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 introduce 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. Our experiments demonstrate that PINT significantly outperforms existing TGNs on several real-world benchmarks.

1 Introduction

Graph neural networks (GNNs) have recently led to breakthroughs in many applications 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. In several scenarios, these temporal graphs are only given as a sequence of timestamped events. Recently, temporal graph nets (TGNs) 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, time encoders, recurrent models, and message passing.

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. Similarly, many other notable advances on GNNs were made possible by untangling their ability to generalize, extrapolate, compute graph properties, and express Boolean classifiers; by uncovering their connections to distributed algorithms, graph kernels, dynamic programming, diffusion processes, graphical models, and combinatorial optimization; and by analyzing their discriminative power. 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?
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) that subsumes a wide variety of methods [e.g., 16, 32, 41]. 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 [37] 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 \( V = \{1, 2, \ldots, n\} \) 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 \((G_1, G_2, \ldots)\) 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 graphs \((G(t_0), G(t_1), \ldots)\) such that \( t_k < t_{k+1} \), and \( G(t_k) \) results from updating \( G(t_k) \) with all events at time \( t_k \). 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 the 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 does not always hold.

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, 27, 31], we focus on CTDGs with edge addition events (see Appendix E for a discussion on deletion). Thus, we can represent temporal graphs as sets \( G(t) = \{(t_k, e_{k}, e) \mid t_k < t\} \). We also assume each distinct node \( v \) in \( G(t) \) 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 neighborhood of node \( v \) at time \( t \) as \( \mathcal{N}(v, t) = \{(u, e_{uv}(t'), t') \mid \exists (u, v, t') \in G(t)\} \), i.e., the set of neighbor-feature/token pairs from all interactions of node \( v \) prior to \( t \).

MP-TGNs compute the temporal representation \( h_v^{(t)}(t) \) of \( v \) at layer \( t \) by recursively applying

\[
\hat{h}_v^{(t)}(t) = A_{\text{GG}}^{(t)}(\{h_v^{(t-1)}(t), t-t_v, e \mid (u, e, t') \in \mathcal{N}(v, t)\})
\]

\[
h_v^{(t)}(t) = \text{UPDATE}^{(t)} \left( h_v^{(t-1)}(t), \hat{h}_v^{(t)}(t) \right),
\]

where \( \{\cdot\} \) denotes multisets, \( h_v^{(0)}(t) \) is the state of \( v \) at time \( t \), and \( A_{\text{GG}}^{(t)} \) and \( \text{UPDATE}^{(t)} \) 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

\[
m_v(t) = \text{MSGG}^{(t)}(\{s_v(t^-), s_v(t^-), t-t_v, e_{uv}(t) \mid \exists (v, u, t) \in \mathcal{J}(v, t)\})
\]

\[
s_v(t) = \text{MEMORY}^{(t)}(s_v(t^-), m_v(t^-)),
\]

where \( s_v(0) = x_v \) (initial node features), \( s_v(t^-) \) denotes the state of \( v \) right before time \( t \), and \( t_v \) denotes the time of the last update to \( v \). MSGG combines information from simultaneous events involving node \( v \) and MEMORY usually implements a GRU [5]. Notably, some MP-TGNs do not use memory, or equivalently, they employ identity memory, i.e., \( s_v(t) = x_v \). See Appendix A for details.

Causal Anonymous Walks (CAWs). Wang et al. [37] proposed CAW as an effective 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 \( v \). An L-length temporal walk is represented as \( W = ((w_1, t_1), (w_2, t_2), \ldots, (w_L, t_L)) \), with \( t_1 > t_2 > \ldots > t_L \), and \((w_{i-1}, w_i, t_i) \in G(t)\) for all \( i \). 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 \( \ell \)-th entry of \( g(w; S_u) \) stores how many times \( w \) appears at the \( \ell \)-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 \( W_\ell \) is \( \ell \)-th pair of \( W \).
To encode a walk $W$ with respect to the sets $S_u$ and $S_v$, CAW applies $\text{Enc}(W; S_u, S_v) = \text{RNN}(f_1(J_{\text{CAW}}(w_i; S_u, S_v)) || f_2(t_{i-1} - t_i)_{i=1}^t)$ where $f_1$ is a permutation-invariant function, $f_2$ is a time encoder, and $t_0 = t$. 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 TGN layers $L$. We will keep the dependence on depth $L$ implicit for notational simplicity. For each element $(u, e, t') \in \mathcal{N}(u, t)$ 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 $\pi^t_u$ the (possibly many-to-one) operator that maps nodes in $T_v(t)$ back to nodes in $\mathcal{G}(t)$, e.g., $\pi^t_u i = u$. Each node $i$ in $T_v(t)$ has a state vector $s_i = s_{\pi^t_u i}(t)$. 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 $\pi^t_u$ simply as $\pi$ for a cleaner notation.

We study the expressiveness 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 $T_u(t)$ and $T_v(t)$ at time $t$ are isomorphic if there is a bijection $f : V(T_u(t)) \rightarrow V(T_v(t))$ between the nodes of the trees such that the following holds:

1. $(u, v, t') \in E(T_u(t)) \iff (f(u), f(v), t') \in E(T_v(t))$
2. $\forall (u, v, t') \in E(T_u(t)) : e_{uv}(t') = e_{f(u)f(v)}(t')$ and $\forall u \in V(T_u(t)) : s_u = s_{f(u)}$ 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 [42].

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 there exists an MP-TGN $Q$ with $L$ layers and injective aggregation and update functions at each layer that 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 $Q_{L+M}$ denote the class of MP-TGNs with recurrent memory and $L$ layers. Similarly, we denote by $Q_L$ the family of memoryless MP-TGNs with $L$ layers. Let $\Delta$ be the temporal diameter of $\mathcal{G}(t)$ (see Definition B2). Then, it holds that:

1. If $L < \Delta$: $Q_{L+M}$ is strictly more powerful than $Q_L$ in distinguishing nodes of $\mathcal{G}(t)$;
2. For any $L : Q_{L+\Delta}$ is at least as powerful as $Q_{L+M}$ in distinguishing nodes of $\mathcal{G}(t)$.
We can extend the notion of node distinguishability to edges/events. We say that a model distinguishes the 3-depth TCTs of \( u \) and \( v \). Remarkably, these models are among the best-performing MP-TGNs. Nonetheless, we can show 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 message aggregator (i.e., using \( \text{MEAN as MSGAGG} \)) 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 message 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 evaluate the resulting vector using an MLP. On the other hand, CAW is originally designed for link prediction tasks and directly computes edge embeddings, bypassing 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 \( b_\gamma \neq b_{\gamma'} \) for \( \gamma \) and \( \gamma' \). **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 MP-TGN’s 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 \( G(t) \) by applying the following iterative procedure:

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

**Refinement:** At step \( \ell \), the colors of all nodes are refined using a hash (injective) function: for all \( v \in V(G(t)) \), we apply \( c^{\ell+1}(v) = \text{HASH}(c^\ell(v), c^\ell(u), c^\ell(t), t) : (u, v, t) \in G(t)) \);
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 $X$, edge features $E$, and timestamps $T$.
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 $G(t)$ and $G'(t)$ differ in diameter ($\infty$ 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 $G(t)$, there is a corresponding one
in $G'(t)$ whose TCTs are isomorphic, e.g., $T_u(t) \equiv T'_{u'}(t)$ for $t > t_5$. 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.

