Enhancing Deep Learning with Statistical Inference for Node Classification in Temporal Graphs

Jan von Pichowski
Vincenzo Perri
Lisi Qarkaxhija
Ingo Scholtes
jan.pichowski@uni-wuerzburg.de
Center for Artificial Intelligence and Data Science (CAIDAS)
University of Würzburg
Würzburg, Germany

Abstract

Modeling temporal and sequential patterns in temporal graphs is a critical research challenge in the development of time-aware GNNs. We propose a significance test based on a graph null model where timestamps are randomly shuffled to identify significant time-respecting paths, i.e., sequences of time-stamped edges that follow a temporal order. By combining this inference method with graph learning, we develop a two-step model, SIT-GNN, capable of capturing significant sequences in time-respecting paths. We demonstrate this novel capability with synthetic data and explain the enhanced classification performance in empirical data through an analysis with respect to the significance test. To the best of our knowledge, our work is the first to introduce statistically informed GNNs that leverage sequential patterns in terms of time-respecting paths. SIT-GNN represents a step towards bridging the gap between statistical graph inference and neural graph representation learning, with potential applications to static GNNs.

CCS Concepts

• **Computing methodologies → Neural networks**; *Supervised learning by classification*; Anomaly detection.

Keywords

graph neural networks, statistical inference, random graph ensembles, temporal graphs, higher-order networks

1 Introduction

Temporal graphs are graphs where time-stamped edges represent events between nodes that occur at specific times. They find wide application in various domains, including social networks [51, 55], analysis of epidemic spreading [37], and fraud detection [9]. This work focuses on classifying static node properties, i.e., properties that that do not change throughout the observation period such as, e.g., the role of nodes. For most methods, either time-stamped edges are aggregated in a static graph, and static labels are assigned, or both the graph and the node labels are considered dynamic. In this work, we consider dynamic aspects of the graph but seek to assign node labels that are static. We focus on two key aspects overlooked by most techniques: (1) capturing sequential patterns in time-respecting paths by employing a specialized graph representation and (2) incorporating the statistical significance of these patterns in temporal graph learning.

(1) In addition to the underlying temporal component, time-stamped edges also contains an overarching structural ordering that is overlooked when only considering edge frequencies on its own. The arrow of time imposes a time dependent relationship on time-stamped edges between nodes. Information can only flow from one node to another through an intermediary if the time-stamped edges are temporally ordered. If the event order is shuffled, the intermediary might lose the information because it interacted with the subsequent member beforehand.

(2) The absolute frequency of time-stamped edges alone is often insufficient for successful classification. For instance, when considering time-stamped interactions at a workplace, assigning roles based solely on the absolute number of interactions can lead to incorrect labeling. Let us assume managers are characterized by a higher number of interactions with other managers. By using the absolute frequency of pairwise interactions only, a standard graph algorithm may fail to correctly classify managers with an overall small number of interactions (e.g., introverted managers). In fact, a more effective metric should consider the relative importance of interactions by accounting for an individual's lower overall interaction frequency, rather than focusing exclusively on absolute counts.

In this work we address both aspects for the static node classification task in temporal graphs. We propose a two-step approach that integrates statistical inference with deep graph learning. Our architecture enhances classification performance in scenarios characterized by non-trivial patterns in the temporal ordering of edges. Importantly, it accounts for edge patterns that cannot be adequately captured by absolute frequencies alone, but require analysis of the statistical significance of temporally ordered edges. Our contributions are as follows:

(i) We introduce a temporal graph neural network designed for static classification that produces state-of-the-art classification results by effectively managing time-respecting paths, with statistically significant sequences.
(ii) We analyze five empirical temporal datasets, and create two synthetic ones, to investigate the patterns in temporal ordering and statistical significance that our method relies on to improve on the state-of-the-art.
(iii) We leverage the synergy between statistical inference methods and deep graph learning, supporting the claims on the fruitfulness of the approach recently suggested in [4].

Throughout the article, we substantiate our contributions by validating the following claims:

(C1) Utilizing edges based on their statistical significance, rather than raw frequency, increases network homophily — a topological property known to enhance GNN performance [68].
(C2) Our method is more efficient than standard temporal GNNs that rely on higher-order line-graphs or directly work with raw temporal event data.
(C3) Through statistical significance testing, our method captures fine-grained temporal patterns in time-respecting paths that remain undetectable even to existing temporal-order-aware approaches.
(C4) Consequently, it outperforms SOTA techniques in static node classification tasks on temporal graphs.

2 Related work

As our work addresses temporal graph data, it is related to the field of temporal GNNs. Temporal GNNs have been developed for both discrete- and continuous-time settings [38]. Discrete-time approaches segment the temporal data into time windows [23, 36, 52], thus aggregating temporal time-stamped edges and losing information on time-respecting paths within those time windows.

In contrast to the discrete-time setting, continuous-time approaches produce time-evolving node embeddings, focusing on the temporal variability of network activity at different time points, rather than on the patterns occurring across temporally-ordered event sequences [12, 34, 49].

These methods are commonly evaluated based on the prediction of dynamically changing node labels, which differs from our focus on predicting static node labels using temporal information. Our research area aligns with [47], which leverages sequential correlations in high-resolution timestamped data. In this context, the graph evolves dynamically, but the prediction target remains static. This contrasts with benchmarks used for temporal dynamic node prediction (TGB [27]) or non-temporal classification (OGB [25]). We specifically address static classification in temporal graphs, as exemplified by the social patterns datasets ([16, 19, 57]).

Furthermore, our method emphasize the importance of time-respecting paths, by applying structural changes to the graph, for accurately capturing the temporal dynamics relevant to static classification tasks. Data augmentation for graphs has been explored from various directions with the goal of allowing machine learning models to better generalize and attend to signal over noise [66].

Many methods have utilized heuristic graph modification strategies like randomly removing nodes [15], edges [48], or subgraphs [59, 62] to improve performance and generalizability. Other works have considered adding virtual nodes [28, 46] or rewiring the network topology, which also addresses oversquashing [1, 56], with graph transformers operating on a fully connected topology representing an extreme case [33, 41, 61]. Additionally, it has been shown that using graph diffusion convolutions instead of raw neighborhoods alleviates problems from noisy and arbitrarily defined edges in real-world graphs [18].

Network data augmentation has also been explored by going beyond pairwise connections, either through mediating node interactions via subgraphs [3, 11, 43, 65] or by utilizing higher-order graphs. Examples of higher-order approaches include simplicial networks [6], cellular complexes [5, 22], hypergraphs [10, 20, 26], and higher-order De Bruijn graphs modeling time-respecting paths [47].

Another area of research focused on learning the graph augmentations from the data. One approach is to perform graph augmentation as a preprocessing step, completely separate from the downstream task, where the graph structure is cleaned before being used as input to the GNN [30, 67]. Others embed the augmentation strategy into an end-to-end differentiable pipeline, jointly learning the optimal graph representation and the downstream task [14, 17, 29, 31, 39].

