Graph State Networks (GSNs): Persistent Nodewise Selective State Space Models

Anonymous authors
Paper under double-blind review

Abstract

Temporal graphs are often observed as streams of timestamped interactions, where accurate prediction requires retaining and selectively using historical information nodes. Existing temporal graph models either (i) recompute representations from a sliding neighborhood/history at query time, or (ii) maintain a memory module but offer limited control and limited theory for what is retained over long horizons. We propose **Graph State Networks (GSNs)**, a bucketed temporal-graph framework that maintains a persistent hidden state per node and updates it online using a content- and time-dependent selective state space update. Concretely, GSNs store node states in an explicit id-indexed state table and for each bucket, *read* the current state, *update* it with a time-aware Mamba-like mechanism, and *commit* the state back via an exponential moving average controlled by commit-rate α. This commit mechanism provides an explicit "retention dial" and enables a clean analysis of forgetting. We develop a **capacity/recall theory** for persistent node memory and show that, under an affine approximation of *blank-bucket* dynamics, the influence of a single past event decays geometrically at a rate governed by α and the induced linearized update. Empirically, GSNs are competitive on standard dynamic link prediction benchmarks. We validate the theory with controlled synthetic write–wait–read probes: measured influence is close to exponential in delay, and fitting short-delay dynamics predicts long horizon recall across commit rates.

1 Introduction

Many real-world relational systems, e.g., communication networks, collaboration graphs, recommender systems, transaction ledgers, are naturally modeled as *temporal graphs*, where edges arrive as timestamped events. A core problem is **dynamic link prediction**: given past interactions, score which edges are likely to occur next. The challenge is not just modeling graph structure, but doing so *online* while handling long and irregular histories.

A common design choice is *where the history is stored.* In one family of methods, node representations are computed (or recomputed) from a window of recent interactions using message passing and/or attention over sampled temporal neighborhoods. This can be effective, but it inherits familiar limitations of deep message passing–difficulty propagating long-range signals due to bottlenecks ("over-squashing") and representation collapse in deep stacks ("over-smoothing"), while neighborhood retrieval can become expensive or brittle under distribution shift. A second family uses more global attention-style mechanisms (temporal Transformers and graph Transformers), which can capture long-range dependencies but often lead to quadratic costs in sequence length or require substantial positional/structural encoding machinery.

A different viewpoint is to treat each node as an **online state machine**: instead of rebuilding a node embedding from scratch at query time, a persistent state is updated as events arrive. This view is present in several temporal-graph architectures (e.g., memory module in Temporal Graph Networks), but two gaps remain in practice:

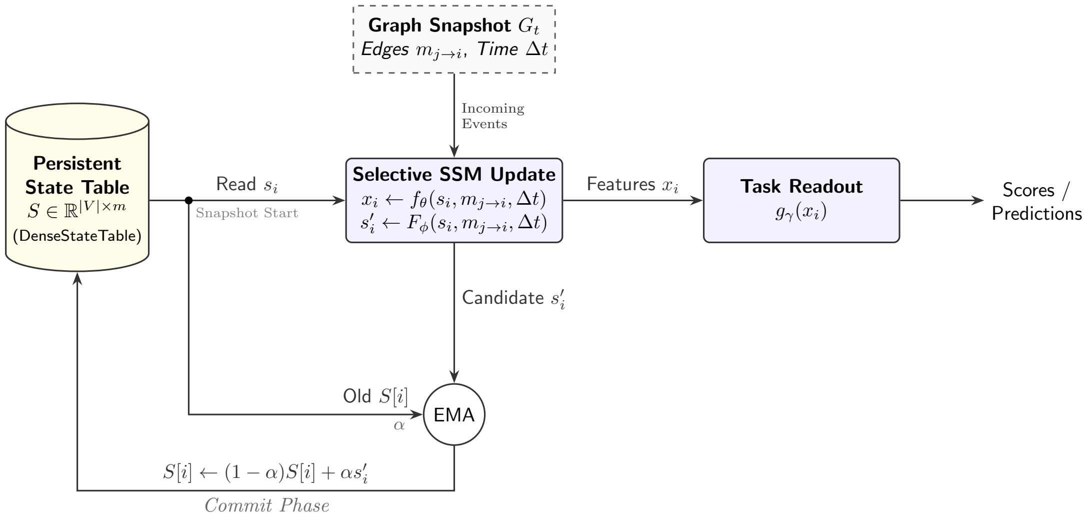


Figure 1: GSN Layer (*read–update–commit*) for a temporal graph snapshot. Here V is the set of nodes and m is the state dimension. Each node i has a persistent state $s_i \in \mathbb{R}^m$ stored in an ID-indexed persistent state table $S \in \mathbb{R}^{|V|\times m}$ that carries across buckets (hence persistent). At snapshot start, the layer reads s_i, then uses incoming messages/events $m_{j\to i}$ and elapsed time Δt in a time-aware selective state-space update to produce features x_i for the task head $g_\gamma(x_i)$ and a proposed new state s_i'. After scoring, the new states are committed to the table via an Exponential Moving Average (EMA) with commit rate α using (1).

1. **Update mechanism:** Many memory updates are hand-designed (GRU-style) or tied closely to specific neighborhood aggregation schemes. Recent progress in **selective state-space models (SSMs)**, notably Mamba (Gu & Dao, 2023), suggests that content-dependent, gated state updates can provide strong long-range modeling with favorable scaling.

2. **Controllability and interpretability of forgetting:** Even when a model maintains memory, it is often unclear *what* information persists and how *quickly* it decays as time passes without a new signal. For deployment, this matters: some applications need long retention; others need rapid adaptation.

This work proposes **Graph State Networks (GSNs)** as a step toward addressing both gaps. GSNs make a simple but strict architectural commitment: every node i owns a persistent hidden state $s_i \in \mathbb{R}^m$ that is **not reinitialized per batch or per forward call**, but evolves over the temporal stream. Concretely, we maintain an explicit state table S indexed by global node IDs. For each temporal bucket/snapshot, we:

- **Read** the current persistent state for nodes present in the bucket;
- **Update** it using a time-aware Mamba-like block;
- **Write/commit** the new state back to the table using an Exponential Moving Average (EMA) with commit rate $\alpha \in (0, 1]$:

$$S[\text{ids}(i)] \leftarrow (1-\alpha)S[\text{ids}(i)] + \alpha s_i, \tag{1}$$

  where s_i is the calculated *new state* of the node i, and ids(i) is the ID of node i as a tuple of (a, b) indexing the row and column of S.

This design cleanly separates (i) *how* information is integrated into a node's memory (the selective SSM update) from (ii) *how aggressively* that memory is refreshed (the commit rule). It also matches our implementation setting: we intentionally use bucketed processing where node states persist across buckets and are advanced via explicit commit steps at bucket boundaries, rather than strict event-by-event constant-time streaming updates (though it must be noted that streaming updates can be achieved by using a bucket size = 1).

