Provably expressive temporal graph networks

Amauri H. Souza[1], **Diego Mesquita**[2], **Samuel Kaski**[1,3], **Vikas Garg**[1,4]
[1]Aalto University [2]Getulio Vargas Foundation [3]University of Manchester [4]YaiYai Ltd
{amauri.souza, samuel.kaski}@aalto.fi, diego.mesquita@fgv.br, vgarg@csail.mit.edu

Abstract

Temporal graph networks (TGNs) have gained prominence as models for embedding dynamic interactions, but little is known about their theoretical underpinnings. We establish fundamental results about the representational power and limits of the two main categories of TGNs: those that aggregate temporal walks (WA-TGNs), and those that augment local message passing with recurrent memory modules (MP-TGNs). Specifically, novel constructions reveal the inadequacy of MP-TGNs and WA-TGNs, proving that neither category subsumes the other. We extend the 1-WL (Weisfeiler-Leman) test to temporal graphs, and show that the most powerful MP-TGNs should use injective updates, as in this case they become as expressive as the temporal WL. Also, we show that sufficiently deep MP-TGNs cannot benefit from memory, and MP/WA-TGNs fail to compute graph properties such as girth.

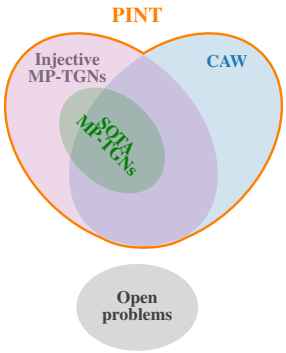
These theoretical insights lead us to PINT — a novel architecture that leverages injective temporal message passing and relative positional features. Importantly, PINT is provably more expressive than both MP-TGNs and WA-TGNs. PINT significantly outperforms existing TGNs on several real-world benchmarks.

1 Introduction

Graph neural networks (GNNs) [11, 30, 36, 39] have recently led to breakthroughs in many applications [7, 28, 31] by resorting to message passing between neighboring nodes in input graphs. While message passing imposes an important inductive bias, it does not account for the dynamic nature of interactions in time-evolving graphs arising from many real-world domains such as social networks and bioinformatics [16, 40]. In several scenarios, these temporal graphs are only given as a sequence of timestamped events. Recently, temporal graph nets (TGNs) [16, 27, 32, 38, 42] have emerged as a prominent learning framework for temporal graphs and have become particularly popular due to their outstanding predictive performance. Aiming at capturing meaningful structural and temporal patterns, TGNs combine a variety of building blocks, such as self-attention [33, 34], time encoders [15, 41], recurrent models [5, 13], and message passing [10].

Unraveling the learning capabilities of (temporal) graph networks is imperative to understanding their strengths and pitfalls, and designing better, more nuanced models that are both theoretically well-grounded and practically efficacious. For instance, the enhanced expressivity of higher-order GNNs has roots in the inadequacy of standard message-passing GNNs to separate graphs that are indistinguishable by the Weisfeiler-Leman isomorphism test, known as 1-WL test or color refinement algorithm [21, 22, 29, 37, 43]. Similarly, many other notable advances on GNNs were made possible by untangling their ability to generalize [9, 17, 35], extrapolate [45], compute graph properties [4, 6, 9], and express Boolean classifiers [1]; by uncovering their connections to distributed algorithms [19, 29], graph kernels [8], dynamic programming [44], diffusion processes [3], graphical models [46], and combinatorial optimization [2]; and by analyzing their discriminative power [20, 23]. In stark contrast, the theoretical foundations of TGNs remain largely unexplored. For instance, unresolved questions include: How does the expressive power of existing TGNs compare? When do TGNs fail? Can we improve the expressiveness of TGNs? What are the limits on the power of TGNs?

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



Overview of the theoretical results	
Relationship between CTDGs and DTDGs	Prop. 1
Injective MP-TGNs $\succeq$ MP-TGNs	Prop. 2
Sufficiently deep MP-TGNs do not need memory	Prop. 3
SOTA MP-TGNs $\prec$ Injective MP-TGNs	Prop. 4
MP-TGNs $\not\succ$ WA-TGNs and WA-TGNs $\not\succ$ MP-TGNs	Prop. 5
Injective MP-TGNs $\cong$ temporal-WL test	Prop. 6
MP-TGNs/CAWs cannot recognize graph properties	Prop. 7
Constructing injective temporal MP	Prop. 8
PINT (ours) $\succ$ both MP-TGNs and WA-TGNs	Prop. 9
Limitations of PINT	Prop. 10

Figure 1: Schematic diagram and summary of our contributions.

We establish a series of results to address these fundamental questions. We begin by showing that discrete-time dynamic graphs (DTDGs) can always be converted to continuous-time analogues (CTDGs) without loss of information, so we can focus on analyzing the ability of TGNs to distinguish nodes/links of CTDGs. We consider a general framework for message-passing TGNs (MP-TGNs) [27] that subsumes a wide variety of methods [e.g., 16, 32, 42]. We prove that equipping MP-TGNs with injective aggregation and update functions leads to the class of most expressive *anonymous MP-TGNs* (i.e., those that do not leverage node ids). Extending the color-refinement algorithm to temporal settings, we show that these most powerful MP-TGNs are as expressive as the temporal WL method. Notably, existing MP-TGNs do not enforce injectivity. We also delineate the role of memory in MP-TGNs: nodes in a network with only a few layers of message passing fail to aggregate information from a sufficiently wide receptive field (i.e., from distant nodes), so memory serves to offset this highly local view with additional global information. In contrast, sufficiently deep architectures obviate the need for memory modules.

Different from MP-TGNs, walk-aggregating TGNs (WA-TGNs) such as CAW [38] obtain representations from anonymized temporal walks. We provide constructions that expose shortcomings of each framework, establishing that WA-TGNs can distinguish links in cases where MP-TGNs fail and vice-versa. Consequently, neither class is more expressive than the other. Additionally, we show that MP-TGNs and CAWs cannot decide temporal graph properties such as diameter, girth, or number of cycles. Strikingly, our analysis unravels the subtle relationship between the walk computations in CAWs and the MP steps in MP-TGNs.

Equipped with these theoretical insights, we propose PINT (short for *position-encoding injective temporal graph net*), founded on a new temporal layer that leverages the strengths of both MP-TGNs and WA-TGNs. Like the most expressive MP-TGNs, PINT defines injective message passing and update steps. PINT also augments memory states with novel relative positional features, and these features can replicate all the discriminative benefits available to WA-TGNs. Interestingly, the time complexity of computing our positional features is less severe than the sampling overhead in CAW, thus PINT can often be trained faster than CAW. Importantly, we establish that PINT is provably more expressive than CAW as well as MP-TGNs.

Our contributions are three-fold:

- a rigorous theoretical foundation for TGNs is laid - elucidating the role of memory, benefits of injective message passing, limits of existing TGN models, temporal extension of the 1-WL test and its implications, impossibility results about temporal graph properties, and the relationship between main classes of TGNs — as summarized in Figure 1;
- explicit injective temporal functions are introduced, and a novel method for temporal graphs is proposed that is provably more expressive than state-of-the-art TGNs;
- extensive empirical investigations underscore practical benefits of this work. The proposed method is either competitive or significantly better than existing models on several real benchmarks for dynamic link prediction, in transductive as well as inductive settings.

2 Preliminaries

We denote a *static graph* G as a tuple $(V, E, \mathcal{X}, \mathcal{E})$, where $V = \{1, 2, \dots, n\}$ denotes the set of nodes and $E \subseteq V \times V$ the set of edges. Each node $u \in V$ has a feature vector $x_u \in \mathcal{X}$ and each edge $(u, v) \in E$ has a feature vector $e_{uv} \in \mathcal{E}$, where $\mathcal{X}$ and $\mathcal{E}$ are countable sets of features.

Dynamic graphs can be roughly split according to their discrete- or continuous-time nature [14]. A *discrete-time dynamic graph* (DTDG) is of a sequence of graph snapshots $(G_1, G_2, \dots)$ usually sampled at regular intervals, each snapshot being a static graph $G_t = (V_t, E_t, \mathcal{X}_t, \mathcal{E}_t)$.

A *continuous-time dynamic graph* (CTDG) evolves with node- and edge-level *events*, such as addition and deletion. We represent a CTDG as a sequence of time-stamped multi-graphs $(\mathsf{G}(t_0), \mathsf{G}(t_1), \dots)$ such that $t_k < t_{k+1}$, and $\mathsf{G}(t_{k+1})$ results from updating $\mathsf{G}(t_k)$ with all events at time t_{k+1}. We assume no event occurs between t_k and t_{k+1}. We denote an interaction (i.e., edge addition event) between nodes u and v at time t as a tuple (u, v, t) associated with a feature vector $e_{uv}(t)$. Unless otherwise stated, interactions correspond to undirected edges, i.e., (u, v, t) is a shorthand for $(\{u, v\}, t)$.

Noting that CTDGs allow for finer (irregular) temporal resolution, we now formalize the intuition that DTDGs can be reduced to and thus analyzed as CTDGs, but the converse may need extra assumptions.

Proposition 1 (Relationship between DTDG and CTDG)**.** *For any DTDG we can build a CTDG with the same sets of node and edge features that contains the same information, i.e., we can reconstruct the original DTDG from the converted CTDG. The converse holds if the CTDG timestamps form a subset of a uniformly spaced countable set.*

Following the usual practice [16, 38, 42], we focus on CTDGs with edge addition events (see Appendix E for a discussion on deletion). Thus, we can represent temporal graphs as sets $\mathcal{G}(t) = \{(u_k, v_k, t_k) \mid t_k < t\}$. We also assume each distinct node v in $\mathcal{G}(t)$ has an initial feature vector x_v.

Message-passing temporal graph nets (MP-TGNs). Rossi et al. [27] introduced MP-TGN as a general representation learning framework for temporal graphs. The goal is to encode the graph dynamics into node embeddings, capturing information that is relevant for the task at hand. To achieve this, MP-TGNs rely on three main ingredients: memory, aggregation, and update. Memory comprises a set of vectors that summarizes the history of each node, and is updated using a recurrent model whenever an event occurs. The aggregation and update components resemble those in message-passing GNNs, where the embedding of each node is refined using messages from its neighbors.

We define the *temporal neighbohood* of node v at time t as $\mathcal{N}(v, t) = \{(u, e_{uv}(t'), t') \mid \exists (u, v, t') \in \mathcal{G}(t)\}$, i.e., the set of neighbor/feature/timestamp triplets from all interactions of node v prior to t. MP-TGNs compute the temporal representation $h_v^{(\ell)}(t)$ of v at layer ℓ by recursively applying

$$\tilde{h}_v^{(\ell)}(t) = \textsc{Agg}^{(\ell)}(\{\!\{(h_u^{(\ell-1)}(t), t - t', e) \mid (u, e, t') \in \mathcal{N}(v, t)\}\!\}) \tag{1}$$

$$h_v^{(\ell)}(t) = \textsc{Update}^{(\ell)}\left(h_v^{(\ell-1)}(t), \tilde{h}_v^{(\ell)}(t)\right), \tag{2}$$

where $\{\!\{\cdot\}\!\}$ denotes multisets, $h_v^{(0)}(t) = s_v(t)$ is the *state* of v at time t, and $\textsc{Agg}^{(\ell)}$ and $\textsc{Update}^{(\ell)}$ are arbitrary parameterized functions. The memory block updates the states as events occur. Let $\mathcal{J}(v, t)$ be the set of events involving v at time t. The state of v is updated due to $\mathcal{J}(v, t)$ as

$$m_v(t) = \textsc{MemAgg}(\{\!\{[s_v(t), s_u(t), t - t_v, e_{vu}(t)] \mid (v, u, t) \in \mathcal{J}(v, t)\}\!\}) \tag{3}$$

$$s_v(t^+) = \textsc{MemUpdate}(s_v(t), m_v(t)), \tag{4}$$

where $s_v(0) = x_v$ (initial node features), $s_v(t^+)$ denotes the updated state of v due to events at time t, and t_v denotes the time of the last update to v. $\textsc{MemAgg}$ combines information from simultaneous events involving node v and $\textsc{MemUpdate}$ usually implements a gated recurrent unit (GRU) [5]. Notably, some MP-TGNs do not use memory, or equivalently, they employ *identity memory*, i.e., $s_v(t) = x_v$ for all t. We refer to Appendix A for further details.

Causal Anonymous Walks (CAWs). Wang et al. [38] proposed CAW as an approach for link prediction on temporal graphs. To predict if an event (u, v, t) occurs, CAW first obtains sets S_u and S_v of temporal walks starting at nodes u and v at time t. An $(L-1)$-length *temporal walk* is represented as $W = ((w_1, t_1), (w_2, t_2), \dots, (w_L, t_L))$, with $t_1 > t_2 > \dots > t_L$ and $(w_{i-1}, w_i, t_i) \in \mathcal{G}(t)$ $\forall i > 1$. Note that when predicting (u, v, t), we have walks starting at time $t_1 = t$. Then, CAW anonymizes walks replacing each node w with a set $I_{\text{CAW}}(w; S_u, S_v) = \{g(w; S_u), g(w; S_v)\}$ of two

feature vectors. The ℓ-th entry of $g(w; S_u)$ stores how many times w appears at the ℓ-th position in a walk of S_u, i.e. $g(w, S_u)[\ell] = |\{W \in S_u : (w, t_\ell) = W_\ell\}|$ where W_ℓ is ℓ-th pair of W.

To encode a walk W with respect to the sets S_u and S_v, CAW applies $\textsc{Enc}(W; S_u, S_v) = \mathrm{RNN}([f_1(I_{\text{CAW}}(w_i; S_u, S_v)) \| f_2(t_{i-1} - t_i)]_{i=1}^{L})$ where f_1 is a permutation-invariant function, f_2 is a time encoder, and $t_0 = t_1 = t$. Finally, CAW combines the embeddings of each walk in $S_u \cup S_v$ using mean-pooling or self-attention to obtain the representation for the event (u, v, t).

In practice, TGNs often rely on sampling schemes for computational reasons. However we are concerned with the expressiveness of TGNs, so our analysis assumes complete structural information, i.e., S_u is the set of all temporal walks from u and MP-TGNs combine information from all neighbors.

3 The representational power and limits of TGNs

We now study the expressiveness of TGNs on node/edge-level prediction. We also establish connections to a variant of the WL test and show limits of specific TGN models. Proofs are in Appendix B.

3.1 Distinguishing nodes with MP-TGNs

We analyze MP-TGNs w.r.t. their ability to map different nodes to different locations in the embedding space. In particular, we say that an L-layer MP-TGN distinguishes two nodes u, v of a temporal graph at time t, if the last layer embeddings of u and v are different, i.e., $h_u^{(L)}(t) \neq h_v^{(L)}(t)$.

We can describe the MP computations of a node v at time t via its *temporal computation tree* (TCT) $T_v(t)$. $T_v(t)$ has v as its root and height equal to the number of MP-TGN layers L. We will keep the dependence on depth L implicit for notational simplicity. For each element $(u, e, t') \in \mathcal{N}(v, t)$ associated with v, we have a node, say i, in the next layer of the TCT linked to the root by an edge annotated with (e, t'). The remaining TCT layers are built recursively using the same mechanism. We denote by $\sharp_v^t$ the (possibly many-to-one) operator that maps nodes in $T_v(t)$ back to nodes in $\mathcal{G}(t)$, e.g., $\sharp_v^t i = u$. Each node i in $T_v(t)$ has a state vector $s_i = s_{\sharp_v^t i}(t)$. To get the embedding of the root v, information is propagated bottom-up, i.e., starting from the leaves all the way up to the root — each node aggregates the message from the layer below and updates its representation along the way. Whenever clear from context, we denote $\sharp_v^t$ simply as $\sharp$ for a cleaner notation.

We study the expressive power of MP-TGNs through the lens of functions on multisets adapted to temporal settings, i.e., comprising triplets of node states, edge features, and timestamps. Intuitively, injective functions 'preserve' the information as it is propagated, so should be essential for maximally expressive MP-TGNs. We formalize this idea in Lemma 1 and Proposition 2 via Definition 1.

Definition 1 (Isomorphic TCTs)**.** *Two TCTs $T_z(t)$ and $T_{z'}(t)$ at time t are isomorphic if there is a bijection $f : V(T_z(t)) \to V(T_{z'}(t))$ between the nodes of the trees such that the following holds:*

$$(u, v, t') \in E(T_z(t)) \iff (f(u), f(v), t') \in E(T_{z'}(t))$$

$$\forall (u, v, t') \in E(T_z(t)) : e_{uv}(t') = e_{f(u)f(v)}(t') \text{ and } \forall u \in V(T_z(t)) : s_u = s_{f(u)} \text{ and } k_u = k_{f(u)}$$

Here, k_u denotes the level (depth) of node u in the tree. The root node has level 0, and for a node u with level k_u, the children of u have level $k_u + 1$.