3 Approach

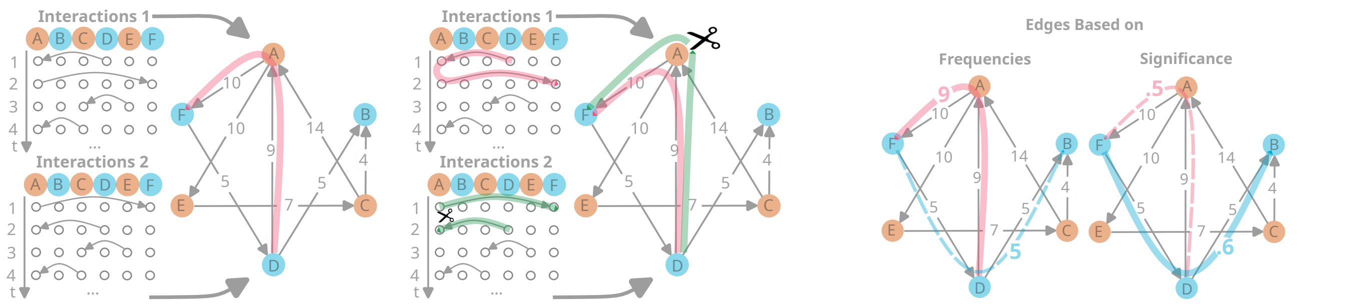
Before introducing our approach, we first clearly state the problem. Our work focuses on high-resolution temporal graph data that consist of time-stamped edges between nodes. We address the static classification task, which involves classifying nodes based on the information derived from these time-stamped edges. This task is significant due to its wide range of applications, such as predicting user roles in social networks, detecting fraudulent members in financial transactions, and identifying influential nodes in communication networks. Most successful solutions, such as those compared in [47], encode time-stamped edges in a static graph, as illustrated in Figure 1a. Here edges are static and weights count the number of time-stamped edges between two nodes. However, this approach ignores the temporal aspect of the data and the significance of time-stamped edge activation frequencies with respect to a random baseline distribution.

We base our method on two concepts: We first introduce a statistical inference technique using random graph ensembles that are based on the node and edge statistics in a temporal graph. We then specifically show how to encode significantly over-represented time-respecting paths in a temporal graph as *time-respecting edges* in higher-order De Bruijn graphs [63], which we can use as basis for a graph neural network. This leads to the definition of *significant edges* that form the base of our *Significance Inferring Temporal Graph Neural Network* (SIT-GNN) model.

3.1 Statistical inference with random graph ensembles

Random graph ensembles are a powerful tool for statistical inference in network analysis. They allow to generate a distribution of graphs that share certain aggregate properties with the observed network, providing a baseline for comparison. By comparing the observed network with a random graph ensemble, we can identify significant edges that deviate from randomness. This information about the significance of the edges can then be used as input to a graph neural network, enhancing its performance.

To this end, we employ the configuration model [42] that randomly connects node stubs. Manually connecting the node stubs for all possible solutions is not analytically tractable. Therefore, we use the soft configuration model [7] that only fixes expected node degrees, allowing the problem to be reformulated as an urn problem. By using hypergeometric distributions as described in [35], we can perform statistical tests to determine if edges are over- or under-represented with respect to the underlying node degrees.



(a) Static time aggregated edges. For both datasets the time-stamped edges are converted to the same static graph by counting the occurrences. There is a path DA, AF in both time aggregated graphs.

(b) Time-respecting paths. A time-respecting path or an edge in the second-order De Bruijn graph respects the temporal order of time-stamped edges such that edge DAF only exists for dataset (1).

(c) Significant edges. Considering time-aggregated 2nd-order edge frequencies leads to the assumption that edge DAF is more important than FDB. However, the underlying 1st-order edges of DAF are also more present. FDB is more significant because it is less frequent but covers its 1st-order edges better.

Figure 1: Illustrative example of our approach, highlighting that less frequent time-stamped edges may be more over-represented than more frequent time-stamped edges.

An edge is over-represented if it is observed more frequently than would be expected by chance.

In our case, only over-represented edges are significant for the graph neural network since they encode edges that stand out from randomness. We compute the *significance* $s(e_{ij})$ of the edge e_{ij} using the probability mass function for the hypergeometric distribution:

$$s(e_{ij}) = \Pr(X_{ij} \leq A_{ij}), \quad \text{with} \quad \Pr(X_{ij} = A_{ij}) = \frac{\binom{\Xi_{ij}}{A_{ij}}\binom{M-\Xi_{ij}}{m-A_{ij}}}{\binom{M}{m}} \tag{1}$$

where $s(e_{ij})$ is the probability of edge e_{ij} being over-represented, A_{ij} is the observed frequency of edge e_{ij} and X_{ij} is any random instance of the ensemble. To obtain a random instance, we could sample $m = \sum_{i,j \in V} A_{ij}$ equiprobable edges without replacement from the multi-set of size $M = \sum_{i,j \in V} \Xi_{ij}$ of in- and out-stub pairs, where the total number of stub combinations between two vertices i, j is given by $\Xi_{ij} = d_i^{\text{out}} d_j^{\text{in}}$. Here, d_i^{out} and d_j^{in} are the out- and in-degrees of nodes i and j, respectively. However, this formula follows a hypergeometric distribution such that we solve it analytically. To ensure that non-existent edges are excluded from consideration, we adapt Ξ by redistributing the stub-pairs as described in [35]. We call an edge *significant edge* if the observed frequency is over-represented with respect to a given predefined threshold, e.g., $s(e_{ij}) \geq 0.95$.

3.2 Encoding time-respecting paths in higher-order De Bruijn graphs

A graph $G = (V, E)$ is defined as a set of nodes V representing the elements of the system, and a set of edges $E \subseteq V \times V$ representing their direct connections. However, it is often important to consider how nodes influence one another through a *path*, which is an ordered sequence $(v_1, v_2, \dots, v_l)$ of nodes $v_i \in V$. In a path, all node transitions must correspond to edges in the graph, i.e., $e_i = (v_i, v_{i+1}) \in E; \forall i \in [0, l-1]$.

Paths are often inferred from edges based on a *transitivity assumption*. This assumption states that if there is an edge (v_0, v_1) with transition probability α, and an edge (v_1, v_2) with transition probability β, then the path (v_0, v_1, v_2) will be observed with probability $\alpha \cdot \beta$. In other words, the transitions are considered to be independent.

The transitivity assumption simplifies the modeling of a path by expressing its probability as the product of the individual edge transition probabilities. However, this assumption often fails in temporal networks $G^t = (V, E^t)$, where $E^t \subseteq V \times V \times \mathbb{N}$ as edges have timestamps. In temporal networks, the ordering of edges can play an important role in determining the likelihood of observing certain paths. A *time-respecting* path is defined as a sequence of edges $((v_0, v_1, t_1), \dots, (v_i, v_{i+1}, t_i), \dots, (v_{n-1}, v_n, t_n))$ that $\forall i \in [0, l-1]$ respects two conditions: (i) transitions respect the order of time $t_i > t_{i-1}$, and (ii) $t_i - t_{i-1} \leq \delta$, where δ is a parameter controlling the maximum time distance for considering interactions temporally adjacent. Therefore, different from what we would get by discarding time and using the transitivity assumption, the two edges (v, w, t_1) and (u, v, t_2) form a *time-respecting path* only if $t_2 > t_1$.

To capture time-respecting sequential patterns, higher-order De Bruijn graphs model the probabilities of path sequences explicitly. These models construct a representation that respects the topology of the original graph and the frequencies of observed paths of a given length k. Specifically, a higher-order network of the k-th order is defined as an ordered pair $G^{(k)} = (V^{(k)}, E^{(k)})$, where $V^{(k)} \subseteq V^k$ are the higher-order vertices, and $E^{(k)} \subseteq V^{(k)} \times V^{(k)}$ are the higher-order edges. Each higher-order vertex $v =: \langle v_0 v_1 \dots v_{k-1} \rangle \in V^{(k)}$ is an ordered tuple of k vertices $v_i \in V$ from the original graph. The higher-order edges connect higher-order nodes that overlap in exactly $k-1$ vertices, similar to the construction of high-dimensional De Bruijn graphs [13]. The weights of the higher-order edges in $G^{(k)}$ represent the frequency of paths of length k in the original graph. Specifically, the weight of the edge $(\langle v_0 \dots v_{k-1} \rangle, \langle v_1 \dots v_k \rangle)$ counts how often the path $\langle v_0 \dots v_k \rangle$ of length k occurs. By explicitly modeling the probabilities of these higher-order path sequences, the higher-order network representation can capture patterns and dependencies that may be missed when relying on the transitivity assumption [54]. Figure 1b illustrates the time-respecting paths.