Beyond architecture, our second goal is to make node retention mechanism measurable. We develop a capacity/recall theory for persistent node states: because a fixed-dimensional state must compress history, it cannot reproduce arbitrary query-time retrieval, but its retention can be characterized in terms of what remains recoverable after many events. A key object is the model's behavior during **blank buckets**, i.e., periods with no new informative input, where the system's intrinsic dynamics determine whether old information decays, persists, or is overwritten by drift. Under a mild affine approximation of the blank update, we obtain closed-form convergence and **rigorous forgetting bounds**: the influence of a single event decays geometrically with a contraction factor determined by α and the induced linear map.

Contributions. In summary, this work:

- introduces **Graph State Networks**, a persistent node-state architecture updated by a time-aware selective SSM and committed via an EMA rule with rate α;
- develops **blank-bucket forgetting theory** that yields closed-form convergence behavior and quantitative bounds on single-event influence decay;
- proposes and evaluates **synthetic write–wait–read probes** that diagnose what the node state retains and show that short-horizon measurements extrapolate to long-horizon predictions;
- demonstrates that **state persistence is compatible with strong predictive performance** on standard temporal link prediction benchmarks, while offering an explicit retention dial.

2 Related Work

A large body of literature studies representation learning on temporal interaction graphs, e.g., (Rossi et al., 2020; Wang et al., 2021b; Xu et al., 2020). Early and influential approaches include **JODIE** (Kumar et al., 2019b), which learns dynamic embedding trajectories for interacting entities, and **DyRep** (Trivedi et al., 2019), which models evolving node representations driven by communication/association processes in continuous time. Attention-based temporal neighborhood aggregation was popularized by **TGAT** (Xu et al., 2020), which introduces functional time encodings and temporal self-attention for inductive temporal graphs. **Temporal Graph Networks (TGN)** (Rossi et al., 2020) unify several lines of work by combining a memory module with message-passing operators over temporal neighborhoods.

Beyond neighbor aggregation, motif/walk-based and Transformer-style models have been explored. **CAWN** (Wang et al., 2021a) uses causal anonymous walks to capture temporal motifs in an inductive way. **TCL** (Wang et al., 2021b) adapts Transformer-style modeling to temporal graphs with contrastive learning objectives. More recently, a line of work argues that carefully designed but simpler architectures can be highly competitive: **GraphMixer** (Cong et al., 2023) proposes an MLP-centric design for temporal link prediction, while **DyGFormer** (Yu et al., 2023) uses a Transformer with co-occurrence encoding and patching to leverage longer histories efficiently. Benchmarking efforts such as the **Temporal Graph Benchmark (TGB)** emphasize realistic evaluation protocols and include strong non-parametric baselines like **EdgeBank** (Huang et al., 2023).

State-space models for sequences and graphs: Structured state space models such as **S4** (Gu et al., 2021) demonstrated that long-range sequence dependencies can be modeled efficiently with principled linear dynamical systems. **Mamba** (Gu & Dao, 2023) introduced *selective* (input-dependent) state updates that achieve strong empirical performance with linear-time scaling in sequence length. Subsequent work like **Mamba-2** (Dao & Gu, 2024) further clarified connections between attention and structured state space computation.

Motivated by these, several papers adapt SSM ideas to graph-structured learning. **S4G** (Song et al., 2024) introduces structured state spaces into graph settings to address long-range dependencies while retaining useful inductive bias. **DyGMamba** (Ding et al., 2025) adapts Mamba-style selective state space models to **continuous-time dynamic graphs** by encoding per-node interaction histories with an SSM and using an additional time-level SSM to select critical historical information, achieving strong dynamic link prediction performance with favorable efficiency. **Graph Mamba Networks** (Wang et al., 2024) propose a general framework that sequentializes graph neighborhoods and processes them with bidirectional selective SSM encoders. These approaches often focus on static graphs or on building graph-to-sequence encoders within each forward pass.

Positioning. GSNs introduced in this paper target the same temporal link prediction setting and evaluation protocol used by these baselines. Conceptually, GSNs are related to memory-based temporal GNNs (e.g., TGN) in that they maintain persistent node memories, but they differ in two important ways: (i) the memory update is implemented as a selective state-space update (SSM) rather than a purely GRU/message-passing update, and (ii) we explicitly analyze forgetting/retention induced by the commit mechanism and blank dynamics, rather than treating the memory as a black box.

As far as SSM-like models are concerned, GSNs use selective SSM machinery but elevate the "sequence position" notion to **node identity**: the state is persistent per node and stored in an explicit table. Unlike many graph-SSM encoders that reconstruct representations from a context window, GSNs are designed around **state persistence across temporal buckets** with explicit read/update/commit semantics. This allows us ask and answer a different question: *not only "can SSMs work on graphs?"* but *"what does a persistent node state retain, and how can we measure and control its forgetting"*?

3 Preliminaries: Persistent Node State Dynamics in GSNs

This section formalizes the state evolution semantics that the rest of the theory studies. The key design choice is that **each node owns a persistent hidden state** that is **stored outside** the forward pass and **updated/committed** only at well-defined times in a bucketed protocol.

3.1 Temporal interaction stream and bucketed protocol

We observe a temporal interaction stream of events $\{e_k\}_{k=1}^{N}$, where each event $e_k = (u_k, v_k, t_k, x_k)$ contains an interacting pair (u_k, v_k), a timestamp t_k, and optional *edge* features x_k. Let V be the global node set and ids(i) map to a stable integer ID.

We partition time into K chronological buckets $B^{(1)}, \dots, B^{(K)}$. For bucket $B^{(k)} = [t_{\text{start}}^{(k)}, t_{\text{end}}^{(k)}]$, we distinguish:

- **History snapshot (pre-bucket):** The graph $G_{\text{prev}}^{(k)}$, containing all events with timestamp $< t_{\text{start}}^{(k)}$.
- **Commit snapshot (up to bucket end):** The graph $G_{\text{end}}^{(k)}$, containing all events with timestamp $\leq t_{\text{end}}^{(k)}$.

This separation is important for leakage-free evaluation: we want predictions inside bucket k to depend only on history before the bucket, while still allowing the model's persistent memory to be updated with the bucket's events after scoring.

3.2 Persistent state table and read–update–commit

A GSN maintains a **persistent state table** $S \in \mathbb{R}^{|V| \times m}$, where the row $S[\text{ids}(i)]$ is node i's memory/state (dimension m). This table persists across buckets and is conceptually a part of the model's state.

For a given snapshot (e.g., $G_{\text{prev}}^{(k)}$ or $G_{\text{end}}^{(k)}$), the model performs:

1. **Read:** for each node i participating in the snapshot, read $s_i \leftarrow S[\text{ids}(i)]$.

2. **Pre-commit update:** produce a proposed new state s_i' via a parameterized update operator $F_\phi : \mathbb{R}^m \times \mathcal{E} \times [0, T] \mapsto \mathbb{R}^m$ such that

$$s_i' = F_\phi(s_i; \text{snapshot context}, \Delta t).$$

   Here, m is the state dimension, $\mathcal{E}$ is the edge set, $\Delta t \in [0, T]$ denotes elapsed time since node i was last updated, and T denotes the last timestamp of the data (See Figure 1). In our implementation, this update is realized with a time-aware selective state-space block (Mamba-style), but the theory below only requires that it induces some map F_ϕ on states during a bucket.