Lemma 1. *If an MP-TGN Q with L layers distinguishes two nodes u, v of a dynamic graph $\mathcal{G}(t)$, then the L-depth TCTs $T_u(t)$ and $T_v(t)$ are not isomorphic.*

For non-isomorphic TCTs, Proposition 2 shows that improving MP-TGNs with *injective* message passing layers suffices to achieve node distinguishability, extending results from static GNNs [43].

Proposition 2 (Most expressive MP-TGNs)**.** *If the L-depth TCTs of two nodes u, v of a temporal graph $\mathcal{G}(t)$ at time t are not isomorphic, then an MP-TGN Q with L layers and injective aggregation and update functions at each layer is able to distinguish nodes u and v.*

So far, we have considered TCTs with general memory modules, i.e., nodes are annotated with memory states. However, an important question remains: *How does the expressive power of MP-TGNs change as a function of the memory*? Our next result - Proposition 3 - shows that adding GRU-based memory does not increase the expressiveness of suitably deep MP-TGNs.

Proposition 3 (The role of memory)**.** *Let $\mathcal{Q}_L^{[M]}$ denote the class of MP-TGNs with recurrent memory and L layers. Similarly, we denote by $\mathcal{Q}_L$ the family of memoryless MP-TGNs with L layers. Let Δ be the temporal diameter of $\mathcal{G}(t)$ (see Definition B2). Then, it holds that:*

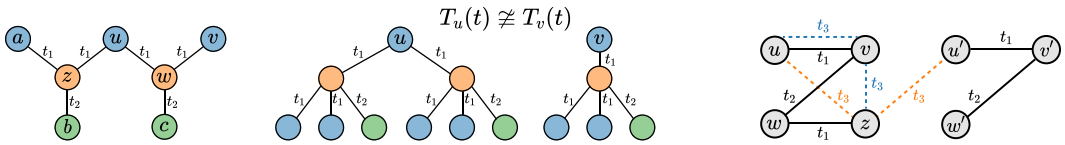


Figure 2: **Limitations of TGNs**. [*Left*] Temporal graph with nodes u, v that TGN-Att/TGAT cannot distinguish. Colors are node features, edge features are identical, and $t_3 > t_2 > t_1$. [*Center*] TCTs of u and v are non-isomorphic. However, the attention layers of TGAT/TGN-Att compute weighted averages over a same multiset of values, returning identical messages for u and v. [*Right*] MP-TGNs fail to distinguish the events (u, v, t_3) and (v, z, t_3) as TCTs of z and u are isomorphic. Meanwhile, CAW cannot separate (u, z, t_3) and (u', z, t_3): the 3-depth TCTs of u and u' are not isomorphic, but the temporal walks from u and u' have length 1, keeping CAW from capturing structural differences.

1. *If $L < \Delta$: $\mathcal{Q}_L^{[M]}$ is strictly more powerful than $\mathcal{Q}_L$ in distinguishing nodes of $\mathcal{G}(t)$;*
2. *For any L : $\mathcal{Q}_{L+\Delta}$ is at least as powerful as $\mathcal{Q}_L^{[M]}$ in distinguishing nodes of $\mathcal{G}(t)$.*

The MP-TGN framework is rather general and subsumes many modern methods for temporal graphs [e.g., 16, 32, 42]. We now analyze the theoretical limitations of two concrete instances of MP-TGNs: TGAT [42] and TGN-Att [27]. Remarkably, these models are among the best-performing MP-TGNs. Nonetheless, we can show that there are nodes of very simple temporal graphs that TGAT and TGN-Att cannot distinguish (see Figure 2). We formalize this in Proposition 4 by establishing that there are cases in which TGNs with injective layers can succeed, but TGAT and TGN-Att cannot.

Proposition 4 (Limitations of TGAT/TGN-Att)**.** *There exist temporal graphs containing nodes u, v that have non-isomorphic TCTs, yet no TGAT nor TGN-Att with mean memory aggregator (i.e., using* MEAN *as* MEMAGG*) can distinguish u and v.*

This limitation stems from the fact that the attention mechanism employed by TGAT and TGN-Att is proportion invariant [26]. The memory module of TGN-Att cannot counteract this limitation due to its mean-based aggregation scheme. We provide more details in Appendix B.6.

3.2 Predicting temporal links

Models for dynamic graphs are usually trained and evaluated on temporal link prediction [18], which consists in predicting whether an event would occur at a given time. To predict an event between nodes u and v at t, MP-TGNs combine the node embeddings $h_u^{(L)}(t)$ and $h_v^{(L)}(t)$, and push the resulting vector through an MLP. On the other hand, CAW is originally designed for link prediction tasks and directly computes edge embeddings, bypasssing the computation of node representations.

We can extend the notion of node distinguishability to edges/events. We say that a model distinguishes two synchronous events $\gamma = (u, v, t)$ and $\gamma' = (u', v', t)$ of a temporal graph if it assigns different edge embeddings $h_\gamma \neq h_{\gamma'}$ for γ and γ'. Proposition 5 asserts that CAWs are not strictly more expressive than MP-TGNs, and vice-versa. Intuitively, CAW's advantage over MP-TGNs lies in its ability to exploit node identities and capture correlation between walks. However, CAW imposes temporal constraints on random walks, i.e., walks have timestamps in decreasing order, which can limit its ability to distinguish events. Figure 2(Right) sketches constructions for Proposition 5.

Proposition 5 (Limitations of MP-TGNs and CAW)**.** *There exist distinct synchronous events of a temporal graph that CAW can distinguish but MP-TGNs with injective layers cannot, and vice-versa.*

3.3 Connections with the WL test

The Weisfeiler-Leman test (1-WL) has been used as a key tool to analyze the expressive power of GNNs. We now study the power of MP-TGNs under a temporally-extended version of 1-WL, and prove negative results regarding whether TGNs can recognize properties of temporal graphs.

Temporal WL test. We can extend the WL test for temporal settings in a straightforward manner by exploiting the equivalence between temporal graphs and multi-graphs with timestamped edges [24]. In particular, the temporal variant of 1-WL assigns colors for all nodes in an input dynamic graph $\mathcal{G}(t)$ by applying the following iterative procedure:

Initialization: The colors of all nodes in $\mathcal{G}(t)$ are initialized using the initial node features: $\forall v \in V(\mathcal{G}(t)), c^0(v) = x_v$. If node features are not available, all nodes receive identical colors;

Refinement: At step ℓ, the colors of all nodes are refined using a hash (injective) function: for all $v \in V(\mathcal{G}(t))$, we apply $c^{\ell+1}(v) = \text{HASH}(c^{\ell}(v), \{\!\{(c^{\ell}(u), e_{uv}(t'), t') : (u, v, t') \in \mathcal{G}(t)\}\!\})$;

Termination: The test is carried out for two temporal graphs at time t in parallel and stops when the multisets of corresponding colors diverge, returning non-isomorphic. If the algorithm runs until the number of different colors stops increasing, the test is deemed inconclusive.

We note that the temporal WL test trivially reduces to the standard 1-WL test if all timestamps and edge features are identical. The resemblance between MP-TGNs and GNNs and their corresponding WL tests suggests that the power of MP-TGNs is bounded by the temporal WL test. Proposition 6 conveys that MP-TGNs with injective layers are as powerful as the temporal WL test.

Proposition 6. *Assume finite spaces of initial node features $\mathcal{X}$, edge features $\mathcal{E}$, and timestamps $\mathcal{T}$. Let the number of events of any temporal graph be bounded by a fixed constant. Then, there is an MP-TGN with suitable parameters using injective aggregation/update functions that outputs different representations for two temporal graphs if and only if the temporal-WL test outputs 'non-isomorphic'.*

A natural consequence of the limited power of MP-TGNs is that even the most powerful MP-TGNs fail to distinguish relevant graph properties, and the same applies to CAWs (see Proposition 7).

Proposition 7. *There exist non-isomorphic temporal graphs differing in properties such as diameter, girth, and total number of cycles, which cannot be differentiated by MP-TGNs and CAWs.*

Figure 3 provides a construction for Proposition 7. The temporal graphs $\mathcal{G}(t)$ and $\mathcal{G}'(t)$ differ in diameter (∞ vs. 3), girth (3 vs. 6), and number of cycles (2 vs. 1). By inspecting the TCTs, one can observe that, for any node in $\mathcal{G}(t)$, there is a corresponding one in $\mathcal{G}'(t)$ whose TCTs are isomorphic, e.g., $T_{u_1}(t) \cong T_{u'_1}(t)$ for $t > t_3$. As a result, the multisets of node embeddings for these temporal graphs are identical. We provide more details and a construction - where CAW fails to decide properties - in the Appendix.

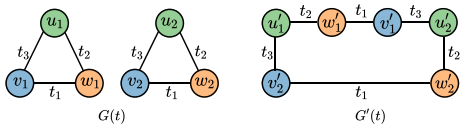


Figure 3: Examples of temporal graphs for which MP-TGNs cannot distinguish the diameter, girth, and number of cycles.

4 Position-encoding injective temporal graph net

We now leverage insights from our analysis in Section 3 to build more powerful TGNs. First, we discuss how to build injective aggregation and update functions in the temporal setting. Second, we propose an efficient scheme to compute positional features based on counts from TCTs. In addition, we show that the proposed method, called ***p**osition-encoding **in**jective **t**emporal graph net* (PINT), is more powerful than both WA-TGNs and MP-TGNs in distinguishing events in temporal graphs.

Injective temporal aggregation. An important design principle in TGNs is to prioritize (give higher importance to) events based on recency [38, 42]. Proposition 8 introduces an injective aggregation scheme that captures this principle using linearly exponential time decay.

Proposition 8 (Injective function on temporal neighborhood)**.** *Let $\mathcal{X}$ and $\mathcal{E}$ be countable, and $\mathcal{T}$ countable and bounded. There exists a function f and scalars α and β such that $\sum_i f(x_i, e_i)\alpha^{-\beta t_i}$ is unique on any multiset $M = \{\!\{(x_i, e_i, t_i)\}\!\} \subseteq \mathcal{X} \times \mathcal{E} \times \mathcal{T}$ with $|M| < N$, where N is a constant.*

Leveraging Proposition 8 and the approximation capabilities of multi-layer perceptrons (MLPs), we propose *position-encoding injective temporal graph net* (PINT). In particular, PINT computes the embedding of node v at time t and layer ℓ using the following message passing steps:

$$\tilde{h}_v^{(\ell)}(t) = \sum_{(u,e,t') \in \mathcal{N}(v,t)} \text{MLP}_{\text{agg}}^{(\ell)} \left(h_u^{(\ell-1)}(t) \parallel e \right) \alpha^{-\beta(t-t')} \tag{5}$$

$$h_v^{(\ell)}(t) = \text{MLP}_{\text{upd}}^{(\ell)} \left(h_v^{(\ell-1)}(t) \parallel \tilde{h}_v^{(\ell)}(t) \right) \tag{6}$$

where $\parallel$ denotes concatenation, $h_v^{(0)} = s_v(t)$, α and β are scalar (hyper-)parameters, and $\text{MLP}_{\text{agg}}^{(\ell)}$ and $\text{MLP}_{\text{upd}}^{(\ell)}$ denote the nonlinear transformations of the aggregation and update steps, respectively.

We note that to guarantee that the MLPs in PINT implement injective aggregation/update, we must further assume that the edge and node features (states) take values in a finite support. In addition, we highlight that there may exist many other ways to achieve injective temporal MP — we have presented a solution that captures the 'recency' inductive bias of real-world temporal networks.

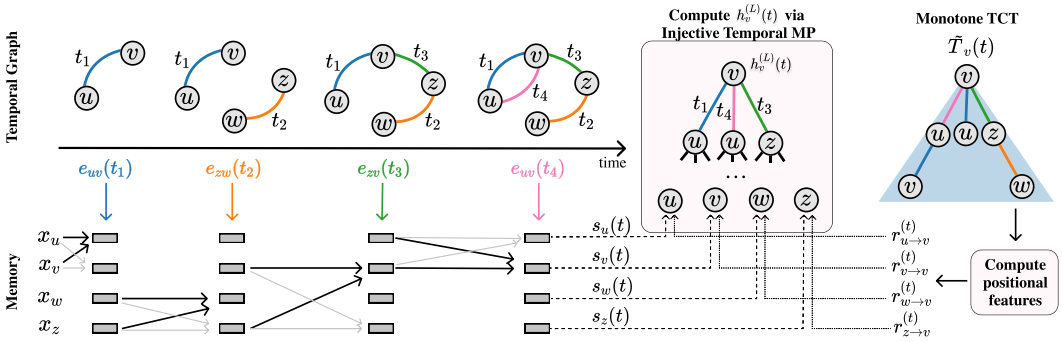


Figure 5: **PINT.** Following the MP-TGN protocol, PINT updates memory states as events unroll. Meanwhile, we use Eqs. (7-11) to update positional features. To extract the embedding for node v, we build its TCT, annotate nodes with memory + positional features, and run (injective) MP.

Relative positional features. To boost the power of PINT, we propose augmenting memory states with *relative* positional features. These features count how many temporal walks of a given length exist between two nodes, or equivalently, how many times nodes appear at different levels of TCTs.

Formally, let P be the $d \times d$ matrix obtained by padding a $(d-1)$-dimensional identity matrix with zeros on its top row and its rightmost column. Also, let $r_{j\to u}^{(t)} \in \mathbb{N}^d$ denote the positional feature vector of node j relative to u's TCT at time t. For each event (u, v, t), with u and v not participating in other events at t, we recursively update the positional feature vectors as

$$\mathcal{V}_i^{(0)} = \{i\} \quad \forall i \tag{7}$$

$$r_{i\to j}^{(0)} = \begin{cases} [1, 0, \dots, 0]^\top & \text{if } i = j \\ [0, 0, \dots, 0]^\top & \text{if } i \neq j \end{cases} \tag{8}$$

$$\mathcal{V}_u^{(t^+)} = \mathcal{V}_v^{(t^+)} = \mathcal{V}_v^{(t)} \cup \mathcal{V}_u^{(t)} \tag{9}$$

$$r_{i\to v}^{(t^+)} = P\, r_{i\to u}^{(t)} + r_{i\to v}^{(t)} \quad \forall i \in \mathcal{V}_u^{(t)} \tag{10}$$

$$r_{j\to u}^{(t^+)} = P\, r_{j\to v}^{(t)} + r_{j\to u}^{(t)} \quad \forall j \in \mathcal{V}_v^{(t)} \tag{11}$$

where we use t^+ to denote values "right after" t. The set $\mathcal{V}_i$ keeps track of the nodes for which we need to update positional features when i participates in an interaction. For simplicity, we have assumed that there are no other events involving u or v at time t. Appendix B.10 provides equations for the general case where nodes can participate in multiple events at the same timestamp.

The value $r_{i\to v}^{(t)}[k]$ (the k-th component of $r_{i\to v}^{(t)}$) corresponds to how many different ways we can get from v to i in k steps through temporal walks. Additionally, we provide in Lemma 2 an interpretation of relative positional features in terms of the so-called monotone TCTs (Definition 2). In this regard, Figure 4 shows how the TCT of v evolves due to an event (u, v, t) and provides an intuition about the updates in Eqs. 10-11. The procedure amounts to appending the monotone TCT of u to the first level of the monotone TCT of v.

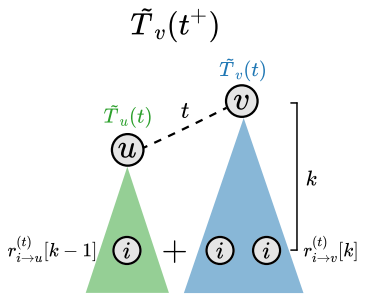


Figure 4: The effect of (u, v, t) on the monotone TCT of v. Also, note how the positional features of a node i, relative to v, can be incrementally updated.

Definition 2. *The monotone TCT of a node u at time t, denoted by $\tilde{T}_u(t)$, is the maximal subtree of the TCT of u s.t. for any path $p = (u, t_1, u_1, t_2, u_2, \dots)$ from the root u to leaf nodes of $\tilde{T}_u(t)$ time monotonically decreases, i.e., we have that $t_1 > t_2 > \dots$.*

Lemma 2. *For any pair of nodes i, u of a temporal graph $\mathcal{G}(t)$, the k-th component of the positional feature vector $r_{i\to u}^{(t)}$ stores the number of times i appears at the k-th layer of the monotone TCT of u.*

Edge and node embeddings. To obtain the embedding h_γ for an event $\gamma = (u, v, t)$, an L-layer PINT computes embeddings for node u and v using L steps of temporal message passing. However, when computing the embedding $h_u^L(t)$ of u, we concatenate node states $s_j(t)$ with the positional features $r_{j\to u}^{(t)}$ and $r_{j\to v}^{(t)}$ for all node j in the L-hop temporal neighborhood of u. We apply the same procedure to obtain $h_v^L(t)$, and then combine $h_v^L(t)$ and $h_u^L(t)$ using a readout function.

