DIFFUSION MODEL FOR CONTINUOUS-TIME DYNAMIC GRAPH GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic graph generation has emerged as a powerful paradigm for modeling evolving structural phenomena. By synthesizing a stream of events, it can capture real-world network evolution over time. However, generation remains limited and slow as dynamic graphs grow in both size and feature dimensionality, limiting practical use. We introduce Dynamic Graph Diffusion (DyGDi), a new dynamic graph generation model that leverages diffusion to jointly model a window of events and their features. DyGDi generates the entire window in a single pass. This yields faster generation and lets events and features attend to one another, allowing the model to learn more coherent dynamics. To address a node-identity issue arising from this parallel generation, we introduce a new event tokenization that injects more information about event appearance, anchoring newly generated nodes to their identities. Altogether, we show that DyGDi trains and generates more efficiently than previous models and scales better. It produces graphs with high-quality node-feature and topological properties, while adapting to zero-shot settings such as link prediction. Code *will be provided* on GitHub.

1 INTRODUCTION

Many real-world systems are naturally described as dynamic graphs, such as social networks, citation graphs, financial transactions, and traffic-control systems (Ji et al., 2025; Yang et al., 2024; Yu & Gu, 2019). Synthetically generating such data streams is challenging, as the generated content must respect the temporal structure and topology of the original data without simply replicating it (Gupta et al., 2022). Prior models face scaling challenges rooted in their adjacency-matrix representation, which is unsuitable for large graphs (Gupta et al., 2022). Beyond the topological structure, a model must also generate realistic, evolving features for nodes and edges. Finally, it must generalize inductively to nodes and timestamps unseen during training. Generating such data has applications in predicting graph evolution over time, anomaly detection, social-network simulation, and recommendation systems (Yang et al., 2024). Given a dynamic graph, we therefore consider the task of inductively generating sequences of unseen events and nodes consistent with the original data dynamics.

Existing works represent a dynamic graph in one of two main ways (Yang et al., 2024), either as a *discrete-time* dynamic graph (DTDG) or as a *continuous-time* dynamic graph (CTDG). DTDG generative models view a graph as a sequence of discrete static graphs, either by aggregating events into a list of graph snapshots or by building a single static graph in which recurring nodes are linked by temporal edges. Both representations enable the reuse of static-graph generation methods, but discrete snapshots suffer a loss of temporal resolution from the discretization process, and the temporal-edge construction scales poorly to large numbers of timestamps due to the size of the resulting graph. CTDG generative models instead view a graph as a sequence of events, each occurring at a single timestamp, where an event may be an edge addition or deletion, a node addition or deletion, or a transformation of node or edge features. Such methods have been shown to scale to large graphs better than discrete-time models (Hosseini et al., 2025). However, existing inductive CTDG generators are autoregressive, sequential event sampling makes generation slow, and accelerating it by sampling batches of events in parallel generates the events of a batch independently, sacrificing their coherence.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Diffusion models, which originated in image generation (Sohl-Dickstein et al., 2015; Ho et al., 2020), mark a major advance in generative modelling and have since been extended to other modalities such as video (Ho et al., 2022) and text (Arriola et al., 2024). They are particularly attractive for their conditioning flexibility (Mayet et al., 2025; Wewer et al., 2025). They operate by corrupting data through a forward process and learning to reverse it back to the original distribution. Recent work has adapted them to static-graph data, for example in molecule design (Vignac et al., 2023; Jo et al., 2022; Hoogeboom et al., 2022), community-network and large-scale graph generation (Niu et al., 2020; Jo et al., 2022; Limnios et al., 2024; Bernecker et al., 2025; Li et al., 2024), and protein design (Trippe et al., 2023). Despite these promising results, such methods face two limitations: they remain confined to static-graph generation, and those that model the full adjacency matrix cannot scale to graphs with large numbers of nodes or timestamps.

We therefore propose Dynamic Graph Diffusion (DyGDi), the first diffusion-based generative model for CTDGs. It generates an entire window of events simultaneously, with node embeddings derived from a memory bank encoder (Rossi et al., 2020). Because this formulation does not rely on fixed node identities, it supports inductive generation of unseen nodes and timestamps. To support inductive generation of new nodes, we introduce event-class information and a random embedding for new nodes. The resulting model trains stably across window sizes, alleviating the scalability and memory issues of prior adjacency-matrix models, and can generate sequences of arbitrary length.
Our main contributions are summarized as follows:
• We propose the first diffusion-based generative model for CTDGs, generating windows of events simultaneously conditioned on past events.
• We identify and formalize memory matching collapse and window matching collapse in inductive generation and resolve them, enabling inductive generation of new nodes.
• We conduct extensive experiments showing that DyGDi achieves state-of-the-art dynamic graph generation, in both topology and feature distribution, and substantially faster training and generation than previous models. Moreover, the diffusion paradigm allows the trained model to be flexibly repurposed for downstream tasks such as zero-shot link prediction, highlighting the versatility of DyGDi and the broader potential of diffusion models for dynamic graph generation.

2 PRELIMINARIES

Continuous-Time Dynamic Graphs. A static graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of n vertices $\mathcal{V} = \{1, \dots, n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Continuous-time dynamic graphs (CTDGs) extend this formalism to graphs that evolve over time. Rather than a sequence of discrete snapshots, a CTDG is an ordered sequence of time-stamped events:

$$\mathcal{G} = \{x(t_1), x(t_2), \dots\} \tag{1}$$

where each event $x(t_i)$, occurring at time t_i, may be an edge or node addition or deletion, or a transformation of node or edge features. Each event is characterised by

$$x(t_i) = (u, v, t_i, \boldsymbol{e}_{u,v}(t_i)) \tag{2}$$

with $u, v \in \mathcal{V}$ and $t_i \in \mathbb{R}$, where the edge feature $\boldsymbol{e}_{u,v}(t_i)$ encodes attributes of the directed edge from u to v, such as edge weights, types, or other relational metadata.

Diffusion Models and Flow Matching. Diffusion models learn a target data distribution p_1 by reversing a noising process. A forward process progressively corrupts a data point $x_1 \sim p_1$ towards a source distribution p_0 (typically Gaussian noise, $x_0 \sim \mathcal{N}(0, \boldsymbol{I})$), yielding intermediate latent variables $x_\tau \sim q(x_\tau \mid x_1)$. Since t is reserved for graph time, we denote the diffusion timestep by τ. Rather than parameterizing the reverse process through noise or score prediction (Ho et al., 2020; Song et al., 2021), we adopt flow matching (Lipman et al., 2023), which learns a continuous velocity field v_θ transporting samples from the source distribution p_0 to the data distribution p_1 via an Ordinary Differential Equation (ODE):

$$\dot{x} = v_\theta(x, \tau) \quad \text{with target} \quad v^*(x_\tau, \tau) = x_1 - x_0. \tag{3}$$

The forward process interpolates linearly between noise and data:

$$x_\tau = (1 - \tau)\, x_0 + \tau\, x_1, \quad \tau \sim \mathcal{U}(0, 1). \tag{4}$$

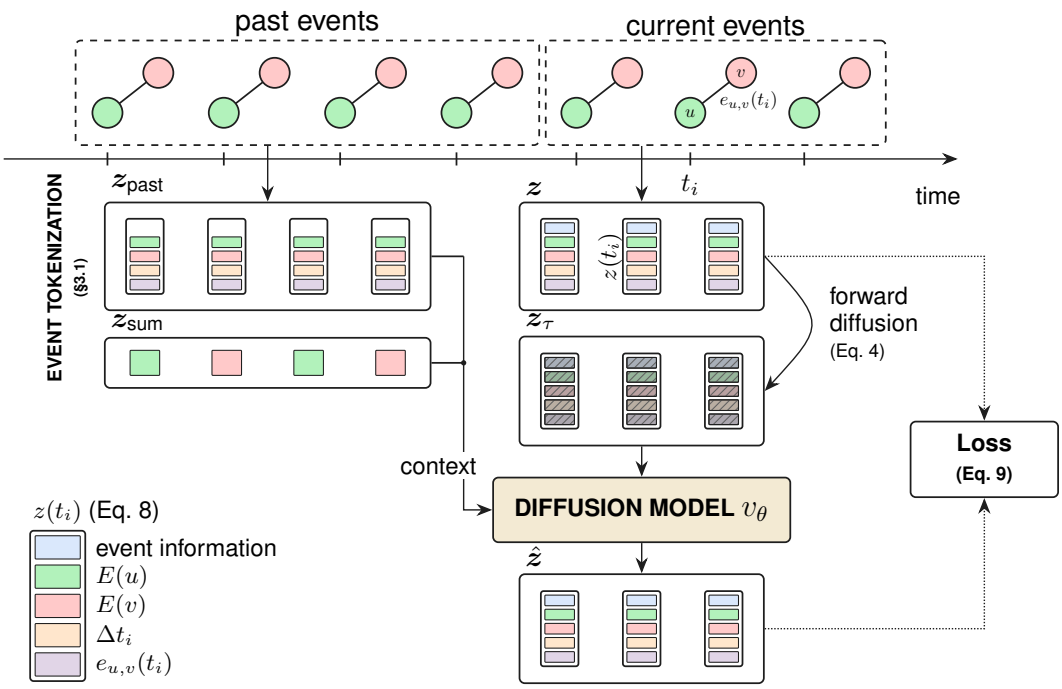

Figure 1: **Dynamic Graph Diffusion (DyGDi) training process.** The graph is represented as a sequence of events and tokenized by adding event information, with each node identity encoded by a continuous graph encoder. A window of past events serves as context, while the current window of events is noised and predicted. The diffusion model generates multiple events and their features simultaneously.

The network v_θ is trained to match the target velocity field with a mean-squared-error objective:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{x_0 \sim p_0,\, x_1 \sim p_1,\, \tau \sim \mathcal{U}(0,1)} \left\| v_\theta\left(x_\tau, \tau\right) - v^*\left(x_\tau, \tau\right) \right\|_2^2 . \tag{5}$$

At generation time, new samples are obtained by integrating the learned ODE in Equation (3) from $x_0 \sim p_0$ to $x_1 \sim p_1$.

Conditional Generation. Generation can be conditioned on auxiliary information c (*e.g.* context, class, or history), by extending the velocity field and its target to $v_\theta(x_\tau, \tau \mid c)$ and $v^*(x_\tau, \tau \mid c) = x_1 - x_0$. The training objective in Equation (5) then becomes

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{x_0 \sim p_0,\, x_1 \sim p_1,\, \tau \sim \mathcal{U}(0,1)} \left\| v_\theta\left(x_\tau, \tau \mid c\right) - v^*\left(x_\tau, \tau\right) \right\|_2^2 , \tag{6}$$

and samples are generated for a fixed condition c by integrating $\dot{x} = v_\theta(x, \tau \mid c)$ from $x_0 \sim p_0$ to $x_1 \sim p_1$.

3 The Proposed Event Diffusion Model

In Dynamic Graph Diffusion (DyGDi), the underlying problem of dynamic graph generation is cast as a sequence-to-sequence problem. Consequently, events are treated as a sequence of tokens, and the diffusion process operates directly on this token sequence. The diffusion model is conditioned on past event embeddings and denoises a window of current event embeddings. The embeddings are produced by a memory module that learns a mapping from node identifiers to latent vectors, and the inverse mapping from an embedding back to a node identifier is performed at generation time via a nearest-neighbour search. An overview of DyGDi is provided in Figure 1.