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 position-encoding injective temporal 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 [37, 41]. Proposition 8 introduces an injective aggregation
scheme that captures this principle employing a linearly exponential time decay.

Proposition 8 (Injective function on temporal neighborhood). Let $X$ and $E$ be countable, and $T$
countable and bounded. There exists a function $f$ and scalars $\alpha$ and $\beta$ such that $\sum_{(x_i, e_i, t_i)} f(x_i, e_i) e^{-\beta t_i}$
is unique on any multiset $M = \{(x_i, e_i, t_i)\} \subseteq X \times E \times T$ 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 $\ell$ using the following message passing steps:

$$h_{v}^{(\ell)}(t) = \sum_{(u, e, t) \in N(v, t)} \text{MLP}_{\text{agg}}^{(\ell)} \left( h_{u}^{(\ell-1)}(t) \| c \right) e^{-\beta (t - t')}$$

$$h_{v}^{(\ell)}(t) = \text{MLP}_{\text{upd}}^{(\ell)} \left( h_{v}^{(\ell-1)}(t) \| h_{v}^{(\ell)}(t) \right)$$

where $\|$ denotes concatenation, $h_{v}^{(0)} = s_v(t)$, $\alpha$ and $\beta$ are scalar (hyper-)parameters, and MLP_{\text{agg}}^{(\ell)}
and 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 from some 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.
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 $r_j^{(t)} \in \mathbb{N}^d$ 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

$$V_i^{(0)} = \{i\} \quad \forall i$$

$$r_{i \rightarrow j}^{(0)} = \begin{cases} [1, 0, \ldots, 0]^T \quad & \text{if } i = j \\ [0, 0, \ldots, 0]^T \quad & \text{if } i \neq j \end{cases}$$

where we use $t^-$ to denote values “right before” $t$. The set $V_i$ 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.

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 $p = (u, t_1, u_1, t_2, u_2, \ldots)$ from the root $u$ to leaf nodes of $\tilde{T}_u(t)$ time monotonically decreases with respect to time, i.e., we have that $t_1 > t_2 > \ldots$.

Lemma 2. For any pair of nodes $i, u$ of a temporal graph $G(t)$, the $k$-th component of the positional feature vector $r_{i \rightarrow u}^{(t)}$ 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 $h_v$ 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 $r_{j \rightarrow u}^{(t)}$ and $r_{j \rightarrow v}^{(t)}$ 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 $h_u^{[L]}(t)$ and $h_v^{[L]}(t)$ 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 $r_{j \rightarrow u}^{(t)}$. Then, we use our injective MP to combine the information stored in $u$ and its neighboring nodes. Figure 4 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, achieving maximally expressive MP-TGNs.
We have implemented experiments using PyTorch [25] and code is available as additional material.

We now assess the performance of PINT on several popular and large-scale benchmarks for TGNs. We compare PINT against five prominent TGNs: Jodie [16], DyRep [32], TGAT [41], TGN-Att [27], and CAW [37]. 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. [41] 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 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...
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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Reddit</th>
<th>Wikipedia</th>
<th>Twitter</th>
<th>UCI</th>
<th>Enron</th>
<th>LastFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAT</td>
<td>97.33 ± 0.2</td>
<td>94.73 ± 0.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>97.65 ± 0.2</td>
<td>93.56 ± 0.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Transductive

<table>
<thead>
<tr>
<th>Method</th>
<th>Reddit</th>
<th>Wikipedia</th>
<th>Twitter</th>
<th>UCI</th>
<th>Enron</th>
<th>LastFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jodie</td>
<td>97.11 ± 0.3</td>
<td>94.62 ± 0.5</td>
<td>98.23 ± 0.1</td>
<td>86.73 ± 1.0</td>
<td>77.31 ± 4.2</td>
<td>69.32 ± 1.0</td>
</tr>
<tr>
<td>DyRep</td>
<td>97.98 ± 0.1</td>
<td>94.59 ± 0.2</td>
<td>98.48 ± 0.1</td>
<td>54.60 ± 3.1</td>
<td>77.68 ± 1.6</td>
<td>69.24 ± 1.4</td>
</tr>
<tr>
<td>TGAT</td>
<td>98.12 ± 0.2</td>
<td>95.34 ± 0.1</td>
<td>98.70 ± 0.1</td>
<td>77.51 ± 0.7</td>
<td>68.02 ± 0.1</td>
<td>54.77 ± 0.4</td>
</tr>
<tr>
<td>TGN-Att</td>
<td>98.70 ± 0.1</td>
<td>98.46 ± 0.1</td>
<td>98.80 ± 0.1</td>
<td>80.40 ± 1.4</td>
<td>79.91 ± 1.3</td>
<td>80.69 ± 0.2</td>
</tr>
<tr>
<td>CAW</td>
<td>98.39 ± 0.1</td>
<td>98.63 ± 0.1</td>
<td>98.72 ± 0.1</td>
<td>92.16 ± 0.1</td>
<td>92.09 ± 0.7</td>
<td>81.29 ± 0.1</td>
</tr>
</tbody>
</table>

PINT (w/o positional features) | 98.62 ± 0.4 | 98.43 ± 0.4 | 98.53 ± 0.1 | 92.68 ± 0.5 | 83.06 ± 2.1 | 81.35 ± 1.6 |