Similarly, to compute representations for node-level prediction, for each node j in the L-hop neighborhood of u, we concatenate node states $s_j(t)$ with features $r^{(t)}_{j \to u}$. Then, we use our injective MP to combine the information stored in u and its neighboring nodes. Figure 5 illustrates the process.

Notably, Proposition 9 states that PINT is strictly more powerful than existing TGNs. In fact, the relative positional features mimic the discriminative power of WA-TGNs, while eliminate their temporal monotonicity constraints. Additionally, PINT can implement injective temporal message passing (either over states or states + positional features), akin to maximally-expressive MP-TGNs.

Proposition 9 (Expressiveness of PINT: link prediction)**.** *PINT (with relative positional features) is strictly more powerful than MP-TGNs and CAWs in distinguishing events in temporal graphs.*

When does PINT fail? Naturally, whenever the TCTs (annotated with positional features) for the endpoints of two edges (u, v, t) and (u', v', t) are pairwise isomorphic, PINT returns the same edge embedding and is not able to differentiate the events. Figure 6 shows an example in which this happens — we assume that all node/edge features are identical. Due to graph symmetries, u and z occur the same number of times in each level of v's monotone TCT. Also, the sets of temporal walks starting at u and z are identical if we swap the labels of these nodes. Importantly, CAWs and MP-TGNs also fail here, as stated in Proposition 9.

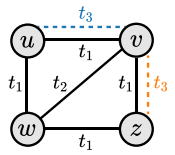


Figure 6: PINT cannot distinguish the events (u, v, t_3) and (v, z, t_3).

Proposition 10 (Limitations of PINT)**.** *There are synchronous events of temporal graphs that PINT cannot distinguish (as seen in Figure 6).*

Implementation and computational cost. The online updates for PINT's positional features have complexity $\mathcal{O}\left(d\,|\mathcal{V}_u^{(t)}| + d\,|\mathcal{V}_v^{(t)}|\right)$. Similarly to CAW's sampling procedure, our online update is a sequential process better done in CPUs. However, while CAW may require significant CPU-GPU memory exchange — proportional to both the number of walks and their depth —, we only communicate the positional features. We can also speed-up the training of PINT by pre-computing the positional features for each batch, avoiding redundant computations at each epoch. Apart from positional features, the computational cost of PINT is similar to that of TGN-Att. Following standard MP-TGN procedure, we control the branching factor of TCTs using neighborhood sampling.

Note that the positional features monotonically increase with time, which is undesirable for practical generalization purposes. Since our theoretical results hold for any fixed t, this issue can be solved by dividing the positional features by a time-dependent normalization factor. Nonetheless, we have found that employing L_1-normalization leads to good empirical results for all evaluated datasets.

5 Experiments

We now assess the performance of PINT on several popular and large-scale benchmarks for TGNs. We run experiments using PyTorch [25] and code is available at `www.github.com/AaltoPML/PINT`.

Tasks and datasets. We evaluate PINT on dynamic link prediction, closely following the evaluation setup employed by Rossi et al. [27] and Xu et al. [42]. We use six popular benchmark datasets: Reddit, Wikipedia, Twitter, UCI, Enron, and LastFM [16, 27, 38, 42]. Notably, UCI, Enron, and LastFM are non-attributed networks, i.e., they do not contain feature vectors associated with the events. Node features are absent in all datasets, thus following previous works we set them to vectors of zeros [27, 42]. Since Twitter is not publicly available, we follow the guidelines by Rossi et al. [27] to create our version. We provide more details regarding datasets in the supplementary material.

Baselines. We compare PINT against five prominent TGNs: Jodie [16], DyRep [32], TGAT [42], TGN-Att [27], and CAW [38]. For completeness, we also report results using two static GNNs: GAT [34] and GraphSage [12]. Since we adopt the same setup as TGN-Att, we use their table numbers for all baselines but CAW on Wikipedia and Reddit. The remaining results were obtained using the implementations and guidelines available from the official repositories. As an ablation study, we also include a version of PINT without relative positional features in the comparison. We provide detailed information about hyperparameters and the training of each model in the supplementary material.

Experimental setup. We follow Xu et al. [42] and use a 70%-15%-15% (train-val-test) temporal split for all datasets. We adopt average precision (AP) as the performance metric. We also analyze separately predictions involving only nodes seen during training (transductive), and those involving

Table 1: **Average Precision** (AP) results for link prediction. We denote the best-performing model (highest mean AP) in **blue**. In 5 out of 6 datasets, PINT achieves the highest AP in the transductive setting. For the inductive case, PINT outperforms previous MP-TGNs and competes with CAW. We also show the performance of PINT with and without relative positional features. For all datasets, adopting positional features leads to significant performance gains.

	Model	Reddit	Wikipedia	Twitter	UCI	Enron	LastFM
Transductive	GAT	97.33 ± 0.2	94.73 ± 0.2	-	-	-	-
	GraphSAGE	97.65 ± 0.2	93.56 ± 0.3	-	-	-	-
	Jodie	97.11 ± 0.3	94.62 ± 0.5	98.23 ± 0.1	86.73 ± 1.0	77.31 ± 4.2	69.32 ± 1.0
	DyRep	97.98 ± 0.1	94.59 ± 0.2	98.48 ± 0.1	54.60 ± 3.1	77.68 ± 1.6	69.24 ± 1.4
	TGAT	98.12 ± 0.2	95.34 ± 0.1	98.70 ± 0.1	77.51 ± 0.7	68.02 ± 0.1	54.77 ± 0.4
	TGN-Att	98.70 ± 0.1	98.46 ± 0.1	98.00 ± 0.1	80.40 ± 1.4	79.91 ± 1.3	80.69 ± 0.2
	CAW	98.39 ± 0.1	98.63 ± 0.1	98.72 ± 0.1	92.16 ± 0.1	$\mathbf{92.09 \pm 0.7}$	81.29 ± 0.1
	PINT (w/o pos. feat.)	$98.62 \pm .04$	$98.43 \pm .04$	98.53 ± 0.1	92.68 ± 0.5	83.06 ± 2.1	81.35 ± 1.6
	PINT	$\mathbf{99.03 \pm .01}$	$\mathbf{98.78 \pm 0.1}$	$\mathbf{99.35 \pm .01}$	$\mathbf{96.01 \pm 0.1}$	88.71 ± 1.3	$\mathbf{88.06 \pm 0.7}$
Inductive	GAT	95.37 ± 0.3	91.27 ± 0.4	-	-	-	-
	GraphSAGE	96.27 ± 0.2	91.09 ± 0.3	-	-	-	-
	Jodie	94.36 ± 1.1	93.11 ± 0.4	96.06 ± 0.1	75.26 ± 1.7	76.48 ± 3.5	80.32 ± 1.4
	DyRep	95.68 ± 0.2	92.05 ± 0.3	96.33 ± 0.2	50.96 ± 1.9	66.97 ± 3.8	82.03 ± 0.6
	TGAT	96.62 ± 0.3	93.99 ± 0.3	96.33 ± 0.1	70.54 ± 0.5	63.70 ± 0.2	56.76 ± 0.9
	TGN-Att	97.55 ± 0.1	97.81 ± 0.1	95.76 ± 0.1	74.70 ± 0.9	78.96 ± 0.5	84.66 ± 0.1
	CAW	97.81 ± 0.1	$\mathbf{98.52 \pm 0.1}$	$\mathbf{98.54 \pm 0.4}$	92.56 ± 0.1	$\mathbf{91.74 \pm 1.7}$	85.67 ± 0.5
	PINT (w/o pos. feat.)	97.22 ± 0.2	97.81 ± 0.1	96.10 ± 0.1	90.25 ± 0.3	75.99 ± 2.3	88.44 ± 1.1
	PINT	$\mathbf{98.25 \pm .04}$	$98.38 \pm .04$	$98.20 \pm .03$	$\mathbf{93.97 \pm 0.1}$	81.05 ± 2.4	$\mathbf{91.76 \pm 0.7}$

novel nodes (inductive). We report mean and standard deviation of the AP over ten runs. For further details, see Appendix D. We provide additional results in the supplementary material.

Results. Table 1 shows that PINT is the best-performing method on five out of six datasets for the transductive setting. Notably, the performance gap between PINT and TGN-Att amounts to over 15% AP on UCI. The gap is also relatively high compared to CAW on LastFM, Enron, and UCI; with CAW being the best model only on Enron. We also observe that many models achieve relatively high AP on the attributed networks (Reddit, Wikipedia, and Twitter). This aligns well with findings from [38], where TGN-Att was shown to have competitive performance against CAW on Wikipedia and Reddit. The performance of GAT and TGAT (static GNNs) on Reddit and Wikipedia reinforces the hypothesis that the edge features add significantly to the discriminative power. On the other hand, PINT and CAW, which leverage relative identities, show superior performance relative to other methods when only time and degree information is available, i.e., on unattributed networks (UCI, Enron, and LastFM). Table 1 also shows the effect of using relative positional features. While including these features boosts PINT's performance systematically, our ablation study shows that PINT w/o positional features still outperforms other MP-TGNs on unattributed networks. In the inductive case, we observe a similar behavior: PINT is consistently the best MP-TGN, and is better than CAW on 3/6 datasets. Overall, PINT (w/ positional features) also yields the lowest standard deviations. This suggests that positional encodings might be a useful inductive bias for TGNs.

Time comparison. Figure 7 compares the training times of PINT against other TGNs. For fairness, we use the same architecture (number of layers & neighbors) for all MP-TGNs: i.e., the best-performing PINT. For CAW, we use the one that yielded results in Table 1. As expected, TGAT is the fastest model. Note that the average time/epoch of PINT gets amortized since positional features are pre-computed. Without these features, PINT's runtime closely matches TGN-Att. When trained for over 25 epochs, PINT runs considerably faster than CAW. We provide additional details and results in the supplementary material.

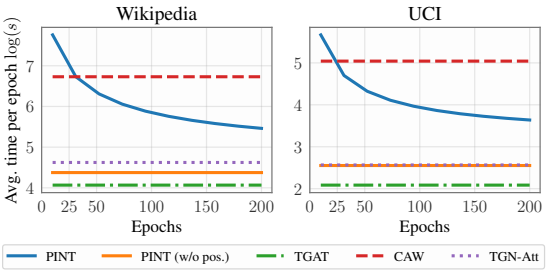


Figure 7: Time comparison: PINT versus TGNs (in log-scale). The cost of pre-computing positional features is quickly diluted as the number of epochs increases.

Table 2: Average precision results for TGN-Att + relative positional features.

	Transductive			Inductive		
	UCI	Enron	LastFM	UCI	Enron	LastFM
TGN-Att	80.40 ± 1.4	79.91 ± 1.3	80.69 ± 0.2	74.70 ± 0.9	78.96 ± 0.5	84.66 ± 0.1
TGN-Att + RPF	95.64 ± 0.1	85.04 ± 2.5	89.41 ± 0.9	92.82 ± 0.4	76.27 ± 3.4	91.63 ± 0.3
PINT	96.01 ± 0.1	88.71 ± 1.3	88.06 ± 0.7	93.97 ± 0.1	81.05 ± 2.4	91.76 ± 0.7

Incorporating relative positional features into MP-TGNs. We can use our relative positional features (RPF) to boost MP-TGNs. Table 2 shows the performance of TGN-Att with relative positional features on UCI, Enron, and LastFM. Notably, TGN-Att receives a significant boost from our RPF. However, PINT still beats TGN-Att+RPF on 5 out of 6 cases. The values for TGN-Att+RPF reflect outcomes from 5 repetitions. We have used the same model selection procedure as TGN-Att in Table 1, and incorporated $d = 4$-dimensional positional features

Dimensionality of relative positional features. We assess the performance of PINT as a function of the dimension d of the relative positional features. Figure 8 shows the performance of PINT for $d \in \{4, 10, 15, 20\}$ on UCI and Enron. We report mean and standard deviation of AP on test set obtained from five independent runs. In all experiments, we re-use the optimal hyperparameters found with $d = 4$. Increasing the dimensionality of the positional features leads to performance gains on both datasets. Notably, we obtain a significant boost for Enron with $d = 10$: 92.69 ± 0.09 AP in the transductive setting and 88.34 ± 0.29 in the inductive case. Thus, PINT becomes the best-performing model on Enron (transductive). On UCI, for $d = 20$, we obtain 96.36 ± 0.07 and 94.77 ± 0.12 (inductive).

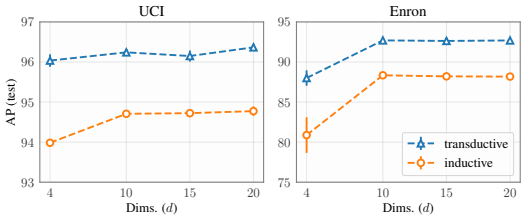


Figure 8: PINT: AP (mean and std) as a function of the dimensionality of the positional features.

6 Conclusion

We laid a rigorous theoretical foundation for TGNs, including the role of memory modules, relationship between classes of TGNs, and failure cases for MP-TGNs. Together, our theoretical results shed light on the representational capabilities of TGNs, and connections with their static counterparts. We also introduced a novel TGN method, provably more expressive than the existing TGNs.

Key practical takeaways from this work: (a) temporal models should be designed to have injective update rules and to exploit both neighborhood and walk aggregation, and (b) deep architectures can likely be made more compute-friendly as the role of memory gets diminished with depth, provably.

Acknowledgments and Disclosure of Funding

This work was supported by the Academy of Finland (Flagship programme: Finnish Center for Artificial Intelligence FCAI and 341763), ELISE Network of Excellence Centres (EU Horizon:2020 grant agreement 951847) and UKRI Turing AI World-Leading Researcher Fellowship, EP/W002973/1. We also acknowledge the computational resources provided by the Aalto Science-IT Project from Computer Science IT. AS and DM also would like to thank Jorge Perez, Jou-Hui Ho, and Hojin Kang for valuable discussions about TGNs, and the latter's input on a preliminary version of this work.

Societal and broader impact

Temporal graph networks have shown remarkable performance in relevant domains such as social networks, e-commerce, and drug discovery. In this paper, we establish fundamental results that delineate the representational power of TGNs. We expect that our findings will help declutter the literature and serve as a seed for future developments. Moreover, our analysis culminates with PINT, a method that is provably more powerful than the prior art and shows superior predictive performance on several benchmarks. We believe that PINT (and its underlying concepts) will help engineers and researchers build better recommendation engines, improving the quality of systems that permeate our lives. Also, we do not foresee any negative societal impact stemming directly from this work.

References

[1] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J.-P. Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[2] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Velickovic. Combinatorial optimization and reasoning with graph neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[3] B. Chamberlain, J. Rowbottom, M. Gorinova, M. M. Bronstein, S. Webb, and E. Rossi. GRAND: graph neural diffusion. In *International Conference on Machine Learning (ICML)*, 2021.

[4] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2020.

[5] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] N. Dehmamy, A.-L. Barabási, and R. Yu. Understanding the representation power of graph neural networks in learning graph topology. In *Advances in neural information processing systems (NeurIPS)*, 2019.

[7] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velickovic. Eta prediction with graph neural networks in google maps. In *Conference on Information and Knowledge Management (CIKM)*, 2021.

[8] S. S. Du, K. Hou, R. Salakhutdinov, B. Póczos, R. Wang, and K. Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[9] V. Garg, S. Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

[11] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2005.

[12] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] S. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[15] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker. Time2vec: Learning a vector representation of time. *ArXiv: 1907.05321*, 2019.

[16] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

[17] R. Liao, R. Urtasun, and R. Zemel. A PAC-bayesian approach to generalization bounds for graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2021.

[18] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[19] A. Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations (ICLR)*, 2020.

[20] A. Loukas. How hard is to distinguish graphs with graph neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[21] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[22] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[23] H. Nguyen and T. Maehara. Graph homomorphism convolution. In *International Conference on Machine Learning (ICML)*, 2020.

[24] F. Orsini, P. Frasconi, and L. D. Raedt. Graph invariant kernels. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NeurIPS - Workshop)*, 2017.

[26] J. Pérez, J. Marinković, and P. Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations (ICLR)*, 2019.

[27] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, E. Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML 2020 Workshop on Graph Representation Learning*, 2020.

[28] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning (ICML)*, 2020.

[29] R. Sato, M. Yamada, and H. Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[30] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[31] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, V. M. Tran, A. Chiappino-Pepe, A. H. Badran, I. W. Andrews, E. J. Chory, G. M. Church, E. D. Brown, T. S. Jaakkola, R. Barzilay, and J. J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688 – 702, 2020.