3.3 Significant time-respecting paths

We combine the two concepts of significant edges and time-respecting paths by applying the statistical inference method to the higher-order De Bruijn graph. Here, the significance of a time respecting-path $\langle v_0, v_1, v_2\rangle$ is determined with respect to the frequency of the time respecting sub-paths $\langle v_0, v_1\rangle$ and $\langle v_1, v_2\rangle$. The path $\langle v_0, v_1, v_2\rangle$ is described by the edges in a second-order De Bruijn graph and the sub-paths are its nodes. As a result, the inference method can directly be applied to the second-order edges of the De Bruijn graph to obtain the *significant edges* that encode time-respecting paths. Figure 1c illustrates $s(e_{ij})$ for second-order edges and shows how the significance score differs from the absolute frequency of the edges.

3.4 SIT-GNN: Significance Inferring Temporal Graph Neural Network

We combine the two key concepts of significant and time-respecting De Bruijn graph edges in our model, SIT-GNN. The model is designed to handle the static classification task for temporal graphs. It consists of two main components: the statistical inference module and the deep graph neural network.

Figure 2 illustrates the architecture of our model. In a preprocessing step, we transform time-stamped edges into a static time-aggregated graph $G = (V, E, F)$ and a second-order De Bruijn graph $G_t = (V_t, E_t, F_t)$, containing time-stamped edge and time-respecting paths with their observed frequencies F and F_t, respectively. For both graphs, we calculate the significance of edges $S = \{s(e_{ij})\,\forall\, e_{ij} \in E\}$ and $S_t = \{s(e_{ij})\,\forall\, e_{ij} \in E_t\}$ using the statistical inference module that solves Equation (1). Edges in the graphs are attributed with these probabilities, leading to the significant graph $\hat{G} = (V, E, S)$ and the significant second-order De Bruijn graph $\hat{G}_t = (V_t, E_t, S_t)$.

The learnable model consists of two GNNs that operate on $\hat{G}$ and $\hat{G}_t$. We encode the significance as edge weights such that the impact of insignificant edges is reduced during message passing. For the used empirical datasets, the significance values S and S_t are mostly either 0 or 1 (Figure 4), such that insignificant edges are ignored during message passing. For simplicity, we use a three-layer GCN, but any GNN that supports edge weights can be integrated into our approach. Following [32], we define the propagation rule with the normalization constant c_{ij} and include the significance $s(e_{ij})$:

$$h_{v_i} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \frac{s(e_{ij})}{c_{ij}} f_{v_j} W\right) \forall\, v_i \in V, \tag{2}$$

$$h_{t_{v_i}} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \frac{s(e_{ij})}{c_{ij}} f_{t_{v_j}} W_c\right) \forall\, v_i \in V_t \tag{3}$$

Our model utilizes provided features $f \in X$ (not to confuse with the random variable X_{ij}) in both GNNs to obtain node embeddings H and H_c, respectively. Since most datasets contain only first-order features, we design a lift operation based on a bipartite graph $B_l = (V, V_t, E_l)$ to map the features f to the second-order graph as f_t. We take a Markovian perspective and map each node $v_i \in V$ to the nodes $v_{ij} \in V_t$ that represent an edge $e_{ij} \in E$ starting in node v_i. After message passing on the time-respecting graph, the features are lifted back to the first-order graph and merged with the inverse bipartite graph $B_m = (V, V_t, E_m)$. The lift and merge operations are again defined with a GNN applied to the bipartite graphs B_l and B_m:

$$f_{t_{v_i}} = \sigma\left(\mathcal{F}\left(\{f_{u_i} \mid u_i \in V \wedge (u_i, v_i) \in E_l\}\right) W_l\right) \forall\, v_i \in V_t \tag{4}$$

$$h_{m_{v_i}} = \sigma\left(\mathcal{F}\left(\{h_{v_i} + h_{m_{u_i}} \mid u_i \in V_t \wedge (u_i, v_i) \in E_m\}\right) W_m\right) \forall\, v_i \in V \tag{5}$$

We use MEAN as the aggregation function $\mathcal{F}$. Finally, the merged embedding H_m is linearly transformed to produce the class output Y'.

The core idea of our model is the integration of statistical inference with deep graph learning. The resulting model respects time-respecting paths and emphasizes significant edges during the message passing process in a higher-order De Bruijn graph. In the following sections, we evaluate the effectiveness of these properties and assess the performance of our model on the static classification task for temporal graphs with high-resolution time-stamped edges.

4 Analysis

Before evaluating the classification performance of SIT-GNN, we analyze the impact of significance testing on the available data **(C1)** and discuss the efficiency of our method **(C2)**. To this end, we employ various benchmark datasets from SocioPatterns (*Hospital* [57], *Student* [53], *Workplace* [19], *School11* and *School12* [16]). Details about these datasets are provided in Appendix A.

4.1 Increased homophily through significance testing (C1)

First, we address claim **C1**, that using only significant edges reveals homophilic structures and thus supports our method. As we are interested in the impact for our method, we focus on the edges in the datasets that facilitate message passing in GCNs. They update a node's embedding based on information from its neighboring nodes. Classification is more straightforward for GCNs when the neighboring nodes are of the same class [68], indicating a homophilic network. As defined in [45], the node homophilic ratio (NHR) is based on the node's v neighborhood $\mathcal{N}(v)$:

$$\text{NHR} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{|\{(u, v) : u \in \mathcal{N}(v) | y_u = y_v\}|}{|\mathcal{N}(v)|} \tag{6}$$

Figure 3a shows that considering only edges with at least 50% significance increases the NHR for all datasets except for *Student*.

However, this ratio does not account for class sizes and ignores edge weights, even though most GCNs can handle them. To address this, we introduce a node homophily ratio considering *balanced* class weights w_c and edge *weights* $w(u, v)$:

$$\text{BWNHR} = \frac{1}{|C|} \sum_{c \in C} w_c \left[\frac{1}{|\mathcal{V}_c|} \sum_{v \in \mathcal{V}_c} \frac{\sum_{u \in \mathcal{N}(v) | y_u = y_v} w(u, v)}{\sum_{u \in \mathcal{N}(v)} w(u, v)}\right], \tag{7}$$

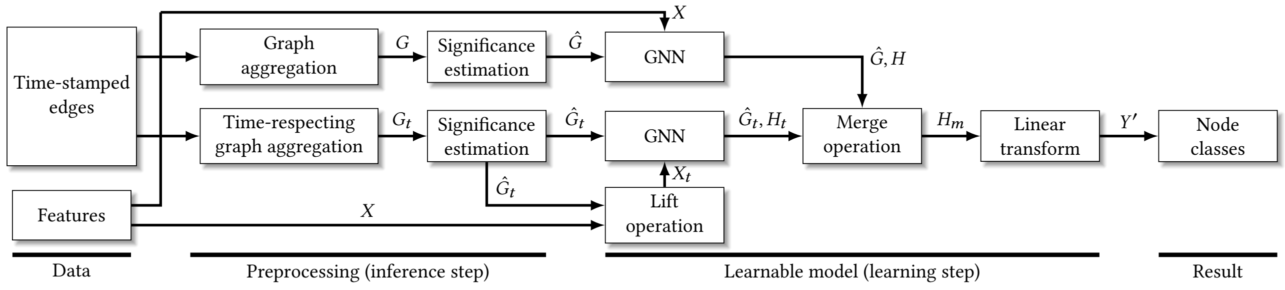
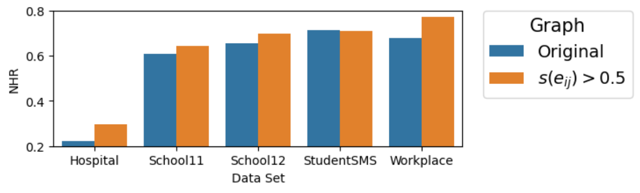
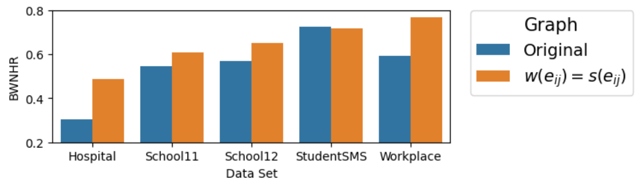


Figure 2: The SIT-GNN architecture consist of two main components: the statistical inference module and the deep graph neural network. Both parts are applied to the original static graph and to the time-respecting encoding De Bruijn graph. Lift and merge operations transform the features and embeddings in the desired space.