PINT | 99.03 ± 0.1 | 98.78 ± 0.1 | 99.35 ± 0.1 | 96.01 ± 0.1 | 88.71 ± 1.3 | 88.06 ± 0.7 |

Inductive

<table>
<thead>
<tr>
<th>Method</th>
<th>Reddit</th>
<th>Wikipedia</th>
<th>Twitter</th>
<th>UCI</th>
<th>Enron</th>
<th>LastFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jodie</td>
<td>94.36 ± 1.1</td>
<td>93.11 ± 0.4</td>
<td>96.06 ± 0.1</td>
<td>75.26 ± 1.7</td>
<td>76.48 ± 3.5</td>
<td>80.32 ± 1.4</td>
</tr>
<tr>
<td>DyRep</td>
<td>95.68 ± 0.2</td>
<td>92.05 ± 0.3</td>
<td>96.33 ± 0.2</td>
<td>50.96 ± 1.9</td>
<td>66.97 ± 3.8</td>
<td>82.03 ± 0.6</td>
</tr>
<tr>
<td>TGAT</td>
<td>96.62 ± 0.3</td>
<td>93.99 ± 0.3</td>
<td>96.33 ± 0.1</td>
<td>70.54 ± 0.5</td>
<td>63.70 ± 0.2</td>
<td>56.76 ± 0.9</td>
</tr>
<tr>
<td>TGN-Att</td>
<td>97.55 ± 0.1</td>
<td>97.81 ± 0.1</td>
<td>95.76 ± 0.1</td>
<td>74.70 ± 0.9</td>
<td>78.96 ± 0.5</td>
<td>84.66 ± 0.1</td>
</tr>
<tr>
<td>CAW</td>
<td>97.81 ± 0.1</td>
<td>98.52 ± 0.1</td>
<td>98.54 ± 0.4</td>
<td>92.56 ± 0.1</td>
<td>91.74 ± 1.7</td>
<td>85.67 ± 0.5</td>
</tr>
</tbody>
</table>

PINT (w/o positional features) | 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 | 98.25 ± 0.4 | 98.38 ± 0.4 | 98.20 ± 0.3 | 93.97 ± 0.1 | 81.05 ± 2.4 | 91.76 ± 0.7 |

Time comparison. Figure 6 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. More details and results are provided in the Appendix.

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 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.
References


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 [41] combine time encoders [15] and self-attention [33]. In particular, the time encoder $\phi$ is given by

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

(S1)

where $\omega_i$'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 $C_v^{(t)}(t)$ whose $u$-th row is $c_v^{(t)}(u) = [h_v^{(t-1)}(t) \parallel \phi(t - tv) \parallel e_{uv}]$ for all $(u, e_{uv}, tv) \in N(v, t)$. Then, the output $\tilde{h}_v^{(t)}(t)$ of the AGG function is given by

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

(S2)

$$\tilde{h}_v^{(t)}(t) = \text{softmax}(qK\top)V$$  

(S3)

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

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.

Equation 1, Equation 2 and Equation 3 might make the reader believe that memory states are updated before prediction, i.e., computing node embeddings (message passing) using the features of the edge we want to predict. However, this would incur information leakage. On the other hand, not doing so prevents us from propagating gradients through the 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.

Revisiting memory with message aggregators. To speed up computations, MP-TGNs employ a form of batch learning where events/messages 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, message aggregators allow removing ambiguity in the way the memory of a node participating in multiple events (at the same timestamp) is updated — two events involving a given node $i$ at the same time could lead to different ways of updating the state of $i$. 

Anonymous Author(s)
Affiliation
email
Suppose the event $\gamma = (i, u, t)$ occurs. MP-TGNs proceed by computing a message function $\text{Msg}_e$ for each endpoint of $\gamma$, i.e.,

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

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

Now, suppose two events $(i, u, t)$ and $(i, v, t)$ happen. MP-TGNs aggregate the messages from these events using a function $\text{MsgAGG}$ to obtain a single message for $i$:

$$\bar{m}_i(t) = \text{MsgAGG}(m_{i,u}(t), m_{i,v}(t))$$

Rossi et al. [27] propose non-learnable message aggregators, such as the mean aggregator (average all messages for a given node), that we denote as $\text{MeanAGG}$ and adopt throughout our analysis. As an example, under events $(i, u, t)$ and $(i, v, t)$, the aggregated message for $i$ is 

$$\bar{m}_i(t) = 0.5([s_i(t^-), s_u(t^-), t - t_i, e_{iu}(t)] + [s_i(t^-), s_v(t^-), t - t_i, e_{iv}(t)])$$

The memory update of our query node $i$ is given by

$$s_i(t) = \text{Memory}(s_i(t^-), \bar{m}_i(t))$$

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 $(w_{i-1}, w_i, t_i) \in G(t)$ for all $i$. 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 $I_{CAW}(w_i; S_u, S_v)$ 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 an RNN:

$$\text{ENC}(W; S_u, S_v) = \text{RNN}(\langle f_1(I_{CAW}(w_i; S_u, S_v)) \rangle_{i=1}^L),$$

where $f_1$ is

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

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

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

where $\omega_i$’s are learned parameters.

To compute the embedding $h_{\gamma}$ 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 $h_{\gamma}$.
B Proofs

B.1 Further definitions and Lemmata

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

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

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 
\[ p = (u_0, t_1, u_1, t_2, u_2, \ldots) \text{ from } T_u(t) \iff p' = (f(u_0), t_1, f(u_1), t_2, f(u_2), \ldots) \text{ from } T_v(t), \]
with 
\[ s_{u_i} = s_{f(u_i)} \text{ and } e_{u_i u_{i+1}}(t_{i+1}) = e_{f(u_i)f(u_{i+1})}(t_{i+1}) \text{ and } k_{u_i} = k_{f(u_i)} \forall i \]
where \( f : V(T_u(t)) \to V(T_v(t)) \) is a bijection.

Assume that \( \tilde{T}_u(t) \not\cong \tilde{T}_v(t) \). Then, either there exists a path \( p_s = (u'_0, t'_1, u'_1, t'_2, \ldots) \) in \( \tilde{T}_u(t) \), such 
that \( t'_k < t'_k \) 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. \( \square \)

Lemma B2. Let \( G(t) \) and \( G'(t) \) be any two non-isomorphic temporal graphs. If an MP-TGN obtains 
different multisets of node embeddings for \( G(t) \) and \( G'(t) \), then, the temporal WL test decides \( G(t) \) 
and \( G'(t) \) 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 \( \Delta \) (temporal diameter) additional layers that does the 
same. Thus, it suffices to show that if the multisets of colors from temporal WL for \( G(t) \) and \( G'(t) \) 
after \( \ell \) iterations are identical, then the multisets of embeddings from the memoryless MP-TGN 
are also identical, i.e., if \( \{e^\ell(u)\}_{u \in V(G(t))} = \{e^\ell(u')\}_{u' \in V(G'(t))} \), then \( \{h^\ell(u)\}_{u \in V(G(t))} = 
\{h^\ell(u')\}_{u' \in V(G'(t))} \). To do so, we repurpose the proof of Lemma 2 in [42].