[32] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. DyRep: Learning representations over dynamic graphs. In *International Conference on Learning Representations (ICLR)*, 2019.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[34] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[35] S. Verma and Z.-L. Zhang. Stability and generalization of graph convolutional neural networks. In *International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

[36] Y. Verma, S. Kaski, M. Heinonen, and V. Garg. Modular flows: Differential molecular generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[37] C. Vignac, A. Loukas, and P. Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[38] Y. Wang, Y. Chang, Y. Liu, J. Leskovec, and P. Li. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations (ICLR)*, 2021.

[39] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020.

[40] D. Xu, W. Cheng, D. Luo, Y. Gu, X. Liu, J. Ni, B. Zong, H. Chen, and X. Zhang. Adaptive neural network for node classification in dynamic networks. In *IEEE International Conference on Data Mining (ICDM)*, 2019.

[41] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Self-attention with functional time representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[42] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

[43] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[44] K. Xu, J. Li, M. Zhang, S. S. Du, K.-I. Kawarabayashi, and S. Jegelka. What can neural networks reason about? In *International Conference on Learning Representations (ICLR)*, 2020.

[45] K. Xu, M. Zhang, J. Li, S. S. Du, K.-I. Kawarabayashi, and S. Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2021.

[46] Z. Zhang, F. Wu, and W. S. Lee. Factor graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Section 4, we describe theoretical limitations of our method – in the paragraph named "When does PINT fail?".
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Proofs are given in Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our code is available in a github repository.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide detailed description of all experiments in the supplementary material, including model selection, and data splits. Please, see Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report mean and standard deviation for all experiments.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We describe the computational resources in the supplementary material (Appendix D).
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We provide references for all codes/datasets we use in the paper.
   (b) Did you mention the license of the assets? [N/A] We use publicly available assets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide code in a github repository.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Provably expressive temporal graph networks (Supplementary material)

A Further details on temporal graph networks

In this section we present more details about the models TGAT, TGN-Att, and CAW.

A.1 Temporal graph attention (TGAT)

Temporal graph attention networks [42] combine time encoders [15] and self-attention [33]. In particular, the time encoder ϕ is given by

$$\phi(t-t') = [\cos(\omega_1(t-t')+b_1), \ldots, \cos(\omega_d(t-t')+b_d)], \tag{S1}$$

where ω_i's and b_i's are learned scalar parameters. The time embeddings are concatenated to the edge features before being fed into a typical self-attention layer, where the query q is a function of a reference node v, and both values V and keys K depend on v's temporal neighbors. Formally, TGAT first computes a matrix $C_v^{(\ell)}(t)$ whose u-th row is $c_{vu}^{(\ell)}(t) = [h_u^{(\ell-1)}(t) \parallel \phi(t-t_{uv}) \parallel e_{uv}]$ for all $(u, e_{uv}, t_{uv}) \in \mathcal{N}(v,t)$. Then, the output $\tilde{h}_v^{(\ell)}(t)$ of the $\textsc{Agg}^{(\ell)}$ function is given by

$$q = [h_v^{(\ell-1)}(t) \parallel \phi(0)]W_q^{(\ell)} \quad K = C_v^{(\ell)}(t)W_K^{(\ell)} \quad V = C_v^{(\ell)}(t)W_V^{(\ell)} \tag{S2}$$

$$\tilde{h}_v^{(\ell)}(t) = \mathrm{softmax}\left(qK^\top\right)V \tag{S3}$$

where $W_q^{(\ell)}, W_K^{(\ell)}$, and $W_V^{(\ell)}$ are model parameters. Regarding the $\textsc{Update}$ function, TGAT applies a multilayer perceptron, i.e., $h_v^{(\ell)}(t) = \mathrm{MLP}^{(\ell)}(h_v^{(\ell-1)}(t) \parallel \tilde{h}_v^{(\ell)}(t))$.

A.2 Temporal graph networks with attention (TGN-Att)

We now discuss details regarding the MP-TGN framework omitted from the main paper for simplicity.

For the sake of generality, Rossi et al. [27] present a formulation for MP-TGNs that can handle node-level events, e.g., node feature updates. These events lead to i) updating node memory states, and ii) using the time-evolving node features as additional inputs for the message-passing functions. Nonetheless, to the best of our knowledge, all relevant CTDG benchmarks comprise only edge events. Therefore, for ease of presentation, we omit node events and temporal node features from our treatment. In Appendix E, we discuss how to handle node-level events.

Note that MP-TGNs update memory states only after an event occurs, otherwise it would incur information leakage. Unless we use the updated states to predict another event later on in the batch, this means that there might be no signal to propagate through memory modules. To get around this problem, Rossi et al. [27] propose updating the memory with messages coming from previous batches, and then predicting the interactions.

To speed up computations, MP-TGNs employ a form of batch learning where events in a same batch are aggregated. In our analysis, we assume that two events belong to the same batch only if they occur at the same timestamp. Importantly, memory aggregators allow removing ambiguity in the way the memory of a node participating in multiple events (at the same timestamp) is updated — without memory aggregator, two events involving a given node i at the same time could lead to different ways of updating the state of i.

Suppose the event $\gamma = (i, u, t)$ occurs. MP-TGNs proceed by computing a memory-message function MEMMSG_e for each endpoint of γ, i.e.,

$$m_{i,u}(t) = \text{MEMMSG}_e(s_i(t), s_u(t), t - t_i, e_{iu}(t))$$
$$m_{u,i}(t) = \text{MEMMSG}_e(s_u(t), s_i(t), t - t_u, e_{iu}(t))$$

Following the original formulation, we assume an identity memory-message function — simply the concatenation of the inputs, i.e., $\text{MEMMSG}_e(s_i(t), s_u(t), t-t_i, e_{iu}(t)) = [s_i(t), s_u(t), t-t_i, e_{iu}(t)]$.

Now, suppose two events (i, u, t) and (i, v, t) happen. MP-TGNs aggregate the memory-messages from these events using a function MEMAGG to obtain a single memory message for i:

$$m_i(t) = \text{MEMAGG}(m_{i,u}(t), m_{i,v}(t))$$

Rossi et al. [27] propose non-learnable memory aggregators, such as the mean aggregator (average all memory messages for a given node), that we denote as MEANAGG and adopt throughout our analysis. As an example, under events (i, u, t) and (i, v, t), the aggregated message for i is $m_i(t) = 0.5([s_i(t), s_u(t), t - t_i, e_{iu}(t)] + [s_i(t), s_v(t), t - t_i, e_{iv}(t)])$.

The memory update of our query node i is given by

$$s_i(t^+) = \text{MEMUPDATE}(s_i(t), m_i(t)).$$

Finally, we note that TGAT does not have a memory module. TGN-Att consists of the model resulting from augmenting TGAT with a GRU-based memory.

A.3 Causal anonymous walks (CAW)

We now provide details regarding how CAW obtains edge embeddings for a query event $\gamma = (u, v, t)$.

A temporal walk is represented as $W = ((w_1, t_1), (w_2, t_2), \ldots, (w_L, t_L))$, with $t_1 > t_2 > \cdots > t_L$ and $(w_{i-1}, w_i, t_i) \in \mathcal{G}(t)$ for all $i > 1$. We denote by $S_u(t)$ the set of maximal temporal walks starting at u of size at most L obtained from the temporal graph at time t. Following the original paper, we drop the time dependence henceforth.

A given walk W gets anonymized through replacing each element w_i belonging to W by a 2-element set of vectors $I_{\text{CAW}}(w_i; S_u, S_v)$ accounting for how many times w_i appears at each position of walks in S_u and S_v. These vectors are denoted by $g(w_i, S_u)$ and $g(w_i, S_v)$. The walk is encoded using a RNN:

$$\text{ENC}(W; S_u, S_v) = \text{RNN}([f_1(I_{\text{CAW}}(w_i; S_u, S_v)) \| f_2(t_i - t_{i-1})]_{i=1}^{L}),$$

where $t_1 = t_0 = t$ and f_1 is

$$f_1(I_{\text{CAW}}(w_i; S_u, S_v)) = \text{MLP}(g(w_i, S_u)) + \text{MLP}(g(w_i, S_v)).$$

We note that the MLPs share parameters. The function f_2 is given by

$$f_2(t) = [\cos(\omega_i t), \sin(\omega_1 t), \ldots, \cos(\omega_d t), \sin(\omega_d t)]$$

where ω_i's are learned parameters.

To compute the embedding h_γ for (u, v, t), CAW considers two readout functions: mean and self-attention. Finally, the final link prediction is obtained from a 2-layer MLP over h_γ.

B Proofs

B.1 Further definitions and Lemmata

Definition B1 (Monotone walk.)**.** *An N-length monotone walk in a temporal graph $\mathcal{G}(t)$ is a sequence $(w_1, t_1, w_2, t_2, \ldots, w_{N+1})$ such that $t_i > t_{i+1}$ and $(w_i, w_{i+1}, t_i) \in \mathcal{G}(t)$ for all i.*

Definition B2 (Temporal diameter.)**.** *We say the temporal diameter of a graph $\mathcal{G}(t)$ is Δ if the longest monotone walk in $\mathcal{G}(t)$ has length (i.e, number of edges) exactly Δ.*

Lemma B1. *If the TCTs of two nodes are isomorphic, then their monotone TCTs (Definition 2) are also isomorphic, i.e., $T_u(t) \cong T_v(t) \Rightarrow \tilde{T}_u(t) \cong \tilde{T}_v(t)$ for two nodes u and v of a dynamic graph.*

Proof. Since $T_u(t) \cong T_v(t)$, we have that

$$p = (u_0, t_1, u_1, t_2, u_2, \ldots) \text{ from } T_u(t) \Longleftrightarrow p' = (f(u_0), t_1, f(u_1), t_2, f(u_2), \ldots) \text{ from } T_v(t),$$
$$\text{with } s_{u_i} = s_{f(u_i)} \text{ and } e_{u_i u_{i+1}}(t_{i+1}) = e_{f(u_i) f(u_{i+1})}(t_{i+1}) \text{ and } k_{u_i} = k_{f(u_i)} \quad \forall i$$

where $f : V(T_u(t)) \to V(T_v(t))$ is a bijection.

Assume that $\tilde{T}_u(t) \ncong \tilde{T}_v(t)$. Then, either there exists a path $p_s = (u'_0, t'_1, u'_1, t'_2, \ldots)$ in $\tilde{T}_u(t)$, such that $t'_{k+1} < t'_k$ for all k (i.e., a monotone walk), with no corresponding one in $\tilde{T}_v(t)$ or vice-versa. Without loss of generality, let us consider the former case.

We can construct the path p'_s in $T_v(t)$ by applying f in all elements of p_s, i.e., $p'_s = (f(u'_0), t'_1, f(u'_1), t'_2, \ldots)$. Note that p'_s is a monotone walk in $T_v(t)$. Since $\tilde{T}_v(t)$ is the maximal monotone subtree of $T_v(t)$, it must contain p'_s, leading to contradiction. □

Lemma B2. *Let $\mathcal{G}(t)$ and $\mathcal{G}'(t)$ be any two non-isomorphic temporal graphs. If an MP-TGN obtains different multisets of node embeddings for $\mathcal{G}(t)$ and $\mathcal{G}'(t)$. Then, the temporal WL test decides $\mathcal{G}(t)$ and $\mathcal{G}'(t)$ are not isomorphic.*

Proof. Recall Proposition 3 shows that if an MP-TGN with memory is able to distinguish two nodes, then there is a memoryless MP-TGN with Δ (temporal diameter) additional layers that does the same. Thus, it suffices to show that if the multisets of colors from temporal WL for $\mathcal{G}(t)$ and $\mathcal{G}'$(t) after ℓ iterations are identical, then the multisets of embeddings from the memoryless MP-TGN are also identical, i.e., if $\{\!\{c^\ell(u)\}\!\}_{u \in V(\mathcal{G}(t))} = \{\!\{c^\ell(u')\}\!\}_{u' \in V(\mathcal{G}'(t))}$, then $\{\!\{h_u^{(\ell)}(t)\}\!\}_{u \in V(\mathcal{G}(t))} = \{\!\{h_{u'}^{(\ell)}(t)\}\!\}_{u' \in V(\mathcal{G}'(t))}$. To do so, we repurpose the proof of Lemma 2 in [43].

More broadly, we show that for any two nodes of a temporal graph $\mathcal{G}(t)$, if the temporal WL returns $c^\ell(u) = c^\ell(v)$, we have that corresponding embeddings from MP-TGN without memory are identical $h_u^\ell(t) = h_v^\ell(t)$. We proceed with a proof by induction.

[*Base case*] For $\ell = 0$, the proposition trivially holds as the temporal WL has the initial node features as colors, and memoryless MP-TGNs have these features as embeddings.

[*Induction step*] Assume the proposition holds for iteration ℓ. Thus, for any two nodes u, v, if $c^{\ell+1}(u) = c^{\ell+1}(v)$, we have

$$(c^\ell(u), \{\!\{(c^\ell(i), e_{iu}(t'), t') : (u, i, t') \in \mathcal{G}(t)\}\!\}) = (c^\ell(v), \{\!\{(c^\ell(j), e_{jv}(t'), t') : (v, j, t') \in \mathcal{G}(t)\}\!\})$$

and, by the induction hypothesis, we know

$$(h_u^{(\ell)}(t), \{\!\{(h_i^{(\ell)}(t), e_{iu}(t'), t') : (u, i, t') \in \mathcal{G}(t)\}\!\}) = (h_v^{(\ell)}(t), \{\!\{(h_j^{(\ell)}(t), e_{jv}(t'), t') : (v, j, t') \in \mathcal{G}(t)\}\!\})$$

We also note that this last identity also implies

$$(h_u^{(\ell)}(t), \{\!\{(h_i^{(\ell)}(t), t - t', e) \mid (i, e, t') \in \mathcal{N}(u, t)\}\!\}) = (h_v^{(\ell)}(t), \{\!\{(h_j^{(\ell)}(t), t - t', e) \mid (j, e, t') \in \mathcal{N}(v, t)\}\!\})$$

since there exists an event $(u, i, t') \in \mathcal{G}(t)$ with feature $e_{ui}(t') = e$ iff there is an element $(i, e, t') \in \mathcal{N}(u, t)$.

As a result, the inputs of the MP-TGN's aggregation and update functions are identical, which leads to identical outputs $h_u^{(\ell+1)}(t) = h_v^{(\ell+1)}(t)$. Therefore, if the temporal WL test obtains identical multisets of colors for two temporal graphs after ℓ steps, the multisets of embeddings at layer ℓ for these graphs are also identical. □

Lemma B3 (Lemma 5 in [43]). *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \varphi\left(\sum_{x \in X} f(x)\right)$ for some function φ.*

B.2 Proof of Proposition 1: Relationship between DTDGs and CTDGs

Proof. We prove the two statements in Proposition 1 separately. In the following, we treat CTDGs as sets of events up to a given timestamp.

Statement 1: For any DTDG we can build a CTDG that contains the same information.

A DTDG consists of a sequence of graphs with no temporal information. We can model this using the CTDG formalism by setting a fixed time difference δ between consecutive elements $\mathsf{G}(t_i), \mathsf{G}(t_{i+1})$ of the CTDG, i.e., $t_{i+1} - t_i = \delta$ for all $i \geq 0$.

Consider a DTDG given by the sequence $(G_1, G_2, \dots)$. To build the equivalent CTDG, we define $S(G_i)$ as the set of edge events corresponding to G_i, i.e., $S(G_i) = \{(u, v, i\delta) : (u, v) \in E(G_i)\}$. We also make the edge features of these events match those in the DTDG, i.e., $e_{uv}(i\delta) = e_{uv} \in \mathcal{E}_i$. To account for node features, for all $u \in V(G_i)$, we create an event $(u, \diamond, i\delta)$ between u and a dummy node $\diamond$, with feature $e_{u\diamond}(i\delta) = x_u \in \mathcal{X}_i$. Let $C(G_i)$ denote the set comprising these node-level events. Then, we can construct the CTDG $\mathsf{G}(t_i) = \cup_{j=1}^{i} S(G_j) \cup C(G_j)$ for $i = 1, \dots$. Reconstructing the DTDG $(G_1, G_2, \dots)$ is trivial. To build G_i, it suffices to select all events at time $i\delta$ in the CTDG. Events involving $\diamond$ determine node features and the remaining ones constitute edges in the DTDG.

Statement 2: The converse holds if the CTDG timestamps form a subset of some uniformly spaced countable set.