3.1 Dynamic Graph Generation as a Sequence-to-Sequence Problem

Event Tokenization. To enable the diffusion process to operate on a CTDG, each event is mapped to a fixed-dimensional token. For every node $i \in \mathcal{V}$, we obtain a latent representation through an

embedding function $E : \mathcal{V} \to \mathbb{R}^{d_n}$ implemented by a memory module (a Temporal Graph Network, or TGN (Rossi et al., 2020), in our instantiation), where d_n denotes the node embedding dimension. The memory module maintains a fixed-size bank of N slots $\boldsymbol{M} = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N\} \subset \mathbb{R}^{d_n}$, where N is a hyperparameter that upper-bounds the number of nodes the model can track. Each slot is associated with at most one node identifier, and slots that have not yet been bound to an observed identifier remain at their zero initialisation. Denoting by $\mathcal{V}_{\text{seen}} \subseteq \mathcal{V}$ the set of node identifiers observed in at least one event prior to the current tokenization step, we define

$$E(i) = \boldsymbol{s}_i \text{ if } i \in \mathcal{V}_{\text{seen}}, \text{ and } \mathbf{0} \text{ otherwise.} \tag{7}$$

Given an event $x(t_k) = (u, v, t_k, \boldsymbol{e}_{u,v}(t_k))$ with source and destination identifiers $u, v \in \mathcal{V}$, its token is obtained by concatenating the source and destination node embeddings, the inter-event interval $\Delta t_k = t_k - t_{k-1}$, and the edge feature:

$$z(t_k) = \text{concat}\left[E(u), E(v), \Delta t_k, \boldsymbol{e}_{u,v}(t_k)\right] \in \mathbb{R}^d. \tag{8}$$

A CTDG is thereby mapped to an ordered sequence of tokens $(z(t_1), z(t_2), \ldots)$.

Predicting a Window of Events. At both training and inference time, DyGDi generates a sliding window of L current event tokens $\boldsymbol{z} = (z(t_{k+1}), \ldots, z(t_{k+L}))$, conditioned on a context of M past tokens $\boldsymbol{z}_{\text{past}} = (z(t_{k-M+1}), \ldots, z(t_k))$ that summarises the recent history of the graph. In addition, DyGDi is provided with a graph-level summary of the CTDG, $\boldsymbol{z}_{\text{sum}} = \{E(i) : i \in \mathcal{V}_{\text{seen}}\}$, the embedding of every seen node (i.e. the raw memory state maintained by the TGN memory module), summarising that node's history up to t_k, although a recurrent network maintaining a hidden state may be used. The model thus learns the joint conditional distribution $p(\boldsymbol{z} \mid \boldsymbol{z}_{\text{past}}, \boldsymbol{z}_{\text{sum}})$, from which the next event tokens are sampled.

Following the conditional flow-matching framework of Equation (6) with condition $c = (\boldsymbol{z}_{\text{past}}, \boldsymbol{z}_{\text{sum}})$, the velocity network v_θ is optimised with the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{z_0 \sim p_0,\, z_1 \sim p_1,\, \tau \sim \mathcal{U}(0,1)} \left\| v_\theta\left(z_\tau, \tau | \boldsymbol{z}_{\text{past}}, \boldsymbol{z}_{\text{sum}}\right) - v^*\left(z_\tau, \tau\right) \right\|_2^2. \tag{9}$$

3.2 Generation Process

Generating a window of events with DyGDi proceeds in three steps, (i) sampling a window of tokens from the conditional distribution $p(\boldsymbol{z} \mid \boldsymbol{z}_{\text{past}}, \boldsymbol{z}_{\text{sum}})$, (ii) matching each token embedding either to an existing node identifier or to a new one, and (iii) updating the memory module accordingly and iteratively repeating. Specifically, let $\boldsymbol{z} \sim q(\boldsymbol{z} \mid \boldsymbol{z}_{\text{past}}, \boldsymbol{z}_{\text{sum}})$ be a generated window of L tokens, and let $\hat{s}$ denote a node embedding extracted from token $z(t_k)$. The associated node identifier is sampled from a categorical distribution over $\mathcal{V}$, $\hat{id} \sim p_{\text{match}}(\cdot \mid \hat{s})$, where the probability of selecting node i is

$$p_{\text{match}}(i \mid \hat{s}) = \frac{e^{d(\hat{s},\, E(i))/\gamma}}{\sum_{j \in \mathcal{V}} e^{d(\hat{s},\, E(j))/\gamma}}, \tag{10}$$

with $d(\cdot, \cdot)$ a similarity score (we use cosine similarity) and γ a temperature parameter.

New-Embedding Collapse. However, the formulation in Equation (10) becomes problematic when node identifiers must be sampled from $\mathcal{V} \setminus \mathcal{V}_{\text{seen}}$, the set of previously unseen nodes. This induces two failure modes, one on each side of the matching score $d(\hat{s},\, E(i))$, namely *memory matching collapse*, a collapse of the match targets $E(i)$, and *window matching collapse*, a collapse of the match queries $\hat{s}$.

• *Memory matching collapse*: For $i, j \in \mathcal{V} \setminus \mathcal{V}_{\text{seen}}$ with $i \neq j$, the memory embeddings collapse, $E(i) = E(j) = \mathbf{0}$ (Equation (7)), so the match targets are indistinguishable, and the same node recurring in the window may be mapped to different identifiers.

• *Window matching collapse*: For $i, j \in \mathcal{V} \setminus \mathcal{V}_{\text{seen}}$ with $i \neq j$, the generated embeddings collapse, $s_i = s_j$, so the match queries are indistinguishable, and different nodes may be mapped to the same identifier.

Event-Class Information and Random Embeddings for New Nodes. To alleviate the aforementioned issues, we introduce *event class information* attached to each token and initialise, during training, every unseen node in a window with a random Gaussian embedding. To track node identity across the window, for each node participating in an event we associate a one-hot class vector $\boldsymbol{c}$

indicating whether the node has already been seen, is a new node appearing for the first time in the window, or is a new node already seen earlier in the window. The tokenization in Equation (8) thus becomes:

$$z(t_k) = \text{concat}\left[\boldsymbol{c}_u, \boldsymbol{c}_v, E(u), E(v), \Delta t_k, \boldsymbol{e}_{u,v}(t_k)\right] \in \mathbb{R}^d. \tag{11}$$

This decomposition allows nodes to be sampled iteratively from the appropriate pool: a node assigned to the *seen* class is sampled from $\mathcal{V}_{\text{seen}}$, a node assigned to the *new* class is sampled from $\mathcal{V} \setminus \mathcal{V}_{\text{seen}}$, and a new node already observed within the current window is sampled from the pool of previously matched new nodes.

Distinct embeddings remove the representational collapse underlying both. During training each new node receives a random Gaussian draw, giving distinct new nodes distinct memory targets, and hence distinct generated queries:

$$E(i) = \boldsymbol{s}_i \text{ if } i \in \mathcal{V}_{\text{seen}}, \text{ and } \xi_i \sim \mathcal{N}(0, \boldsymbol{I}) \text{ otherwise.} \tag{12}$$

These distinct targets resolve window matching collapse and give the class-vector routing the separation to re-identify a recurring new node, which resolves memory matching collapse. Both the class vectors and the random embeddings serve as input during training. At generation time, the class vectors are instead generated by the model and route the iterative sampling described above. The contribution of these components, together with the conditioning signals $\boldsymbol{z}_{\text{sum}}$ and $\boldsymbol{z}_{\text{past}}$, is examined in the ablation study of Appendix A.9.

3.3 Link Prediction

DyGDi enables zero-shot, parameter-free link prediction. To derive a link predictor from a trained CTDG generative diffusion model, we leverage the density-estimation (Liu et al., 2025) properties of the diffusion process and estimate the conditional edge probability between a node pair (u, v) from the reconstruction error after a forward-diffusion step. Intuitively, positive edges drawn from the real data distribution should yield a lower reconstruction error than negative edges. We adopt a simple formulation for the reconstruction score $score$:

$$score = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\boldsymbol{I}), \tau \sim Strategy}\left[-\sum_{f=1}^{F} \frac{1}{\|z^f\|} d^f\left(z_\tau^f + (1-\tau) v_\theta(z_\tau, \tau)^f, z^f\right)\right] \tag{13}$$

where F is the number of components of z (Equation (11)), z^f denotes the f-th component of z, and $d^f(\cdot)$ is a distance function (we use the mean absolute error). The distances are averaged over multiple random noise samples and time steps (we use uniform time-step sampling, although a fixed τ^* could be further fine-tuned). This formulation is similar to casting link prediction as anomaly detection (Wolleb et al., 2022), where negative edges lie off the learned data manifold and so produce a high reconstruction error.

4 Related Work

Dynamic graph generative models operate in either a transductive or an inductive setting; we review both, then prior couplings of diffusion models with graphs for signal generation.

Transductive dynamic graph generation. Transductive models generate interactions only among existing nodes. They cannot introduce new nodes and often replicate the training interactions (Gupta et al., 2022). TagGen (Zhou et al., 2020) builds a static graph with timestamped edges, samples synthetic random walks, keeps those a classifier judges realistic, and aggregates them into the dynamic graph. Temporal edges restrict generation to known timesteps, and it supports neither node features nor new identities. Gupta et al. (2022) report that it regenerates 80% of the training edges and, by modelling an adjacency matrix, captures only short-range dependencies, limiting it to roughly 200 timestamps. Dymond (Zeno et al., 2021) models 3-node motifs that arrive at a constant rate and persist once formed. Unlike TagGen, it does not replicate the training data but instead deviates from the source graph's statistics, while remaining confined to a similarly short horizon (Hosseini et al., 2025).

Inductive dynamic graph generation. These models additionally introduce new nodes, enabling operation on unseen graphs while preventing leakage of source-graph node identities. Tigger (Gupta

et al., 2022) autoregressively models random walks with an RNN and merges them into the synthetic graph. Tigger-I extends this to the inductive setting by reducing the continuous graph to a static one and applying GraphSAGE (Hamilton et al., 2017) and a WGAN (Arjovsky et al., 2017). The static reduction hampers scaling to large graphs, and the node-time-sequence representation prevents the use of node and edge features. Furthermore, random-walk sampling and merging captures walk-level statistics but prevents precise conditioning on individual temporal interactions. DG-Gen (Hosseini et al., 2025) pairs a TGN (Rossi et al., 2020) encoder with a small decoder that factorises the interaction probability over edge components, enabling large-scale inductive autoregressive generation with node and edge features and arbitrary timestamps. From the TGN node embeddings, the decoder samples, for each event, a source node, then a destination conditioned on the source, and finally the time and edge features via an RNN conditioned on the source–destination edge. However, generation remains autoregressive and therefore slow. To accelerate it, DG-Gen samples batches of events, where each event in a batch is sampled independently of the others, sacrificing the coherence that sequential sampling provided.