More broadly, we show that for any two nodes of a temporal graph \( G(t) \), if the temporal WL returns 
\( e^\ell(u) = e^\ell(v) \), we have that corresponding embeddings from MP-TGN without memory are identical 
\( h^\ell(u) = h^\ell(v) \). 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 \( \ell \). Thus, for any two nodes \( u, v, \) if 
\( e^{\ell+1}(u) = e^{\ell+1}(v) \), we have 
\[ (e^{\ell}(u), \{e^{\ell}(v), e_{iu(t')}, t') : (u, i, t') \in G(t)) = (e^{\ell}(v), \{e^{\ell}(v), e_{ju(t')}, t') : (v, j, t') \in G(t)) \]
and, by the induction hypothesis, we know 
\[ (h^\ell(u), \{h^\ell(v), e_{iu(t')}, t') : (u, i, t') \in G(t)) = 
(h^\ell(v), \{h^\ell(v), e_{ju(t')}, t') : (v, j, t') \in G(t)) \]

We also note that this last identity also implies 
\[ (h^\ell(u), \{h^\ell(u), t - t', e \} : (i, e, t') \in N(u, t)) = 
(h^\ell(v), \{h^\ell(v), t - t', e \} : (j, e, t') \in N(v, t)) \]
since there exists an event \( (u, i, t') \in G(t) \) with feature \( e_{u(t')} = e \) if there is an element \( (i, e, t') \in 
N(u, t). \)
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 \( \ell \) steps, the multisets of embeddings at layer \( \ell \) for these graphs are also identical.

**Lemma 3** (Lemma 5 in [42]). Assume \( \mathcal{X} \) is countable. There exists a function \( f : \mathcal{X} \rightarrow \mathbb{R}^n \) so that \( h(X) = \sum_{x \in \mathcal{X}} f(x) \) is unique for each multiset \( X \subset \mathcal{X} \) of bounded size. Moreover, any multiset function \( g \) can be decomposed as \( g(X) = \varphi(\sum_{x \in \mathcal{X}} f(x)) \) for some function \( \varphi \).

**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 \( \delta \) between consecutive elements \( G(t_i), G(t_{i+1}) \) of the CTDG, i.e., \( t_{i+1} - t_i = \delta \) for all \( i \geq 0 \).

Consider a DTDG given by the sequence \((G_1, G_2, \ldots)\). 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 \((u, \phi, i\delta)\) between \( u \) and a dummy node \( \phi \), with feature \( e_{u\phi}(i\delta) = x_u \in \mathcal{X}_i \). Let \( C(G_i) \) denote the set comprising these node-level events. Then, we can construct the CTDG \( G(t_i) = \bigcup_{j=1}^i S(G_j) \cup C(G_j) \) for \( i = 1, \ldots \).

Reconstructing the DTDG \((G_1, G_2, \ldots)\) is trivial. To build \( G_i \), it suffices to select all events at time \( i\delta \) in the CTDG. Events involving \( \phi \) 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 \((a_1, a_2, \ldots)\) is the ordered sequence formed from elements \( a_r \) of \( A \), i.e., \( a_1 < a_2 < \ldots < a_i < a_{i+1}, \ldots \).

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 \( T \subseteq \bar{T} \) such that \( \bar{T} \) is countable and uniformly spaced. Our idea is to construct a DTDG sequence with timestamps that coincide with the elements in \( T \). Then, since \( T \subseteq \bar{T} \), 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 \( \bar{T} \) 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 \( \{G(t_1), G(t_2), \ldots\} \) such that \( G(t_i) = \{(u, v, t) : t \in T \text{ and } t \leq t_i\} \) for \( t_i \in T \).

Also, let us denote \( H(t_i) = \{(u, v, t) \in G(t_i) : t = t_i\} \) the set of events at time \( t_i \in T \). We can build a corresponding DTDG \((G_1, G_2, \ldots)\) such that for all \( \tau_k \in T \) the k-th snapshot \( G_k \) is

\[
V(G_k) = \begin{cases} 
\{ (u, t_\tau, \tau_k) \in H(\tau_k) \}, & \text{if } \tau_k \in T; \\
\emptyset, & \text{otherwise.}
\end{cases}
\]

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

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

\[
\bar{I} = \{(i, k) \in \mathbb{N} \times \mathbb{N} : \tau_k = t_i \text{ for } t_i \in T \text{ and } \tau_k \in T\}.
\]

Note that we can treat \( \bar{I} \) as a map by defining \( \bar{I}(i) = k \) if and only if \((i, k) \in \bar{I} \). 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 \( G(\bar{t}_i) = \bigcup_{j : j \leq \bar{I}(i)} S(G_j) \) for \( t_i \in T \).
B.3 Proof of Lemma 1

Proof. Here we show that if two nodes \( u \) and \( v \) have isomorphic \((L,\ell)\)-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 \( \ell \) 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 \( \ell \). 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 (with annotated edges) \( f(i) \) in \( T_{v,1} \) with \( s_i = f(i) \). Recall that the \( \ell \)-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 \( \ell \)-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 \( v \).

\( \square \)

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 triplets 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 observed 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 \( \ell \) layers. We want to prove that, under the injectivity assumption, if \( T_{u,\ell}(t) \not\cong T_{v,\ell}(t) \), then \( h_u^{(\ell)}(t) \not\equiv h_v^{(\ell)}(t) \) 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 \( \ell \). Also, keep in mind that \( \varphi_\ell = \text{UPDATE}^{(\ell)} \circ \text{AGG}^{(\ell)} \) is injective for any \( \ell \).

[Base case] For \( \ell = 1 \), if \( T_{u,1} \not\cong T_{v,1} \) then the root node states are different (i.e., \( s_u \neq s_v \)) or the multiset of states/edge features/ timestamps triplets from \( u \) and \( v \)’s children are different. In both cases, the inputs of \( \varphi_\ell \) are different and it therefore outputs different embeddings for \( u \) and \( v \).

[Induction step] The inductive hypothesis is \( T_{u,\ell-1} \not\cong T_{v,\ell-1} \Rightarrow h_u^{(\ell-1)} \not\equiv h_v^{(\ell-1)} \) for any pair of nodes \( u \) and \( v \). If \( T_{u,\ell} \not\cong T_{v,\ell} \), 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, \( \varphi_\ell \) 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 \( \varphi \) will output different embeddings for \( u \) and \( v \).

\( \square \)

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 \( L < \Delta \): \( \mathcal{Q}^{[M]}_L \) is strictly stronger than \( \mathcal{Q}_L \).
We now show that the memory for a node $n$-layer MP-TGNs with memory comprises the family of $n$-layer MP-TGNs without memory (we can assume identity memory). Therefore, $Q^{[M]}_L$ is at least as powerful as $Q_L$. To show that $Q^{[M]}_L$ is strictly stronger (more powerful) than $Q_L$, 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. We assume that $t_2 > t_1$.](image)

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

It suffices to show if $Q_{L+\Delta}$ cannot distinguish a pair of nodes $u$ and $v$, $Q^{[M]}_L$ cannot distinguish them too. Let $T_u^{[M]}(t)$ and $T_u(t)$ denote the $L$-depth TCTs of $u$ with and without memory respectively.