   Commit (persistent write): update the table using an exponential moving average with commit rate α:

$$S[\text{ids}(i)] \leftarrow (1 - \alpha)S[\text{ids}(i)] + \alpha s_i',$$

   where $\alpha \in (0, 1]$ can be fixed or scheduled.

Intuitively, the pre-commit update computes "what the node state should become given the bucket", while the commit step controls how aggressively that proposal overwrites persistent memory. The scalar α with reappear in the theory as a direct knob on forgetting and mixing.

3.3 Stateful training/evaluation semantics with no leakage

Within bucket $B^{(k)}$, we use two-snapshot protocol:

- **Score step:** run the model on $G^{(k)}_{\text{prev}}$ to obtain node representations, then score candidate edges whose timestamp lies in bucket $B^{(k)}$.
- **Commit step:** after scoring, update/commit the persistent states using $G^{(k)}_{\text{end}}$, so that the next bucket starts from memory that has incorporated bucket-k interactions.

This makes the stateful model well-defined: predictions for bucket k are conditioned on history strictly before the bucket, but the state entering the bucket $k + 1$ reflects everything up to $t^{(k)}_{\text{end}}$.

3.4 Quantities the theory will track

The remaining theory asks: **what does a bounded-dimensional persistent state retain, and how does it forget?** To make this precise, we define three objects:

1. **The committed bucket operator.** Fix a bucket snapshot structure and the exogenous inputs it induces, e.g., features, time deltas, etc. The combination "pre-commit update + EMA commit" defines an operator T_α on node states: $s \mapsto (1 - \alpha)s + \alpha F(s)$, where F is the pre-commit update map induced by that bucket. In the next section, we study this operator in the special but revealing case of "blank buckets".
2. **Influence of an event (counterfactual sensitivity).** Consider two executions that are identical except for a small perturbation to the state caused by changing a single ingested event (or write payload) at time 0. Let s_t and $\hat{s}_t$ be the resulting node states after t subsequent bucket commits under identical later inputs. **The influence at delay** t is measured by $\|s_t - \hat{s}_t\|$. This is the central object for "forgetting".
3. **Recall through a readout.** A downstream task uses a decoder/readout g (e.g., link predictor or probe head) applied to the node state. We will bound not just state differences, but also readout differences $\|g(s_t) - g(\hat{s}_t)\|$ under mild regularity assumptions, connecting "state retention" to "task-visible retention".

With these definitions in place, the next section derives explicit convergence and geometric forgetting rates for repeated blank-bucket commits, and shows how α and the blank dynamics jointly determine memory half-life.

4 Blank-Bucket Theory: Convergence and Geometric Forgetting

This section formalizes the "retention dial" perspective of the commit rate α by analyzing what happens when a node experiences *no new informative input* for multiple buckets.

4.1 Blank buckets

A **blank bucket** is a bucket in which the node receives no new task-relevant signal, e.g., features are zeroed, or messages carry a fixed "blank token", but the model is still executed and the resulting state is committed to the persistent state table (as in the write–wait–read protocol).

Let s_t be the persistent state of a fixed node after t blank-bucket commits following some "write" event. The GSN update semantics from the previous section imply a committed recursion of the form:

- pre-commit proposal: $s \mapsto F(s)$ (this is "whatever the backbone does on a blank bucket")
- commit: $s_{t+1} = (1-\alpha)s_t + \alpha F(s_t)$

4.2 Main result: closed form, convergence, and influence decay

The next result, a consequence of classical Lyapunov theory, provides us with means to characterize the "forgetting" properties of GSNs.

Theorem 1. **(Contraction for arbitrary commit maps):** Let $F : \mathbb{R}^d \mapsto \mathbb{R}^d$ be $\mathcal{C}^1$ and fix $\alpha \in (0,1]$. Define the (time-invariant) update map

$$f(s) \coloneqq (1-\alpha)s + \alpha F(s). \tag{2}$$

Assume there exists an equilibrium $s^\star$ with $f(s^\star) = s^\star$. Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a convex **forward-invariant** set, (i.e., $f(\mathcal{D}) \subseteq \mathcal{D}$) containing $s^\star$. Suppose that there exist a matrix $P \succ \mathbf{0}$ and a scalar $\rho \in (0,1)$, such that for all $s \in \mathcal{D}$,

$$\left((1-\alpha)I + \alpha\nabla F(s)\right)^\top P \left((1-\alpha)I + \alpha\nabla F(s)\right) \preceq \rho^2 P. \tag{3}$$

Then along any trajectory $s_{t+1} = f(s_t)$ with $s_0 \in \mathcal{D}$, we have that

1. (**Exponential Stability**) In Euclidean norm,
$$\|s_t - s^\star\| \le \sqrt{\frac{\lambda_{\max}(P)}{\lambda_{\min}(P)}}\rho^t \|s_0 - s^\star\|.$$
2. (**Incremental contraction**) For any two trajectories $x_{t+1} = f(x_t)$ and $y_{t+1} = f(y_t)$ with $x_0, y_0 \in \mathcal{D}$,
$$\|x_t - y_t\|_P \le \rho^t \|x_0 - y_0\|_P, \quad \text{where} \quad \|v\|_P = \sqrt{v^\top P v}$$

See the appendix for a proof.

To make this result analyzable in practice, we adopt a local modeling step used throughout the paper's empirical theory validation: **during blank buckets, the learned update behaves approximately as an affine map**

$$F(s) = Ws + u$$

for some matrix W and vector u determined by the trained backbone and the blank-bucket exogenous inputs.

Under this approximation, the committed recursion becomes:

$$s_{t+1} = Ms_t + \alpha u, \tag{4}$$

where,

$$M \coloneqq (1-\alpha)I + \alpha W \tag{5}$$

This assumption isolates *all* memory retention questions into the spectrum/size of a single operator M, and makes α's role explicit as a convex interpolation between "do nothing" (I) and "apply blank dynamics" W.

Corollary 1. **(Blank-bucket dynamics under affine update):** Assume that during blank buckets the pre-commit update is affine: $F(s) = Ws + u$. Let $\alpha \in (0, 1]$ and define M as in (5). Let $\|\cdot\|$ denote the operator norm for a matrix and the ℓ_2 norm for a vector. If $\|W\| \le \beta < 1$, then:

1. **Unique fixed point (α-invariant):** there is a unique fixed point $s^\star$ satisfying $s^\star = Ws^\star + u$, equivalently $s^\star = (I - W)^{-1}u$, and it doesn't depend on α.
2. **Closed form:** for all $t \ge 0$,
$$s_t = s^\star + M^t(s_0 - s^\star). \tag{6}$$
3. **Geometric convergence:** We have,
$$\|s_t - s^\star\| \le r^t \|s_0 - s^\star\|, \tag{7}$$
where $r = \|M\| \le (1 - \alpha) + \alpha\beta = 1 - \alpha(1 - \beta) < 1$.
4. **Geometric Forgetting (single-event influence):** for any two runs with identical blank buckets but different initial states s_0 and $\hat{s}_0$,
$$\|s_t - \hat{s}_t\| \le r^t \|s_0 - \hat{s}_0\|.$$
In particular, if $s_0 - \hat{s}_0$ is the perturbation induced by changing only the write event, then the influence of that event decays at most geometrically in the number of blank bucket commits.