(a) NHR for original graphs compared to graphs with only sufficient significant ($s(e_{ij}) > 0.5$) edges.



(b) BWNHR for original graphs without edge weights compared to edges with significance score $s(e_{ij})$.

Figure 3: Considering the significance for the impact of edges leads to an increase of homophily, measured in NHR and BWNHR, for most of the used empirical datasets.

with $w_c = |\mathcal{V}|/(|\mathcal{V}_c||C|)$. Using this score, we assess whether weighting the edges with the significance score results in nodes having, on average, more adjacent nodes of the same class. Figure 3b demonstrates that weighting with the significance score yields a higher BWNHR for all datasets except *Student*. Notably, for *Hospital* and *Workplace*, the differences are substantial, with an increase of nearly 0.2. This indicates that most datasets contain a homophilic relationship revealed by the significance test. Therefore, we anticipate that our model will perform best on *Hospital* and *Workplace*.

Both scores lead to similar results even though for one score we consider a pruning at $s(e_{ij}) > 0.5$ and for the other we consider the significance score as edge weight. Considering Figure 4 we observe that the significance scores are mostly either 0 or 1, such that they effectively prune edges. For the *Student* dataset the distribution is more uniform, preventing an increase of homophily through pruning.

It is worth noting that the *Student* dataset is the only dataset based on real link data, whereas the other datasets are derived from proximity data. Hence, the noise in the other dataset is more pronounced, and it disguises the homophily because the noise consist of predominantly heterophilic edges. This explains why removing noisy edges with the significance test effectively increases homophily.

4.2 Runtime complexity (C2)

In this section, we address claim **C2** that our method works efficiently on empirical data and is in particular more efficient than standard temporal GNNs relying on higher-order line-graphs or raw observations of time-stamped edges. We first discuss theoretical arguments limiting the runtime and then provide empirical observations to support our claims. In the performance evaluation, we measure that our approach has a runtime in the same or lower order of magnitude as the used baselines.

Our method involves message passing on both first-order graphs and second-order De Bruijn graphs. Thus, the runtime is primarily influenced by the potentially larger second-order De Bruijn graph. A line graph represents an upper bound for the De Bruijn graph with a complexity of $|E_{\text{LINE}}| = |E|^2 = O(|V|^4)$. However, the De Bruijn graph only includes time-respecting paths of length two, reducing the complexity further to $O(|V|^3)$, which is significantly lower than that of a line graph. Typically for empirical data, first-order graphs are sparse, with $|E| = O(|V|)$, leading to $|E|^2 = O(|V|^2)$. Consequently, for sparse graphs, the number of edges in a second-order De Bruijn graph has an upper limit significant lower than $O(|V|^2)$, or even $O(|V|)$ depending on the temporal patterns in the graphs. [47] further refine this by noting that these edges are bounded by $\sum_{ij} A_{ij}^2 \leq |V|^2$, where A^2 is the second power of the binary adjacency matrix A of the first-order graph.

While these theoretical upper bounds provide an idea of the worst-case limits, we now consider statistics from empirical data. In Figure 5, we compare the major drivers for runtime complexity of baseline methods used in Section 5.2 with respect to the datasets. These include the node count $|V|$, first-order edge count $|E|$, second-order De Bruijn graph edge count $|E^{(2)}|$, and number of

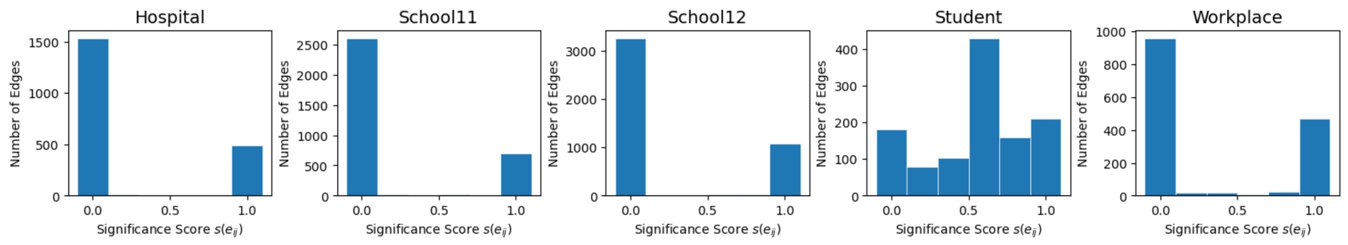


Figure 4: Distribution of first-order edges with respect to the significance scores $s(e_{ij})$.

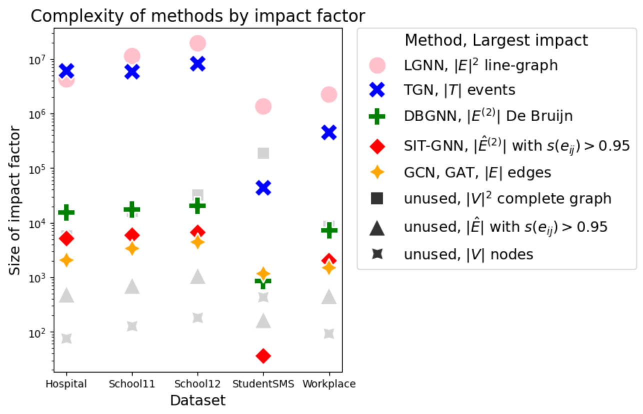


Figure 5: This graph shows how often message passing (MP) is performed in each epoch. TGN performs MP event-wise, GCN for each edge and DBGNN for each edge in the second-order De Bruijn graph. Our method relies on the second-order De Bruijn graph but only considers significant edges. As a result, our method performs MP in the same order of magnitude as GCN for the given empirical datasets.

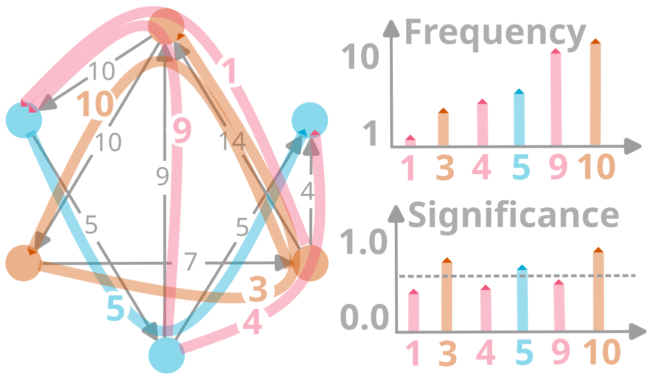


Figure 6: Small synthetic example showing two-step transitions between nodes of two classes. The frequency of intra (orange, blue) and inter (pink) class transitions are indistinguishable. Considering the one-step transitions as base distribution leads to significance values that makes it possible to distinguish between intra and inter class transitions. Thus, a transductive classification where the class is determined based on a subset of known nodes becomes simpler due to the stronger association of same class nodes.

time-stamped edges, respectively events, $|T|$. We also include statistics for the graphs with insignificant edges removed $|\hat{E}^{(2)}|$ and for the upper limit of the line graph $|E|^2$. Note that for most methods including ours the number of time-stamped edges does not impact the complexity because they are aggregated for each underlying edge and transformed to a significance score in a preprocessing step. For the datasets considered, the higher-order De Bruijn graph with pruned insignificant edges is of the same magnitude as the first-order graph or even smaller for very sparse data. This is due to the sparsity of the first-order graph and the pruning of non-time-respecting paths. Pruning insignificant edges significantly reduces the edge count, making our method as efficient as previous works. The effective speedup depends on the pruning constraint, here 0.95. Even without pruning, our method's worst-case runtime is bounded by DBGNN [47], making it significantly more efficient than both TGN [12] and LGNN [24]. In the experimental evaluation, we confirm this observation by measuring the method's per epoch training time (Table 1).