We say that a countable set $A \subset \mathbb{R}$ is uniformly spaced if there exists some $\delta \in \mathbb{R}$ such that $a_{i+1} - a_i = \delta$ for all i where $(a_1, a_2, \dots)$ is the ordered sequence formed from elements a_r of A, i.e., $a_1 < a_2 < \dots < a_i < a_{i+1}, \dots$

Note that DTDGs are naturally represented by a set of uniformly spaced timestamps. This is because DTDGs correspond to sequences that do not contain any time information. Let us denote the set of CTDG timestamps $T \subseteq \mathcal{T}$ such that $\mathcal{T}$ is countable and uniformly spaced. Our idea is to construct a DTDG sequence with timestamps that coincide with the elements in $\mathcal{T}$. Then, since $T \subseteq \mathcal{T}$, we do not lose any information pertaining to events occurring at timestamps given by T. Without loss of generality, in the following we assume that the elements of T and $\mathcal{T}$ are arranged in their increasing order respectively, i.e., $t_i < t_{i+1}$ for all i, and $\tau_k < \tau_{k+1}$ for all k.

Consider a CTDG $(\mathsf{G}(t_1), \mathsf{G}(t_2), \dots)$ such that $\mathsf{G}(t_i) = \{(u, v, t) : t \in T \text{ and } t \leq t_i\}$ for $t_i \in T$. Also, let us denote $H(t_i) = \{(u, v, t) \in \mathsf{G}(t_i) : t = t_i\}$ the set of events at time $t_i \in T$. We can build a corresponding DTDG $(G_1, G_2, \dots)$ such that for all $\tau_k \in \mathcal{T}$ the k-th snapshot G_k is

$$V(G_k) = \begin{cases} \{u : (u, \cdot, \tau_k) \in H(\tau_k)\}, & \text{if } \tau_k \in T; \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$E(G_k) = \begin{cases} \{(u, v) : (u, v, \tau_k) \in H(\tau_k)\}, & \text{if } \tau_k \in T; \\ \emptyset, & \text{otherwise.} \end{cases}$$

To recover the original CTDG, we can adapt the reconstruction procedure we used in the previous part of the proof. We define

$$\tilde{I} = \{(i, k) \in \mathbb{N} \times \mathbb{N} : \tau_k = t_i \text{ for } t_i \in T \text{ and } \tau_k \in \mathcal{T}\}. \tag{S4}$$

Note that we can treat $\tilde{I}$ as a map by defining $\tilde{I}(i) = k$ if and only if $(i, k) \in \tilde{I}$. To recover the original CTDG, we first create the set of events $S(G_k) = \{(u, v, k\delta) : (u, v) \in E(G_k)\}$. Then, we build $\mathsf{G}(t_i) = \cup_{j : j \leq \tilde{I}(i)} S(G_j)$ for $t_i \in T$. □

B.3 Proof of Lemma 1

Proof. Here we show that if two nodes u and v have isomorphic (L-depth) TCTs, then MP-TGNs (with L-layers) compute identical embeddings for u and v. Formally, let $T_{u,\ell}(t)$ denote the TCT of u with ℓ layers. We want to show that $T_{u,\ell}(t) \cong T_{v,\ell}(t) \Rightarrow h_u^{(\ell)}(t) = h_v^{(\ell)}(t)$. We employ a proof by induction on ℓ. Since there is no ambiguity, we drop the dependence on time in the following.

[*Base case*] Consider $\ell = 1$. By the isomorphism assumption $T_{u,1} \cong T_{v,1}$, $h_u^{(0)} = s_u = s_v = h_v^{(0)}$ — roots of both trees have the same states. Also, for any children i of u in $T_{u,1}$ there is a corresponding one $f(i)$ in $T_{v,1}$ with $s_i = s_{f(i)}$. Recall that the ℓ-th layer aggregation function $\text{AGG}^{(\ell)}(\cdot)$ acts on multisets of triplets of previous-layer embeddings, edge features and timestamps of temporal neighbors (see Equation 1). Since the temporal neighbors of u correspond to its children in $T_{u,1}$, then the output of the aggregation function for u and v are identical: $\tilde{h}_u^{(1)} = \tilde{h}_v^{(1)}$. In addition, since the initial embeddings of u and v are also equal (i.e., $h_u^{(0)} = h_v^{(0)}$), we can ensure that the update function returns $h_u^{(1)} = h_v^{(1)}$.

[*Induction step*] Assuming that $T_{u,\ell-1} \cong T_{v,\ell-1} \Rightarrow h_u^{(\ell-1)} = h_v^{(\ell-1)}$ for any pair of nodes u and v, we will show that $T_{u,\ell} \cong T_{v,\ell} \Rightarrow h_u^{(\ell)} = h_v^{(\ell)}$. For any children i of u, let us define the subtree of $T_{u,\ell}$ rooted at i by T_i. We know that T_i has depth $\ell - 1$, and since $T_{u,\ell} \cong T_{v,\ell}$, there exists a corresponding subtree of T_v (of depth $\ell - 1$) rooted at $f(i)$ such that $T_i \cong T_{f(i)}$. Using the induction hypothesis, we obtain that the multisets of embeddings from the children i of u and children $f(i)$ of v are identical. Note that if two ℓ-depth TCTs are isomorphic, they are also isomorphic up to depth $\ell - 1$, i.e., $T_{u,\ell} \cong T_{v,\ell}$ implies $T_{u,\ell-1} \cong T_{v,\ell-1}$ and, consequently, $h_u^{(\ell-1)} = h_v^{(\ell-1)}$ (by induction hypothesis). Thus, the input of the aggregation and update functions are identical and they compute the same embeddings for u and v. □

B.4 Proof of Proposition 2: Most expressive MP-TGNs

Proof. Consider MP-TGNs with parameter values that make $\text{AGG}^{(\ell)}(\cdot)$ and $\text{UPDATE}^{(\ell)}(\cdot)$ injective functions on multisets of triples of hidden representations, edge features and timestamps. The existence of these parameters is guaranteed by the fact that, at any given time t, the space of node states (and hidden embeddings from temporal neighbors), edge features and timestamps is finite (see Lemma B3).

Again, let $T_{u,\ell}(t)$ denote the TCT of u with ℓ layers. We want to prove that, under the injectivity assumption, if $T_{u,\ell}(t) \ncong T_{v,\ell}(t)$, then $h_u^{(\ell)}(t) \neq h_u^{(\ell)}(t)$ for any two nodes u and v. In the following, we simplify notation by removing the dependence on time. We proceed with proof by induction on the TCT's depth ℓ. Also, keep in mind that $\varphi_\ell = \text{UPDATE}^{(\ell)} \circ \text{AGG}^{(\ell)}$ is injective for any ℓ.

[*Base case*] For $\ell = 1$, if $T_{u,1} \ncong T_{v,1}$ then the root node states are different (i.e., $s_u \neq s_v$) or the multiset of states/edge features/ timestamps triples from u and v's children are different. In both cases, the inputs of φ_ℓ are different and it therefore outputs different embeddings for u and v.

[*Induction step*] The inductive hypothesis is $T_{u,\ell-1} \ncong T_{v,\ell-1} \Rightarrow h_u^{(\ell-1)} \neq h_v^{(\ell-1)}$ for any pair of nodes u and v. If $T_{u,\ell} \ncong T_{v,\ell}$, at least one of the following holds: i) the states of u and v are different, ii) the multisets of edges (edge features/ timestamps) with endpoints in u and v are different, or iii) there is no pair-wise isomorphism between the TCTs rooted at u and v's children. In the first two cases, φ_ℓ trivially outputs different embeddings for u and v. We are left with the case in which only the latter occurs. Using our inductive hypothesis, the lack of a (isomorphism ensuring) bijection between the TCTs rooted at u and v's children implies there is also no bijection between their multiset of embeddings. In turn, this guarantees that φ will output different embeddings for u and v. □

B.5 Proof of Proposition 3: The role of memory

Proof. We prove the two parts of the proposition separately. In the following proofs, we rely on the concept of monotone TCTs (see Definition 2).

Statement 1: If $L < \Delta$: $\mathcal{Q}_L^{[M]}$ is strictly stronger than $\mathcal{Q}_L$.

We know that the family of L-layer MP-TGNs with memory comprises the family of L-layer MP-TGNs without memory (we can assume identity memory). Therefore, $\mathcal{Q}_L^{[M]}$ is at least as powerful as $\mathcal{Q}_L$. To show that $\mathcal{Q}_L^{[M]}$ is strictly stronger (more powerful) than $\mathcal{Q}_L$, when $L < \Delta$, it suffices to create an example for which memory can help distinguish a pair of nodes. We provide a trivial example in Figure S1 for $L = 1$. Note that the 1-depth TCTs of u and v are isomorphic when no memory is used. However, when equipped with memory, the interaction (b, c, t_1) affects the states of v and c, making the 1-depth TCTs of u and v (at time $t > t_2$) no longer isomorphic.

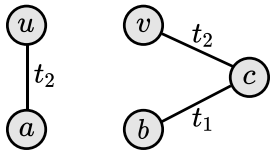


Figure S1: Temporal graph where all initial node features and edge features are identical, and $t_2 > t_1$.

Statement 2: For any L : $\mathcal{Q}_{L+\Delta}$ is at least as powerful as $\mathcal{Q}_L^{[M]}$.

It suffices to show that if $\mathcal{Q}_{L+\Delta}$ cannot distinguish a pair of nodes u and v , $\mathcal{Q}_L^{[M]}$ cannot distinguish them too. Let $T_{u,L}^M(t)$ and $T_{u,L}(t)$ denote the L-depth TCTs of u with and without memory respectively. Using Lemma 1, this is equivalent to showing that $T_{u,L+\Delta}(t) \cong T_{v,L+\Delta}(t) \Rightarrow T_{u,L}^M(t) \cong T_{v,L}^M(t)$, since no MP-TGN can separate nodes associated with isomorphic TCTs. In the following, when we omit the number of layers from TCTs, we assume TCTs of arbitrary depth.

Step 1: Characterizing the dependence of memory on initial states and events in the dynamic graph.

We now show that the memory for a node u, after processing all events with timestamp $\leq t_n$, depends on the initial states of a set of nodes $\mathcal{V}_u^n$, and a set of events annotated with their respective timestamps and features $\mathcal{B}_u^n$. If at time t_n no event involves a node z , we set $\mathcal{B}_z^n = \mathcal{B}_z^{n-1}$ and $\mathcal{V}_z^n = \mathcal{V}_z^{n-1}$. We also initialize $\mathcal{B}_u^0 = \emptyset$ and $\mathcal{V}_u^0 = \{u\}$ for all nodes u. We proceed with a proof by induction on the number of observed timestamps n.

[*Base case*] Let $\mathcal{I}_1(u) = \{v : (u, v, t_1) \in \mathcal{G}(t_1^+)\}$ be the set of nodes interacting with u at time t_1, where $\mathcal{G}(t_1^+)$ is the temporal graph right after t_1. Similarly, let $\mathcal{J}_1(u) = \{(u, \cdot, t_1) \in \mathcal{G}(t_1^+)\}$ be the set of events involving u at time t_1. Recall that up until t_1, all memory states equal initial node features (i.e., $s_u(t_1) = s_u(0)$). Then, the updated memory (see Equation 3 and Equation 4) for u depends on $\mathcal{V}_u^1 = \mathcal{V}_u^0 \cup_{v \in \mathcal{I}(u)} \mathcal{V}_v^0$, $\mathcal{B}_u^1 = \mathcal{B}_u^0 \cup \mathcal{J}_1(u)$.

[*Induction step*] Assume that for timestamp t_{n-1} the proposition holds. We now show that it holds for t_n. Since the proposition holds for $n-1$ timestamps, we know that the memory of any w that interacts with u in t_n, i.e. $w \in \mathcal{I}_n(u)$, depends on $\mathcal{V}_w^{n-1}$ and $\mathcal{B}_w^{n-1}$, and the memory of u so far depends on $\mathcal{V}_u^{n-1}$ and $\mathcal{B}_u^{n-1}$. Then, the updated memory for u depends on $\mathcal{V}_u^n = \mathcal{V}_u^{n-1} \cup_{w \in \mathcal{I}(u)} \{w, u\} \cup \mathcal{V}_w^{n-1}$ and $\mathcal{B}_u^n = \mathcal{B}_u^{n-1} \cup \mathcal{J}_n(u) \cup_{w \in \mathcal{I}(u)} \mathcal{B}_w^{n-1}$.

Step 2: $(z, w, t_{zw}) \in \mathcal{B}_u^n$ if and only if there is a path $(u_k, t_k = t_{zw}, u_{k+1})$ in $\tilde{T}_u(t_n^+)$ — the monotone TCT of u (see Definition 2) after processing events with timestamp $\leq t_n$ — with either $\sharp u_k = z, \sharp u_{k+1} = w$ or $\sharp u_k = w, \sharp u_{k+1} = z$.

[*Forward direction*] An event (z, w, t_{zw}) with $t_{zw} \leq t_n$ will be in $\mathcal{B}_u^n$ only if $z = u$ or $w = u$, or if there is a subset of events $\{(u, \sharp u_1, t_1), (\sharp u_1, \sharp u_2, t_2), \ldots, (\sharp u_k, \sharp u_{k+1}, t_{zw})\}$ with $\sharp u_k = z$ and $\sharp u_{k+1} = w$ such that $t_n \geq t_1 > \cdots > t_{zw}$. In either case, this will lead to root-to-leaf path in $\tilde{T}_u(t_n^+)$ passing through (u_k, t_{zw}, u_{k+1}). This subset of events can be easily obtained by backtracking edges that caused unions/updates in the procedure from Step 1.

[*Backward direction*] Assume there is a subpath $p = (u_k, t_k = t_{zw}, u_{k+1}) \in \tilde{T}_u(t_n^+)$ with $\sharp u_k = z$ and $\sharp u_{k+1} = w$ such that $(z, w, t_{zw}) \notin \mathcal{B}_u^n$. Since we can obtain p from $\tilde{T}_u(t_n^+)$, we know that the sequence of events $r = ((u, \sharp u_1, t_1), \ldots, (\sharp u_{k-2}, \sharp u_{k-1} = z, t_{k-1}), (z, w, t_k = t_{zw}))$ happened and that $t_i > t_{i+1} \quad \forall i$. However, since $(z, w, t_{zw}) \notin \mathcal{B}_u^n$, there must be no monotone walk starting from u going through the edge (z, w, t_{zw}) to arrive at w, which is exactly what r characterizes. Thus, we reach contradiction.

Note that the nodes in $\mathcal{V}_u^n$ are simply the nodes that have an endpoint in the events $\mathcal{B}_u^n$, and therefore are also nodes in $\tilde{T}_u(t_n^+)$ and vice-versa.

Step 3: For any node u, there is a bijection that maps $(\mathcal{V}_u^n, \mathcal{B}_u^n)$ to $\tilde{T}_u(t_n^+)$.

First, we note that $(\mathcal{V}_u^n, \mathcal{B}_u^n)$ depends on a subset of all events, which we represent as $\mathcal{G}' \subseteq \mathcal{G}(t_n^+)$. Since $\mathcal{B}_u^n$ contains all events in $\mathcal{G}'$ and $(\mathcal{V}_u^n, \mathcal{B}_u^n)$ can be uniquely constructed from $\mathcal{G}'$, then there is a bijection g that maps from $\mathcal{G}'$ to $(\mathcal{V}_u^n, \mathcal{B}_u^n)$.

Similarly, $\tilde{T}_u(t_n^+)$ also depends on a subset of events which we denote by $\mathcal{G}'' \subseteq \mathcal{G}(t_n^+)$. We note that the unique events in $\tilde{T}_u(t_n^+)$ correspond to $\mathcal{G}''$, and we can uniquely build the tree $\tilde{T}_u(t_n^+)$ from $\mathcal{G}''$. This implies that there is a bijection h that maps from $\mathcal{G}''$ to $\tilde{T}_u(t_n^+)$.

Previously, we have shown that all events in $\mathcal{B}_u^n$ are also in $\tilde{T}_u(t_n^+)$ and vice-versa. This implies that both sets depend on the same events, and thus on the same subset of all events, i.e., $\mathcal{G}' = \mathcal{G}'' = \mathcal{G}_S$. Since there is a bijection g between $\mathcal{G}_S$ and $(\mathcal{V}_u^n, \mathcal{B}_u^n)$, and a bijection h between $\mathcal{G}_S$ and $\tilde{T}_u(t_n^+)$, there exists a bijection f between $(\mathcal{V}_u^n, \mathcal{B}_u^n)$ and $\tilde{T}_u(t_n^+)$.

Step 4: If $T_{u,L+\Delta}(t^+) \cong T_{v,L+\Delta}(t^+)$, then $T_{u,L}^M(t^+) \cong T_{v,L}^M(t^+)$.

To simplify notation, we omit here the dependence on time.