Proof sketch. Corollary 1 is the affine specialization of Theorem 1. With $F(s) = Ws + u$ and $\|W\| \le \beta < 1$, the committed map f from Theorem 1 (defined by (2)) becomes the affine recursion
$$s_{t+1} = Ms_t + \alpha u,$$
where M is as defined in (5). The fixed point $s^\star$ is characterized by the fixed-point equation $f(s^\star) = s^\star$ (the equilibrium required by Theorem 1). Using the specific form of f this time we get the explicit fixed-point as $s^\star = (I - W)^{-1}u$ independent of α.

To get geometric convergence and forgetting, apply Theorem 1 with $\mathcal{D} = \mathbb{R}^d$ and $P = I$. Here $\nabla F(s) = W$ is constant, so the Jacobian $\nabla f(s)$ is the constant matrix M and Theorem 1's matrix inequality holds with $\rho = \|M\|$ because $M^\top M \preceq \|M\|^2 I$.

Finally, bound $\|M\|$ by triangle inequality: $\|M\| \le (1 - \alpha) + \alpha\|W\| \le (1 - \alpha) + \alpha\beta =: r < 1$. This gives the stated geometric decay rates. $\square$

Two immediate design takeaways:

- **Smaller α implies longer retention.** Since $r \le 1 - \alpha(1 - \beta)$, decreasing α pushes r toward 1, slowing geometric decay, i.e., increasing memory horizon.
- **State norm can grow while influence decays.** The recursion can converge toward a nonzero $s^\star$ even if the *difference* between two counterfactual runs shrinks. So it is possible for $\|s_t\|$ to increase (drift towards $s^\star$) while $\|s_t - \hat{s}_t\|$ decreases (forgetting of the write). This is why the experiments track an explicit influence rather than just the state magnitude.

5 Experiments

In this section, we discuss experiments that distinctively fall into two different genres, *viz.* (i) validation and suitability of GSNs in doing dynamic link prediction and (ii) validation of capacity/recall theory. Before presenting the results, it is worth distinguishing the inference regimes targeted by different architectures. While recent state-space baselines such as DyGMamba achieve strong performance, their reliance on bidirectional scanning (forward and reverse passes) during graph encoding implies a batch-processing paradigm. This structure is highly effective for offline analysis but introduces necessary latency in strictly online, streaming settings where predictions must be made instantly upon event arrival without waiting for future context. In contrast, GSNs are designed as causally strictly forward-only systems: the persistent node state is updated immediately as events arrive, maintaining constant-time inference compatibility regardless of the sequence length.

Having set the paradigm, we now present the metrics for GSNs. We start with dynamic link prediction.

5.1 GSNs for Dynamic Link Prediction

We train GSNs on seven different dynamic link prediction datasets: Wikipedia (Kumar et al., 2019a), MOOC (Kumar et al., 2019a), Enron (Shetty & Adibi, 2004), UCI (Panzarasa et al., 2009), US Legis (Poursafaei et al., 2022), Can. Parl (Poursafaei et al., 2022), and Contact (Poursafaei et al., 2022). The results are summarized in Table 1 and Table 2 respectively.

Protocol and evaluation. The training process is detailed in Algorithm 1. We follow the standard temporal link prediction protocol used by prior work on these datasets. Each interaction (u, v, t, attr) is processed in timestamp order, and the model is evaluated by scoring the true edge against sampled negatives. We report results under both the commonly used negative sampling regimes shown in the tables (rnd/ind).

Baselines. We compare against representative memory-based and attention-based temporal graph models (e.g., JODIE, DyRep, TGAT, TGN, CAWN, GraphMixer, DyGFormer, DyGMamba), and also against recent sequence-modeling approaches (DyG-Mamba). Where available, we include strong non-neural heuristics (EdgeBank) and contrastive pretraining baselines (TCL), using the benchmark's reported mean±std across runs.

Table 1: Average Precision (AP, %) for **transductive** dynamic link prediction with random (rnd) and inductive (ind) negative sampling strategies. APs are reported as mean±std. AP for all other models are taken from Ding et al. (2025).

NSS	Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DyG-Mamba	GSN
rnd	Wikipedia	96.50±0.14	94.86±0.06	96.94±0.06	98.45±0.06	98.76±0.03	90.37±0.00	96.47±0.04	97.25±0.04	99.03±0.04	**99.08±0.03**	97.05±0.08
	MOOC	80.23±2.44	81.97±0.49	85.84±0.15	89.15±1.60	80.15±0.25	57.97±0.00	82.38±0.24	82.78±0.15	87.52±0.49	90.25±0.09	**90.76±0.10**
	Enron	84.77±0.30	82.38±3.36	71.12±0.97	86.53±1.11	89.56±0.09	83.53±0.00	79.70±0.71	82.25±0.16	92.47±0.12	**93.14±0.08**	91.04±0.21
	UCI	89.43±1.09	65.14±2.30	79.63±0.70	92.34±1.04	95.18±0.06	76.20±0.00	89.57±1.63	93.25±0.57	95.79±0.17	**96.14±0.14**	92.23±0.63
	Can.Parl.	69.26±0.31	66.54±2.76	70.73±0.72	70.88±2.34	69.82±2.34	64.55±0.00	68.67±2.67	77.04±0.46	97.36±0.45	**98.20±0.52**	93.48±1.82
	USLegis	75.05±1.52	75.34±0.39	68.52±3.16	75.99±0.58	70.58±0.48	58.39±0.00	69.59±0.48	70.74±1.02	71.11±0.59	73.66±1.13	**87.80±1.08**
	Contact	95.31±1.33	95.98±0.15	96.28±0.09	96.89±0.56	90.26±0.28	92.58±0.00	92.44±0.12	91.44±0.03	98.29±0.01	**98.38±0.01**	97.90±0.03
ind	Wikipedia	75.65±0.79	70.21±1.58	87.00±0.16	85.62±0.44	74.06±2.62	80.63±0.00	86.76±0.72	88.59±0.17	79.29±5.38	87.06±0.86	**90.78±0.78**
	MOOC	65.23±2.19	61.66±0.95	75.95±0.64	77.50±2.91	73.51±0.94	49.43±0.00	74.65±0.54	74.27±0.92	**81.24±0.69**	81.19±2.02	80.07±0.91
	Enron	68.96±0.98	67.79±1.53	63.94±1.36	70.89±2.72	75.15±0.58	73.89±0.00	71.29±0.32	75.01±0.79	77.41±0.89	77.46±0.90	**84.31±0.95**
	UCI	65.99±1.40	54.79±1.76	68.67±0.84	70.94±0.71	64.61±0.48	57.43±0.00	76.01±1.11	80.10±0.51	72.25±1.71	77.75±1.56	**82.58±0.90**
	Can.Parl.	48.42±0.66	58.61±0.86	68.82±1.21	65.34±2.87	67.75±1.00	62.16±0.00	65.85±1.75	69.48±0.63	95.44±0.57	**97.29±0.96**	85.68±0.70
	USLegis	50.27±5.13	83.44±1.16	61.91±5.82	67.57±6.47	65.81±8.52	64.74±0.00	78.15±3.34	79.63±0.84	81.25±3.62	**85.61±1.66**	83.43±1.42
	Contact	93.43±1.78	94.18±0.10	94.35±0.48	90.18±3.28	89.31±0.27	85.00±0.00	91.35±0.21	90.87±0.35	94.75±0.28	94.63±0.06	**95.43±0.09**