Other commonly used benchmark datasets, such as TGB [27] or OGB [25], often have a larger number of time-stamped edges and/or nods but still result in sparse graphs. Therefore, we do not anticipate that applying our method to these tasks will be constrained by runtime complexity. Instead, the main challenge lies in adapting our method to different tasks as presented in those datasets, which is an area for future work.

5 Experimental Evaluation

Previously, we have shown that significance testing reveals a more homophilic relationship in empirical data. Next, we further pinpoint this observation to a concrete synthetic example to address whether this significance test also increases the learning capabilities of our model. Lastly, we show that our approach is also valid in empirical data leading to superior performance.

5.1 Capturing temporal patterns (C3)

We claim (C3) that our model can learn patterns that other models cannot. To demonstrate this, we generate synthetic data with a specific pattern that is published with our code.

To give a connection to a practicable case: We identify the role of managers in workplace based on their communication. Even though the managers interact mainly with other managers the raw interaction count between them is not meaningful enough because the total interaction activity of the managers varies heavily.

Formally, we address a transductive node classification task for a graph where node activations (in terms of degrees), edges, and time-respecting edge frequencies vary. We bias time-respecting edges towards predominant class association patterns, meaning nodes of the same class are more often connected by a time-respecting edge. The dataset is published together with the code (Appendix B). Our hypothesis is that our model can learn this pattern and classify unseen nodes based on dominant time-respecting paths, whereas other methods relying on mere edge frequency fail due to the varying frequencies.

Figure 6 illustrates a reduced example with varying time-respecting edge frequencies. Distinguishing associative from non-associative edges based on frequency alone is impossible. However, associative time-respecting second-order edges appear more frequently than expected when considering underlying first-order edge frequencies, making them detectable with a cutoff value.

Figure 7 shows that only our model can learn this distinction and separate the two node classes. Randomizing edge order removes the bias in terms of significant time-respecting information, making the data purely random and reducing our model's performance to random guessing. This confirms that our model learns the encoded temporal pattern and not random fluctuations in the data.

5.2 Evaluation on empirical data (C4)

To address claim (C4), we compare SIT-GNN to competitive baselines on five common empirical datasets for static node classification on temporal graphs. We consider baselines from the deep learning and representation learning domains, as well as methods from other domains, such as static node classification and dynamic node classification.

HONEM [60] and EVO [2] are two representation learning methods specifically designed for temporal graphs. LGNN [8] and DBGNN [47] are deep learning methods tailored for temporal graphs. We adapt the static node classification methods Node2Vec and GCN to the temporal setting by considering the temporal graph as a static graph. This helps to understand whether considering temporal information is beneficial for solving the task. GAT [58] is added to understand if the attention mechanism can replicate the performance of our significance testing GNN.

Models for dynamic node classification need to be thoroughly adapted to the static setting, as they are designed for a different task. This is a challenging problem on its own, but for reference, we consider an adaption of TGN [49], which is currently state-of-the-art in the TGB [27] leaderboard, except for the simple Moving Average heuristic, which cannot be mapped to our task. This helps to understand whether dynamic methods are relevant for the static temporal case.

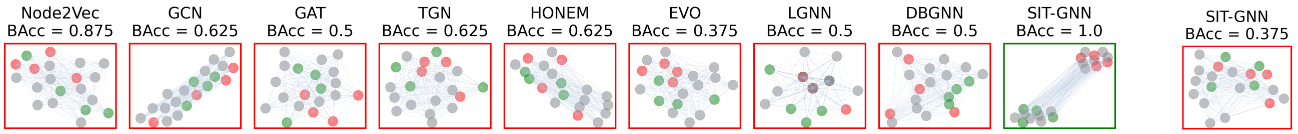
We conduct an extensive parameter search for all models to ensure fair comparison. For the representation learning models Node2Vec and EVO we adhere to the original configurations, i.e. we use an embedding size of $d = 128$ and a random walk length of $l = 80$, repeated $r = 10$ times. As context size we use $k = 10$. For Node2Vec we select the return parameter (p) and the in-out parameter (q) from the set $0.25, 0.5, 1, 2, 4$. The deep learning models (GCN, GAT, LGNN, DBGNN, and our proposed model) consist of three layers. Following the approach of [47], we set the size of the last layer to $h_2 = 16$, while the sizes of the preceding layers are determined during model selection. The study range for h_0 and h_1 encompasses $4, 8, 16, 32$ over a maximum of 5000 epochs as per [47]. The time-respecting path length is fixed to $k = 2$ for SIT-GNN and DBGNN because it is shown as optimal in [47] for the given datasets. Stochastic Gradient Descent (SGD) serves as our optimization function, with the learning rate set to 0.001, which showed the best performances on average for all tested models. We use dropout regularization with a dropout rate of 0.4 to mitigate overfitting and we incorporate class weights in the loss function to address imbalanced training datasets.

To compare various GNN architectures, we adopt a conventional approach. We split the data into a 10% test set and a 90% training set, which is further divided into a training and validation set (80%/20%). Subsequently, we select the best-performing model and epoch based on its validation set performance. Finally, we evaluate the chosen model's performance on the test set, reporting the mean and standard deviation of the balanced accuracy across all 10 repetitions. We evaluate the models using the balanced accuracy metric to account for class imbalance. For comparability, we use the same folds and splits for all experiments. Besides the random splits, the random initialization of the model also contributes to the variability captured by the standard deviation. For reproducibility, we fix the random splits and reuse a common seed in every repetition for the random initialization of model weights and dropout candidates. Additional details about the models and experiments are outlined in Appendix B and C, including an ablation study about the different edge types.

Table 1 shows the results. Our model outperforms all other models on all datasets except for the *Student* dataset where it is on par with the best model. This confirms the analysis from section 4.1 and meets our expectations. The observed gain is strongest for the hospital and workplace datasets. Except for *Student*, neither static nor dynamic methods are competitive, indicating that the temporal aspect is crucial for the task. Complementing our analysis in section 4.2, we also show that the training time of our method is in the same order of magnitude as for static GNNs and significantly lower than for competing baseline methods.

5.3 Limitations

In this work, we focus on sequences of time-stamped edges, but do not incorporate additional temporal features or edge attributes that could further enhance our method. We made a limited use of the framework of hypergeometric statistical ensemble, which can be extended with non-homogeneous edge propensities [7]. Additionally, time-respecting edges of order $k > 2$ could be considered even though $k = 2$ has been shown to be optimal for these datasets [47]. Other random graph ensembles may be explored to introduce domain-specific, non-trivial biases. We demonstrate that our approach also increases homophily in static first-order graphs, suggesting that applying this method to non-temporal static graphs is a promising direction for future research. Our inference method performs particularly well on temporal graphs for static classification; Future work could explore augmenting standard temporal



(a) Data set with biased temporal order of edges towards predominant class association. **(b) Randomized**

Figure 7: Test balanced accuracy (BAcc) and t-SNE plot of learned embeddings (train split is gray). Only our method successfully learns the pattern in the synthetic data. When the data is randomized, our method's performance drops to random guessing, indicating that it effectively captures the underlying pattern, which is disrupted by randomization.