Using [Lemma 1](#), this is equivalent to showing that $T_u(t) \equiv T_u^{[M]}(t) \Rightarrow T_u^L(t) \equiv T_u^{[L]}(t)$, 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 $V_u^n$ and a set of events annotated with their respective timestamps and features $B_u$. If at time $t_n$ no event involves a node $z$, we set $B_u = B_u^{n-1}$ and $V_u^n = V_u^{n-1}$. We also initialize $B_u^0 = \emptyset$ and $V_u^0 = \{u\}$ for all nodes $u$. We proceed with a proof by induction on the number of observed timestamps $n$.

**Base case** Let $I_1(u) = \{v : (u, v, t_1) \in G(t_1)\}$ be the set of nodes interacting with $u$ at time $t_1$, where $G(t_1)$ is the temporal graph right after $t_1$. Similarly, let $f_1(u) = \{(u, v, t_1) \in G(t_1)\}$ be the set of events involving $u$ at time $t_1$. Recall that before $t_1$, all memory states equal initial node features (i.e., $s_u(t_1) = s_u(0)$). Then, the updated memory (see [Equation 3](#)) for $u$ depends on $V_u^1 = V_u^0 \cup \{v \in I_1(u)\}$, $V_u^0 = V_u^0 \cup f_1(u)$.

**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., $w \in I_n(u)$, depends on $V_u^{n-1}$ and $B_u^{n-1}$, and the memory of $w$ so far depends on $V_u^{n-1}$ and $B_u^{n-1}$. Then, the updated memory for $w$ depends on $V_w = V_u^{n-1} \cup w \in I_n(u)$, $V_u^{n-1}$, and $B_u = B_u^{n-1} \cup f_n(u) \cup w \in I_n(u) B_w^{n-1}$.

**Step 2:** $(z, w, t_{zw}) \in B_u^n$ if and only if there is a path $(u_k, t_k = t_{zw}, u_{k+1})$ in $T_u(t_n)$ — the monotonous TCT of $u$ (see [Definition 2](#)) after processing events with timestamp $\leq t_n$ — with either $z u_k = z, z u_{k+1} = w$ or $z u_k = w, z u_{k+1} = z$.

**Forward direction** An event $(z, w, t_{zw})$ with $t_{zw} \leq t_n$ will be in $B_u^n$ only if $z = u$ or $w = u$, or if there is a subset of events $\{(u, z u_1, t_1), (z u_2, t_2), \ldots, (z u_k, z u_{k+1}, t_{zw})\}$ with $z u_k = z$ and $z u_{k+1} = w$ such that $t_n \geq t_1 > \cdots > t_{zw}$. In either case, this will lead to a root-to-leaf path in $T_u(t_n)$ 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 $T_u(t_n)$ with $z u_k = z$ and $z u_{k+1} = w$ such that $(z, w, t_{zw}) \notin B_u^n$. Since we can obtain $p$ from $T_u(t_n)$, we know that the sequence of events $r = (u, z u_1, t_1), \ldots, (z u_{k-2}, z u_{k-1} = z, t_{k-1}), (z, w, t_k = t_{zw})$ happened and that $t_i > t_{i+1} \forall i$. However, since $(z, w, t_{zw}) \notin B_u^n$, 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 $V^u_n$ are simply the nodes that have an endpoint in the events $B^u_n$, 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 $(V^u_n, B^u_n)$ to $\tilde{T}_u(t^+_n)$.

First, we note that $(V^u_n, B^u_n)$ depends on a subset of all events, which we represent as $G' \subseteq G(t^+_n)$.

Since $B^u_n$ contains all events in $G'$ and $(V^u_n, B^u_n)$ can be uniquely constructed from $G'$, there is a bijection $g$ that maps from $G'$ to $(V^u_n, B^u_n)$.

Similarly, $\tilde{T}_u(t^+_n)$ also depends on a subset of events which we denote by $G'' \subseteq G(t^+_n)$. We note that the unique events in $\tilde{T}_u(t^+_n)$ correspond to $G''$, and we can uniquely build the tree $\tilde{T}_u(t^+_n)$ from $G''$.

This implies that there is a bijection $h$ that maps from $G''$ to $\tilde{T}_u(t^+_n)$.

Previously, we have shown that all events in $B^u_n$ 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., $G' = G'' = G_S$.

Since there is a bijection $g$ between $G_S$ and $(V^u_n, B^u_n)$, and a bijection $h$ between $G_S$ and $\tilde{T}_u(t^+_n)$, there exists a bijection $f$ between $(V^u_n, B^u_n)$ and $\tilde{T}_u(t^+_n)$.

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

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 $w$.

Since the depth of $T_{w}'$ is at least $\Delta$, we know that the $T_{w}' \cong T_{w}$ — i.e., imposing time-constraints to $T_{w}'$ results in the monotone TCT of node $w$. Also, because the memory of $w$ depends on $T_{w}$, $T_{w}'$ comprises the information used to compute the memory state of $w$. 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 $T_{u,L}^M \cong T_{v,L}^M$. \[ \square \]

**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 $h_{u,1}^{\ell}(t) \neq h_{v,1}^{\ell}(t)$, 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.](image)

[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 $\ell$ layers, we have that $h_{u,\ell}^{\ell}(t) = h_{v,\ell}^{\ell}(t)$.

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


We can then use this to show that $h_w^{(t)}(t) = h_z^{(t)}(t)$ for any TGAT with arbitrary number of layers $\ell$.

**Step 2:** There is no TGAT such that $h_w^{(t)}(t) \neq h_z^{(t)}(t)$.

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

Note that $\mathcal{N}(u, t) = \{(z, e, t_1), (w, e, t_4)\}$ and $\mathcal{N}(v, t) = \{(w, e, t_1)\}$, where $e$ denotes an edge feature vector. Also, we have previously shown that $h_w^{(t-1)}(t) = h_z^{(t-1)}(t)$.

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

Since all events have the common edge features $e$, the matrices $C_u^{(t)}$ and $C_v^{(t)}$ share the same vector in their rows. The single-row matrix $C_u^{(t)}$ is given by $C_u^{(t)} = [h_w^{(t-1)}(t)(0)$, while the two-row matrix $C_v^{(t)} = [h_v^{(t-1)}(t)(0), h_v^{(t-1)}(t)(1)]$, where $h_v^{(t-1)}(t) = h_v^{(t-1)}(t)$. We can express $C_u^{(t)} = [1, 1]^T r$ and $C_v^{(t)} = r$, where $r$ denotes the row vector $r = [h_v^{(t-1)}(t)](0)$.

