

[65] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation learning for dynamic graphs: A survey. *The Journal of Machine Learning Research* 21, 1 (2020), 2648–2720. 1

[66] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation learning for dynamic graphs: A survey. *The Journal of Machine Learning Research* 21, 1 (2020), 2648–2720. 1, 7

[67] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 20

[68] Kevin Kiningham, Philip Levis, and Christopher Ré. 2020. GReTA: Hardware Optimized Graph Processing for GNNs. In *ReCoML*. 8

[69] Kevin Kiningham, Christopher Re, and Philip Levis. 2020. GRIP: a graph neural network accelerator architecture. *arXiv preprint arXiv:2007.13828* (2020). 8

[70] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016). 8

[71] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1269–1278. 1, 2, 9

[72] Vijay Kumar and Anjan Babu. 2015. Domain Suitable Graph Database Selection: A Preliminary Report. In *3rd International Conference on Advances in Engineering Sciences & Applied Mathematics, London, UK*. 26–29. 9

[73] Shen Li et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020). 8

[74] Xiaohan Li, Mengqi Zhang, Shu Wu, Zheng Liu, Liang Wang, and S Yu Philip. 2020. Dynamic graph collaborative filtering. In *2020 IEEE international conference on data mining (ICDM)*. IEEE, 322–331. 1

[75] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE TOC* (2020). 8

[76] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting* 37, 4 (2021), 1748–1764. 2

[77] Songtao Liu, Lingwei Chen, Hanze Dong, Zihao Wang, Dinghao Wu, and Zengfeng Huang. 2019. Higher-order weighted graph convolutional networks. *arXiv preprint arXiv:1911.04129* (2019). 2

[78] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 10012–10022. 2

[79] Yuhong Luo and Pan Li. 2022. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*. PMLR, 1–1. 2

[80] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. Neugraph: parallel deep neural network computation on large graphs. In *USENIX ATC*. 8

[81] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 719–728. 1, 2

[82] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE CVPR*. 8

[83] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4602–4609. 2, 8

[84] Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. 2019. Hierarchical transformers for long document classification. In *2019 IEEE automatic speech recognition and understanding workshop (ASRU)*. IEEE, 838–844. 2

[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. 5, 19

[86] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. 2023. RWKV: Reinventing RNNs for the Transformer Era. *arXiv preprint arXiv:2305.13048* (2023). 2

[87] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. 2020. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409* (2020). 1

[88] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. 2022. Towards Better Evaluation for Dynamic Link Prediction. *arXiv preprint arXiv:2207.10128* (2022). 5

[89] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018). 2

[90] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507* (2019). 2

[91] Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. 2023. Neural graph reasoning: Complex logical query answering meets graph databases. *arXiv preprint arXiv:2303.14617* (2023). 9

[92] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* (2020). 1, 2, 5, 9

[93] Ryan A. Rossi, Nesreen K. Ahmed, and Eunyee Koh. 2018. Higher-Order Network Representation Learning. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 3–4. https://doi.org/10.1145/3184558.3186900 8

[94] Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi Yadkori. 2018. HONE: Higher-Order Network Embeddings. *arXiv:1801.09303 [cs, stat]* (May 2018). arXiv:1801.09303 [cs, stat] 8

[95] Sherif Sakr et al. 2020. The Future is Big Graphs! A Community View on Graph Processing Systems. *arXiv preprint arXiv:2012.06171* (2020). 1, 9

[96] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. 2020. Learning to simulate complex physics with graph networks. In *ICML*. 1, 8

[97] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80. 8

[98] Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202* (2020). 19

[99] Ann E Sizemore and Danielle S Bassett. 2018. Dynamic graph metrics: Tutorial, toolbox, and tale. *NeuroImage* 180 (2018), 417–427. 1, 7

[100] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. 2021. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access* 9 (2021), 79143–79168. 1, 7

[101] Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. 2019. Session-based social recommendation via dynamic graph attention networks. In *Proceedings of the Twelfth ACM international conference on web search and data mining*. 555–563. 1

[102] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *NeurIPS* (2016). 8

[103] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/8fb21ee7a2207526da55a679f0332de2-Paper.pdf 1

[104] Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. 2022. A length-extrapolatable transformer. *arXiv preprint arXiv:2212.10554* (2022). 19

[105] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. 2

[106] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735* (2018). 8

[107] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems* 34 (2021), 24261–24272. 9

[108] Leo Torres, Ann S Blevins, Danielle Bassett, and Tina Eliassi-Rad. 2021. The why, how, and when of representations for complex systems. *SIAM Rev.* 63, 3 (2021), 435–485. 2

[109] Alok Tripathy, Katherine Yelick, and Aydın Buluç. 2020. Reducing communication in graph neural network training. In *ACM/IEEE Supercomputing*. 8

[110] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*. 1, 2, 9

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 1, 2

[112] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017). 8, 9

[113] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2022. Marius++: Large-Scale Training of Graph Neural Networks on a Single Machine. *arXiv preprint arXiv:2202.02365* (2022). 8

[114] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling Sampling. *MLSys* (2022).