Table 2: AUC-ROC (%) for **transductive** dynamic link prediction with random (rnd) and inductive (ind) negative sampling strategies. AUC-ROCs are reported as mean±std. AUC-ROCs for all other models are taken from Ding et al. (2025).

NSS	Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	FreeDyG	DyGFormer	DyG-Mamba	GSN
rnd	Wikipedia	96.33±0.07	94.37±0.09	96.67±0.07	98.37±0.07	98.54±0.04	90.78±0.00	95.84±0.18	96.92±0.03	**99.41±0.01**	98.91±0.02	99.01±0.09	94.14±0.02
	MOOC	83.81±2.09	85.08±0.58	87.11±0.19	**91.21±1.15**	80.38±0.26	60.86±0.00	83.12±0.18	84.01±0.17	89.93±0.35	87.91±0.58	91.01±0.14	81.52±1.35
	Enron	87.96±0.52	84.89±3.00	68.89±1.10	88.32±0.99	90.45±0.14	87.05±0.00	75.74±0.72	84.38±0.21	**94.01±0.11**	93.33±0.13	93.05±0.17	87.26±0.18
	UCI	90.44±0.49	68.77±2.34	78.53±0.74	92.03±1.13	93.87±0.08	77.30±0.00	87.82±1.36	91.81±0.67	95.00±0.21	94.49±0.26	**95.32±0.18**	84.52±0.51
	Can.Parl.	78.21±0.23	73.35±3.67	75.69±0.78	76.99±1.80	75.70±3.27	64.14±0.00	72.46±3.23	83.17±0.53	N/A	97.76±0.41	**98.67±0.29**	86.96±0.40
	USLegis	82.85±1.07	82.28±0.32	75.84±1.99	83.34±0.43	77.16±0.39	62.57±0.00	76.27±0.63	76.96±0.79	N/A	77.90±0.58	78.19±0.64	75.60±1.10
	Contact	96.66±0.89	96.48±0.14	96.95±0.08	97.54±0.35	89.99±0.34	94.34±0.00	94.15±0.09	93.94±0.02	N/A	98.53±0.01	**98.59±0.00**	95.80±0.10
ind	Wikipedia	70.96±0.78	67.36±0.96	81.93±0.22	80.97±0.31	70.95±0.95	81.73±0.00	82.19±0.48	84.28±0.30	82.74±0.32	75.09±3.70	82.30±1.81	**87.83±1.31**
	MOOC	66.63±2.30	63.26±1.01	73.18±0.33	77.44±2.86	70.32±1.43	48.18±0.00	70.36±0.37	72.45±0.72	78.47±0.94	80.76±0.76	**82.05±1.38**	60.15±1.00
	Enron	70.92±1.05	68.73±1.34	60.45±2.12	71.34±2.46	75.17±0.50	75.00±0.00	67.64±0.86	71.53±0.85	77.27±0.61	74.07±0.64	74.49±0.48	**80.38±0.80**
	UCI	64.14±1.26	54.25±2.01	60.80±1.01	64.11±1.04	58.06±0.26	58.03±0.00	70.05±1.86	74.59±0.74	**75.39±0.57**	65.96±1.18	73.23±1.03	65.21±1.71
	Can.Parl.	52.88±0.80	63.53±0.65	72.47±1.18	69.57±2.81	72.93±1.78	61.41±0.00	69.47±2.12	70.52±0.94	N/A	96.70±0.59	**97.30±0.97**	71.35±0.63
	USLegis	59.05±5.52	89.44±0.71	71.62±5.42	78.12±4.46	76.45±7.02	68.66±0.00	82.54±3.91	84.22±0.91	N/A	87.96±1.80	**90.28±0.50**	66.85±0.78
	Contact	94.47±1.08	94.23±0.18	94.10±0.41	91.64±0.72	87.68±0.24	85.87±0.00	91.23±0.19	90.96±0.27	N/A	95.01±0.15	95.08±0.01	90.70±0.14

Discussion. Across the seven datasets, GSNs are competitive with recent temporal graph baselines under Average Precision (AP) and across negative sampling regimes. The strongest gains tend to appear when the evaluation stresses *generalization under distribution shift* (e.g., inductive splits or inductive negatives), which is consistent with the intuition that a persistent node state can act as a compact summary that is updated online rather than recomputed from scratch. At the same time, the results also highlight that there remains headroom from careful tuning and architectural choices (e.g., how the edge-scoring head consumes node states, and how aggressively one commits state between buckets). Overall, these benchmarks serve as an end-to-end sanity check: GSN-style state persistence does not sacrifice predictive performance on standard dynamic link prediction, and it motivates a more targeted question—*what memory is the model actually retaining, and how does it decay with delay?* We answer that question next using controlled write–wait–read probes that directly test the capacity/recall theory.

5.2 Capacity/Recall Theory: Synthetic Write–Wait–Read Probes

Setting. To isolate the memory behavior of a node state, we use a synthetic "write–wait–read" protocol. A node receives a *write* event carrying a discrete symbol (the item to be stored), followed by a gap or *delay* of d time steps in which the node receives only a *blank* event (no new information), and finally a *read* query that asks the model to recover the written symbol. We vary the delay $d \in \{0, 1, 2, 5, 10, 20, 30, 50\}$ and the GSN commit rate $\alpha \in \{0.05, 0.1, 0.2, 0.4, 1\}$ (higher α means more aggressive state commits).

We consider two padding variants during the "wait": *baseline blanks* reuse a single blank token, whereas *blank shift* perturbs/changes the blank token to rule out trivial positional shortcuts.

Metrics. We report *recall accuracy* (fraction of correctly recovered symbols at read time) and an *influence* diagnostic that measures how much the readout changes when the written symbol is perturbed, normalized relative to the $d = 0$ case. The theory predicts an approximately exponential decay of influence with delay, governed by an effective contraction factor (e.g., the spectral norm of an induced linearized update), and corresponding lower/upper envelopes on recall.