Any node $w \in T_{u,L}^M$ also appears in $T_{u,L+\Delta}$ at the same level. The subtree of $T_{u,L+\Delta}$ rooted at w, denoted here by T'_w, has depth at least $k \geq \Delta$. Note that T'_w corresponds to the k-depth TCT of $\sharp w$. Since the depth of T'_w is at least Δ, we know that the $\tilde{T}'_w \cong \tilde{T}_{\sharp w}$ — i.e., imposing time-constraints to T'_w results in the monotone TCT of node $\sharp w$. Also, because the memory of $\sharp w$ depends on $\tilde{T}_{\sharp w}$, T'_w comprises the information used to compute the memory state of $\sharp w$. Note that this applies to any w in $T_{u,L}^M$; thus, $T_{u,L+\Delta}$ contains all we need to compute the states of any node of the dynamic graph that appears in $T_{u,L}^M$. The same argument applies to $T_{v,L+\Delta}$ and $T_{v,L}^M$. Finally, since $T_{u,L}^M$ can be uniquely computed from $T_{u,L+\Delta}$, and $T_{v,L}^M$ from $T_{v,L+\Delta}$, if $T_{u,L+\Delta} \cong T_{v,L+\Delta}$, then $T_{u,L}^M \cong T_{v,L}^M$. □

B.6 Proof of Proposition 4: Limitations of TGAT and TGN-Att

Proof. In this proof, we first provide an example of a dynamic graph where the TCTs of two nodes u and v are not isomorphic. Then, we show that we can not find a TGAT model such that $h_u^{(L)}(t) \neq h_v^{(L)}(t)$, i.e., TGAT does not distinguish u and v. Next, we show that even if we consider TGATs with memory (TGN-Att), it is still not possible to distinguish nodes u and v in our example.

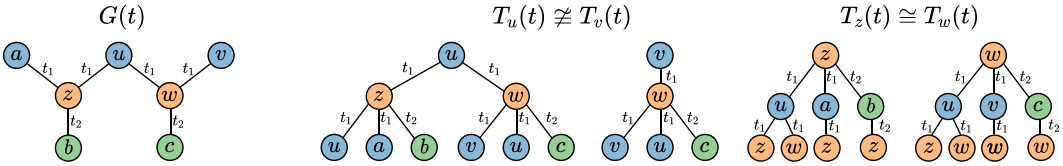


Figure S2: (**Leftmost**) Example of a temporal graph for which TGN-Att and TGAT cannot distinguish nodes u and v even though their TCTs are non-isomorphic. Colors denote node features and all edge features are identical, and $t_2 > t_1$ (and $t > t_2$). (**Right**) The 2-depth TCTs of nodes u, v, z and w. The TCTs of u and v are non-isomorphic whereas the TCTs of z and w are isomorphic.

Figure S2(leftmost) provides a temporal graph where all edge events have the same edge features. Colors denote node features. As we can observe, the TCTs of nodes u and v are not isomorphic. In the following, we consider node distinguishability at time $t > t_2$.

Statement 1: TGAT cannot distinguish the nodes u and v in our example.

Step 1: For any TGAT with ℓ layers, we have that $h_w^{(\ell)}(t) = h_z^{(\ell)}(t)$.

We note that the ℓ-layer TCTs of nodes w and z are isomorphic, for any ℓ. To see this, one can consider the symmetry around node u that allows us to define a node permutation function (bijection) f given by $f(z) = w, f(w) = z, f(a) = v, f(u) = u, f(b) = c, f(c) = b, f(v) = a$. Figure S2(right) provides an illustration of the 2-depth TCTs of z and w at time $t > t_2$.

By Lemma 1, if the ℓ-layer TCTs of two nodes z and w are isomorphic, then no ℓ-layer MP-TGN can distinguish them. Thus, we conclude that $h_w^{(\ell)}(t) = h_z^{(\ell)}(t)$ for any TGAT with arbitrary number of layers ℓ.

Step 2: There is no TGAT such that $h_v^{(\ell)}(t) \neq h_u^{(\ell)}(t)$.

To compute $h_v^{(\ell)}(t)$, TGAT aggregates the messages of v's temporal neighbors at layer $\ell - 1$, and then combines $h_v^{(\ell-1)}(t)$ with the aggregated message $\tilde{h}_v^{(\ell-1)}(t)$ to obtain $h_v^{(\ell)}(t)$.

Note that $\mathcal{N}(u,t) = \{(z,e,t_1),(w,e,t_1)\}$ and $\mathcal{N}(v,t) = \{(w,e,t_1)\}$, where e denotes an edge feature vector. Also, we have previously shown that $h_w^{(\ell-1)}(t) = h_z^{(\ell-1)}(t)$.

Using the TGAT aggregation layer (Equation S2), the query vectors of u and v are $q_u = [h_u^{(\ell-1)}(t)||\phi(0)]W_q^{(\ell)}$ and $q_v = [h_v^{(\ell-1)}(t)||\phi(0)]W_q^{(\ell)}$, respectively.

Since all events have the common edge features e, the matrices $C_u^{(\ell)}$ and $C_v^{(\ell)}$ share the same vector in their rows. The single-row matrix $C_v^{(\ell)}$ is given by $C_v^{(\ell)} = [h_w^{(\ell-1)}(t)||\phi(t-t_1)||e]$, while the two-row matrix $C_u^{(\ell)} = \Big[[h_w^{(\ell-1)}(t)||\phi(t-t_1)||e]; [h_z^{(\ell-1)}(t)\|\phi(t-t_1)||e]\Big]$, with $h_w^{(\ell-1)}(t) = h_z^{(\ell-1)}(t)$. We can express $C_u^{(\ell)} = [1,1]^\top r$ and $C_v^{(\ell)} = r$, where r denotes the row vector $r = [h_z^{(\ell-1)}(t)||\phi(t-t_1)||e]$.

Using the key and value matrices of node v, i.e., $K_v = C_v^{(\ell)} W_K^{(\ell)}$ and $V_v = C_v^{(\ell)} W_V^{(\ell)}$, we have that

$$
\begin{aligned}
\tilde{h}_v^{(\ell)}(t) &= \mathrm{softmax}(q_v K_v^\top) V_v \\
&= \underbrace{\mathrm{softmax}(q_v K_v^\top)}_{=1} r W_V^{(\ell)} && \text{[softmax of a single element is 1]} \\
&= r W_V^{(\ell)} \\
&= \underbrace{\mathrm{softmax}(q_u K_u^\top)[1,1]^\top}_{=1} r W_V^{(\ell)} = \tilde{h}_u^{(\ell)}(t) && \text{[softmax outputs a convex combination]}
\end{aligned}
$$

We have shown that the aggregated messages of nodes u and v are the same at any layer ℓ. We note that the initial embeddings are also identical $h_v^{(0)}(t) = h_u^{(0)}(t)$ as u and v have the same color. Recall that the update step is $h_v^{(\ell)}(t) = \mathrm{MLP}(h_v^{(\ell-1)}(t), \tilde{h}_v^{(\ell)}(t))$. Therefore, if the initial embeddings are identical, and the aggregated messages at each layer are also identical, we have that $h_u^{(\ell)}(t) = h_v^{(\ell)}(t)$ for any ℓ.

Statement 2: TGN-Att cannot distinguish the nodes u and v in our example.

We now show that adding a memory module to TGAT produces node states such that $s_u(t) = s_v(t) = s_a(t)$, $s_z(t) = s_w(t)$, and $s_b(t) = s_c(t)$. If that is the case, then these node states could be treated as node features in a equivalent TGAT model of our example in Figure S2, proving that there is no TGN-Att such that $h_v^{(\ell)}(t) \neq h_u^{(\ell)}(t)$. In the following, we consider TGN-Att with average memory aggregators (see Appendix A).

We begin by showing that $s_a(t) = s_u(t) = s_v(t)$ after memory updates. We note that the memory message node a receives is $[e\|t_1\|s_z(t_1)]$. The memory message node u receives is $\textsc{MeanAgg}([e\|t_1\|s_w(t_1)], [e\|t_1\|s_z(t_1)])$, but since $s_w(t_1) = s_z(t_1)$, both messages are the same, and the average aggregator outputs $[e\|t_1\|s_z(t_1)]$. Finally, the message that node v receives is $[e\|t_1\|s_w(t_1)] = [e\|t_1\|s_z(t_1)]$. Since all three nodes receive the same memory message and have the same initial features, their updated memory states are identical.

Now we show that $s_z(t) = s_w(t)$, for $t_1 < t \leq t_2$. Note that the message that node z receives is $\textsc{MeanAgg}([e\|t_1\|s_a(t_1)], [e\|t_1\|s_u(t_1)]) = [e\|t_1\|s_u(t_1)]$, with $s_u(t_1) = s_a(t_1)$. The message that node w receives is $\textsc{MeanAgg}([e\|t_1\|s_u(t_1)], [e\|t_1\|s_v(t_1)]) = [e\|t_1\|s_u(t_1)]$. Again, since the initial features and the messages received by each node are equal, $s_z(t) = s_w(t)$ for $t_1 < t \leq t_2$.

We can then use this to show that $s_z(t) = s_w(t)$ for $t > t_2$. Note that at time t_2, the message that nodes z and w receive are $[e\|t_2 - t_1\|s_b(t_2)]$ and $[e\|t_2 - t_1\|s_c(t_2)]$, respectively. Also, note that

$s_b(t_2) = s_c(t_2) = s_b(0) = s_c(0)$ as the states of b and c are only updated right after t_2. Because the received messages and the previous states (up until t_2) of z and w are identical, we have that $s_z(t) = s_w(t)$ for $t > t_2$.

Finally, we show that $s_b(t) = s_c(t)$. Using that $s_z(t_2) = s_w(t_2)$ in conjunction with the fact that node b receives message $[[e \| t_2 - t_1 \| s_z(t_2)]]$, and node c receives $[e \| t_2 - t_1 \| s_w(t_2)]$, we obtain $s_b(t) = s_c(t)$ since initial memory states and messages that the nodes received are the same. □

B.7 Proof of Proposition 5: Limitations of MP-TGN and CAWs

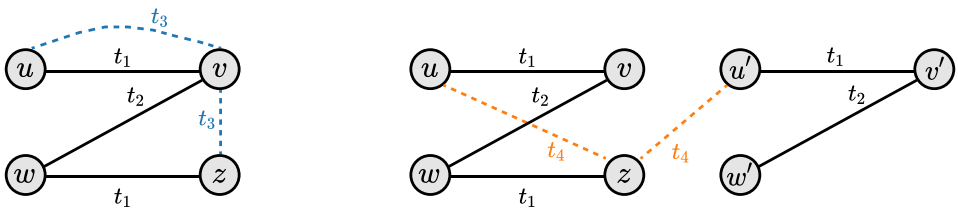


Figure S3: (**Left**) Example of a temporal graph for which CAW can distinguish the events (u, v, t_3) and (z, v, t_3) but MP-TGNs cannot. We assume that all edge and node features are identical, and $t_{k+1} > t_k$ for all k. (**Right**) Example for which MP-TGNs can distinguish (u, z, t_4) and (u', z, t_4) but CAW cannot.

Proof. Using the example in Figure S3(Left), we adapt a construction by Wang et al. [38] to show that CAW can separate events that MP-TGNs adopting node embedding concatenation cannot. We first note that the TCTs of u and z are isomorphic. Thus, since v is a common endpoint in (u, v, t_3) and (z, v, t_3), no MP-TGN can distinguish these two events. Nonetheless, CAW obtains the following anonymized walks for the event (u, v, t_3):

$$\underbrace{\{[1,0,0],[0,1,0]\}}_{I_{\text{CAW}}(u;S_u,S_v)} \xrightarrow{t_1} \underbrace{\{[0,1,0],[2,0,0]\}}_{I_{\text{CAW}}(v;S_u,S_v)}$$

$$\underbrace{\{[0,1,0],[2,0,0]\}}_{I_{\text{CAW}}(v;S_u,S_v)} \xrightarrow{t_1} \underbrace{\{[1,0,0],[0,1,0]\}}_{I_{\text{CAW}}(u;S_u,S_v)}$$

$$\underbrace{\{[0,1,0],[2,0,0]\}}_{I_{\text{CAW}}(v;S_u,S_v)} \xrightarrow{t_2} \underbrace{\{[0,0,0],[0,1,0]\}}_{I_{\text{CAW}}(w;S_u,S_v)} \xrightarrow{t_1} \underbrace{\{[0,0,0],[0,0,1]\}}_{I_{\text{CAW}}(z;S_u,S_v)}$$

and the walks associated with (z, v, t_3) are (here we omit underbraces for readability):

$$\{[1,0,0],[0,0,1]\} \xrightarrow{t_1} \{[0,1,0],[0,1,0]\}$$

$$\{[0,0,0],[2,0,0]\} \xrightarrow{t_1} \{[0,0,0],[0,1,0]\}$$

$$\{[0,0,0],[2,0,0]\} \xrightarrow{t_2} \{[0,1,0],[0,1,0]\} \xrightarrow{t_1} \{[1,0,0],[0,0,1]\}$$

In this example, assume that MLPs used to encode each walk correspond to identity mappings. Then, the sum of the elements in each set is injective since each element of the sets in the anonymized walks are one-hot vectors. We note that, in this example, we can simply choose a RNN that sums the vectors in each sequence (walks), and then apply a mean readout layer (or pooling aggregator) to obtain distinct representations for (u, v, t_3) and (z, v, t_3).

We now use the example in Figure S3(Right) to show that MP-TGNs can separate events that CAW cannot. To see why MP-TGNs can separate the events (u, z, t_4) and (u', z, t_4), it suffices to observe that the 4-depth TCTs of u and u' are non-isomorphic. Thus, a MP-TGN with injective layers could distinguish such events. Now, let us take a look at the anonymized walks for (u, z, t_4):

$$\underbrace{\{[1,0],[0,0]\}}_{I_{\text{CAW}}(u;S_u,S_z)} \xrightarrow{t_1} \underbrace{\{[0,1],[0,0]\}}_{I_{\text{CAW}}(v;S_u,S_z)}$$

$$\underbrace{\{[0,0],[1,0]\}}_{I_{\text{CAW}}(z;S_u,S_z)} \xrightarrow{t_1} \underbrace{\{[0,0],[0,1]\}}_{I_{\text{CAW}}(w;S_u,S_z)}$$

and for (u', z, t_4):

$$\underbrace{\{[1,0],[0,0]\}}_{I_{\text{CAW}}(u';S_{u'},S_z)} \xrightarrow{t_1} \underbrace{\{[0,1],[0,0]\}}_{I_{\text{CAW}}(v';S_{u'},S_z)}$$
$$\underbrace{\{[0,0],[1,0]\}}_{I_{\text{CAW}}(z;S_{u'},S_z)} \xrightarrow{t_1} \underbrace{\{[0,0],[0,1]\}}_{I_{\text{CAW}}(w;S_{u'},S_z)}$$

Since the sets of walks are identical, they must have the same embedding. Therefore, there is no CAW model that can separate these two events. □

B.8 Proof of Proposition 6: Injective MP-TGNs and the temporal WL test

We want to prove that injective MP-TGNs can separate two temporal graphs if and only if the temporal WL does the same. Our proof comprises two parts. We first show that if an MP-TGN produces different multisets of embeddings for two non-isomorphic temporal graphs $\mathcal{G}(t)$ and $\mathcal{G}'(t)$, then the temporal WL decides these graphs are not isomorphic. Then, we prove that, if the temporal WL decides $\mathcal{G}(t)$ and $\mathcal{G}'(t)$ are non-isomorphic, there is an injective MP-TGN (i.e., with injective message-passing layers) that outputs distinct multisets of embeddings.

Statement 1: Temporal WL is at least as powerful as MP-TGNs.

See Lemma B2 for proof.

Statement 2: Injective MP-TGN is at least as powerful as temporal WL.

Proof. To prove this, we can repurpose the proof of Theorem 3 in [43]. In particular, we assume MP-TGNs that meet the injective requirements of Proposition 2, i.e., MP-TGNs that implement injective aggregate and update functions on multisets of hidden representations from temporal neighbors. Following their footprints, we prove that there is a injection φ to the set of embeddings of all nodes in a temporal graph from their respective colors in the temporal WL test. We do so via induction on the number of layers ℓ. To achieve our purpose, we can assume identity memory without loss of generality.

The base case ($\ell = 0$) is straightforward since the temporal WL test initializes colors with node features. We now focus on the inductive step. Suppose the proposition holds for $\ell - 1$. Note that our update function:

$$h_v^{(\ell)}(t) = \text{UPDATE}^{(\ell)}\left(h_v^{(\ell-1)}(t), \text{AGG}^{(\ell)}(\{\!\{(h_u^{(\ell-1)}(t), t-t', e) \mid (u,e,t') \in \mathcal{N}(v,t)\}\!\})\right)$$

can be rewritten using φ as a function of node colors:

$$h_v^{(\ell)}(t) = \text{UPDATE}^{(\ell)}\left(\varphi(c^{\ell-1}(v)), \text{AGG}^{(\ell)}(\{\!\{(\varphi(c^{\ell-1}(u)), t-t', e) \mid (u,e,t') \in \mathcal{N}(v,t)\}\!\})\right).$$

Note that the composition of injective functions is also injective. In addition, time-shifting operations are also injective. Thus we can construct an injection ψ such that:

$$\begin{aligned} h_v^{(\ell)}(t) &= \psi\left(c^{\ell-1}(v), \{\!\{(c^{\ell-1}(u), t', e) \mid (u,e,t') \in \mathcal{N}(v,t)\}\!\}\right) \\ &= \psi\left(c^{\ell-1}(v), \{\!\{(c^{\ell-1}(u), t', e_{uv}(t')) \mid (v,u,t') \in \mathcal{G}(t)\}\!\}\right) \end{aligned}$$

since there exists an element $(u,e,t') \in \mathcal{N}(v,t)$ if and only if there is an event $(u,v,t') \in \mathcal{G}(t)$ with feature $e_{uv}(t') = e$.