[115] Cheng Wan, Youjie Li, Cameron R Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. 2022. PipeGCN: Efficient full-graph training of graph convolutional networks with pipelined feature communication. *arXiv preprint arXiv:2203.10428* (2022). 8

[116] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. 2021. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944* (2021). 1, 2, 5, 9

[117] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv:1909.01315* (2019). 8

[118] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. 2021. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 international conference on management of data*. 2628–2638. 1, 2

[119] Yuke Wang et al. 2021. GNNAdvisor: An Efficient Runtime System for GNN Acceleration on GPUs. In *OSDI*. 8

[120] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive representation learning in temporal networks via causal anonymous walks. *arXiv preprint arXiv:2101.05974* (2021). 1, 2, 5, 9

[121] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12. 8

[122] Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. 2021. Fastformer: Additive attention can be all you need. *arXiv preprint arXiv:2108.09084* (2021). 2

[123] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871. 8

[124] Yidi Wu, Kaihao Ma, Zhenkun Cai, Tatiana Jin, Boyang Li, Chenguang Zheng, James Cheng, and Fan Yu. 2021. Seastar: vertex-centric programming for graph neural networks. In *EuroSys*. 8

[125] Zonghan Wu et al. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020). 8

[126] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121* (2019). 1

[127] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962* (2020). 1, 2, 9

[128] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018). 2, 8

[129] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. 2022. Dynamic network embedding survey. *Neurocomputing* 472 (2022), 212–223. 1, 7

[130] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. Hygcn: A gcn accelerator with hybrid architecture. In *IEEE HPCA*. IEEE, 15–29. 8

[131] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017). 1

[132] Le Yu, Bowen Du, Xiao Hu, Leilei Sun, Liangzhe Han, and Weifeng Lv. 2021. Deep spatio-temporal graph convolutional network for traffic accident prediction. *Neurocomputing* 423 (2021), 135–147. 1

[133] Le Yu, Zihang Liu, Tongyu Zhu, Leilei Sun, Bowen Du, and Weifeng Lv. 2022. Modelling Evolutionary and Stationary User Preferences for Temporal Sets Prediction. *arXiv preprint arXiv:2204.05490* (2022). 1

[134] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. 2023. Towards Better Dynamic Graph Learning: New Architecture and Unified Library. arXiv:2303.13047 [cs.LG] 1, 2, 3, 5, 6, 9, 20

[135] Le Yu, Guanghui Wu, Leilei Sun, Bowen Du, and Weifeng Lv. 2022. Element-guided Temporal Graph Representation Learning for Temporal Sets Prediction. In *Proceedings of the ACM Web Conference 2022*. 1902–1913. 1

[136] Dalong Zhang et al. 2020. Agl: a scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454* (2020). 8

[137] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691* (2018). 8

[138] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. 2022. Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4741–4753. 1

[139] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020). 8

[140] Zizhao Zhang, Han Zhang, Long Zhao, Ting Chen, Sercan Ö Arik, and Tomas Pfister. 2022. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 3417–3425. 2

[141] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, Qidong Su, Minjie Wang, Chao Ma, and George Karypis. 2021. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Graphs. *arXiv:2112.15345* (2021). 8

[142] Jie Zhou et al. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. 8

[143] Rong Zhu et al. 2019. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730* (2019). 8

Appendix

A Model Design: Additional Details

We provide more details on the model design.

A.1 Vanilla Transformer

We detail some parts of the vanilla Transformer that are used when constructing the Block-Recurrent Transformer.

A.1.1 Positional Encoding

Before the sequence arrives at the encoder, it requires positional encoding, such that the Transformer has a notion of the order of the inputs x_i in the input sequence. The original paper suggests the following transformation of the inputs:

$$Z_{i,j} = X_{i,j} + \begin{cases} \sin\left(i/1000^{j/d}\right) & \text{if } j \text{ even,} \\ \cos\left(i/1000^{(j-1)/d}\right) & \text{otherwise.} \end{cases} \tag{5}$$

The matrix Z is then fed into the multi-head self-attention module of the encoder.