Using the key and value matrices of node $v$, i.e., $K_v = C_v^{(t)} W_K^{(t)}$ and $V_v = C_v^{(t)} W_V^{(t)}$, we have that
\[
\tilde{h}_v^{(t)}(t) = \text{softmax}(q_v K_v^T) V_v = \text{softmax}(q_v K_v^T) r W_v^{(t)} = r W_v^{(t)} = \text{softmax}(q_v K_v^T [1, 1]^T r W_v^{(t)} = h_v^{(t)}(t)
\]

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

**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 $s_u(t) = s_v(t) = s_u(t)$, $s_v(t) = s_v(t)$, and $s_v(t) = s_v(t)$. If that is the case, then these node states could be treated as node features in an equivalent TGAT model of our example in Figure S2, proving that there is no TGN-Att such that $h_u^{(t)}(t) \neq h_v^{(t)}(t)$. In the following, we consider TGN-Att with average message aggregators (see Appendix A).

We begin by showing that $s_u(t) = s_u(t) = s_v(t)$ after memory updates. We note that the message node receives is $[e \parallel t_1 \parallel s_w(t_2)]$. The message node receives is $\text{MEANAGG}([e \parallel t_1 \parallel s_w(t_2)], [e \parallel t_1 \parallel s_z(t_2)], [e \parallel t_1 \parallel s_z(t_2)])$, but since $s_u(t_2) = s_z(t_2)$, both messages are the same, and the average aggregator outputs $[e \parallel t_1 \parallel s_z(t_2)]$. Finally, the message that node $v$ receives is $[e \parallel t_1 \parallel s_w(t_2)] = [e \parallel t_1 \parallel s_z(t_2)]$. Since all three nodes receive the same message and have the same initial features, their updated memory states are identical.

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

We can then use this to show that $s_{z}(t) = s_{w}(t)$ for $t \geq t_2$. Note that at time $t_2$, the message that nodes $z$ and $w$ receive are $[e \parallel t_2 - t_1 \parallel s_w(t_2)]$ and $[e \parallel t_2 - t_1 \parallel s_z(t_2)]$, respectively. Also, note...
that \( s_b(t_2^-) = s_c(t_2^-) = s_v(0) = s_c(0) \) as the states of \( b \) and \( c \) are only updated at \( t_2 \). Because
the received messages and the previous states (before \( t_2 \)) of \( z \) and \( w \) are identical, we have that
\( s_z(t) = s_w(t) \) for \( t \geq t_2 \).

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-TGNs and CAWs

![Diagram](image1.png)

**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. [37] 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 \((u, v, t_3)\):

\[
\begin{align*}
\{[1,0,0],[0,1,0]\} & \xrightarrow{I_{CAW}(u;S_u,S_e)} \{[0,1,0],[2,0,0]\} \\
\{[0,1,0],[2,0,0]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[1,0,0],[0,1,0]\} \\
\{[0,1,0],[2,0,0]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[0,0,0],[0,1,0]\} \\
\{[0,0,0],[0,1,0]\} & \xrightarrow{I_{CAW}(z;S_z,S_e)} \{[0,0,0],[0,0,1]\}
\end{align*}
\]

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

\[
\begin{align*}
\{[1,0,0],[0,0,1]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[0,1,0],[0,1,0]\} \\
\{[0,0,0],[2,0,0]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[0,0,0],[0,1,0]\} \\
\{[0,0,0],[2,0,0]\} & \xrightarrow{I_{CAW}(z;S_z,S_e)} \{[1,0,0],[0,0,1]\} \\
\end{align*}
\]

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 \((u, z, t_4)\):

\[
\begin{align*}
\{[1,0,0],[0,0,0]\} & \xrightarrow{I_{CAW}(u;S_u,S_e)} \{[0,1,0],[0,0,0]\} \\
\{[0,0,0],[1,0,0]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[0,0,0],[0,1,0]\} \\
\{[0,0,0],[1,0,0]\} & \xrightarrow{I_{CAW}(v;S_v,S_e)} \{[0,0,0],[0,0,1]\}
\end{align*}
\]
and for \((u', z, t_4)\):

\[
\begin{align*}
\{[1, 0], [0, 0]\} & \xrightarrow{L_{CAW}(u'; S_{u'}, S_z)} \{[0, 1], [0, 0]\} \\
\{[0, 0], [1, 0]\} & \xrightarrow{L_{CAW}(v'; S_{u'}, S_z)} \{[0, 0], [0, 1]\}
\end{align*}
\]

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 \(G(t)\) and \(G'(t)\), then the temporal WL decides these graphs are not isomorphic. Then, we prove that, if the temporal WL decides \(G(t)\) and \(G'(t)\) 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 [42]. 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 \(\phi\) 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 \(\ell\). 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:

\[
h_{u}^{(\ell)}(t) = \text{UPDATE}^{(\ell)}\left(h_{u}^{(\ell-1)}(t), \text{AGG}^{(\ell)}(\{(h_{u}^{(\ell-1)}(t), t - t', e) \mid (u, e, t') \in \mathcal{N}(v, t)\})\right)
\]

can be rewritten using \(\varphi\) as a function of node colors:

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

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

\[
\begin{align*}
h_{u}^{(\ell)}(t) &= \psi\left(c^{\ell-1}(v), c^{\ell-1}(u), t', e \mid (u, e, t') \in \mathcal{N}(v, t)\right) \\
&= \psi\left(c^{\ell-1}(v), (c^{\ell-1}(u), t', e_{uv}(t')) \mid (v, u, t') \in \mathcal{G}(t)\right)
\end{align*}
\]

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:

\[
\begin{align*}
h_{u}^{(\ell)}(t) &= \psi \circ \text{HASH}^{-1} \circ \text{HASH}\left(c^{\ell-1}(v), (c^{\ell-1}(u), t', e_{uv}(t')) \mid (u, v, t') \in \mathcal{G}(t)\right) \\
&= \psi \circ \text{HASH}^{-1}(c^{(\ell)}(v))
\end{align*}
\]

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. □
We now show that the $k$-th component of the relative positional features $r_u^{(t)}$, $r_v^{(t)}$ 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, $r_u^{(0)}$ is the zero vector if $u \neq v$, indicating that node $u$ does not belong to the TCT of $v$. If $u = v$, then $r_u^{(0)} = [1, 0, \ldots, 0]$ corresponds to count 1 for the root of the TCT of $u$. Thus, for $t = 0$, the proposition holds.