Other diffusion models on graphs and signal generation. Prior work on signal generation has coupled diffusion models with graphs, generating signals on a graph rather than the graph itself. DiffSTG (Wen et al., 2023) performs probabilistic forecasting of node signals on a spatio-temporal graph, *i.e*. a fixed sensor graph (*e.g*. traffic, air quality) whose node values evolve over time. The topology is a static input, and only signal values are generated, which amounts to predicting new node values on a static graph. TSGDiff (Shen et al., 2026) generates time series through a representation it calls "dynamic graphs", but these are a deterministic intermediate encoding. Each signal window becomes a graph whose nodes are time steps and whose edges link consecutive and Fourier-periodic neighbours. A VAE-style GNN embeds this graph, an MLP diffusion models the latent, and the decoder reconstructs only the series values. The adjacency is never decoded or sampled. Both approaches contrast with our task, where the generated object is the graph itself: a continuous-time stream of interaction events, with node identities and inter-event times part of the sampled distribution. To the best of our knowledge, this work is the first diffusion-based generative model for CTDGs.

5 Results and Discussion

This section shows that DyGDi produces higher-quality features than existing graph generative models (Section 5.2); generates higher-quality topologies (Section 5.2); scales to more feature dimensions (Section 5.3), with fast training (Appendix A.4) and generation (Section 5.4); remains efficient in speed and memory across generation window and context sizes (Appendices A.5 and A.6); and performs link prediction without fine-tuning, highlighting the potential and flexibility of diffusion models for graph generation (Appendix A.7).

5.1 Experimental Methodology

Baselines and Datasets. For the *generative task*, DyGDi is evaluated against the state-of-the-art inductive CTDG generative models Tigger (Gupta et al., 2022) and DG-Gen (Hosseini et al., 2025), both detailed in Section 4. Recent studies (Hosseini et al., 2024; Gupta et al., 2022; Rossi et al., 2020) report that very few approaches can perform inductive CTDG generation, and to the best of our knowledge only DG-Gen can perform inductive generation jointly with message features. For the *link prediction task*, we benchmark against JODIE, DyRep, TGAT, TGN, CAWN, EdgeBank, TCL, GraphMixer, DyGFormer, and DG-Gen. All methods are evaluated on standard benchmarks for CTDG generation and link prediction, namely the Reddit, Wikipedia, LastFM, and MOOC datasets (Kumar et al., 2019). See Appendix A.2 for dataset statistics and further implementation details. Following DG-Gen, and for a fair comparison, all models are first trained and evaluated on the first 12 features of Reddit and Wikipedia. We additionally scale to the full 172-feature set (Section 5.3).

Evaluation Metrics. The quality of the generated graphs is evaluated using established metrics (Hosseini et al., 2025; Gupta et al., 2022; Hosseini et al., 2024). *Feature quality*: is evaluated using the Jensen–Shannon (JS) distance, computed between the histograms of the marginal distributions of all interaction features and between the joint histograms of every pair of features. The Wasserstein distance (WD) and Maximum Mean Discrepancy (MMD), both computed directly on the generated features without binning, are also reported as complementary measures.

Table 1: **Feature generation quality.** Mean and standard deviation of the Jensen–Shannon distance between the joint pairwise feature histograms, and of the Wasserstein distance computed on the raw features. The per-feature (1D) JS distance and the MMD are reported in Table 7 (Appendix A.3).

	Model	Pairwise JS↓	Wasserstein↓
Reddit	DG-Gen	0.27 ± 0.08	0.26 ± 0.09
	DyGDi	$\mathbf{0.19 \pm 0.12}$	$\mathbf{0.06 \pm 0.03}$
Wiki.	DG-Gen	0.18 ± 0.14	0.11 ± 0.10
	DyGDi	$\mathbf{0.17 \pm 0.18}$	$\mathbf{0.05 \pm 0.02}$
MOOC	DG-Gen	0.15 ± 0.11	$\mathbf{0.09 \pm 0.04}$
	DyGDi	$\mathbf{0.11 \pm 0.04}$	0.10 ± 0.04

Topological properties: As in Gupta et al. (2022); Hosseini et al. (2025), the generated graphs are evaluated by computing static descriptors on discrete-time graph snapshots at multiple temporal resolutions and comparing them to those of the real graphs; the Median Absolute Error is reported for each descriptor. For node-level metrics capturing local topology, we report Node Degree (ND), Closeness Centrality (CLO), and Betweenness Centrality (BC). For snapshot-level metrics capturing global structure, we use Wedge Count (WC), Power Law Exponent of the node degree (PLE), Number of Connected Components (NCC), Largest Connected Component (LCC), and Relative Edge Distribution Entropy (REL). These metrics probe four complementary axes: ND measures the graph density. WC, PLE, and REL capture the shape of the degree distribution. CLO and BC assess the path structure. NCC and LCC evaluate the connectivity of the generated graphs. In addition, we report the global Johnson–Lindenstrauss (JL) similarity metric (Hosseini et al., 2024). The JL metric applies a random matrix projection to the graph and summarises sensitivity to topology, features, and time within a single scalar value. For the link prediction task, the Average Precision (AP) and Area Under the Curve (AUC) are reported.

5.2 Generation Quality

Feature Quality. Table 1 reports the pairwise JS distance and the WD for each model. The per-feature (1D) JS distance and the MMD are detailed in Appendix A.3. As Tigger does not produce features, it is omitted from this table. The LastFM dataset contains no features and is also omitted. We observe that DyGDi produces high-quality features and consistently outperforms or matches DG-Gen across datasets and metrics, including on both metrics on Reddit and Wikipedia. The gains are largest on the Wasserstein distance, indicating that DyGDi places the feature mass correctly and does so uniformly across features rather than just on average. The pairwise JS distance improves on all three datasets, showing that this improvement extends to the dependencies between features and not only to the marginals (Appendix A.3).

On Reddit, DyGDi reduces the Wasserstein distance by a factor of ~ 4.2. This suggests that DG-Gen more often generates out-of-distribution feature values, whereas DyGDi fits the data distribution more tightly. Results are closer on Wikipedia, where both models obtain similar pairwise JS and Wasserstein distances. On MOOC, DyGDi improves the pairwise JS distance by 40% with a much tighter cross-feature spread, while both models obtain comparable Wasserstein distances.

Topology Quality. Table 2 evaluates the topological properties of the generated graphs. Following previous works (Gupta et al., 2022; Hosseini et al., 2024; 2025), we compute per-window graph statistics and compare them against those of the corresponding real window. As a complement, we report the JL similarity (Hosseini et al., 2024), which embeds the dynamic graph and jointly captures temporal structure, features, and topology—information that feature-only or topology-only window-level statistics cannot see. As Tigger cannot generate features, its JL similarity is omitted on all datasets except LastFM, which has no features. Notably, DyGDi achieves the best JL similarity on all four datasets, outperforms DG-Gen on 21 of the 36 dataset–metric comparisons (with 4 ties), and Tigger on 32 out of 36. This shows that DyGDi generates topologically faithful graphs, with the density, degree shape, and path structure of its windows remaining within the scale of the real statistics, whereas both baselines produce spurious hubs and inflated components. Because the JL similarity embeds the full event stream rather than marginal per-window statistics, this gain further shows that DyGDi reproduces the real joint temporal structure, not merely individually plausible windows. In Appendix A.8, we verify the robustness of these results to different time-window sizes (512 to 4096), confirming the advantages of DyGDi.

Reddit has the largest dynamic range (its real windows exhibit high WC, NCC, and LCC values) and is where DyGDi performs best, outperforming Tigger and DG-Gen on 8 of 9 metrics and coming very close to DG-Gen on PLE. Both baselines approximate the real data density (their ND remains

Table 2: **Topology generation quality.** Each cell reports the median absolute error between the generated and real per-snapshot statistics. The (percentage) beside it rescales this error by the magnitude of the real statistic (its median over windows). The *Real Data* rows give the median of each real statistic over windows, *i.e.* the scale against which each error should be judged.

	Model	ND ↓	CLO ↓	BC ↓	WC ↓	PLE ↓	NCC ↓	LCC ↓	REL ↓	JL↑
Reddit	Real Data	1.32	1.39e−2	9.62e−5	178	3.18	58	14	9.57e−1	
	Tigger	1.17 (88%)	2.09e−1 (1502%)	4.10e−3 (4265%)	9855 (5537%)	8.54e−1 (27%)	29 (50%)	351 (2507%)	1.09e−1 (11%)	-
	DG-Gen	6.01e−1 (45%)	2.83e−1 (2040%)	8.79e−3 (9140%)	4207 (2363%)	**2.58e−1 (8%)**	48 (83%)	104 (743%)	1.97e−1 (21%)	0.23
	DyGDi	**1.49e−1 (11%)**	**3.90e−3 (28%)**	**7.21e−5 (75%)**	**146 (82%)**	2.99e−1 (9%)	**15 (26%)**	**7 (50%)**	**2.74e−2 (3%)**	**0.26**
Wikipedia	Real Data	1.04	3.51e−2	6.17e−5	1	5.09	15	3	9.96e−1	
	Tigger	3.13e−1 (30%)	1.20e−2 (34%)	1.13e−3 (1840%)	53 (5300%)	2.23 (44%)	9 (60%)	13 (433%)	3.45e−2 (3%)	-
	DG-Gen	**3.23e−2 (3%)**	**9.70e−3 (28%)**	**6.21e−5 (101%)**	**1 (100%)**	8.96e1 (1761%)	**5 (33%)**	**1 (33%)**	**3.37e−3 (0%)**	0.12
	DyGDi	3.68e−2 (4%)	1.13e−2 (32%)	8.52e−5 (138%)	**1 (100%)**	**1.26 (25%)**	**5 (33%)**	**1 (33%)**	3.83e−3 (0%)	**0.13**
MOOC	Real Data	1.97	1.28e−1	2.25e−2	133	2.09	6	36	9.54e−1	
	Tigger	8.15 (414%)	2.45e−1 (192%)	3.53e−2 (157%)	634 991 (477 437%)	**4.68e−1 (22%)**	**5 (83%)**	1052 (2922%)	1.03e−1 (11%)	-
	DG-Gen	**3.99e−1 (20%)**	**7.49e−2 (59%)**	**1.88e−2 (84%)**	203 (153%)	6.33e−1 (30%)	8 (133%)	**27 (75%)**	4.96e−2 (5%)	0.06
	DyGDi	5.00e−1 (25%)	1.06e−1 (83%)	2.05e−2 (91%)	**129 (97%)**	6.17e−1 (30%)	45 (750%)	**27 (75%)**	**2.03e−2 (2%)**	**0.32**
LastFM	Real Data	1.05	1.43e−1	3.42e−4	1	4.65	5	3	9.94e−1	
	Tigger	1.37 (130%)	1.49e−1 (104%)	2.37e−2 (6931%)	419 (41 900%)	2.32 (50%)	**2 (40%)**	63 (2100%)	8.92e−2 (9%)	0.11
	DG-Gen	5.88e−2 (6%)	**8.89e−2 (62%)**	4.90e−4 (143%)	1 (100%)	9.03e1 (1941%)	**2 (40%)**	1 (33%)	7.00e−3 (1%)	0.15
	DyGDi	**0.0 (0%)**	2.67e−1 (187%)	**0.0 (0%)**	**0 (0%)**	**0.0 (0%)**	3 (60%)	**0 (0%)**	**1.13e−4 (0%)**	**0.17**

within the real-data range) but concentrate the edges on only a few nodes, creating large hubs and clusters that are absent from the real data: their WC errors reach 23–55× the real values, their LCC errors 7–25× the real sizes, and their CLO and BC errors 15–91× the real values. In contrast, DyGDi is the only model whose errors remain below the real-data scale on density, degree shape, path structure, and connectivity.