Table 1: Model per epoch training time and performance in terms of balanced accuracy (BAcc) for the temporal static node classification task (temp.) across various empirical datasets. We mark the best models in bold and second in *italic.* We include a diverse set of baselines, comprising deep learning (deep) and representation learning (emb.) methods. For reference, we also adapt and include methods from other domains, such as static node classification (static) and dynamic node classification (dyn.).

Method			Hospital		School11		School12		Student		Workplace	
Model	Type	Graph	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)
Node2Vec	emb.	static	*56.0* ± 7.4	8.9 ± 0.6	54.0 ± 5.5	11.7 ± 0.7	59.0 ± 10.6	13.6 ± 0.9	56.1 ± 2.5	41.9 ± 6.0	*76.5* ± 2.9	10.1 ± 0.2
GCN	deep	static	42.8 ± 2.5	2.7 ± 0.6	48.8 ± 5.5	3.3 ± 0.2	*61.3* ± 4.8	4.0 ± 0.3	56.5 ± 5.1	3.2 ± 0.4	30.6 ± 8.8	3.2 ± 0.1
GAT	deep	static	24.6 ± 0.6	5.5 ± 0.3	54.7 ± 5.9	4.8 ± 0.3	44.0 ± 12.0	5.4 ± 0.4	**62.9** ± 6.3	5.2 ± 0.2	28.8 ± 9.2	4.8 ± 0.3
TGN	deep	dyn.	39.1 ± 6.2	1173.7 ± 184.2	55.2 ± 6.2	1225.7 ± 40.3	52.1 ± 12.4	2147.0 ± 285.4	49.3 ± 5.0	1360.8 ± 40.9	54.9 ± 14.9	454.8 ± 10.6
EVO	emb.	temp.	26.5 ± 5.4	12.2 ± 0.6	52.2 ± 7.1	12.8 ± 1.7	50.7 ± 8.2	18.6 ± 1.4	45.8 ± 2.96	48.5 ± 10.1	19.7 ± 6.3	13.6 ± 1.5
HONEM	emb.	temp.	48.8 ± 6.0	173.9 ± 9.4	*57.9* ± 6.7	246.3 ± 26.2	53.3 ± 5.9	480.1 ± 54.1	57.9 ± 3.56	289.5 ± 37.4	75.1 ± 8.9	170.7 ± 69.5
LGNN	deep	temp.	44.3 ± 5.7	15.4 ± 0.9	55.1 ± 8.7	14.7 ± 0.3	48.1 ± 7.5	15.9 ± 1.1	56.8 ± 5.54	15.4 ± 1.3	75.1 ± 7.2	15.1 ± 0.7
DBGNN	deep	temp.	40.3 ± 5.9	4.0 ± 0.6	39.8 ± 8.2	4.7 ± 0.7	59.1 ± 9.1	5.0 ± 0.4	51.2 ± 6.17	4.7 ± 0.7	69.2 ± 3.7	4.6 ± 0.1
SIT-GNN	deep	temp.	**58.8** ± 6.2	6.8 ± 0.1	**58.7** ± 5.5	6.1 ± 0.2	**63.77** ± 9.47	6.9 ± 0.4	*62.6* ± 4.2	5.7 ± 1.4	**78.0** ± 8.2	5.5 ± 0.2

GNN architectures. For such cases, a suitable null model for the evolving graph structure is needed, ideally one that fits into an online inference setup.

6 Conclusion

In this work, we deepen the synergy between statistical inference methods and deep graph learning, leading to the proposal of SIT-GNN, a novel deep graph learning architecture that accounts for time-respecting paths and significant edges in high resolution temporal graph data. Unlike existing graph learning methods that employ message passing on temporal data, our method facilitates a two-step approach. First, it infers significant sequential patterns based on an analytically traceable null model that preserves both topology and frequency, but not the temporal ordering of time-stamped edges. Second, we capture these significant patterns in a De Bruijn graph that encodes time-respecting paths, enabling our model to learn significant sequences in time-respecting paths. We demonstrate this novel capability on a synthetic dataset and explain the additional benefit of increased homophily on empirical datasets. An empirical evaluation shows that our proposed method reliably improves node classification performance over competitive baselines, with peak performance observed in datasets that experience the largest increase in homophily through significance testing. This finding highlights that the innovative combination of statistical inference and neural message passing, which is the key contribution of our work, leads to considerable advantages for temporal graph learning.

Bridging the gap between the application of statistical graph ensembles in network science and deep graph learning, we finally argue that our work opens broader perspectives for the integration of statistical graph inference, graph augmentation, and neural message passing. In particular, applying our method to the inference of (first-order) edges in static graphs could be a promising approach to address the issue that empirical graphs are rarely unspoiled reflections of reality, but are often subject to measurement errors and noise. Our analysis highlights the homophily-enhancing properties of our method, which also extend to first-order graphs, paving the way for further research. The need to combine graph inference techniques with neural message passing [40, 44, 64] has recently been identified as a major challenge for deep graph learning, and our work can be seen as a step in this direction.

Acknowledgments

JP and IS acknowledge funding by the German Ministry of Science and Technology (BMBF) under grant agreement No. 01IS24072A (COMFORT).

References

[1] Federico Barbero, Ameya Velingker, Amin Saberi, Michael M. Bronstein, and Francesco Di Giovanni. 2024. Locality-Aware Graph Rewiring in GNNs. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=4Ua4hKiAJX

[2] Caleb Belth, Fahad Kamran, Donna Tjandra, and Danai Koutra. 2020. When to remember where you came from: node representation learning in higher-order networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (Vancouver, British Columbia,

Canada) *(ASONAM '19)*. Association for Computing Machinery, New York, NY, USA, 222–225. doi:10.1145/3341161.3342911

[3] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. 2022. Equivariant Subgraph Aggregation Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=dFbKQaRk15w

[4] Christopher Blöcker, Martin Rosvall, Ingo Scholtes, and Jevin D. West. 2025. Insights from Network Science can advance Deep Graph Learning. arXiv:2502.01177 [cs.LG] https://arxiv.org/abs/2502.01177

[5] 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.

[6] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. 2021. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*. PMLR, 1026–1037.

[7] Giona Casiraghi and Vahan Nanumyan. 2021. Configuration models as an urn problem. *Scientific Reports* 11, 1 (June 2021). doi:10.1038/s41598-021-92519-y

[8] Zhengdao Chen, Lisha Li, and Joan Bruna. 2019. Supervised Community Detection with Line Graph Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1g0Z3A9Fm

[9] Dawei Cheng, Yao Zou, Sheng Xiang, and Changjun Jiang. 2025. Graph neural networks for financial fraud detection: a review. *Frontiers of Computer Science* 19, 9 (Jan. 2025). doi:10.1007/s11704-024-40474-y

[10] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. 2022. You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=hpBTIv2uy_E

[11] Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. 2021. Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems* 34 (2021), 1713–1726.

[12] da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJeW1yHYwH

[13] Nicolaas Govert De Bruijn. 1946. A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam* 49, 7 (1946), 758–764.

[14] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 22667–22681.

[15] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems* 33 (2020), 22092–22103.

[16] Julie Fournet and Alain Barrat. 2014. Contact Patterns among High School Students. *PLoS ONE* 9, 9 (Sept. 2014), e107878. doi:10.1371/journal.pone.0107878

[17] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *International conference on machine learning*. PMLR, 1972–1982.

[18] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *Advances in neural information processing systems* 32 (2019).