**B.10 Proof of Lemma 2**

We now show that the $k$-th component of the relative positional features $r_u^{(t)}$, $r_v^{(t)}$ 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, $r_u^{(0)}$ is the zero vector if $u \neq v$, indicating that node $u$ does not belong to the TCT of $v$. If $u = v$, then $r_u^{(0)} = [1, 0, \ldots, 0]$ corresponds to count 1 for the root of the TCT of $u$. Thus, for $t = 0$, the proposition holds.
We note that the updates of the positional features of
all nodes and any time instant before \( t \). We will
show that after the event \( \gamma = (u, v, t) \) at time \( t \), the
proposition remains true.

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

Without loss of generality, let us now consider the
impact of \( \gamma \) on the monotone TCT of \( v \). Figure S6
shows how the TCT of \( v \) changes after \( \gamma \), i.e., how it
goes from \( \mathcal{T}_v(t^-) \) to \( \mathcal{T}_v(t) \). 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 node \( i \)
in \( \mathcal{T}_u(t^-) \) regarding how many times it appears in
\( \mathcal{T}(t) \). We do so by adding the counts in \( \mathcal{T}_u(t^-) \) (i.e.,
\( r_{i \rightarrow u}^{(t^-)} \)) to \( \mathcal{T}_v(t) \) (i.e., \( r_{i \rightarrow v}^{(t^-)} \)), accounting for the 1-
layer mismatch, since \( \mathcal{T}_u(t^-) \) 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 \( \mathcal{T}_u(t^-) \), i.e.,
\[
\begin{align*}
r_{i \rightarrow v}^{(t)} &= Pr_{i \rightarrow u}^{(t^-)} + r_{i \rightarrow v}^{(t^-)} \quad \forall i \in \mathcal{V}_u^{(t^-)},
\end{align*}
\]
where \( \mathcal{V}_u^{(t^-)} \) comprises the nodes of the original graph that belong to \( \mathcal{T}_u(t^-) \).

Similarly, the event \( \gamma \) also affects the counts of nodes in the TCT of \( v \) w.r.t. the TCT of \( u \). To account
for that change, we follow the same procedure and update \( r_{j \rightarrow u}^{(t)} = Pr_{j \rightarrow v}^{(t^-)} + r_{j \rightarrow u}^{(t^-)}, \forall j \in \mathcal{V}_v^{(t^-)}. \)

**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
\[
\begin{align*}
r_{i \rightarrow v}^{(t)} &= P \sum_{j=1}^{J} r_{i \rightarrow u_j}^{(t^-)} + r_{i \rightarrow v}^{(t^-)} \quad \forall i \in \bigcup_{j=1}^{J} \mathcal{V}_{u_j}^{(t^-)} \\
\mathcal{V}_v^{(t)} &= \mathcal{V}_v^{(t^-)} \bigcup_{j=1}^{J} \mathcal{V}_{u_j}^{(t^-)}.
\end{align*}
\]
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
\[
z(n_1, n_2) = \frac{\left( n_1 + n_2 \right) (n_1 + n_2 + 1)}{2} + n_2, \text{ provides a proof for that.}
\]
Here, we consider multiset \( M = \{(x_i, e_i, t_i)\} \) whose tuples take values on the Cartesian product of
the countable sets \( X, E, \) and \( T \) — 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 \( X \) and \( E \) are countable,
there exists an enumeration function $\psi : X \times E \rightarrow \mathbb{N}$ for all $M$. Without loss of generality, we assume $T = \{1, 2, t_{\text{max}}\}$. We want to show that there exists a function of the form $\sum_i 10^{-k\psi(x_i, e_i)} \alpha^{-\beta e_i}$ 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 $t_{\text{max}}$ 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 $(x_i, e_i, t_j = j)$ 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 $t_{\text{max}}$ slots of $\beta$ decimals, we set $k = \beta(t_{\text{max}} + 1)$. Therefore, the function

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

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

\section*{B.12 Proof of Proposition 9: Expressiveness of PINT: link prediction}

\textbf{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.

\textbf{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.

\textbf{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 $G(t)$. 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 $\{\text{ENC}(W; S_u, S_v)\}_{W \in \{S_u \cup S_v\}} = \{\text{ENC}(W'; S_{u'}, S_{v'})\}_{W' \in \{S_{u'} \cup S_{v'}\}}$, where 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: $\bar{T}_u(t) \cong \bar{T}_{u'}(t)$, and $\bar{T}_v(t) \cong \bar{T}_{v'}(t)$ with associated bijections $f_1 : V(\bar{T}_u(t)) \rightarrow V(\bar{T}_{u'}(t))$ and $f_2 : V(\bar{T}_v(t)) \rightarrow V(\bar{T}_{v'}(t))$.

We can construct a tree $T_{uv}$ by attaching $\bar{T}_u(t)$ and $\bar{T}_v(t)$ 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 $\bar{T}_{u'}(t)$ and $\bar{T}_{v'}(t)$ 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 $f : V(T_{uv}) \rightarrow V(T_{u'v'})$ 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 $V(\bar{T}_u(t))$ to a node in $V(\bar{T}_{u'}(t))$, for instance. We have that

$$\sum_{i \in V(T_{uv}) \setminus \{uv\}} [r_{\phi i u}^{(t)} \| r_{\phi i u'}^{(t)}] = [r_{\phi f i u}^{(t)} \| r_{\phi f i u'}^{(t)}]$$

Note that we use the function $\phi$ (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 $\text{ENC}(W; S_u, S_v) = \text{ENC}(W'; S_{u'}, S_{v'})$, it suffices to show that the anonymized walks are equal. Thus, we turn our problem into showing that for any walk $W = (w_0, t_0, w_1, t_1, \ldots) \in S_u \cup S_v$, there exists a corresponding one.
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 $\zeta$. Using this equivalence, it suffices to show that

$$g(\bar{z}; S_u) = g(\bar{z}; f(i); S'_v) \quad \text{and} \quad g(\bar{z}; S_v) = g(\bar{z}; f(i); S'_u) \quad \forall i \in V(T_{uv}) \setminus \{uv\}$$

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

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

$$g(\bar{z}; S_u)[\ell] = \sum_{\psi \in \Psi} \mathcal{O}(\psi; \bar{T}_u(t)).$$

Since we assume $g(\bar{z}; S_u)[\ell] \neq g(\bar{z}; f(i); S'_u')[\ell]$, then it holds that

$$g(\bar{z}; S_u)[\ell] \neq g(\bar{z}; f(i); S'_u')[\ell] \Rightarrow \sum_{\psi \in \Psi} \mathcal{O}(\psi; \bar{T}_u(t)) \neq \sum_{\psi \in \Psi} \mathcal{O}(f(\psi); \bar{T}'_u(t)) \quad (S5)$$

Note that the subtree of $\bar{T}_u$ rooted at $\psi$ should be isomorphic to the subtree of $\bar{T}_u'$ 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 $\mathcal{O}(\psi; \bar{T}_u) \neq \mathcal{O}(f(\psi); \bar{T}'_u)$, reaching a contradiction. The same argument can be applied to $v$ and $v'$ to prove that $g(\bar{z}; S_v) = g(\bar{z}; f(i); S'_v)$.