On Wikipedia and LastFM, both DG-Gen and DyGDi clearly outperform Tigger on the topological metrics and match the real window statistics, the exception being DG-Gen's degree-tail shape (PLE), a poor fit, whereas DyGDi stays within 25% of the real value. This error should be read with caution, as the power-law exponent is highly sensitive and becomes ill-posed when almost all nodes have degree one. The JL similarity, which sees topology and features jointly, still discriminates between the models and favors DyGDi over Tigger and DG-Gen on both datasets.

On MOOC, DG-Gen and DyGDi perform comparably on most topological metrics, except NCC, where DyGDi's error is large (∼7.5× the real-data scale), over-fragmenting its graphs into too many connected components. Despite this, the JL similarity rates DG-Gen five times worse than DyGDi, indicating that even though each individual DG-Gen window is plausible, the windows do not compose into a realistic stream of events.

5.3 Scaling to More Features

To evaluate the ability of DyGDi to scale to a higher number of features, additional trainings are performed on the Reddit and Wikipedia datasets using all 172 features instead of the first 12 (Section 5.2). Table 3 reports the feature and topology metrics and the JL similarity for this setting. To save space, the MMD and per-feature JS distance are omitted. Because the added features are far more atomic than the first 12 (the most frequent value covers a median of 89% of the samples on Reddit and 97% on Wikipedia), the MMD saturates at the same value for both models, as on MOOC (Appendix A.3). Likewise, only a reduced set of topological metrics that still covers the four structural axes is reported (ND; WC; CLO and BC; NCC), omitting PLE, REL, and LCC, which overlap with a reported metric on the same axis.

DyGDi scales better to a high number of features, as illustrated by the pairwise JS distance (7% and 38% lower on Reddit and Wikipedia, respectively) and by the Wasserstein distance (2.3× and 1.7× lower), the latter indicating that DG-Gen produces out-of-distribution feature values that heavily penalize the distance. The topology of DG-Gen remains plausible on both datasets, on par with DyGDi. This can be explained by its sequential generation, where the nodes of an event are sampled first and the associated features are then generated conditionally, so the feature dimensionality does not directly affect edge selection. The JL similarity is the most telling indicator. From 12 (Table 2) to 172 features, that of DG-Gen drops by 44% on Reddit (resp. 65% on Wikipedia), whereas that of DyGDi drops by only 14% (resp. 16%). This widens the gap between the two models from ∼1.1× to 1.7× (resp. 2.6×). Since the per-window topology of DG-Gen is preserved, this drop is explained

Table 3: Feature and topology generation quality, using all 172 features of Reddit and Wikipedia.

	Model	Pairwise JS↓	Wasserstein↓	ND ↓	CLO ↓	BC ↓	WC ↓	NCC ↓	JL↑
Reddit	Real Data	- ± -	- ± -	1.32	1.39e−2	9.62e−5	178	58	
	DG-Gen	0.19 ± 0.11	0.24 ± 0.13	1.55e−1 (12%)	7.76e−3 (56%)	8.54e−5 (89%)	**102 (57%)**	34 (59%)	0.13
	DyGDi	**0.17 ± 0.12**	**0.10 ± 0.03**	**8.54e−2 (6%)**	**5.61e−3 (40%)**	**7.36e−5 (76%)**	130 (73%)	**13 (22%)**	**0.22**
Wiki.	Real Data	- ± -	- ± -	1.04	3.51e−2	6.17e−5	1	15	
	DG-Gen	0.18 ± 0.11	0.19 ± 0.15	**4.04e−2 (4%)**	**1.02e−2 (29%)**	1.14e−4 (185%)	**2 (200%)**	**4 (27%)**	0.04
	DyGDi	**0.11 ± 0.11**	**0.11 ± 0.03**	5.88e−2 (6%)	1.41e−2 (40%)	**1.09e−4 (177%)**	3 (300%)	8 (53%)	**0.11**

Table 4: **Generation Speed** on the Reddit dataset for 12 features (Reddit 12F) and 172 features (Reddit 172F). DyGDi lines with a lower Pairwise JS distance than DG-Gen are highlighted in green. For DyGDi, the number of generation steps used is reported next to the name. Speeds are averaged over 10 runs.

	Model	Pairwise JS↓	Wasserstein↓	ND ↓	CLO ↓	BC ↓	WC ↓	NCC ↓	JL↑	Events/s↑
Reddit 12F.	Tigger	- ± -	- ± -	1.17 (88%)	2.09e−1 (1502%)	4.10e−3 (4265%)	9855 (5537%)	29 (50%)	-	425
	DG-Gen	0.27 ± 0.08	0.26 ± 0.09	6.01e−1 (45%)	2.83e−1 (2040%)	8.79e−3 (9140%)	4207 (2363%)	48 (83%)	0.23	326
	DyGDi 1	0.70 ± 0.07	0.61 ± 0.07	1.93e−1 (15%)	9.04e−3 (65%)	9.01e−5 (94%)	119 (67%)	55 (95%)	0.12	10 192
	DyGDi 5	0.29 ± 0.10	0.20 ± 0.02	1.83e−1 (14%)	3.30e−3 (24%)	6.16e−5 (64%)	154 (87%)	16 (28%)	0.18	5 694
	DyGDi 10	0.23 ± 0.11	0.12 ± 0.02	2.01e−1 (15%)	4.33e−3 (31%)	6.71e−5 (70%)	158 (89%)	22 (38%)	0.19	3 536
	DyGDi 20	0.21 ± 0.12	0.09 ± 0.02	1.97e−1 (15%)	6.04e−3 (43%)	9.58e−5 (100%)	155 (87%)	25 (43%)	0.22	2 061
	DyGDi 50	0.19 ± 0.12	0.07 ± 0.02	2.11e−1 (16%)	5.99e−3 (43%)	9.00e−5 (94%)	164 (92%)	23 (40%)	0.16	915
	DyGDi 100	0.19 ± 0.12	0.06 ± 0.03	1.49e−1 (11%)	3.90e−3 (28%)	7.21e−5 (75%)	146 (82%)	15 (26%)	0.26	498
Reddit 172F.	DG-Gen	0.19 ± 0.11	0.24 ± 0.13	1.55e−1 (12%)	7.76e−3 (56%)	8.54e−5 (89%)	102 (57%)	34 (59%)	0.13	24
	DyGDi 1	0.43 ± 0.27	0.41 ± 0.18	2.77e−1 (21%)	9.52e−3 (69%)	9.38e−5 (97%)	158 (89%)	63 (109%)	0.11	9 539
	DyGDi 5	0.22 ± 0.12	0.17 ± 0.05	8.84e−2 (7%)	3.14e−3 (23%)	4.72e−5 (49%)	84 (47%)	10 (17%)	0.15	6 559
	DyGDi 10	0.19 ± 0.12	0.13 ± 0.03	7.59e−2 (6%)	2.81e−3 (20%)	4.64e−5 (48%)	94 (53%)	12 (21%)	0.19	4 169
	DyGDi 20	0.18 ± 0.12	0.11 ± 0.03	6.79e−2 (5%)	5.39e−3 (39%)	7.05e−5 (73%)	128 (72%)	12 (21%)	0.19	2 426
	DyGDi 50	0.17 ± 0.12	0.11 ± 0.03	8.06e−2 (6%)	5.28e−3 (38%)	7.16e−5 (74%)	134 (75%)	11 (19%)	0.19	1 070
	DyGDi 100	0.17 ± 0.12	0.10 ± 0.03	8.54e−2 (6%)	5.61e−3 (40%)	7.36e−5 (76%)	130 (73%)	13 (22%)	0.22	583

by the degradation of its feature quality within the joint stream. Together, these results show that DyGDi scales better to a high number of features.

5.4 Quality vs. Generation Efficiency

In Table 4, we report the average number of events generated per second on Reddit with both 12 (Reddit 12F) and 172 (Reddit 172F) features for DyGDi and DG-Gen (Tigger does not generate features). For DyGDi, the number of flow-matching sampling steps (NBS) is listed alongside. All models were run under the same setting using an A100, with speeds averaged over 10 runs. Tigger's inductive generation runs differently from DG-Gen and DyGDi, as it requires sampling random walks and then merging them in a discrete fashion to match window statistics to real-graph statistics. Note that DyGDi's generation speed is largely unaffected by the feature count, since its cost is dominated by the transformer operating over the fixed-length event sequence and the feature dimension enters only through a lightweight projection. On Reddit 12F, DyGDi is faster than DG-Gen at every step count, from $2\times$ at 100-NBS up to $11\times$ at 10-NBS, which the flow-matching framework enables with minimal loss in generation quality (Pairwise JS stays almost the same). Scaling to 172 features on Reddit 172F slows DG-Gen by $14\times$ relative to its 12-feature counterpart, as its RNN does not generate in a single step. The gap to DyGDi thus widens to $25\times$ at 100-NBS and $175\times$ at 10-NBS while matching DG-Gen's generation quality. This highlights the potential of the DyGDi framework not only to train faster (Appendix A.4) but also to generate dynamic graphs in real time at scale.

6 Conclusion

In this paper, we introduced DyGDi, a framework for dynamic graph generation. Unlike prior dynamic graph generation approaches, DyGDi uses a diffusion-based method to generate both the topology and the event features. This design enables faster training, better scaling, and generation of large graphs, in terms of both features and topology. As a result, DyGDi is a promising approach for dynamic graph generation in real-world applications, where the number of events and features can be very large. Furthermore, the use of diffusion enables flexible generation, opening up new applications such as zero-shot link prediction.

AI USE STATEMENT

In this work, we used generative AI tools for implement methods: especially for checking the correctness of the code, metrics, and data loading, assist with translation, support qualitative and thematic data analysis, interpret results: by asking the tools to verify the soundness and correctness of our analysis and, in other cases, to provide feedback and corrections, create or modify scientific figures or images: by using the tools to assist in generating the code that creates the figures, edit a research paper to improve readability: by asking the tools to check for spelling mistakes and grammar errors .

We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, clean and reformat dataset .

We have reviewed all AI-assisted work, by checking generated text and code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce our results throughout the paper and appendix. The full experimental setup of DyGDi is described in Appendix A.2, and all model, training, and generation hyperparameters are reported in Appendix A.2.3 and Table 6. All experiments use only publicly available benchmark datasets, so no private or newly collected data is required to reproduce our results. Upon acceptance, we will release our source code and the trained model checkpoints in a public GitHub repository, whose link will be added in the final version of the paper.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. Pmlr, 2017.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. arXiv, October 2024. URL https://openreview.net/forum?id=tyEyYT267x.

Tobias Bernecker, Ghalia Rehawi, Francesco Paolo Casale, Janine Knauer-Arloth, and Annalisa Marsico. Random walk diffusion for efficient large-scale graph generation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=tSFpsfndE7.

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 *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ayPPc0SyLv1.

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

Shubham Gupta, Sahil Manchanda, Srikanta Bedathur, and Sayan Ranu. TIGGER: Scalable Generative Modelling for Temporal Interaction Graphs. *AAAI*, 36(6):6819–6828, June 2022. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v36i6.20638. URL http://arxiv.org/abs/2203.03564. arXiv:2203.03564 [cs].