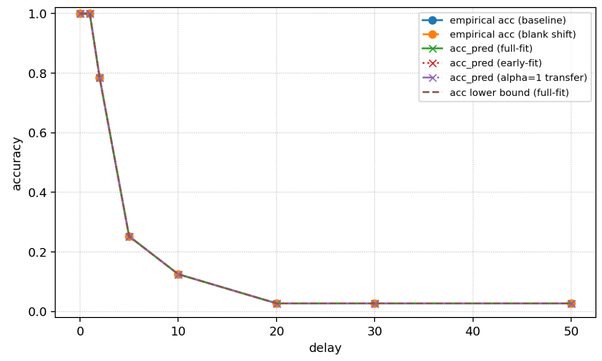
Fitting procedure (theory-to-measurement). Our theory predicts that, after a single write, the contribution of that write to a future read decays approximately exponentially with delay, controlled by an effective contraction factor that depends on the commit rate α and the linearized state update. Operationally, we estimate this contraction factor from *short-delay* measurements of influence (to avoid numerical floor effects), and then use the fitted parameter to extrapolate influence and recall to longer delays. This mirrors the intended use case of the theory: predict long-horizon recall from easily measurable local dynamics.

Results. We sweep delay and commit rate to map out when a single write remains recoverable. Consistent with the theory, recall is near-perfect at small delays and then degrades as delay grows, with the "memory horizon" shifting systematically with α (smaller α yields longer retention). To understand *why* accuracy drops, we turn to the influence metric: the measured influence curves are close to exponential in delay (Figure 2, top-right), indicating that forgetting is governed by an effective contraction of the node-state dynamics. Fitting the contraction factor on short delays is sufficient to predict longer-delay influence and, through the channel-style bound, the corresponding recall envelopes (Figure 2, top-left). Finally, the bottom row connects the fitted dynamics to architectural control knobs: the estimated contraction factor varies smoothly with α, and the extrapolation error on held-out long delays remains small across a broad range of commit rates. As an additional sanity check, we repeat the "wait" phase with a *blank-shift* padding token (same length, different identity) and obtain the same qualitative decay, ruling out trivial padding artifacts.

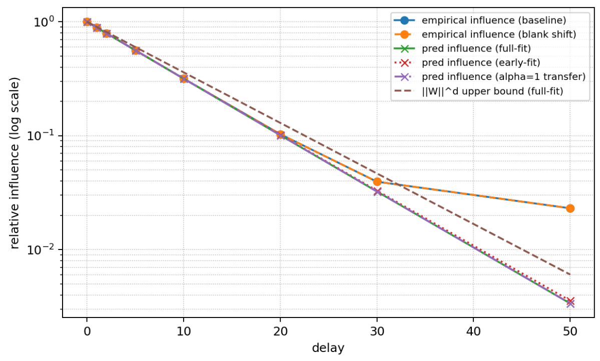
Fitting the theoretical model on short-delay measurements yields accurate long-horizon predictions of both influence and recall. This supports the central modeling assumption of the capacity/recall analysis: after linearization, the dominant forgetting mechanism behaves like a contraction whose rate can be summarized by a single scalar. From a practical perspective, these curves provide an actionable diagnostic for *choosing* the commit rate: if an application requires retaining information for D steps, the fitted contraction factor predicts whether the signal will remain above the effective noise floor at that horizon. Conversely, when rapid adaptation is more important than long retention, larger α intentionally shortens the memory horizon.

Together with the dynamic link prediction results, these controlled probes suggest that GSNs can match strong temporal-graph baselines while offering a principled, theory-backed handle on memory retention in persistent node states.

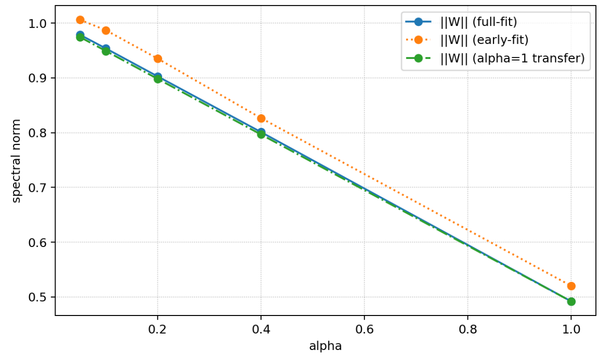
Late-delay deviation/influence floor. In Fig 2b, the deterministic affine prediction can undershoot the empirical influence at the largest delays, where the measured curve approaches a small, nonzero floor. This is expected: the theory assumes a time-invariant affine blank operator, whereas in practice blank buckets induce time-varying operators (e.g., different blank graphs), nonlinear state-dependent dynamics, and effective process noise (including any explicit state noise), which together imply an irreducible long-horizon influence floor under repeated blank updates. Accordingly, we fit the contraction factor using short delays (to avoid floor effects) and evaluate extrapolation primarily over the regime where the influence remains above this noise floor and recall transitions



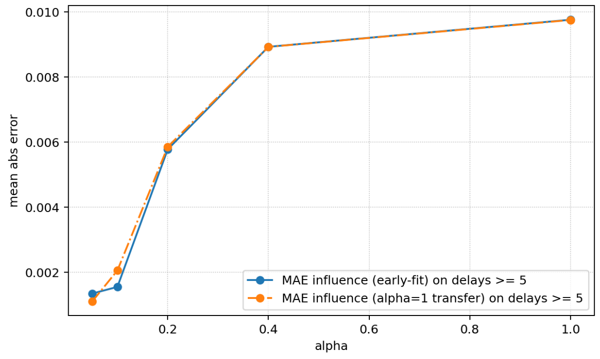
(a) Accuracy vs. delay ($\alpha = 0.2$).



(b) Influence decay vs. delay ($\alpha = 0.2$, log y-scale).



(c) Estimated contraction factor vs. α.



(d) Out-of-fit-horizon influence error vs. α.

Figure 2: Validating the capacity/recall theory on synthetic write–wait–read probes. **Top-left:** recall accuracy versus delay for a representative commit rate ($\alpha = 0.2$); the dashed curve and shaded band show the theory-based prediction obtained by fitting short-delay influence and propagating it through the recall bound. **Top-right:** normalized influence versus delay for the same setting (log y-axis), illustrating an approximately exponential decay. **Bottom-left:** estimated contraction factor as a function of α (smaller implies slower forgetting), computed from short-delay fits and compared to a spectral-norm proxy of the linearized update. **Bottom-right:** mean absolute error between predicted and measured influence on long delays (held out from fitting), showing that local dynamics extrapolate to long-horizon behavior.

6 Conclusion

This work proposed **Graph State Networks (GSNs)**: a temporal-graph architecture that treats each node as a **persistent stateful system** stored in an explicit ID-indexed table and updated online via a **time-aware selective state-space block**, with writes committed through an EMA rule controlled by a single **commit rate** α. This separation–*how* information is integrated (the backbone update) versus *how aggressively* it is committed (α)–yields a simple but powerful "retention dial" that is easy to reason about and tune.