[19] Mathieu Génois, Christian L Vestergaard, Julie Fournet, André Panisson, Isabelle Bonmarin, and Alain Barrat. 2015. Data on face-to-face contacts in an office building suggest a low-cost vaccination strategy based on community linkers. *Network Science* 3, 3 (2015), 326–347. http://www.sociopatterns.org/datasets/contacts-in-a-workplace/

[20] Dobrik Georgiev Georgiev, Marc Brockschmidt, and Miltiadis Allamanis. 2022. HEAT: Hyperedge Attention Networks. *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=gCmQK6McbR

[21] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[22] Mustafa Hajij, Ghada Zamzmi, Theodore Papamarkou, Nina Miolane, Aldo Guzmán-Sáenz, Karthikeyan Natesan Ramamurthy, Tolga Birdal, Tamal K Dey, Soham Mukherjee, Shreyas N Samaga, et al. 2022. Topological deep learning: Going beyond graph data. *arXiv preprint arXiv:2206.00606* (2022).

[23] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. *Advances in neural information processing systems* 32 (2019).

[24] Frank Harary and R. Z. Norman. 1960. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo* 9 (1960), 161–168. https://api.semanticscholar.org/CorpusID:122473974

[25] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *CoRR* abs/2005.00687 (2020). arXiv:2005.00687 https://arxiv.org/abs/2005.00687

[26] Jing Huang and Jie Yang. 2021. UniGNN: a Unified Framework for Graph and Hypergraph Neural Networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*.

[27] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2023. Temporal Graph Benchmark for Machine Learning on Temporal Graphs. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 2056–2073. https://proceedings.neurips.cc/paper_files/paper/2023/file/066b98e63313162f6562b35962671288-Paper-Datasets_and_Benchmarks.pdf

[28] EunJeong Hwang, Veronika Thost, Shib Sankar Dasgupta, and Tengfei Ma. 2021. Revisiting virtual nodes in graph neural networks for link prediction. (2021).

[29] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11313–11320.

[30] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 66–74.

[31] Anees Kazi, Luca Cosmo, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael M Bronstein. 2022. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 2 (2022), 1606–1617.

[32] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJU4ayYgl

[33] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems* 34 (2021), 21618–21629.

[34] 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 (KDD '19)*. ACM, 1269–1278. doi:10.1145/3292500.3330895

[35] Timothy LaRock, Vahan Nanumyan, Ingo Scholtes, Giona Casiraghi, Tina Eliassi-Rad, and Frank Schweitzer. 2020. *HYPA: Efficient Detection of Path Anomalies in Time Series Data on Networks*. Society for Industrial and Applied Mathematics, 460–468. doi:10.1137/1.9781611976236.52

[36] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology* 58, 7 (2007), 1019–1031. doi:10.1002/asi.20591

[37] Zewen Liu, Guancheng Wan, B. Aditya Prakash, Max S.Y. Lau, and Wei Jin. 2024. A Review of Graph Neural Networks in Epidemic Modeling. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6577–6587. doi:10.1145/3637528.3671455

[38] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, franco scarselli, and Andrea Passerini. 2023. Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities. *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=pHCdMat0gI

[39] Jianglin Lu, Yi Xu, Huan Wang, Yue Bai, and Yun Fu. 2023. Latent Graph Inference with Limited Supervision. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=tGuMwFnRZX

[40] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. 2019. A flexible generative framework for graph-based semi-supervised learning. *Advances in Neural Information Processing Systems* 32 (2019).

[41] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. 2021. GraphiT: Encoding Graph Structure in Transformers. *CoRR* abs/2106.05667 (2021). arXiv:2106.05667 https://arxiv.org/abs/2106.05667

[42] Michael Molloy and Bruce Reed. 1995. A critical point for random graphs with a given degree sequence. *Random Struct. Alg.* 6, 2-3 (1 March 1995), 161–180. doi:10.1002/rsa.3240060204

[43] Federico Monti, Karl Otness, and Michael M Bronstein. 2018. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE data science workshop (DSW)*. IEEE, 225–228.

[44] Soumyasundar Pal, Saber Malekmohammadi, Florence Regol, Yingxue Zhang, Yishi Xu, and Mark Coates. 2020. Non parametric graph learning for bayesian graph neural networks. In *Conference on uncertainty in artificial intelligence*. PMLR, 1318–1327.

[45] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1e2agrFvS

[46] Trang Pham, Truyen Tran, Hoa Dam, and Svetha Venkatesh. 2017. Graph classification via deep learning with virtual nodes. *arXiv preprint arXiv:1708.04357* (2017).

[47] Lisi Qarkaxhija, Vincenzo Perri, and Ingo Scholtes. 2022. De Bruijn Goes Neural: Causality-Aware Graph Neural Networks for Time Series Data on Dynamic

Graphs. In *Proceedings of the First Learning on Graphs Conference (Proceedings of Machine Learning Research, Vol. 198)*. PMLR, 51:1–51:21. https://proceedings.mlr.press/v198/qarkaxhija22a.html

[48] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hkx1qkrKPr

[49] 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).

[50] Mandana Saebi, Giovanni Luca Ciampaglia, Lance M. Kaplan, and Nitesh V. Chawla. 2020. HONEM: Learning Embedding for Higher Order Networks. *Big Data* 8, 4 (Aug. 2020), 255–269. doi:10.1089/big.2019.0169

[51] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. 2021. Graph Neural Networks for Friend Ranking in Large-scale Social Platforms. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 2535–2546. doi:10.1145/3442381.3450120

[52] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*. 519–527.

[53] Piotr Sapiezynski, Arkadiusz Stopczynski, David Dreyer Lassen, and Sune Lehmann. 2019. Interaction data from the Copenhagen Networks Study. *Scientific data* 6 (11 Dec. 2019). doi:10.1038/s41597-019-0325-x

[54] Ingo Scholtes. 2017. When is a network a network? Multi-order graphical model selection in pathways and temporal networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1037–1046.

[55] Kartik Sharma, Yeon-Chang Lee, Sivagami Nambi, Aditya Salian, Shlok Shah, Sang-Wook Kim, and Srijan Kumar. 2024. A Survey of Graph Neural Networks for Social Recommender Systems. *ACM Comput. Surv.* 56, 10, Article 265 (June 2024), 34 pages. doi:10.1145/3661821

[56] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. 2021. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522* (2021).

[57] Philippe Vanhems, Alain Barrat, Ciro Cattuto, Jean-François Pinton, Nagham Khanafer, Corinne Régis, Byeul-a Kim, Brigitte Comte, and Nicolas Voirin. 2013. Estimating Potential Infection Transmission Routes in Hospital Wards Using Wearable Proximity Sensors. *PLoS ONE* 8, 9 (Sept. 2013), e73970. doi:10.1371/journal.pone.0073970

[58] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[59] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2020. Graphcrop: Subgraph cropping for graph classification. *arXiv preprint arXiv:2009.10564* (2020).

[60] Jian Xu, Thanuka L. Wickramarathne, and Nitesh V. Chawla. 2016. Representing higher-order dependencies in networks. *Science Advances* 2, 5 (May 2016). doi:10.1126/sciadv.1600028

[61] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in neural information processing systems* 34 (2021), 28877–28888.

[62] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in neural information processing systems* 33 (2020), 5812–5823.

[63] Fu Ji Zhang and Guo Ning Lin. 1987. On the de Bruijn - Good graphs. *Acta Mathematica Sinica* 30, 2 (1987), 195–205.

[64] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. 2019. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5829–5836.

[65] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. 2022. From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Mspk_WYKoEH

[66] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. 2023. Graph Data Augmentation for Graph Machine Learning: A Survey. *IEEE Data Eng. Bull.* 46, 2 (2023), 140–165. http://sites.computer.org/debull/A23june/p140.pdf

[67] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 35. 11015–11023.

[68] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: current limitations and effective designs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 653, 12 pages.