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022.

Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pp. 8867–8887. PMLR, 2022.

Ryien Hosseini, Filippo Simini, Venkatram Vishwanath, Rebecca Willett, and Henry Hoffmann. Quality Measures for Dynamic Graph Generative Models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, October 2024. URL https://openreview.net/forum?id=8bjspmAMBk.

Ryien Hosseini, Filippo Simini, Venkatram Vishwanath, and Henry Hoffmann. A Deep Probabilistic Framework for Continuous Time Dynamic Graph Generation. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pp. 17249–17257. AAAI Press, 2025. doi: 10.1609/AAAI.V39I16.33896. URL https://doi.org/10.1609/aaai.v39i16.33896.

Jiarui Ji, Wenda Wang, Runlin Lei, Jialin Bi, Lei Wang, Rui Wang, Qiang Wang, Bin Tong, Xu Chen, and Zhewei Wei. A Systematic Survey on Generative Models for Graph Generation, December 2025. URL https://www.preprints.org/manuscript/202512.0114/v1.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International conference on machine learning*, pp. 10362–10383. PMLR, 2022.

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*, pp. 1269–1278, 2019.

Mufei Li, Eleonora Kreacic, Vamsi K. Potluru, and Pan Li. Graphmaker: Can diffusion models generate large attributed graphs? *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=0q4zjGMKoA.

Stratis Limnios, Praveen Selvaraj, Mihai Cucuringu, Carsten Maple, Gesine Reinert, and Andrew Elliott. Sagess: A sampling graph denoising diffusion model for scalable graph generation. In *ECAI 2024: 27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain–Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, pp. 2950–2957. SAGE Publications Pvt. Ltd 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Jing Liu, Zhenchao Ma, Zepu Wang, Chenxuanyin Zou, Jiayang Ren, Zehua Wang, Liang Song, Bo Hu, Yang Liu, and Victor Leung. A survey on diffusion models for anomaly detection. *arXiv preprint arXiv:2501.11430*, 2025.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11451–11461. IEEE, 2022.

Tsiry Mayet, Simon Bernard, Romain HÉRAULT, and Clement Chatelain. Multiple noises in diffusion model for semi-supervised multi-domain translation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=vYdT26kDYM.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International conference on artificial intelligence and statistics*, pp. 4474–4484. PMLR, 2020.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35: 32928–32941, 2022.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs, October 2020. URL http://arxiv.org/abs/2006.10637. arXiv:2006.10637 [cs].

Lifeng Shen, Xuyang Li, and Lele Long. Tsgdiff: Rethinking synthetic time series generation from a pure graph perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 25304–25311, 2026.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Brian L Trippe, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and Tommi S Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. In *The Eleventh International Conference on Learning Representations*, 2023.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Representation Learning over Dynamic Graphs, March 2018. URL http://arxiv.org/abs/1803.04051. arXiv:1803.04051 [cs].

Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete Denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=UaAD-Nu86WX.

Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. TCL: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944*, 2021a.

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*, 2021b. URL https://openreview.net/forum?id=KYPz4YsCPj.

Haomin Wen, Youfang Lin, Yutong Xia, Huaiyu Wan, Qingsong Wen, Roger Zimmermann, and Yuxuan Liang. Diffstg: Probabilistic spatio-temporal graph forecasting with denoising diffusion models. In *Proceedings of the 31st ACM international conference on advances in geographic information systems*, pp. 1–12, 2023.

Christopher Wewer, Bartlomiej Pogodzinski, Bernt Schiele, and Jan Eric Lenssen. Spatial reasoning with denoising models. In *International Conference on Machine Learning*, pp. 66706–66725. PMLR, 2025.

Julia Wolleb, Florentin Bieder, Robin Sandkühler, and Philippe C Cattin. Diffusion models for medical anomaly detection. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 35–45. Springer, 2022.

Leshanshui Yang, Clément Chatelain, and Sébastien Adam. Dynamic graph representation learning with neural networks: A survey. *Ieee Access*, 12:43460–43484, 2024.

James Jian Qiao Yu and Jiatao Gu. Real-time traffic speed estimation with graph convolutional generative autoencoder. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3940–3951, 2019. doi: 10.1109/TITS.2019.2910560.

Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xHNzWHbklj.

Giselle Zeno, Timothy La Fond, and Jennifer Neville. DYMOND: DYnamic MOtif-NoDes Network Generative Model. In *Proceedings of the Web Conference 2021*, pp. 718–729, April 2021. doi: 10.1145/3442381.3450102. URL http://arxiv.org/abs/2308.00770. arXiv:2308.00770 [cs].

Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 401–411, 2020.

A Appendix

A.1 Limitations

While we have made significant progress, our model is still far from addressing every challenge in dynamic graph generation. We believe that DyGDi can achieve significantly better results in both quality and efficiency. First, our model currently uses fixed condition and generation window sizes, which may not be suitable for all applications. Furthermore, decoupling these parameters from the training dynamics could help achieve better generalization. Second, our model still requires multiple steps to generate a window of events. Diffusion methods that require fewer generation steps would be better suited to resource-constrained devices.

A.2 Implementation Details

A.2.1 Datasets and Baselines

Dataset statistics for Reddit, Wikipedia, LastFM, and MOOC (Kumar et al., 2019) are summarized in Table 5.

Table 5: Dataset statistics.

Dataset	#Nodes	#Edges	#Features	#Train	#Valid	#Test	Time span (days)
Reddit	10,984	672,447	172	537,957	67,245	67,245	31.0
Wikipedia	9,227	157,474	172	125,979	15,747	15,748	31.0
MOOC	7,144	411,749	4	329,400	41,174	41,175	29.8
LastFM	1,980	1,293,103	0	1,034,482	129,310	129,311	1586.9

Following Hosseini et al. (2025), training is performed on a 70/15/15 train/val/test split. For baselines, we use the existing code for Tigger and DG-Gen. For the link prediction task, we use the results reported by Yu et al. (2023) for all baselines except DG-Gen, following the negative-sampling protocol of Poursafaei et al. (2022), and reproduce the DG-Gen experiments using the available code.

A.2.2 Model Implementation

In DyGDi, the event encoder E (Equation (7)) is a pre-trained TGN (Rossi et al., 2020), with all hyperparameters fixed to those of Hosseini et al. (2025). The denoiser v_θ is an encoder–decoder transformer, where the encoder processes $\boldsymbol{z}_{\text{past}}$ and $\boldsymbol{z}_{\text{sum}}$, and the decoder predicts the denoised $\boldsymbol{z}$. Both $\boldsymbol{z}_{\text{past}}$ and $\boldsymbol{z}_{\text{sum}}$ are encoded by 4-layer transformer blocks with hidden size 384, 4 heads, and dropout rate 0.1. The decoder is a small DiT (Peebles & Xie, 2023) adapted for event processing, with depth 12, hidden size 384, 6 heads, and full cross-attention. We use RoPE (Su et al., 2024) positional embeddings in the decoder and in the $\boldsymbol{z}_{\text{past}}$ encoder.

A.2.3 Training and Generation Hyperparameters

The pre-trained TGN encoder E reuses the per-dataset configuration of Hosseini et al. (2025), namely the memory, embedding, and time-encoding dimensions, the neighbour budget, and whether the inter-event interval Δt is log-compressed before entering the TGN. TGN encoder is kept frozen during diffusion training.

All datasets share the same diffusion training setup, with the velocity network optimised with AdamW ($\beta = (0.9, 0.999)$, weight decay 0), a linear learning-rate warm-up over the first 5 epochs (from $0.3\times$ to $1\times$ the base rate), an exponential moving average of the weights (decay 0.999), and gradient-norm clipping at 1.0. The TGN memory is updated in chunks of 100 events, and the diffusion model consumes $\log(1+\Delta t)$ in place of the raw inter-event interval. We found the log-encoding to be necessary for LastFM mainly, as the inter-event Δt can be very large.

At generation time we integrate the flow-matching ODE with a fixed-step Euler solver for 100 steps, and match each generated embedding to a node identifier by cosine similarity (Equation (10)) with stochastic sampling.

For link prediction, the reconstruction score $score$ (Equation (13)) is averaged over 100 noise samples.

All per-dataset hyperparameters are summarised in Table 6.

Table 6: **Training and generation hyperparameters of DyGDi.** Pre-trained TGN encoder dimensions follow Hosseini et al. (2025). L and M are the sizes of the generated window $\boldsymbol{z}$ and the past context $\boldsymbol{z}_{\text{past}}$, and γ is the matching temperature. Generation always uses a 100-step Euler solver with cosine node matching (Equation (10)) and stochastic sampling. The last two rows use the full 172-dimensional feature set (Section 5.3).

	Pre-trained TGN encoder					Diffusion training			Generation
Dataset	d_{mem}	d_{emb}	d_{time}	#Nbr	$\log \Delta t$	L	M	LR	γ
Reddit	76	76	100	100	Yes	4096	4096	10^{-4}	0.05
Wikipedia	76	100	300	200	Yes	2048	4096	10^{-4}	0.3
MOOC	300	200	200	10	Yes	2048	4096	10^{-4}	0.3
LastFM	300	76	800	40	No	4096	4096	10^{-4}	0.05
Reddit (172F)	76	76	100	100	Yes	4096	4096	10^{-3}	0.3
Wikipedia (172F)	76	100	300	200	Yes	512	4096	10^{-4}	0.05

A.2.4 DG-Gen Full-Feature Adaptation

For the feature-scaling comparison (Section 5.3), we train DG-Gen on the full feature set (172 features on Reddit and Wikipedia) from the authors' released code.

The only change we make to DG-Gen's released configuration concerns its feature-noise schedule. DG-Gen adds Gaussian noise of scale ϵ to the message features and anneals it during training, rescaling ϵ by 0.1 whenever the feature loss improves by less than a relative threshold over the last T epochs, down to a floor of $\epsilon/100$, yielding three successive noise regimes. This schedule was tuned for the 12-feature setting over 50 epochs, where on Reddit it spends roughly $22\%/20\%/58\%$ of training in the three regimes. With 172 features, however, the per-epoch cost grows sharply and only about 9 epochs fit the same compute budget of 48h.

Since the first rescale cannot fire before epoch $T{+}1$ (with the released $T{=}10$ on Reddit and $T{=}20$ on Wikipedia), the entire run stays pinned at the initial, highest noise level, where the features are buried in noise and no feature learning occurs.

To restore a fair comparison, we keep the initial noise ϵ and its floor unchanged (final noise level matches the 12-feature) and only shorten the annealing window T so that each noise regime receives the same *proportion* of the (now shorter) epoch budget as before, namely $T{=}1$ on Reddit (rescaling at epochs 2 and 3, *i.e.* $\approx 22\%/11\%/67\%$) and $T{=}5$ on Wikipedia. We additionally clip the gradient norm to 1.0 for stability and, on Wikipedia, lower the learning rate to 10^{-4}.

A.3 Feature Generation Quality: 1D-Jensen–Shannon Distance and Maximum Mean Discrepancy

Table 7 extends Table 1 with per-feature (1D) JS distance and MMD. The per-feature JS distance improves on all three datasets, as DyGDi reduces it by 68% on Reddit and by 32% on MOOC, and has a slight edge on Wikipedia, showing that its marginal feature distributions are also closer to the real data. On Reddit, DyGDi further obtains a lower MMD, with a smaller gain than on the other metrics.