### 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. [47] 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. [51] 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 [50, 59, 60], 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 [46, 56]. However, these models are referred to be hard to converge and to obtain noisy predictions [60].

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 [60], 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 [13], walk-aggregating methods [52], and combinations of sequence models with GNNs [54, 57]. For instance, Seo et al. [57] used a spectral graph convolutional network [48] to encode graph snapshots followed by a graph-level LSTM [13].
Manessi et al. [53] followed a similar approach but employed a node-level LSTM, with parameters shared across the nodes. Sankar et al. [55] proposed a fully attentive model based on graph attention networks [44]. Pareja et al. [54] applied a recurrent neural net to dynamically update the parameters of a GCN. Gao and Ribeiro [49] 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. [58].

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 [37]. 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 [61] consists of a memory-based TGN that uses attention mechanism to update memory states using multi-hop temporal neighborhood information.
D Datasets and implementation details

D.1 Datasets
In our empirical evaluation, we have considered six datasets for dynamic link prediction: Reddit, Wikipedia, UCI, LastFM, Enron, 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 [41].

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

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 172, 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 \(\alpha \in \{0.01, 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, 10.0, 100.0\} \times 10^{-4}\); ii) number of walks \(M \in \{1, 2, 3, 4, 5\}\); and iii) walk length \(L \in \{32, 64, 128\}\). The best combination of hyperparameters is shown in

http://snap.stanford.edu/jodie/reddit.csv
http://snap.stanford.edu/jodie/wikipedia.csv
http://konect.cc/networks/opsahl-ucforum/
http://snap.stanford.edu/jodie/lastfm.csv
https://www.cs.cmu.edu/~./enron/
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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Time decay $\alpha$</th>
<th>#Walks</th>
<th>Walk length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>$10^{-8}$</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>$4 \times 10^{-6}$</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>Twitter</td>
<td>$10^{-6}$</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>UCI</td>
<td>$10^{-5}$</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>Enron</td>
<td>$10^{-6}$</td>
<td>64</td>
<td>5</td>
</tr>
<tr>
<td>LastFM</td>
<td>$10^{-6}$</td>
<td>64</td>
<td>2</td>
</tr>
</tbody>
</table>

PINT. We use $\alpha = 2$ (in the exponential aggregation function), and experiment with learned and fixed $\beta$. We apply a relu function to avoid negative values of $\beta$, 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 $\beta \in \{0.001, 0.0001, 0.00001\}$ and always apply memory as in the original implementation of TGN-Att. We consider number message passing layers $\ell$ 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 $\beta$ led to better results.

Table S3: Optimal hyperparameters for PINT.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\beta$</th>
<th>#Neighbors</th>
<th>#Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>$10^{-5}$</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>$10^{-4}$</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Twitter</td>
<td>$10^{-5}$</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>UCI</td>
<td>$10^{-5}$</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Enron</td>
<td>$10^{-5}$</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>LastFM</td>
<td>$10^{-4}$</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Hardware. For all experiments, we use Tesla V100 GPU cards and consider a memory budget of 32GB of RAM. We omit further details to preserve anonymity during the review process.

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 6, 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.

Dimensionality of positional features. We now assess the performance of PINT as a function of the dimensionality \( d \) of the relative positional features. Figure S8 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 hyper-parameters found with \( d = 4 \) (the configuration shown in the main paper). Increasing the dimensionality of the positional features leads to performance gains on both datasets. Notably, we obtain a significant boost for Enron with \( d = 10 \): 92.696 ± 0.095% AP in the transductive setting and 88.342 ± 0.294% in the inductive case. Thus, PINT becomes the best-performing model on Enron (transductive). On the UCI dataset, higher \( d \) results in more significant gains in the inductive case. On UCI, with \( d = 20 \), we obtain 96.36 ± 0.067 (transductive) and 94.77 ± 0.12 (inductive).

Figure S8: PINT: AP (mean and std) as a function of the dimensionality of the positional features.

Incorporating relative positional encoding into MP-TGNs. We can also use PINT’s relative positional encoding to boost the performance of MP-TGNs. As a proof of concept, we have implemented TGN-Att with PE. We use the same model selection procedure as TGN-Att in Table 1, and incorporate \( d = 4 \)-dimensional positional features.

Table S4 shows the performance of TGN-Att+PE on three unattributed link prediction benchmarks (UCI, Enron, and LastFM). Notably, TGN-Att receives a significant boost from our PE. However, PINT still beats TGN-Att+PE on 5 out of 6 cases. The values for TGN-Att+PE reflect the outcome of 5 repetitions.
Table S4: Link prediction: TGN-Att + relative positional encoding (AP).

<table>
<thead>
<tr>
<th></th>
<th>Transductive</th>
<th>Inductive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UCI</td>
<td>Enron</td>
</tr>
<tr>
<td>TGN-Att</td>
<td>80.40 ± 1.4</td>
<td>79.91 ± 1.3</td>
</tr>
<tr>
<td>TGN-Att + PE</td>
<td>95.64 ± 0.6</td>
<td>85.04 ± 2.54</td>
</tr>
<tr>
<td>PINT</td>
<td>96.01 ± 0.1</td>
<td>88.71 ± 1.3</td>
</tr>
</tbody>
</table>

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 S5 shows that PINT ranks first on Reddit and second on Wikipedia. The values for PINT reflect the outcome of 5 repetitions.

Table S5: Results for node classification (AUC).

<table>
<thead>
<tr>
<th></th>
<th>Wikipedia</th>
<th>Reddit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTDNE</td>
<td>75.89 ± 0.5</td>
<td>59.43 ± 0.6</td>
</tr>
<tr>
<td>JODIE</td>
<td>84.84 ± 1.2</td>
<td>61.83 ± 2.7</td>
</tr>
<tr>
<td>TGAT</td>
<td>83.69 ± 0.7</td>
<td>65.56 ± 0.7</td>
</tr>
<tr>
<td>DyRep</td>
<td>84.59 ± 2.2</td>
<td>62.91 ± 2.4</td>
</tr>
<tr>
<td>TGN-Att</td>
<td>87.81 ± 0.3</td>
<td>67.06 ± 0.9</td>
</tr>
<tr>
<td>PINT</td>
<td>87.59 ± 0.6</td>
<td>67.31 ± 0.2</td>
</tr>
</tbody>
</table>

G Societal and broader impact

Temporal graph networks have shown remarkable performance in relevant domains such as social networks, e-commerce, and education. 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 in a series of 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. Furthermore, we do not foresee any negative societal impact stemming directly from our developments.

Supplementary References