Then, we can write:

$$\begin{aligned} h_v^{(\ell)}(t) &= \psi \circ \text{HASH}^{-1} \circ \text{HASH}\left(c^{\ell-1}(v), \{\!\{(c^{\ell-1}(u), t', e_{uv}(t')) \mid (u,v,t') \in \mathcal{G}(t)\}\!\}\right) \\ &= \psi \circ \text{HASH}^{-1}(c^{(\ell)}(v)) \end{aligned}$$

Note $\varphi = \psi \circ \text{HASH}^{-1}$ is injective since it is a composition of two injective functions. We then conclude that if the temporal WL test outputs different multisets of colors, then a suitable MP-TGN outputs different multisets of embeddings. □

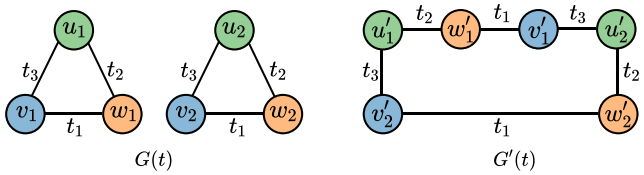


Figure S4: Examples of temporal graphs for which MP-TGNs cannot distinguish the diameter, girth, and number of cycles. For any node in $\mathcal{G}(t)$ (e.g., u_1), there is a corresponding one in $\mathcal{G}'(t)$ (u'_1) whose TCTs are isomorphic.

B.9 Proof of Proposition 7: MP-TGNs and CAWs fail to decide some graph properties

Statement 1: MP-TGNs fail to decide some graph properties.

Proof. Adapting a construction by Garg et al. [9], we provide in Figure S4 an example that demonstrates Proposition 7. Colors denote node features, and all edge features are identical. The temporal graphs $\mathcal{G}(t)$ and $G'(t)$ are non-isomorphic and differ in properties such as diameter (∞ for $\mathcal{G}(t)$ and 3 for $\mathcal{G}'(t)$), girth (3 for $\mathcal{G}(t)$ and 6 for $\mathcal{G}'(t)$), and number of cycles (2 for $\mathcal{G}(t)$ and 1 for $\mathcal{G}'(t)$). In spite of that, for $t > t_3$, the set of embeddings of nodes in $\mathcal{G}(t)$ is the same as that of nodes in $\mathcal{G}'(t)$ and, therefore, MP-TGNs cannot decide these properties. In particular, by constructing the TCTs of all nodes at time $t > t_3$, we observe that the TCTs of the pairs (u_1, u'_1), (u_2, u'_2), (v_1, v'_1), (v_2, v'_2), (w_1, w'_1), (w_2, w'_2) are isomorphic and, therefore, they can not be distinguished (Lemma 1). □

Statement 2: CAWs fail to decide some graph properties.

Since CAW does not provide a recipe to obtain graph-level embeddings, we first define such a procedure. Let $\mathcal{G}(t)$ be a temporal graph given as a set of events. We sequentially compute event embeddings h_γ for each event $\gamma = (u, v, t') \in \mathcal{G}(t)$ respecting the temporal order (two or more events at the same timestamp are computed in parallel). We then apply a readout layer to the set of event embeddings to obtain a graph-level representation. We provide a proof assuming this procedure.

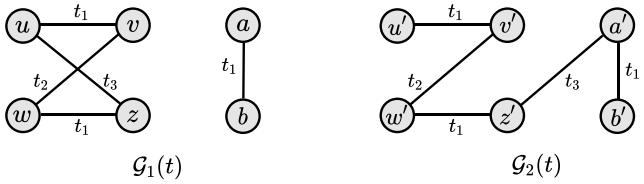


Figure S5: Examples of temporal graphs with different static properties, such as diameter, girth, and number of cycles. CAWs fail to distinguish $\mathcal{G}_1(t)$ and $\mathcal{G}_2(t)$.

Proof. We can adapt our construction in Figure 2 [rightmost] to extend Proposition 7 to CAW. The idea consists of creating two temporal graphs with different diameters, girths, and numbers of cycles that comprise events that CAW cannot separate — Figure S5 provides one such construction. In particular, CAW obtains identical embeddings for (u, z, t_3) and (a', z', t_3) (as shown in Proposition 5). The remaining events are the same up to node re-labelling and thus also lead to identical embeddings. Therefore, CAW cannot distinguish $\mathcal{G}_1(t)$ and $\mathcal{G}_2(t)$ although they clearly differ in diameter, girth, and number of cycles. □

B.10 Proof of Lemma 2

We now show that the k-th component of the relative positional features $r^{(t)}_{u\to v}$ corresponds to the number of occurrences of u at the k-th layer of the monotone TCT of v, and this is valid for all pairs of nodes u and v of the dynamic graph. We proceed with a proof by induction.

[*Base case*] Let us consider $t = 0$, i.e., no events have occurred. By definition, $r^{(0)}_{u\to v}$ is the zero vector if $u \neq v$, indicating that node u does not belong to the TCT of v. If $u = v$, then $r^{(0)}_{u\to u} = [1, 0, \ldots, 0]$ corresponds to count 1 for the root of the TCT of u. Thus, for $t = 0$, the proposition holds.

[*Induction step*] Assume that the proposition holds for all nodes and any time instant up to t. We will show that after the event $\gamma = (u, v, t)$ at time t, the proposition remains true.

Note that the event γ only impacts the monotone TCTs of u and v. The reason is that the monotone TCTs of all other nodes have timestamps lower than t, which prevents the event γ from belonging to any path (with decreasing timestamps) from the root.

Without loss of generality, let us now consider the impact of γ on the monotone TCT of v. Figure S6 shows how the TCT of v changes after γ, i.e., how it goes from $\tilde{T}_v(t)$ to $\tilde{T}_v(t^+)$. In particular, the process attaches the TCT of u to the root node v. Under this change, we need to update the counts of all nodes i in $\tilde{T}_u(t)$ regarding how many times it appears in $\tilde{T}_v(t^+)$. We do so by adding the counts in $\tilde{T}_u(t)$ (i.e., $r^{(t)}_{i\to u}$) to $\tilde{T}_v(t)$ (i.e., $r^{(t)}_{i\to v}$), accounting for the 1-layer mismatch, since $\tilde{T}_u(t)$ is attached to the first layer. This can be easily achieved with the shift matrix $P = \begin{bmatrix} 0 & 0 \\ I_{d-1} & 0 \end{bmatrix}$ applied to the counts of any node i in $\tilde{T}_u(t)$, i.e.,

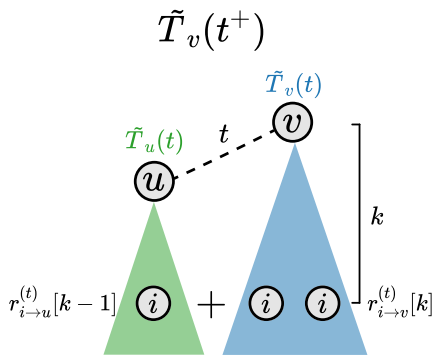


Figure S6: Illustration of how the monotone TCT of v changes after an event between u and v at time t. This allows us to see how to update the positional features of any node i of the dynamic graph that belongs to $\tilde{T}_u(t)$ relative to v.

$$r^{(t^+)}_{i\to v} = P\, r^{(t)}_{i\to u} + r^{(t)}_{i\to v} \qquad \forall i \in \mathcal{V}^{(t)}_u,$$

where $\mathcal{V}^{(t)}_u$ comprises the nodes of the original graph that belong to $\tilde{T}^{(t)}_u$.

Similarly, the event γ also affects the counts of nodes in the monotone TCT of v w.r.t. the monotone TCT of u. To account for that change, we follow the same procedure and update $r^{(t^+)}_{j\to u} = P\, r^{(t)}_{j\to v} + r^{(t)}_{j\to u}, \forall j \in \mathcal{V}^{(t)}_v$.

Handling multiple events at the same time. We now consider the setting where a given node v interacts with multiple nodes $u_1, u_2, \ldots, u_J$ at time t. We can extend the computation of positional features to this setting in a straightforward manner by noting that each event leads to an independent branch in the TCT of v. Therefore, the update of the positional features with respect to v is given by

$$r^{(t^+)}_{i\to v} = P \sum_{j=1}^{J} r^{(t)}_{i\to u_j} + r^{(t)}_{i\to v} \qquad \forall i \in \bigcup_{j=1}^{J} \mathcal{V}^{(t)}_{u_j}$$

$$\mathcal{V}^{(t^+)}_v = \mathcal{V}^{(t)}_v \bigcup_{j=1}^{J} \mathcal{V}^{(t)}_{u_j}.$$

We note that the updates of the positional features of $u_1, \ldots, u_J$ remain untouched if they do not interact with other nodes at time t.

B.11 Proof of Proposition 8: Injective function on temporal neighborhood

Proof. To capture the intuition behind the proof, first consider a multiset M such that $|M| < 4$. We can assign a unique number $\psi(m) \in \{1, 2, 3, 4\}$ to any distinct element $m \in M$. Also, the function $h(m) = 10^{-\psi(m)}$ denotes the decimal expansion of $\psi(m)$ and corresponds to reserving one decimal place for each unique element $m \in M$. Since there are less than 10 elements in the multiset, note that $\sum_m h(m)$ is unique for any multiset M.

To prove the proposition, we also leverage the well-known fact that the Cartesian product of two countable sets is countable — the Cantor's (bijective) pairing function $z : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, with $z(n_1, n_2) = \frac{(n_1+n_2)(n_1+n_2+1)}{2} + n_2$, provides a proof for that.

Here, we consider multisets $M = \{\!\{(x_i, e_i, t_i)\}\!\}$ whose tuples take values on the Cartesian product of the countable sets $\mathcal{X}$, $\mathcal{E}$, and $\mathcal{T}$ — the latter is also assumed to be bounded. In addition, we assume

the lengths of all multisets are bounded by N, i.e., $|M| < N$ for all M. Since $\mathcal{X}$ and $\mathcal{E}$ are countable, there exists an enumeration function $\psi : \mathcal{X} \times \mathcal{E} \to \mathbb{N}$ for all M. Without loss of generality, we assume $\mathcal{T} = \{1, 2, t_{\max}\}$. We want to show that exists a function of the form $\sum_i 10^{-k\psi(x_i,e_i)}\alpha^{-\beta t_i}$ that is unique on any multiset M.

Our idea is to separate a range of k decimal slots for each unique element $(x_i, e_i, \cdot)$ in the multiset. Each such a range has to accommodate at least $t_{\max}$ decimal slots (one for each value of t_i). Finally, we need to make sure we can add up to N values at each decimal slot.

Formally, we map each tuple $(x_i, e_i, \cdot)$ to one of k decimal slots starting from $10^{-k\psi(x_i,e_i)}$. In particular, for each element $(x_i, e_i, t_i = j)$ we add one unit at the j-th decimal slot after $10^{-k\psi(x_i,e_i)}$. Also, to ensure the counts for (x_i, e_i, j) and $(x_i, e_i, l \neq j)$ do not overlap, we set $\beta = \lceil \log_{10} N \rceil$ since no tuple can repeat more than N times. We use $\alpha = 10$ as we shift decimals. Finally, to guarantee that each range encompasses $t_{\max}$ slots of β decimals, we set $k = \beta(t_{\max} + 1)$. Therefore, the function

$$\sum_i 10^{-k\psi(x_i,e_i)}\alpha^{-\beta t_i}$$

is unique on any multiset M. We note that, without loss of generality, one could choose a different basis (other than 10). □

B.12 Proof of Proposition 9: Expressiveness of PINT: link prediction

Proof. We now show that PINT (with relative positional features) is strictly more powerful than MP-TGN and CAW in distinguishing edges of temporal graphs. Leveraging Proposition 5, it suffices to show that PINT is at least as powerful as both CAW and MP-TGN.

Statement 1: PINT is at least as powerful as MP-TGNs.

Since PINT is a generalization of MP-TGNs with injective aggregation/update layers, it derives that it is at least as powerful as MP-TGNs. We can set the model's parameters associated with positional features to zero and obtain an equivalent MP-TGN.

Statement 2: PINT is at least as powerful as CAW.

We wish to show that for any pair of events that PINT cannot distinguish, CAW also cannot distinguish it. Let us consider the events (u, v, t) and (u', v', t) of a temporal graph. Formally, we want to prove that if $\{\!\{T_u(t), T_v(t)\}\!\} = \{\!\{T_{u'}(t), T_{v'}(t)\}\!\}$ (i.e., the multisets contain TCTs that are pairwise isomorphic), then $\{\!\{\textsc{Enc}(W; S_u, S_v)\}\!\}_{W \in \{S_u \cup S_v\}} = \{\!\{\textsc{Enc}(W'; S_{u'}, S_{v'})\}\!\}_{W' \in \{S_{u'} \cup S_{v'}\}}$, where $\textsc{Enc}$ denotes the walk-encoding function of CAW. Importantly, for the sake of this proof, we assume that all TCTs here are augmented with positional features, characterizing edge embeddings obtained from PINT.

Without loss of generality, we can assume that $T_u(t) \cong T_{u'}(t)$ and $T_v(t) \cong T_{v'}(t)$. By Lemma B1, we know that the corresponding monotone TCTs are also isomorphic: $\tilde{T}_u(t) \cong \tilde{T}_{u'}(t)$, and $\tilde{T}_v(t) \cong \tilde{T}_{v'}(t)$ with associated bijections $f_1 : V(\tilde{T}_u(t)) \to V(\tilde{T}_{u'}(t))$ and $f_2 : V(\tilde{T}_v(t)) \to V(\tilde{T}_{v'}(t))$.

We can construct a tree T_{uv} by attaching $\tilde{T}_u(t)$ and $\tilde{T}_v(t)$ to a (virtual) root node uv — without loss of generality, u and v are the left-hand and right-hand child of uv, respectively. We can follow the same procedure and create the tree $T_{u'v'}$ by attaching the TCTs $\tilde{T}_{u'}(t)$ and $\tilde{T}_{v'}(t)$ to a root node $u'v'$. Since the left-hand and right-hand subtrees of T_{uv} and $T_{u'v'}$ are isomorphic, then T_{uv} and $T_{u'v'}$ are also isomorphic. Let $f : V(T_{uv}) \to V(T_{u'v'})$ denote the bijection associated with the augmented trees. We also assume f is constructed by preserving the bijections f_1 and f_2 defined between the original monotone TCTs: this ensures that f does not map any node in $V(\tilde{T}_u(t))$ to a node in $V(\tilde{T}_{v'}(t))$, for instance. We have that

$$[r^{(t)}_{\sharp i \to u} \| r^{(t)}_{\sharp i \to v}] = [r^{(t)}_{\sharp f(i) \to u'} \| r^{(t)}_{\sharp f(i) \to v'}] \quad \forall i \in V(T_{uv}) \setminus \{uv\}$$

Note that we use the function $\sharp$ (that maps nodes in the TCT to nodes in the dynamic graph) here because the positional feature vectors are defined for nodes in the dynamic graph.

To guarantee that two encoded walks are identical $\textsc{Enc}(W; S_u, S_v) = \textsc{Enc}(W'; S_{u'}, S_{v'})$, it suffices to show that the anonymized walks are equal. Thus, we turn our problem into showing that for any walk $W = (w_0, t_0, w_1, t_1, \dots)$ in $S_u \cup S_v$, there exists a corresponding one

$W' = (w'_0, t_0, w'_1, t_1, \dots)$ in $S_{u'} \cup S_{v'}$ such that $I_{\text{CAW}}(w_i; S_u, S_v) = I_{\text{CAW}}(w'_i; S_{u'}, S_{v'})$ for all i. Recall that $I_{\text{CAW}}(w_i; S_u, S_v) = \{g(w_i; S_u), g(w_i; S_v)\}$, where $g(w_i; S_u)$ is a vector whose k-component stores how many times w_i appears in a walk from S_u at position k.