On Wikipedia and MOOC, DG-Gen performs better on the MMD. This lower MMD performance of DyGDi is explained by the near-discrete nature of the features, on which the median-heuristic kernel bandwidth collapses, making the MMD a less suitable metric in this case. On Wikipedia, this follows from the atom-like structure of the dataset. For each of the first 12 features, the single most frequent value alone accounts for at least 47% of the real samples. This causes the median-heuristic bandwidth of the Gaussian kernel to collapse, so the MMD heavily penalizes values that lie near, but not exactly on, these point masses. In DG-Gen, each feature is produced by an RNN with a GMM head trained by maximum likelihood. When 47% of a feature's samples concentrate on a single

Table 7: **Feature generation quality (full metric set).** Mean and standard deviation of the Jensen–Shannon distance between the marginal per-feature histograms and between the joint pairwise histograms. The Maximum Mean Discrepancy (MMD) and the Wasserstein distance are computed on the raw features.

Model	Dataset	Per-feature JS↓	Pairwise JS↓	MMD↓	Wasserstein↓
DG-Gen	Reddit	0.17 ± 0.09	0.27 ± 0.08	0.06 ± 0.14	0.26 ± 0.09
DyGDi	Reddit	$\mathbf{0.10 \pm 0.12}$	$\mathbf{0.19 \pm 0.12}$	$\mathbf{0.04 \pm 0.15}$	$\mathbf{0.06 \pm 0.03}$
DG-Gen	Wikipedia	0.11 ± 0.12	0.18 ± 0.14	$\mathbf{0.18 \pm 0.33}$	0.11 ± 0.10
DyGDi	Wikipedia	$\mathbf{0.09 \pm 0.15}$	$\mathbf{0.17 \pm 0.18}$	0.25 ± 0.35	$\mathbf{0.05 \pm 0.02}$
DG-Gen	MOOC	0.09 ± 0.11	0.15 ± 0.11	$\mathbf{0.56 \pm 0.38}$	$\mathbf{0.09 \pm 0.04}$
DyGDi	MOOC	$\mathbf{0.07 \pm 0.04}$	$\mathbf{0.11 \pm 0.04}$	0.57 ± 0.38	0.10 ± 0.04

value, the likelihood is maximized by placing one component's mean on that atom and shrinking its scale. GMMs can undergo component collapse and, on this dataset, are encouraged to do so. By contrast, DyGDi uses a diffusion process, which may be less subject to variance collapse. The low Wasserstein distance indicates that DyGDi does place the feature mass correctly as a continuous generator, but it does not reproduce the repeated values as closely as the GMM of DG-Gen. The four MOOC features are even more atomic (the most frequent value covers 87%, 83%, 64%, and 94% of the data, respectively). This makes the MMD near-saturated for both DG-Gen and DyGDi, as the bandwidth collapse turns it into an exact-match statistic that neither continuous generator can satisfy, though we still report it for completeness.

A.4 Training Speed

A.4.1 With 12 Features

Figure 2 compares the convergence speed of DyGDi and DG-Gen trained on the first 12 features. DyGDi reaches its best validation loss substantially earlier in training time than DG-Gen, and, unlike DG-Gen, does not require a noise scheduler to stabilize feature-generation convergence (Hosseini et al., 2025). This training-speed gap widens further when scaling to a higher number of features (Section 5.3). DyGDi exploits two levels of parallelism, generating all events and all features simultaneously, whereas DG-Gen relies on an RNN to generate features sequentially and processes only a small number of events in parallel during training. Figure 3 additionally reports Tigger for comparison. Note that Tigger does not generate features and follows a multi-component training pipeline. We report only its main training stage, excluding the GraphSAGE and WGAN components of Tigger-I.

A.4.2 Scaling to More Features

When scaling to a higher number of features, the convergence-speed gap between DyGDi and DG-Gen widens further, as shown in Figure 4. As above, this is due to DyGDi's two levels of parallelism, generating all windows and all features simultaneously, whereas DG-Gen relies on an RNN to generate features sequentially and processes only a small number of events in parallel during training.

A.5 Generation Speed vs. Window Size

In Figure 5, we compare the generation speed of DyGDi for different L on the Reddit 12F dataset, fixing $M = 4096$ and, more importantly, initializing $\boldsymbol{z}_{\text{sum}}$ to all $10{,}984$ nodes of Reddit 12F (Table 5). Initializing $\boldsymbol{z}_{\text{sum}}$ to such a high value gives a worst-case scenario, since earlier in inference the model typically operates with a smaller $\boldsymbol{z}_{\text{sum}}$. This also explains why, in Section 5.4, Reddit 12F is slower than its Reddit 172F counterpart. We find that on Reddit 172F DyGDi tends to reuse past nodes more, whereas on Reddit 12F it creates more new nodes early in the generation, causing $\boldsymbol{z}_{\text{sum}}$ to grow sooner. Even for smaller window sizes, DyGDi remains highly efficient in generation speed.

A.6 VRAM Consumption

In Figure 6, we report the peak VRAM consumption of DyGDi on Reddit 12F across z generation window sizes L and $\boldsymbol{z}_{\text{past}}$ context sizes M, both ranging from 256 to 4096, keeping $\boldsymbol{z}_{\text{sum}}$ full (Appendix A.5).

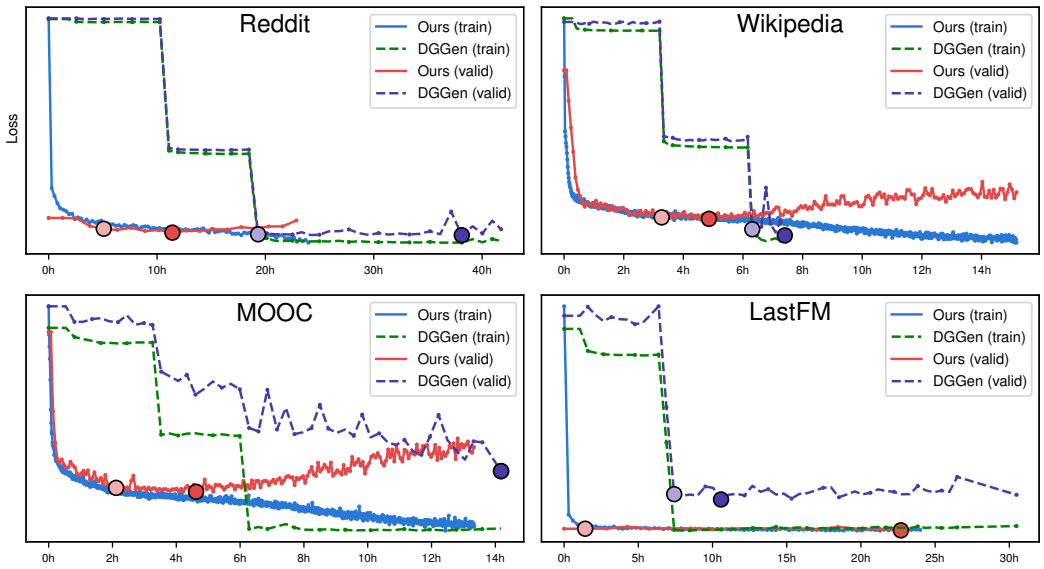

Figure 2: **Convergence speed**. Colored dots (●●) indicate the best validation loss, while the lighter dots (●●) indicate the first value within 5% of the best validation loss. Loss magnitudes should not be compared across models.

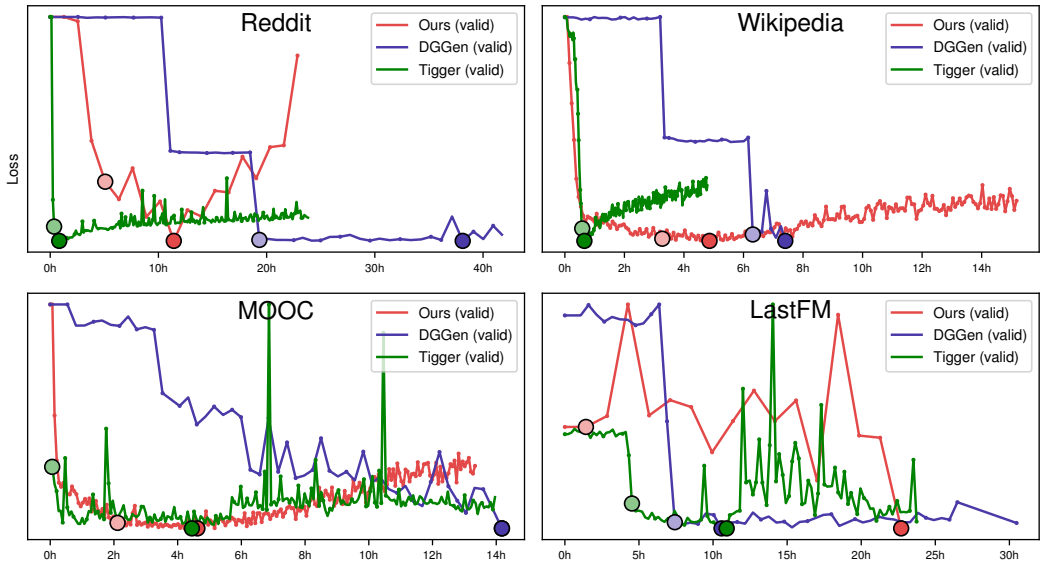

Figure 3: **Convergence speed**. Colored dots (●●●) indicate the best validation loss, while the lighter dots (●●●) indicate the first value within 5% of the best validation loss. Loss magnitudes should not be compared across models. For visibility, we omit the training loss and show only the validation loss.

Across the entire grid, the peak footprint remains modest, from 0.74 GB for the smallest configuration ($L = M = 256$) to 3.52 GB for the largest ($L = M = 4096$), so that even the most demanding setting fits comfortably on a single consumer-grade GPU.

The generation window size L is the dominant driver, as it sets the transformer sequence length. At fixed $M = 256$, increasing L from 256 to 4096 raises the footprint by roughly $4.4\times$, whereas increasing M over the same range at fixed $L = 256$ raises it by only about $2.5\times$.

Overall, DyGDi keeps memory requirements low even in its worst-case configuration, confirming its scalability to large generation windows and long context lengths.

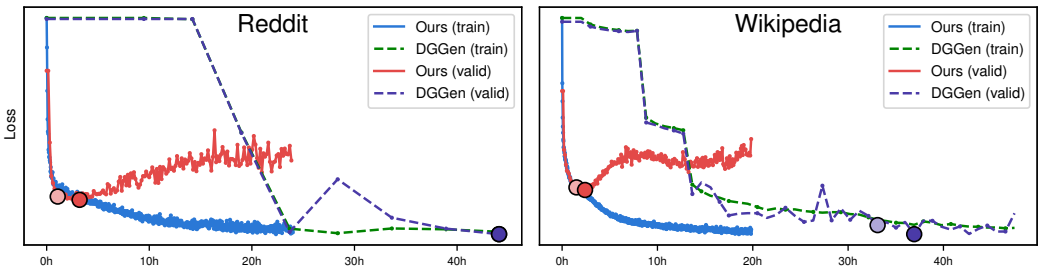

Figure 4: **Convergence speed**. Colored dots (●●) indicate the best validation loss, while the lighter dots (●●) indicate the first value within 5% of the best validation loss. Loss magnitudes should not be compared across models.