On the theory side, we analyzed **blank-bucket dynamics** and showed that, under a mild affine approximation of the blank update, the influence of a single past write **decays geometrically** with a contraction factor determined jointly by α and the induced linearized operator. This motivates a practical workflow: measure short-horizon influence to estimate the effective contraction, then extrapolate to predict long-horizon retention. Controlled **write–wait–read** probes support this picture: influence curves are close to exponential over the regime that matters for recall, and short-delay fits predict longer-delay behavior well, while deviations at very long delays can be understood as an effective noise floor from time-varying blank operators and backbone nonlinearity. Finally, in standard dynamic link prediction benchmarks (including both transductive and inductive evaluations), GSNs are broadly competitive with strong temporal-graph baselines, with particularly strong behavior in inductive settings—consistent with the value of persistent, selectively-updated node memory.

A few directions are especially promising. First, pushing GSNs closer to strict **event-by-event streaming** (or learning variable bucket sizes) would broaden applicability. Second, α is currently a simple global control knob; learning **adaptive** or **node-specific commit rates** could better trade off retention and adaptation across heterogeneous nodes. Third, extending the analysis beyond affine blanks—e.g., to explicitly **stochastic** or **time-varying** blank dynamics and to readout mismatch under distributional drift—would make the theory mode predictive in the far tail. Overall, GSNs suggest a useful middle-ground between recomputing history at query time and opaque memory modules: **persistent node memory with a measurable, tunable retention mechanism.**

7 Broader Impact

Graph State Networks (GSNs) are a methodological contribution to temporal graph learning: they propose a persistent per-node state with controllable update/commit dynamics and provide tools to characterize retention and forgetting. Potential positive impacts include improved modeling of time-evolving relational data in applications such as anomaly detection, forecasting in interaction networks, recommendation, and scientific or industrial monitoring, where long-range temporal dependencies and distribution shift are common. The explicit commit-rate control may also support more predictable behavior under changing environments by making "how quickly the model forgets" easier to tune and diagnose.

At the same time, temporal link prediction models can enable harmful uses when applied to sensitive social or behavioral data (e.g., surveillance, targeted manipulation, or inference of private relationships). Persistent node memory could amplify these risks by making it easier to aggregate signals over long time horizons. These concerns are not unique to GSNs, but are relevant given the model's focus on retention. Mitigations include careful dataset governance and privacy-preserving practices (data minimization, access control, auditing), restricting deployment in high-risk settings, and evaluating models for leakage or unintended inference on sensitive attributes. The commit-rate mechanism may also be used as a partial safeguard by limiting retention when long-horizon accumulation is undesirable, although it is not a substitute for privacy protections.

Finally, like other temporal graph models, GSNs may inherit and potentially reinforce biases present in observational interaction data (e.g., exposure or popularity bias in recommendation logs). Future work should examine fairness-related metrics in temporal settings, explore calibration and debiasing strategies, and document intended use cases and limitations.

References

Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? In *International Conference on Learning Representations*, 2023. doi: 10.48550/arXiv.2302.11636. URL https://arxiv.org/abs/2302.11636.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. ML Research Press, 2024. doi: 10.48550/arXiv.2405.21060. URL https://arxiv.org/abs/2405.21060.

Zifeng Ding, Yifeng Li, Yuan He, Antonio Norelli, Jingcheng Wu, Volker Tresp, Yunpu Ma, and Michael M. Bronstein. DyGMamba: Efficiently Modeling Long-term Temporal Dependency on Continuous-Time Dynamic Graphs with State Space Models, 2025. URL https://openreview.net/forum?id=f3gCs2a4ZD.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *Conference on Neural Information Processing Systems*, 2021. doi: 10.48550/arXiv.2111.00396. URL https://arxiv.org/abs/2111.00396.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In *Conference on Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/066b98e63313162f6562b35962671288-Abstract-Datasets_and_Benchmarks.html.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pp. 1269–1278, New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330895. URL https://doi.org/10.1145/3292500.3330895.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1269–1278. Association for Computing Machinery, 2019b. doi: 10.1145/3292500.3330895. URL https://doi.org/10.1145/3292500.3330895.

Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *J. Assoc. Inf. Sci. Technol.*, 60:911–932, 2009. URL https://api.semanticscholar.org/CorpusID:263266966.

Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Conference on Neural Information Processing Systems*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/d49042a5d49818711c401d34172f9900-Abstract-Datasets_and_Benchmarks.html.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *International Conference on Machine Learning*, 2020. doi: 10.48550/arXiv.2006.10637. URL https://arxiv.org/abs/2006.10637.

Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. 2004. URL https://api.semanticscholar.org/CorpusID:59919272.

Yunchong Song, Siyuan Huang, Jiacheng Cai, Xinbing Wang, Chenghu Zhou, and Zhouhan Lin. S4g: Breaking the bottleneck on graphs with structured state spaces, 2024. URL https://openreview.net/forum?id=0Z6lN4GYrO.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyePrhR5KX`.

Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. *ArXiv*, abs/2402.00789, 2024. URL `https://api.semanticscholar.org/CorpusID:267364853`.

Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*, 2021a. doi: 10.48550/arXiv.2101.05974. URL `https://arxiv.org/abs/2101.05974`.

Yu Wang, Tong Zhang, Yu Meng, Jia Pan, and Shiyu Chang. TCL: Transformer-based dynamic graph modelling via contrastive learning, 2021b. URL `https://arxiv.org/abs/2105.07944`.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*, 2020. doi: 10.48550/arXiv.2002.07962. URL `https://arxiv.org/abs/2002.07962`.

Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. In *Conference on Neural Information Processing Systems*, 2023. doi: 10.48550/arXiv.2303.13047. URL `https://arxiv.org/abs/2303.13047`. Accepted at NeurIPS 2023.

A Proof of Theorem 1

Proof. Let

$$J_f(s) = \nabla f(s) = (1-\alpha)I + \alpha \nabla F(s).$$

Condition (3) says $J_f(s)^\top P J_f(s) \preceq \rho^2 P$. For any vector v,

$$\|J_f(s)v\|_P^2 = v^\top J_f(s)^\top P J_f(s) v \le \rho^2 v^\top P v = \rho^2 \|v\|_P^2.$$

So for all $s \in \mathcal{D}$,

$$\|J_f(s)v\|_P \le \rho \|v\|_P \quad \text{for all } v. \tag{8}$$

Now, take any x, $y \in \mathcal{D}$. Because $\mathcal{D}$ is convex, the segment $z(\tau) = y + \tau(x-y)$ lies in $\mathcal{D}$ for $\tau \in [0,1]$. Define $g(\tau) = f(z(t))$. Then by chain rule,

$$g'(\tau) = J_f(z(\tau))(x-y).$$

Now using the fundamental theorem of calculus and triangle inequality in $\|\cdot\|_P$:

$$\|f(x) - f(y)\|_P = \|g(1) - g(0)\|_P \le \int_0^1 \|g'(\tau)\|_P \quad d\tau.$$

Apply (8) pointwise:

$$\|g'(\tau)\|_P = \|J_f(z(\tau))(x-y)\|_P \le \rho\|x-y\|_P.$$

Therefore,

$$\|f(x) - f(y)\|_P \le \int_0^1 \rho\|x-y\|_P \, d\tau = \rho\|x-y\|_P \tag{9}$$