A.1.2 Multi-Head Self-Attention

Here, the matrix Z is first projected onto three different matrices $Q = ZW_Q \in \mathbb{R}^{n \times d_k}$, $K = ZW_K \in \mathbb{R}^{n \times d_k}$ and $V = ZW_V \in \mathbb{R}^{n \times d_v}$, where W_Q, W_K and W_V are parameter matrices. The matrix of outputs is then calculated as follows:

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{6}$$

In order to capture information from different representation subspaces we can perform Multi-Head Attention. In this case, the matrix of outputs is calculated as follows:

$$\mathrm{MH}\left(Q, K, V\right) = \mathrm{Concat}\left(\mathrm{head}_1, \ldots, \mathrm{head}_h\right) W_O \tag{7}$$

where $W_O \in \mathbb{R}^{hd_v \times d}$, and

$$\mathrm{head}_i = \mathrm{Attention}\left(QW_Q^i, KW_K^i, VW_V^i\right) \tag{8}$$

where $W_Q^i \in \mathbb{R}^{d \times d_k}$, $W_K^i \in \mathbb{R}^{d \times d_k}$, $W_V^i \in \mathbb{R}^{d \times d_v}$ are parameter matrices.

The output of this module is then added to Z over a residual connection [56] and normalised using Layer Normalisation [6]. It is then fed into an MLP.

A.1.3 Feed-Forward Network

The MLP consists of a single ReLu-activated hidden layer. It is applied to each element of the sequence separately. The output is then, as before, added to the output of the previous module over a residual connection and normalised using Layer Normalisation.

The output of the encoder layer is then fed back into another encoder layer. The number of layers in the encoder depends on the implementation. The original paper suggests a total of 6 layers. Afterwards, the output is fed into the decoder, which computes the final output probabilities.

A.2 Block-Recurrent Transformer

A.2.1 Model Overview

The first block within each segment must be able to attend to previous segments. As such, following an idea from Transformer-XL [37], the model stores the keys and values of the computed state vectors of the previous segment in cache.

The Block-Recurrent Transformer is composed of various modules called recurrent cells. There are two types of recurrent cells, vertical cells, which calculate next state vectors, and horizontal cells, which calculate output token embeddings. These can be stacked in any desired way. The blocks are fed one by one to the stack of recurrent cells, and the outputs of the last cell are then concatenated to form the output of the model.

Vertical cell: Just like in the vanilla Transformer, in the vertical recurrent cell, the input token embeddings undergo self-attention. However, unlike the vanilla Transformer, this type of cell also employs cross-attention on the input embeddings and the current state vectors, i.e., it computes attention scores using a query matrix (Q) based on the input embeddings and key and value matrices (K, V) based on the current state vectors. The results from both attention modules are then projected to some feature space and added to the input token embeddings over a residual connection. The resulting values are then fed into an MLP and added to its output to form the output token embeddings.

Horizontal cell: In the horizontal cell the roles of the current state vectors and the input token embeddings are inverted. While the current state vectors undergo self-attention and contribute to the cross-attention module with a query matrix (Q), the input token embeddings only contribute to the cross-attention module with the key and value matrices (K, V). The results of the attention modules are projected onto some feature space. The resulting values h_t are then fed into a group of gates, one per current state vector c_t. Note, that t refers to the index of the current block within the sequence. Each gate realizes the following computations:

$$z_t = W_z h_t + b_z \tag{9}$$

$$g = \sigma(b_g) \tag{10}$$

$$c_{t+1} = c_t \odot g + z_t \odot (1 - g) \tag{11}$$

where W_z, b_z and b_g are trainable parameters, σ is the sigmoid function and $\odot$ is the element-wise multiplication. The outputs of the gates c_{t+1} are then concatenated to form the next state vectors and the output of this cell.

A.2.2 Model Details

After constructing the matrix Z, the model feeds it into a Block-Recurrent Transformer consisting of one horizontal layer and one vertical layer. The matrix Z is divided into segments of size S and further into blocks of size B. In the following we assume $|Z| < S$ and outline the process undergone by Z in this case. The general case is described after.

The blocks of Z are fed into the layers of the Block-Recurrent Transformer one by one, starting from the top of Z. Let Z_b be the b-th block in Z. After the predecessors of Z_b have been processed, Z_b is fed into the **horizontal cell** along with the current state vectors C. Here, the state vectors are first normalized and graced with learned positional embeddings as follows:

$$\overline{C} = \overline{C} + W_p, \tag{12}$$

where W_p is a trainable parameter.