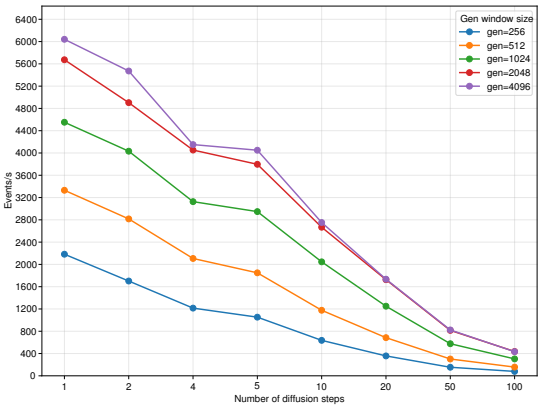

Figure 5: **Generation speed** (events/s) of DyGDi on Reddit 12F as a function of the z generation window size L, with $M = 4096$ and $\boldsymbol{z}_{\text{sum}}$ initialized to all 10,984 nodes (worst case). DyGDi stays fast even for small window sizes.

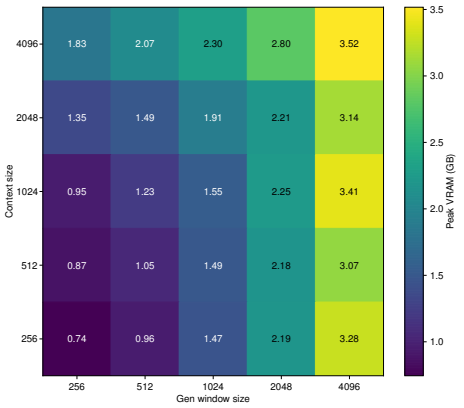

Figure 6: **Peak VRAM consumption** of DyGDi on Reddit 12F as a function of the z generation window size L and $\boldsymbol{z}_{\text{past}}$ the context size M with $\boldsymbol{z}_{\text{sum}}$ full. Even the largest configuration stays below 3.5 GB.

A.7 Link Prediction

DyGDi performs zero-shot, parameter-free link prediction, using the reconstruction score $score$ from Section 3.3. Because Equation (13) depends on the event features, we additionally evaluate a variant that computes $score$ after *feature imputation*: the features of the candidate positive or negative event are removed and reconstructed with RePaint (Lugmayr et al., 2022) (without resampling, using 20 sampling steps of the model) before scoring. The resulting score therefore never accesses the ground-truth event features. We denote this variant DyGDi $_{\text{ip}}$ in all tables. Contrasting it with DyGDi isolates the contribution of the edge features to the reconstruction score, and thus also provides a measure of feature importance.

We benchmark it against the supervised temporal link predictors JODIE (Kumar et al., 2019), DyRep (Trivedi et al., 2018), TGAT (da Xu et al., 2020), TGN (Rossi et al., 2020), CAWN (Wang et al., 2021b), TCL (Wang et al., 2021a), GraphMixer (Cong et al., 2023), and DyGFormer (Yu et al., 2023), the memorization baseline EdgeBank (Poursafaei et al., 2022), and the generative model DGGen (Hosseini et al., 2025), using the same evaluation setup as Poursafaei et al. (2022). EdgeBank is a non-learned heuristic that predicts an edge as positive whenever the node pair has been observed before. It serves as a pure-memorization control that succeeds only when test edges recur. Its memory can be maintained in several ways: retaining every past edge (unlimited memory), only edges within a recent time window (whose length is set either to the test-split duration or to the recurrence interval of edges), or only edges seen more than a threshold number of times. Following Yu et al. (2023), the best-performing variant for each dataset and sampling strategy is reported. The model is presented with positive events and negative events with a sampled negative destination node. There are three sampling strategies, in order of increasing difficulty:

- random sampling, which picks a negative destination node at random, making this the easiest setting, where ranking becomes unreliable (Poursafaei et al., 2022),
- historical sampling, which samples from edges previously seen in training but absent from the current batch,
- and inductive sampling, which samples edges seen in the test set but not in the training set and absent from the current batch.

Average Precision (AP) results for random, historical, and inductive sampling are reported in Tables 8 to 10. For completeness, Tables 11 to 13 report the corresponding AU-ROC under random, historical, and inductive sampling. We further evaluate the harder *new-node* (inductive) setting, in which at least one endpoint of every test edge is a node never seen during training, reporting AP and AU-ROC under all three negative samplers in Tables 14 to 19. The baselines are taken from the inductive evaluation of Yu et al. (2023), which reports no EdgeBank column. This evaluation follows the inductive protocol of Poursafaei et al. (2022), under which we randomly hold out 10% of the nodes and remove every training edge involving them, so that they remain unseen during training and appear only at validation and test time.

Transductive link prediction with features. DyGDi performs strongly compared to existing baselines trained specifically for link prediction, despite never incorporating a link-prediction term in its loss function (unlike every learned baseline, as DG-Gen includes a link-prediction loss). This highlights the potential of the DyGDi framework to serve not only as a generative model, but also as a discriminative one for downstream tasks without additional training.

On Reddit, Wikipedia, and MOOC, DyGDi ranks first on every historical and inductive-sampling comparison, and its AP varies by at most 0.02 across the three sampling strategies, whereas the strongest supervised baselines lose 0.1–0.2 AP from random to historical negatives (*e.g.* TGN from 0.99 to 0.81 on Reddit). This stability suggests that DyGDi is less prone to the memorization shortcuts that supervised models may overfit to. EdgeBank remains competitive under random sampling on Reddit and Wikipedia, but DyGDi surpasses it by a wide margin under the harder historical and inductive regimes on Reddit, Wikipedia, and MOOC, indicating that DyGDi does not merely memorize recurring edges.

The exception is LastFM, where DyGDi performs well on the easier random setting but collapses on the more challenging historical and inductive sampling. For random sampling, DyGDi correctly predicts that a user will prefer to listen to a familiar artist rather than a random one. For historical sampling, DyGDi's AP and AUC performances are the lowest. This indicates that DyGDi gives more importance to previously-seen edges than to the current one. This might be explained by the fact that LastFM has more recurrence in its edges than the other three datasets. On a graph where replaying past interactions is the dominant behaviour and no features are available, previously observed edges are nearly indistinguishable from current ones under a reconstruction score. The score therefore rates historical negatives as no less plausible than the positives. This failure is confined to the two hard regimes on the single most adversarial dataset for a manifold-based score, and it identifies the setting in which an explicit link-prediction loss becomes preferable to a reconstruction score: the competing models, including DG-Gen, are trained with a direct link-prediction objective, a strong discriminative signal that DyGDi lacks and that may explain their stronger performance under these hard negatives. The gap is largest for DyGFormer, the best learned model on LastFM's hard regimes; the pure-memorization EdgeBank even overtakes it under inductive sampling, underscoring how strongly this dataset rewards recurrence.

Transductive link prediction with feature imputation. Contrasting DyGDi with its feature-imputed variant DyGDi $_{\text{ip}}$ reveals how much the edge features drive the reconstruction score. On Reddit and Wikipedia the two variants agree to within 0.005 AP and AU-ROC across all three sampling strategies, showing that on these datasets the model can still discriminate positive from negative events from node identity and temporal-structural signal alone, even without access to the event features. On MOOC the behaviour reverses: imputing the features sharply degrades performance (*e.g.* historical AP from 0.93 to 0.72 and inductive AU-ROC from 0.93 to 0.66), showing that the event features are an important part of MOOC's discriminative signal. LastFM carries no features, so no imputed variant is reported. Beyond probing feature importance in a simple, training-free manner for each dataset, this imputed variant also enables a fairer comparison, since several of the baselines do not access the features of the current event when scoring it.

Table 8: AP (↑), random sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.98	0.98	**0.99**	**0.99**	**0.99**	0.95	0.98	0.97	**0.99**	0.79	**0.99**	**0.99**
Wikipedia	0.97	0.95	0.97	0.98	**0.99**	0.90	0.96	0.97	**0.99**	0.96	0.97	0.98
MOOC	0.80	0.82	0.86	0.89	0.80	0.58	0.82	0.83	0.88	0.90	**0.95**	0.82
LastFM	0.71	0.72	0.73	0.77	0.87	0.79	0.67	0.76	0.93	0.87	**0.95**	-

Table 9: AP (↑), historical sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.80	0.80	0.80	0.81	0.81	0.74	0.77	0.78	0.82	0.71	**0.99**	**0.99**
Wikipedia	0.83	0.80	0.87	0.87	0.71	0.73	0.89	0.91	0.82	0.82	0.96	**0.97**
MOOC	0.79	0.76	0.82	0.87	0.74	0.61	0.77	0.78	0.86	0.83	**0.93**	0.72
LastFM	0.74	0.75	0.72	0.77	0.70	0.73	0.59	0.72	**0.82**	0.80	0.53	-

Table 10: AP (↑), inductive sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.87	0.86	0.90	0.88	0.92	0.85	0.87	0.85	0.91	0.74	**0.99**	**0.99**
Wikipedia	0.76	0.70	0.87	0.86	0.74	0.81	0.87	0.89	0.78	0.82	**0.97**	**0.97**
MOOC	0.65	0.62	0.76	0.78	0.74	0.49	0.75	0.74	0.81	0.59	**0.94**	0.71
LastFM	0.63	0.64	0.71	0.66	0.67	**0.75**	0.58	0.68	0.74	0.74	0.62	-

Table 11: AU-ROC (↑), random sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.98	0.98	0.98	**0.99**	**0.99**	0.95	0.97	0.97	**0.99**	0.73	**0.99**	**0.99**
Wikipedia	0.96	0.94	0.97	0.98	**0.99**	0.91	0.96	0.97	**0.99**	0.94	0.97	0.98
MOOC	0.84	0.85	0.87	0.91	0.80	0.61	0.83	0.84	0.88	0.91	**0.94**	0.79
LastFM	0.70	0.71	0.72	0.78	0.86	0.84	0.64	0.74	0.93	0.86	**0.94**	-

Table 12: AU-ROC (↑), historical sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.81	0.80	0.79	0.81	0.80	0.79	0.76	0.78	0.81	0.64	**0.99**	0.98
Wikipedia	0.81	0.78	0.83	0.83	0.68	0.77	0.86	0.88	0.79	0.76	0.96	**0.97**
MOOC	0.83	0.81	0.81	0.88	0.72	0.62	0.72	0.77	0.87	0.83	**0.93**	0.68
LastFM	0.75	0.75	0.64	0.78	0.68	0.78	0.47	0.64	**0.79**	0.77	0.44	-

Table 13: AU-ROC (↑), inductive sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.84	0.83	0.87	0.85	0.88	0.86	0.85	0.82	0.86	0.69	**0.99**	0.98
Wikipedia	0.71	0.67	0.82	0.81	0.71	0.82	0.82	0.84	0.75	0.75	**0.97**	**0.97**
MOOC	0.67	0.63	0.73	0.77	0.70	0.48	0.70	0.72	0.81	0.51	**0.93**	0.66
LastFM	0.61	0.62	0.64	0.65	0.68	**0.77**	0.47	0.60	0.69	0.68	0.55	-

New-node link prediction. The new-node setting preserves the transductive trends under hard negatives. For both historical and inductive sampling, DyGDi, with or without feature imputation, attains the highest AP and AU-ROC on Reddit, Wikipedia, and MOOC, ahead of every supervised baseline and of DG-Gen by a wide margin. Against the generative baseline DG-Gen, our closest comparator, DyGDi is stronger on every new-node comparison for Reddit, Wikipedia, and MOOC. LastFM remains the exception. As in the transductive hard regimes, DyGDi collapses under historical and inductive sampling on new nodes, where DG-Gen and the supervised baselines keep the advantage. The same failure appears in both the transductive and new-node settings, once more indicating the regime where an explicit link-prediction loss is preferable to a reconstruction score. Aside from LastFM, new-node is the harder regime, where the supervised baselines lose 0.07 to 0.14 AP on average across Reddit, Wikipedia, and MOOC moving from transductive to new-node evaluation, and up to 0.28 on Reddit, whereas DyGDi loses at most 0.03 AP, pointing to strong inductive generalization to unseen nodes.