Set $y = s^\star$ in (9). Since $f(s^\star) = s^\star$,

$$\|f(s) - s^\star\|_P = \|f(s) - f(s^\star)\|_P \le \rho\|s - s^\star\|_P.$$

Squaring both sides and setting $s = s_t$, $s_{t+1} = f(s_t)$:

$$(s_{t+1} - s^\star)^\top P (s_{t+1} - s^\star) \le \rho^2 (s_t - s^\star)^\top P (s_t - s^\star),$$

Finally, the euclidean bound follows from

$$\lambda_{\min}(P)\|v\|^2 \leq v^\top P v \leq \lambda_{\max}(P)\|v\|^2.$$

Incremental contraction is just (9) iterated along trajectories

$$\|x_{t+1} - y_{t+1}\|_P = \|f(x_t) - f(y_t)\|_P \leq \rho\|x_t - y_t\|_P \leq \cdots \leq \rho^t\|x_0 - y_0\|_P$$

$\square$

B GSN Training Algorithm

Algorithm 1 Leakage-free bucketed training of a Graph State Network (GSN) for temporal link prediction

```
Require: Temporal event stream $\mathbf{E}$; cached negatives Neg; bucket size $B_e$ (events per bucket); optimizer Opt; accumulation
    steps $A$; commit schedule $\alpha(\cdot)$ (or constant $\alpha$); write-penalty weight $\lambda_{\text{wr}}$; scorer head Score$(\cdot)$.
 1: Initialize persistent state table(s) $\mathbf{S}$ (indexed by global node IDs).
 2: Initialize gradient accumulators $\{\Delta\theta\} \leftarrow 0$ and counter $c \leftarrow 0$.
 3: for epoch $= 1, 2, \ldots, E$ do
 4:     for bucket $k$ in chronological order do
 5:         $(\mathbf{u}, \mathbf{v}^+, \mathbf{t}, \mathbf{v}^-) \leftarrow$ NextTimeBatchWithNeg$(\mathbf{E}, \text{Neg}, B_e)$
 6:         $[t^{(k)}_{\text{start}}, t^{(k)}_{\text{end}}] \leftarrow [\min(\mathbf{t}), \max(\mathbf{t})]$
 7:         $G^{(k)}_{\text{prev}} \leftarrow$ Snapshot$\left(\mathbf{E};\ t < t^{(k)}_{\text{start}}\right)$                ▷ history-only
 8:         $G^{(k)}_{\text{end}} \leftarrow$ Snapshot$\left(\mathbf{E};\ t \le t^{(k)}_{\text{end}}\right)$                ▷ history + bucket
 9:         $(\mathbf{P}, \mathbf{s}) \leftarrow$ BuildCandidates$(\mathbf{u}, \mathbf{v}^+, \mathbf{v}^-)$
10:         TrainStep$(G^{(k)}_{\text{prev}}, \mathbf{P}, \mathbf{s}, \lambda_{\text{wr}};\ \mathbf{S}, \alpha(\cdot), \text{Opt}, \{\Delta\theta\}, c, A)$
11:         CommitToEnd$(G^{(k)}_{\text{end}};\ \mathbf{S}, \alpha(\cdot))$                ▷ updates $\mathbf{S}$ for bucket $k$+1
12:     end for
13: end for
14: function BuildCandidates$(\mathbf{u}, \mathbf{v}^+, \mathbf{v}^-)$
15:     $\mathbf{P} \leftarrow [\,],\ \ \mathbf{s} \leftarrow [\,]$
16:     for $i = 1$ to $|\mathbf{u}|$ do
17:         $\mathbf{C}_i \leftarrow [\,v^+_i\,] \,\|\, \mathbf{v}^-_i$                ▷ positive first
18:         $\mathbf{s}[i] \leftarrow |\mathbf{C}_i|$
19:         for each $v \in \mathbf{C}_i$ do
20:             append pair $(u_i, v)$ to $\mathbf{P}$
21:         end for
22:     end for
23:     return $(\mathbf{P}, \mathbf{s})$
24: end function
25: function TrainStep$(G, \mathbf{P}, \mathbf{s}, \lambda_{\text{wr}};\ \mathbf{S}, \alpha(\cdot), \text{Opt}, \{\Delta\theta\}, c, A)$
26:     Let $\{u_i\}_{i=1}^{B}$ be the source IDs for each query (one per block), inferred from $\mathbf{P}$ and $\mathbf{s}$.
27:     Read pre-states: $\mathbf{s}_{\text{pre}} \leftarrow \mathbf{S}[u_1, \ldots, u_B]$
28:     Forward + commit on history snapshot:
29:         Run GSN on $G$ with commit enabled to obtain (i) node embeddings $\mathbf{H}$ for nodes in $G$ and (ii) post-update states
    $\mathbf{s}_{\text{post}}$ for touched sources.
30:     Score candidates (post): $\ell_{\text{post}} \leftarrow \text{Score}(\mathbf{s}_{\text{post}}, \mathbf{H}, \mathbf{P})$
31:     Ranking loss: $L_{\text{rank}} \leftarrow$ RankLoss$(\ell_{\text{post}}, \mathbf{s})$                ▷ CE/BCE over each block
32:     Score-change-normalized write penalty:
33:         $\Delta\mathbf{s} \leftarrow \mathbf{s}_{\text{post}} - \text{stopgrad}(\mathbf{s}_{\text{pre}})$
34:         $d_s \leftarrow \|\Delta\mathbf{s}\|_2$                ▷ per-source norm
35:         $\ell_{\text{pre}} \leftarrow \text{Score}(\text{stopgrad}(\mathbf{s}_{\text{pre}}), \mathbf{H}, \mathbf{P})$
36:         $d_{\text{score}} \leftarrow$ BlockMean$(|\ell_{\text{post}} - \ell_{\text{pre}}|, \mathbf{s})$                ▷ mean over each block
37:         $L_{\text{wr}} \leftarrow \lambda_{\text{wr}} \cdot \text{mean}\left(\dfrac{d_s}{\text{stopgrad}(d_{\text{score}}) + \varepsilon}\right)$
38:     $L \leftarrow L_{\text{rank}} + L_{\text{wr}}$
39:     Backprop + accumulate: $g \leftarrow \nabla_\theta L;\quad \Delta\theta \leftarrow \Delta\theta + g;\quad c \leftarrow c + 1$
40:     if $c \bmod A = 0$ then
41:         Opt.Step$(\Delta\theta / A)$;   $\Delta\theta \leftarrow 0$
42:     end if
43: end function
44: function CommitToEnd$(G_{\text{end}};\ \mathbf{S}, \alpha(\cdot))$
45:     Run GSN on $G_{\text{end}}$ with commit enabled (no scoring), updating persistent table(s) $\mathbf{S}$
46:         via EMA commit: $\mathbf{S}[i] \leftarrow (1-\alpha)\mathbf{S}[i] + \alpha\mathbf{s}'_i$ (applied by the model)
47: end function
```