Next, we extract the keys and values matrices (K_b, V_b) of Z_b as follows:

$$K_b = \overline{Z_b} W_K^{Z_b} \text{ and } V_b = \overline{Z_b} W_V^{Z_b}, \tag{13}$$

where $W_K^{Z_b} \in \mathbb{R}^{8d \times Id_k}$, and $W_V^{Z_b} \in \mathbb{R}^{8d \times Id_v}$ are trainable parameters. These matrices are stored for later use in the vertical layer as well.

The computation proceeds as follows:

$$A_h^{(1)} = \text{MH}\left(\overline{C} W_Q^{(1)}, \overline{C} W_K^C, \overline{C} W_V^C\right), \tag{14}$$

$$A_h^{(2)} = \text{MH}\left(\overline{C} W_Q^{(2)}, [K_{b-1}; K_b], [V_{b-1}; V_b]\right), \tag{15}$$

where $W_Q^{(1)} \in \mathbb{R}^{8d \times Id_k}$, $W_K^C \in \mathbb{R}^{8d \times Id_k}$, $W_V^C \in \mathbb{R}^{8d \times Id_v}$, and $W_Q^{(2)} \in \mathbb{R}^{8d \times Id_k}$ are trainable parameters, I denotes the number of attention heads, and $[*; *]$ refers to vertical concatenation. If $b = 1$, the concatenation is skipped. $A_h^{(1)}$ is the result of self-attention on the current state vector, while $A_h^{(2)}$ results from cross-attending the state vectors with the input Z_b. Note, that the function MH refers to the Multi-Head Attention module as described in Section A.1.2.

The matrices $A_h^{(1)}$ and $A_h^{(2)}$ are then concatenated horizontally and projected to a suitable dimension as follows:

$$P_h = (A^{(1)} \| A^{(2)}) W_h + b_h, \tag{16}$$

where $W_h \in \mathbb{R}^{2Id_v \times 8d}$ and $b_h \in \mathbb{R}^{8d}$ are trainable parameters. Subsequently, P_h enters a gating mechanism along with the current state vectors. There, each state vector is multiplied element-wise with the sigmoid of a trainable vector $b_g \in \mathbb{R}^{8d}$, while the rows of P_h are multiplied element-wise with the vector $1 - \sigma(b_g)$. These are then added together as shown below.

$$N = C \odot \sigma(b_g) + P_h \odot (1 - \sigma(b_g)), \tag{17}$$

where $\odot$ denotes element-wise multiplication, and σ denotes the sigmoid function. N is now referred to as the next state vectors and is fed, along with Z_b to a vertical cell.

In the **vertical cell**, the state vectors are first normalized and graced with learned positional embeddings as before:

$$\overline{N} = \overline{N} + W_p. \tag{18}$$

Then, they are processed as follows:

$$A^{(3)} = \text{MHR}\left(\overline{Z_b} W_Q^{(3)}, [K_{b-1}; K_b], [V_{b-1}; V_b]\right), \tag{19}$$

$$A^{(4)} = \text{MH}\left(\overline{Z_b} W_Q^{(4)}, \overline{N} W_K^C, \overline{N} W_V^C\right), \tag{20}$$

where $W_Q^{(3)} \in \mathbb{R}^{8d \times Id_k}$, and $W_Q^{(4)} \in \mathbb{R}^{8d \times Id_k}$ are trainable parameters, and W_K^C and W_V^C are the same parameters used to project the state vectors in the horizontal layer. Unlike before, here, the input Z_b undergoes self-attention, while cross-attending with the state vectors extracted from the previous cell N. Note, that the function MHR refers to a Multi-Head Attention module with Extrapolatable Position Embeddings [104].

As before, the matrices $A^{(3)}$ and $A^{(4)}$ are concatenated horizontally and projected to a suitable dimension:

$$P_v = (A^{(3)} \| A^{(4)}) W_v + b_v, \tag{21}$$

where $W_v \in \mathbb{R}^{2Id_v \times 8d}$ and $b_v \in \mathbb{R}^{8d}$ are trainable parameters. Z_b is then added to P_v over a residual connection. The resulting matrix is fed into an FFN_{GEGLU} [98] and added to its output over a residual connection:

$$O_b = \text{FFN}_{GEGLU}(P_v + Z_b) + P_v + Z_b. \tag{22}$$

O_b is the output of the Block-Recurrent Transformer for the block Z_b.