A Properties of used datasets

We use five datasets from the SocioPatterns repository and two synthetic datasets. The datasets are collected from real-world social interactions and are publicly available. We perform the same preprocessing as in [47] to obtain the first-order graph and the second-order De Bruijn graph. The datasets are summarized in Table 2. All preprocessed datasets are included with the code.

B Details for the experimental evaluation

We compare our architecture with graph representation learning methods (**EVO** [2], **HONEM** [50], and **Node2Vec** [21]) and deep graph learning methods (**GCN** [32], **GAT** [58], **LGNN** [8] and **DBGNN** [47]). Finally, we also consider the state of the art dynamic node prediction method **TGN** [49]. This method was developed for predicting changes of nodes labels over time, and not for the prediction of static node labels that depend on the sequences of interactions. Therefore, we adapt the original training procedure to fit the static task as outlined in Appendix B.2.

B.1 Experiment resources and reproducibility

We performed the experiments on NVIDIA L40 GPUs with 48 GB memory. The training time is listed in the evaluation tables.

To reproduce the experiments, we provide a reference implementation at https://github.com/jvpichowski-research/2025-SIT-GNN together with synthetic and empirical datasets and their splits and licenses. For the implementations of the baselines we attribute the reused implementations from the DBGNN reference paper [47]. They also parsed and provide the used empirical datasets.

B.2 TGN adaptations

We implement TGN as proposed in [49]. Instead of a link prediction layer, we add a node prediction layer as the last stage. The embedding size is fixed to 32, the maximum for the other models. For TGN the training procedure is adapted due to its dynamic origin. The proposed training procedure for dynamic node predictions splits the time-stamped edges into fixed-size temporal batches and predicts the next node state for the nodes affected by the time-stamped edges. The batches are temporally divided into train and test batches. Opposing, the static prediction task splits the nodes into train and test sets. We try to keep as much from the original training procedure as possible to favor the memory based architecture. Hence, we train the model on all event batches of size 200 but restrict the training nodes to the train set with fixed class. The last prediction for the given test nodes is used to evaluate the performance. This is not necessary in the last batch of time-stamped edges. Averaging the predictions over all event batches to obtain an increase in performance turned out to be non trivial. This again shows that dynamic models are not easily transferred to the static case and leads to new open questions for further work. For the synthetic data the batch size is increased to 200.000 since each of the 2^{23} time-stamped edges has its own timestamps which leads to infeasible training time with lower batch sizes. Compared to the other deep learning methods the model losses are updated more often because they are updated for every event batch and not only for every node batch. Consequently, we adapt the learning rate

Table 2: Overview of time series data and ground truth node classes used in the experiments. δ describes the maximum time difference for edges to be considered part of a casual walk.

Dataset	Ref.	Time-stamped edges	$\|V\|$	$\|E\|$	$\|V^{(2)}\|$	$\|E^{(2)}\|$	Classes (Sizes)	δ
School11	[16]	28561	126	3355	3042	17141	2 (85/41)	4
School12	[16]	45047	180	4399	3965	20614	2 (132/48)	4
Hospital	[57]	32424	75	2052	2028	15500	4 (29/27/11/8)	4
Student	[53]	24333	429	1160	733	846	2 (314/115)	40
Workplace	[19]	9827	92	1491	1431	7121	5 (34/26/15/13/4)	4
Synthetic	Ours	8388608	20	400	400	1600	2 (10/10)	1
Synthetic Randomized	Ours	8388608	20	400	400	1600	2 (10/10)	1

Table 3: Model per epoch training time and performance in terms of balanced accuracy for the temporal static node classification task (temp.) across various empirical datasets. The best-performing models are marked in bold.

Edges		Hosptial		School11		School12		Student		Workplace	
Time-respecting	Significant	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)	BAcc (%)	Time (ms)
✗	✗	42.83 ± 2.51	2.79 ± 0.61	48.89 ± 5.53	3.35 ± 0.29	61.38 ± 4.82	4.09 ± 0.34	56.58 ± 5.18	3.28 ± 0.49	30.67 ± 8.89	3.28 ± 0.16
✓	✗	40.33 ± 5.95	4.09 ± 0.60	39.86 ± 8.21	4.72 ± 0.72	59.15 ± 9.10	5.03 ± 0.43	51.29 ± 6.17	4.71 ± 0.72	69.22 ± 3.79	4.63 ± 0.09
✓	✓	**58.83 ± 6.27**	6.87 ± 0.18	**58.75 ± 5.57**	6.15 ± 0.25	**63.77 ± 9.47**	6.95 ± 0.41	**62.60 ± 4.26**	5.76 ± 1.43	**78.06 ± 8.27**	5.50 ± 0.20

Table 4: Experiment results (BAcc) with 10-fold cross-validation evaluation procedure.

Model	School11	School12	Hospital	Student	Workplace
Node2Vec	54.64 ± 17.70	49.65 ± 12.97	24.58 ± 10.92	52.31 ± 7.70	20.54 ± 9.51
GCN	55.00 ± 13.37	59.35 ± 11.13	43.47 ± 9.03	54.50 ± 6.40	73.33 ± 12.60
GAT	58.80 ± 13.44	52.92 ± 16.66	27.50 ± 4.02	**64.65 ± 6.25**	54.08 ± 16.70
TGN	61.52 ± 11.25	41.52 ± 6.19	50.27 ± 14.83	50.67 ± 4.10	80.16 ± 18.71
EVO	43.68 ± 10.91	50.05 ± 7.30	25.83 ± 8.29	55.05 ± 6.39	26.50 ± 12.08
HONEM	59.00 ± 10.61	50.49 ± 9.31	39.44 ± 17.57	53.81 ± 7.28	83.17 ± 11.14
LGNN	57.72 ± 9.85	51.43 ± 17.94	44.03 ± 9.03	52.71 ± 6.63	84.83 ± 14.77
DBGNN	61.54 ± 11.13	64.93 ± 15.26	52.50 ± 19.27	57.72 ± 5.29	84.42 ± 15.59
SIT-GNN	**63.25 ± 16.18**	**66.41 ± 10.24**	**76.39 ± 17.12**	60.66 ± 6.11	**88.29 ± 10.51**

to 0.0001 and the optimizer to the originally used one (Adam) to obtain improved results.

B.3 Experiments with cross-validation evaluation procedure

Typically, for model evaluation, the dataset is split once, and the models are repeatedly trained with different random initializations on this fixed split. For instance, the OGB benchmark [25] provides such a predefined split. We follow this widely used evaluation procedure in the main body of the text.

However, we argue that the less commonly used 10-fold cross-validation (CV) — which divides the dataset into 10 folds and uses a different fold as the test set in each iteration — offers a more robust evaluation method as it avoids the potential bias of a "lucky" initial split. The primary drawback of this approach is a higher standard deviation due to the varying test sets. In Table 4, we present the same experiments using 10-fold CV. The results confirm the findings from the main body of the text using the fixed splits.

C Ablation study about the impact of time-respecting and significant edges

We conduct an ablation study to analyze the impact of time-respecting and significant edges on the performance of our model. We consider three different settings:

- **Only static edges**: We neither use time-respecting edges nor apply a significance test.
- **Time-respecting edges**: We use all time-respecting edges but do not apply a significance test.
- **Time-respecting and significant edges**: We use the time-respecting edges that are significant according to the hypergeometric test.

The results are shown in Table 3. The results show that the model performs best when using both time-respecting and significant edges. When only using static or time-respecting edges, the performance drops, indicating that the model relies on the significant edges to learn the underlying patterns in the data. For some datasets the performance drops when introducing time-respecting edges compared to only using static edges. We assume that too many insignificant time-respecting edges are introduced that dilute the information or reduce homophily. The results confirm our hypothesis that the model relies on significant edges to learn the underlying patterns in the data.