A key observation is that there is an equivalence between deanonymized root-leaf paths in T_{uv} and walks in $S_u \cup S_v$ (disregarding the virtual root node). By deanonymized, we mean paths where node identities (in the temporal graph) are revealed by applying the function $\sharp$. Using this equivalence, it suffices to show that

$$g(\sharp i; S_u) = g(\sharp f(i); S_{u'}) \text{ and } g(\sharp i; S_v) = g(\sharp f(i); S_{v'}) \quad \forall i \in V(T_{uv}) \setminus \{uv\}$$

Suppose there is an $i \in V(T_{uv}) \setminus \{uv\}$ such that $g(\sharp i; S_u) \neq g(\sharp f(i); S_{u'})$. Without loss of generality, suppose this holds for the ℓ-th entry of the vectors.

We know there are exactly $r^{(t)}_{a \to u}[\ell]$ nodes at the ℓ-th level of $\tilde{T}_u(t)$ that are associated with $a = \sharp i \in V(\mathcal{G}(t))$. We denote by Ψ the set comprising such nodes. It also follows that computing $g(\sharp i; S_u)[\ell]$ is the same as summing up the amount of leaves of each subtree of $\tilde{T}_u(t)$ rooted at $\psi \in \Psi$, which we denote as $\mathscr{L}(\psi; \tilde{T}_u(t))$, i.e.,

$$g(\sharp i; S_u)[\ell] = \sum_{\psi \in \Psi} \mathscr{L}(\psi; \tilde{T}_u(t)).$$

Since we assume $g(\sharp i; S_u)[\ell] \neq g(\sharp f(i); S_{u'})[\ell]$, then it holds that

$$g(\sharp i; S_u)[\ell] \neq g(\sharp f(i); S_{u'})[\ell] \Rightarrow \sum_{\psi \in \Psi} \mathscr{L}(\psi; \tilde{T}_u(t)) \neq \sum_{\psi \in \Psi} \mathscr{L}(f(\psi); \tilde{T}_{u'}(t)) \tag{S5}$$

Note that the subtree of $\tilde{T}_u$ rooted at ψ should be isomorphic to the subtree of $\tilde{T}_{u'}$ rooted at $f(\psi)$, and therefore have the same number of leaves. However, the RHS of Equation S5 above implies there is a $\psi \in \Psi$ for which $\mathscr{L}(\psi; \tilde{T}_u) \neq \mathscr{L}(f(\psi); \tilde{T}_{u'})$, reaching a contradiction. The same argument can be applied to v and v' to prove that $g(\sharp i; S_v) = g(\sharp f(i); S_{v'})$. □

C Additional related works

Structural features for static GNNs. Using structural features to enhance the power of GNNs is an active research topic. Bouritsas et al. [48] improved GNN expressivity by incorporating counts of local structures in the message-passing procedure, e.g, the number triangles a node appears on. These counts depend on identifying subgraph isomorphisms and, naturally, can become intractable depending on the chosen substructure. Li et al. [52] proposed increasing the power of GNNs using distance encodings, i.e., augmenting original node features with distance-based ones. In particular, they compute the distance between a node set whose representation is to be learned and each node in the graph. To alleviate the cost of distance encoding, an alternative is to learn absolute position encoding schemes [51, 61, 62], that try to summarize the role each node plays in the overall graph topology. We note that another class of methods uses random features to boost the power of GNNs [47, 58]. However, these models are referred to be hard to converge and obtain noisy predictions [62].

The most trivial difference between these approaches and PINT is that our relative positional features account for temporal information. On a deeper level, our features summarize the role each node plays in each other's monotone TCT instead of measuring, e.g., pair-wise distances in the original graph or counting substructures. Also, our scheme leverages the temporal aspect to achieve computational tractability, updating features incrementally as events unroll. Finally, while some works proposing structural features for static GNNs present marginal gains [62], PINT exhibits significant performance gains in real-world temporal link prediction tasks.

Other models for temporal graphs. Representation learning for dynamic graphs is a broad and diverse field. In fact, strategies to cope with the challenge of modeling dynamic graphs can come in many flavors, including simple aggregation schemes [18], walk-aggregating methods [53], and combinations of sequence models with GNNs [56, 59]. For instance, Seo et al. [59] used a spectral graph convolutional network [49] to encode graph snapshots followed by a graph-level LSTM [13]. Manessi et al. [55] followed a similar approach but employed a node-level LSTM, with parameters shared across the nodes. Sankar et al. [57] proposed a fully attentive model based on graph attention

networks [34]. Pareja et al. [56] applied a recurrent neural net to dynamically update the parameters of a GCN. Gao and Ribeiro [50] compared the expressive power of two classes of models for discrete dynamic graphs: time-and-graph and time-then-graph. The former represents the standard approach of interleaving GNNs and sequence (e.g., RNN) models. In the latter class, the models first capture node and edge dynamics using RNNs, and are then feed into graph neural networks. The authors showed that time-then-graph has expressive advantage over time-and-graph approaches under mild assumptions. For an in-depth review of representation learning for dynamic graphs, we refer to the survey works by Kazemi et al. [14] and Skarding et al. [60].

While most of the early works focused on discrete-time dynamic graphs, we have recently witnessed a rise in interest in models for event-based temporal graphs (i.e., CTDGs). The reason is that models for DTDGs may fail to leverage fine-grained temporal and structural information that can be crucial in many applications. In addition, it is hard to specify meaningful time intervals for different tasks. Thus, modern methods for temporal graphs explicitly incorporate timestamp information into sequence/graph models, achieving significant performance gains over approaches for DTDGs [38]. Appendix A provides a more detailed presentation of CAW, TGN-Att, and TGAT, which are among the best performing models for link prediction on temporal graphs. Besides these methods, JODIE [16] applies two RNNs (for the source and target nodes of an event) with a time-dependent embedding projection to learn node representations of item-user interaction networks. Trivedi et al. [32] employed RNNs with a temporally attentive module to update node representations. APAN [63] consists of a memory-based TGN that uses attention mechanism to update memory states using multi-hop temporal neighborhood information. Makarov et al. [54] proposed incorporating edge embeddings obtained from CAW into MP-TGNs' memory and message-passing computations.

D Datasets and implementation details

D.1 Datasets

In our empirical evaluation, we have considered six datasets for dynamic link prediction: Reddit[1], Wikipedia[2], UCI[3], LastFM[4], Enron[5], and Twitter. Reddit is a network of posts made by users on subreddits, considering the 1,000 most active subreddits and the 10,000 most active users. Wikipedia comprises edits made on the 1,000 most edited Wikipedia pages by editors with at least 5 edits. Both Reddit and Wikipedia networks include links collected over one month, and text is used as edge features, providing informative context. The LastFM dataset is a network of interactions between user and the songs they listened to. UCI comprises students' posts to a forum at the University of California Irvine. Enron contains a collection of email events between employees of the Enron Corporation, before its bankruptcy. The Twitter dataset is a non-bipartite net where nodes are users and interactions are retweets. Since Twitter is not publicly available, we build our own version by following the guidelines by Rossi et al. [27]. We use the data available from the 2021 Twitter RecSys Challenge and select 10,000 nodes and their associated interactions based on node participation: number of interactions the node participates in. We also apply multilingual BERT to obtain text representations of retweets (edge features).

Table S1 reports statistics of the datasets such as number of temporal nodes and links, and the dimensionality of the edge features. We note that UCI, Enron, and LastFM represent non-attributed networks and therefore do not contain feature vectors associated with the events. Also, the node features for all datasets are vectors of zeros [42].

Table S1: Summary statistics of the datasets.

Dataset	#Nodes	#Events	#Edge feat.	Bipartite?
Reddit	10,984 (10,000 / 984)	672,447	172	Yes
Wikipedia	9,227 (8,227 / 1,000)	157,474	172	Yes
Twitter	8,925	406,564	768	No
UCI	1,899	59,835	-	No
Enron	184	125,235	-	No
LastFM	1,980 (980 / 1,000)	1,293,103	-	Yes

D.2 Implementation details

We train all models in link prediction tasks in a self-supervised approach. During training, we generate negative samples: for each actual event (z, w, t) (class 1), we create a fake one (z, w', t) (class 0) where w' is uniformly sampled from the set of nodes, and both events have the same edge feature vector.

To ensure a fair comparison, we mainly rely on the original repositories and guidelines. For instance, regarding the training of MP-TGNs (including PINT), we mostly follow the setup and choices in the implementation available in [27]. In particular, we apply the Adam optimizer with learning rate 10^{-4} during 50 epochs with early stopping if there is no improvement in validation AP for 5 epochs. In addition, we use batch size 200 for all methods. We report statistics (mean and standard deviation) of the performance metric (AP) over ten runs.

MP-TGNs. For TGN-Att, we follow Rossi et al. [27] and sample either ten or twenty temporal neighbors with memory dimensionality equal to 32 (Enron), 100 (UCI, Twitter), or 172 (Reddit, Wikipedia, LastFM), node embedding dimension equal to 100, and two attention heads. We use a memory unit implemented as a GRU, and update the state of each node based on only its most recent message. For TGAT, we use twenty temporal neighbors and two layers.

CAW. We conduct model selection using grid search over: i) time decay $\alpha \in \{0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 100.0\} \times 10^{-6}$, ii) number of walks $M \in \{1, 2, 3, 4, 5\}$;

[1] `http://snap.stanford.edu/jodie/reddit.csv`

[2] `http://snap.stanford.edu/jodie/wikipedia.csv`

[3] `http://konect.cc/networks/opsahl-ucforum/`

[4] `http://snap.stanford.edu/jodie/lastfm.csv`

[5] `https://www.cs.cmu.edu/~./enron/`

and iii) walk length $L \in \{32, 64, 128\}$. The best combination of hyperparameters is shown in Table S2. The remaining training choices follows the default values from the original implementation. Importantly, we note that TGN-Att's original evaluation setup is different from CAW's. Thus, we adapted CAW's original repo to reflect these differences and ensure a valid comparison.

Table S2: Optimal hyperparameters for CAW.

Dataset	**Time decay** α	**#Walks**	**Walk length**
Reddit	10^{-8}	32	3
Wikipedia	4×10^{-6}	64	4
Twitter	10^{-6}	64	3
UCI	10^{-5}	64	2
Enron	10^{-6}	64	5
LastFM	10^{-6}	64	2

PINT. We use $\alpha = 2$ (in the exponential aggregation function), and experiment with learned and fixed β. We apply a relu function to avoid negative values of β, which could lead to unstable training. We do grid search as the follow: when learning beta, we consider initial values for $\beta \in \{0.1, 0.5\}$; for the fixed case (`requires_grad=False`), we evaluate $\beta \in \{10^{-3} \times |\mathcal{N}|, 10^{-4} \times |\mathcal{N}|, 10^{-5} \times |\mathcal{N}|\}$ — $|\mathcal{N}|$ denotes the number of temporal neighbors — and always apply memory as in the original implementation of TGN-Att. We consider number message passing layers ℓ in $\{1, 2\}$. Also, we apply neighborhood sampling with the number of neighbors in $\{10, 20\}$, and update the state of a node based on its most recent message. We then carry out model selection based on AP values obtained during validation. Overall, the models with fixed β led to better results. Table S3 reports the optimal hyperparameters for PINT found via automatic model selection.

In all experiments, we use relative positional features with $d = 4$ dimensions. For computational efficiency, we update the relative positional features only after processing a batch, factoring in all events from that batch. Note that this prevents information linkage as these positional features take effect after prediction. In addition, since temporal events repeat (in the same order) at each epoch, we also speed up PINT's training procedure by precomputing and saving the positional features for each batch. To save up space, we store the positional features as sparse matrices.

Table S3: Optimal hyperparameters for PINT.

Dataset	$\beta/\lvert\mathcal{N}\rvert$	**#Neighbors** ($\lvert\mathcal{N}\rvert$)	**#Layers**
Reddit	10^{-5}	10	2
Wikipedia	10^{-4}	10	2
Twitter	10^{-5}	20	2
UCI	10^{-5}	10	2
Enron	10^{-5}	20	2
LastFM	10^{-4}	10	2

Hardware. For all experiments, we use Tesla V100 GPU cards and consider a memory budget of 32GB of RAM.

E Deletion and node-level events

Rossi et al. [27] propose handling edge deletions by simply updating memory states of the edge's endpoints, as if we were dealing with a usual edge addition. However, it is not discussed whether the edge in question should be excluded from the event list or if we should just add a novel event with edge features that characterize deletion. If we choose the former, we may be unable to recover the memory state of a node from its monotone TCT and the original node features. Removing an edge from the event list also affects the computation of node embeddings. Therefore, we advise practitioners to do the latter when using PINT. It is worth mentioning that the vast majority of models for temporal interaction prediction do not consider the possibility of deletion events.

Regarding node-level events, PINT can accommodate node addition by simply creating novel memory states. To deal with node feature updates, we can create an edge event with both endpoints on that

node, inducing a self-loop in the dynamic graph. Also, we can combine (e.g., concatenate) the temporal features in message-passage operations, similarly to the general formulation of the MP-TGN framework [27]. Finally, we can deal with the removal of a node v by following our previous (edge deletion) procedure to delete all edges with endpoints in v.

F Additional experiments

Time comparison. Figure S7 compares the time per epoch for PINT and for the prior art (CAW, TGN-Att, and TGAT) in the Enron and LastFM datasets. Following the trend in Figure 7, Figure S7 further supports that PINT is generally slower than other MP-TGNs but, after a few training epochs, is orders of magnitude faster than CAW. In the case of Enron, the time CAW takes to complete an epoch is much higher than the time we need to preprocess PINT's positional features.

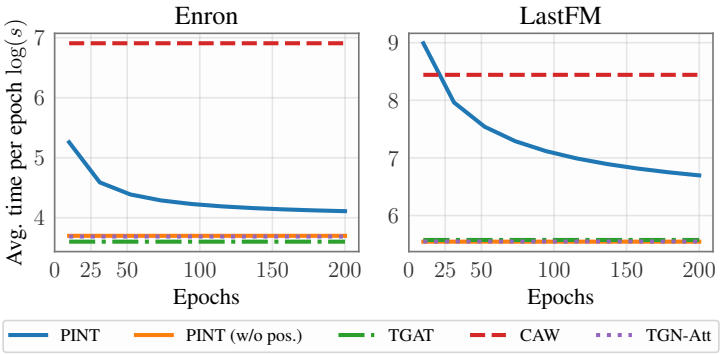


Figure S7: Time comparison: PINT versus TGNs (in log-scale) on Enron and LastFM.

Experiments on node classification. For completeness, we also evaluate PINT on node-level tasks (Wikipedia and Reddit). We follow closely the experimental setup in Rossi et al. [27] and compare against the baselines therein. Table S4 shows that PINT ranks first on Reddit and second on Wikipedia. The values for PINT reflect the outcome of 5 repetitions.

Table S4: Results for node classification (AUC).

	Wikipedia	**Reddit**
CTDNE	75.89 ± 0.5	59.43 ± 0.6
JODIE	84.84 ± 1.2	61.83 ± 2.7
TGAT	83.69 ± 0.7	65.56 ± 0.7
DyRep	84.59 ± 2.2	62.91 ± 2.4
TGN-Att	87.81 ± 0.3	67.06 ± 0.9
PINT	87.59 ± 0.6	67.31 ± 0.2

Supplementary References

[47] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[48] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. In *Arxiv e-prints*, 2020.

[49] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[50] J. Gao and B. Ribeiro. On the equivalence between temporal and static graph representations for observational predictions. *ArXiv*, 2103.07016, 2021.

[51] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Letourneau, and P. Tossou. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[52] P. Li, Y. Wang, H. Wang, and J. Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[53] S. Mahdavi, S. Khoshraftar, and A. An. dynnode2vec: Scalable dynamic network embedding. In *International Conference on Big Data*, 2018.

[54] I. Makarov, A. V. Savchenko, A. Korovko, L. Sherstyuk, N. Severin, A. Mikheev, and D. Babaev. Temporal graph network embedding with causal anonymous walks representations. *ArXiv*, 2108.08754, 2021.

[55] F. Manessi, A. Rozza, and M. Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97, 2020.

[56] A. Pareja, G. Domeniconi, J. Chen, T. Ma, H. Kanezashi T. Suzumura, T. Kaler, T. B. Schardl, and C. E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[57] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *International Conference on Web Search and Data Mining (WSDM)*, 2020.

[58] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *SIAM International Conference on Data Mining (SDM)*, 2021.

[59] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing (ICONIP)*, 2018.

[60] J. Skarding, B. Gabrys, and K. Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[61] B. Srinivasan and B. Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations (ICLR)*, 2020.

[62] H. Wang, H. Yin, M. Zhang, and P. Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2022.

[63] X. Wang, D. Lyu, M. Li, Y. Xia, Q. Yang, X. Wang, X. Wang, P. Cui, Y. Yang, B. Sun, and Z. Guo. APAN: Asynchronous propagation attention network for real-time temporal graph embedding. *International Conference on Management of Data*, 2021.