If Z can be divided into multiple segments, then the keys and values of the last block of the first segment are cached to be utilised by the first block of the second segment, and so on.

The outputs of all blocks $O_1, \ldots, O_B$ are then concatenated vertically:

$$H = [O_1; \ldots; O_B] \in \mathbb{R}^{\max\{l_u, l_v\} \times 8d}. \tag{23}$$

Average Pooling: The d_{out}-dimensional node representations of u and v are finally extracted from H:

$$h_u = \text{Mean}\left(H[:, : 4d]\right) W_{out} + b_{out} \in \mathbb{R}^{d_{out}}, \tag{24}$$

$$h_v = \text{Mean}\left(H[:, 4d :]\right) W_{out} + b_{out} \in \mathbb{R}^{d_{out}}, \tag{25}$$

where $W_{out} \in \mathbb{R}^{4d \times d_{out}}$ and $b_{out} \in \mathbb{R}^{d_{out}}$ are trainable parameters, and d_{out} is the output dimension.

A.3 Decoder

We inherit the decoders for the two downstream tasks, dynamic link prediction and dynamic node classification, from DyGLib. For dynamic link prediction, the decoder is a simple MLP with one ReLu-activated hidden layer. For dynamic node classification, the decoder is an MLP with two ReLu-activated hidden layers. The outputs of both decoders are just one value.

B Evaluation Methodology: Additional Details

B.1 Evaluation Metrics

We use two metrics: the **Average Precision (AP)** and the **Area Under the ROC (AUC)**.

In AP, the model being evaluated outputs a probability p for each query. The threshold τ is the probability at which the predictions are considered positive, i.e., for $p \geq \tau$ the query is considered positive, while for $p < \tau$ the query is considered negative. Average Precision is given by AP $= \frac{1}{N} \sum_{i=1}^{N} \left(r(\tau_i) - r(\tau_{i-1})\right) p(\tau_i)$, where $p(\tau_i)$ and $r(\tau_i)$ are the precision and recall values for the classifier with threshold τ_i. We set $r(\tau_0) = 0$. We use scikit's [85] implementation of this metric.

Area Under the ROC (AUC-ROC): The ROC curve plots recall against the false positive rate at various classification thresholds. AUC-ROC measures the area under the ROC curve, where higher scores are desirable. Intuitively, one can think of AUC-ROC as the probability that the model can distinguish a random positive sample from a random negative sample. Again, we use scikit's [85] implementation of this metric.

B.2 Design Details

Our implementation is integrated into DyGLib [134]. Our implementation of BRT is heavily based on Phil Wang's implementation, which can be found under the following link: `https://github.com/lucidrains/block-recurrent-transformer-pytorch/tree/main`.

B.3 Model Parameters

We split the datasets into train, val, and test by the ratio of 70%-15%-15%. We set the mini-batch size to 100, and use the Adam optimizer [67]. We use a learning rate of 0.0001, and train for 50 epochs. We employ early stopping with a patience of either 0 or 2 depending on the dataset. The node mini-batch sampling is done sequentially in order to follow the chronological order of the interactions. Other model parameters are as follows:

- Dimension of aligned encoding d: 50
- Dimension of time encoding d_T: 100
- Dimension of neighbor co-occurrence encoding d_C: 50
- Dimension of output representation d_{out}: 172
- Number of BRT cells: 2
- Position of the BRT horizontal cell: 1
- Number of attention heads I: 4
- BRT block size: 16
- BRT segment size: 32
- BRT number of state vectors: 32

Our methodology for hyperparameter selection follows DyGLib, an established library and benchmarking infrastructure for dynamic graph learning [134]. The values used for the parameter s_2 (defined in 3.3) were found by means of a grid search over the universe $\{0, 1\}$ for the datasets MOOC and LastFM, and over $\{0, 1, 2, 4\}$ for the dataset CanParl. These universes were chosen so as to test the model both with and without higher order structures. In the latter case, we keep the amount of considered 2-hop interactions low (in the order of magnitude of the amount of 1-hop interactions s_1). This allows us to consider higher order structures while avoiding adding potential noise. Finer tuning was limited by the available compute resources.

B.4 Dataset Details

The details of the datasets illustrated in Section 4 as well as dataset dependent parameters are in Table 1.

Data	Interactions	Sequence length	Patch size	Dropout rate	Patience
MOOC	411,749	256	8	0.1	2
LastFM	1,293,103	512	16	0.1	0
Can. Parl.	74,478	2048	64	0.1	2

Table 1: Overview of model configurations over various datasets.