A.8 Topology Quality vs. Window Size

Table 20 extends Table 2 by reporting the topological metrics of Tigger, DG-Gen, and DyGDi across time-snapshot window sizes from 512 to 4096. The results are consistent with Table 2 at every resolution, with DyGDi often outperforming Tigger and DG-Gen. On Reddit, its errors remain within the scale of the real statistics, and it obtains the best value on at least eight of the nine metrics.

Table 14: New-node AP (↑), random sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.97	0.96	0.97	0.98	**0.99**	0.94	0.95	**0.99**	0.77	0.96	0.95
Wikipedia	0.95	0.92	0.96	0.98	0.98	0.96	0.97	**0.99**	0.94	0.97	0.98
MOOC	0.80	0.81	0.86	0.89	0.81	0.81	0.81	0.87	0.78	**0.95**	0.82
LastFM	0.82	0.83	0.79	0.81	0.89	0.74	0.82	**0.94**	0.86	0.91	-

Table 15: New-node AP (↑), historical sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.62	0.62	0.63	0.65	0.64	0.61	0.65	0.65	0.62	**0.96**	0.95
Wikipedia	0.69	0.62	0.84	0.82	0.67	0.82	0.88	0.71	0.71	0.94	**0.95**
MOOC	0.63	0.63	0.77	0.77	0.75	0.74	0.74	0.81	0.60	**0.93**	0.75
LastFM	0.70	0.71	**0.76**	0.67	0.71	0.66	**0.76**	**0.76**	**0.76**	0.59	-

Table 16: New-node AP (↑), inductive sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.62	0.62	0.63	0.65	0.64	0.61	0.64	0.65	0.62	**0.96**	0.95
Wikipedia	0.69	0.62	0.84	0.82	0.67	0.82	0.88	0.71	0.71	**0.94**	**0.94**
MOOC	0.63	0.63	0.77	0.77	0.75	0.74	0.74	0.81	0.59	**0.93**	0.75
LastFM	0.70	0.71	**0.76**	0.69	0.71	0.66	**0.76**	**0.76**	**0.76**	0.59	-

Table 17: New-node AU-ROC (↑), random sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.97	0.96	0.97	0.97	0.98	0.94	0.95	**0.99**	0.71	0.96	0.95
Wikipedia	0.94	0.91	0.96	**0.98**	**0.98**	0.96	0.96	**0.98**	0.91	0.97	0.97
MOOC	0.83	0.84	0.87	0.91	0.82	0.81	0.83	0.88	0.77	**0.94**	0.79
LastFM	0.81	0.82	0.77	0.83	0.88	0.71	0.80	**0.94**	0.84	0.90	-

Table 18: New-node AU-ROC (↑), historical sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.62	0.60	0.64	0.65	0.65	0.61	0.64	0.65	0.57	**0.95**	**0.95**
Wikipedia	0.62	0.58	0.78	0.76	0.62	0.80	0.83	0.68	0.62	**0.94**	**0.94**
MOOC	0.64	0.64	0.74	0.78	0.72	0.70	0.73	0.81	0.52	**0.92**	0.71
LastFM	0.68	0.69	0.70	0.67	0.68	0.56	0.70	**0.71**	0.70	0.51	-

Table 19: New-node AU-ROC (↑), inductive sampling.

Dataset	JODIE	DyRep	TGAT	TGN	CAWN	TCL	GraphMixer	DyGFormer	DG-Gen	DyGDi	DyGDi $_{ip}$
Reddit	0.62	0.60	0.64	0.65	0.65	0.61	0.64	0.65	0.56	**0.95**	**0.95**
Wikipedia	0.62	0.58	0.78	0.76	0.62	0.80	0.83	0.68	0.62	**0.94**	**0.94**
MOOC	0.64	0.64	0.74	0.78	0.72	0.70	0.73	0.81	0.52	**0.92**	0.71
LastFM	0.68	0.69	0.70	0.67	0.68	0.56	0.70	**0.71**	0.70	0.51	-

A.9 Ablation Study

In Table 21 we ablate the components of DyGDi by removing them one at a time from the token representation and the conditioning. Starting from the full model, we successively strip the random Gaussian embedding for new nodes ($-\xi$), the event-class information ($-\boldsymbol{c}$), the graph-level summary ($-\boldsymbol{z}_{\text{sum}}$), and the past-event context ($-\boldsymbol{z}_{\text{past}}$). Each row omits every component listed in its label. Feature-quality metrics are essentially unchanged across all variants, as expected. Events can be generated almost independently, one by one, without information about the rest of the dynamic graph. The ablation should instead surface in the global JL similarity and in the per-window topology metrics. Two distinct behaviours emerge. On Reddit and MOOC, the JL similarity declines as components are removed while the topology metrics stay plausible, whereas on Wikipedia and LastFM the JL similarity is largely unchanged but the power-law exponent of the node degree (PLE) collapses.

On Reddit and MOOC, every feature and topology metric remains plausible, yet the JL similarity degrades as more components are removed. Since JL embeds the full event stream whereas the topology metrics are computed per window, this indicates that the generated windows stay individually realistic but no longer compose into a coherent continuous graph. Each window statistic is plausible, but their joint distribution deteriorates with every removed component.

Table 20: **Topology generation quality across snapshot window sizes.** Extension of Table 2 to window sizes 512–4096. Each cell reports the median absolute error between the generated and real per-snapshot statistics. The (percentage) beside it rescales this error by the magnitude of the real statistic, and the *Real Data* rows give the median of each real statistic over windows.

	Model	Window	ND ↓	CLO ↓	BC ↓	WC ↓	PLE ↓	NCC ↓	LCC ↓	REL ↓
Reddit	Real Data	512	1.32 -	1.39e−2 -	9.62e−5 -	178 -	3.18 -	58 -	14 -	9.57e−1 -
	Tigger	512	1.17 (88%)	2.09e−1 (1502%)	4.10e−3 (4265%)	9855 (5537%)	8.54e−1 (27%)	29 (50%)	351 (2507%)	1.09e−1 (11%)
	DG-Gen	512	6.01e−1 (45%)	2.83e−1 (2040%)	8.79e−3 (9140%)	4207 (2363%)	**2.58e−1 (8%)**	48 (83%)	104 (743%)	1.97e−1 (21%)
	DyGDi	512	**1.49e−1 (11%)**	**3.90e−3 (28%)**	**7.21e−5 (75%)**	**146 (82%)**	2.99e−1 (9%)	**15 (26%)**	**7 (50%)**	**2.74e−2 (3%)**
	Real Data	1024	1.43 -	1.25e−2 -	8.96e−5 -	656 -	3.03 -	86 -	28 -	9.42e−1 -
	Tigger	1024	1.05 (73%)	2.1 e−1 (1683%)	4.12e−3 (4602%)	9449 (1440%)	7.11e−1 (23%)	57 (66%)	342 (1221%)	9.33e−2 (10%)
	DG-Gen	1024	6.01e−1 (42%)	3.20e−1 (2561%)	5.78e−3 (6454%)	16 405 (2501%)	1.66e−1 (5%)	76 (88%)	210 (750%)	1.98e−1 (21%)
	DyGDi	1024	**1.43e−1 (10%)**	**4.66e−3 (37%)**	**7.76e−5 (87%)**	**525 (80%)**	**1.43e−1 (5%)**	**20 (23%)**	**16 (57%)**	**3.1 e−2 (3%)**
	Real Data	2048	1.55 -	1.36e−2 -	1.82e−4 -	2158 -	2.87 -	117 -	69 -	9.27e−1 -
	Tigger	2048	9.37e−1 (60%)	2.07e−1 (1521%)	3.89e−3 (2144%)	7965 (369%)	5.34e−1 (19%)	89 (76%)	292 (423%)	7.86e−2 (8%)
	DG-Gen	2048	5.53e−1 (36%)	3.44e−1 (2529%)	3.18e−3 (1752%)	63 779 (2955%)	1.35e−1 (5%)	107 (91%)	398 (577%)	1.98e−1 (21%)
	DyGDi	2048	**1.61e−1 (10%)**	**9.50e−3 (70%)**	**3.13e−4 (172%)**	**1440 (67%)**	**1.15e−1 (4%)**	**44 (38%)**	**47 (68%)**	**3.80e−2 (4%)**
	Real Data	4096	1.70 -	2.28e−2 -	8.78e−4 -	7041 -	2.72 -	144 -	300 -	9.13e−1 -
	Tigger	4096	8.18e−1 (48%)	2.00e−1 (880%)	3.08e−3 (351%)	4785 (68%)	4.00e−1 (15%)	116 (81%)	120 (40%)	6.54e−2 (7%)
	DG-Gen	4096	4.56e−1 (27%)	3.48e−1 (1529%)	**1.08e−3 (123%)**	238 110 (3382%)	2.01e−1 (7%)	135 (94%)	617 (206%)	1.95e−1 (21%)
	DyGDi	4096	**1.25e−1 (7%)**	**2.00e−2 (88%)**	1.21e−3 (138%)	**4605 (65%)**	**1.75e−1 (6%)**	**55 (38%)**	**116 (39%)**	**4.02e−2 (4%)**
Wikipedia	Real Data	512	1.04 -	3.51e−2 -	6.17e−5 -	1 -	5.09 -	15 -	3 -	9.96e−1 -
	Tigger	512	3.13e−1 (30%)	1.20e−2 (34%)	1.13e−3 (1840%)	53 (5300%)	2.23 (44%)	9 (60%)	13 (433%)	3.45e−2 (3%)
	DG-Gen	512	**3.23e−2 (3%)**	**9.70e−3 (28%)**	**6.21e−5 (101%)**	**1 (100%)**	8.96e1 (1761%)	**5 (33%)**	**1 (33%)**	**3.37e−3 (0%)**
	DyGDi	512	3.68e−2 (4%)	1.13e−2 (32%)	8.52e−5 (138%)	**1 (100%)**	**1.26 (25%)**	**5 (33%)**	**1 (33%)**	3.83e−3 (0%)
	Real Data	1024	1.07 -	2.16e−2 -	5.81e−5 -	4 -	4.42 -	25 -	4 -	9.93e−1 -
	Tigger	1024	2.78e−1 (26%)	1.16e−2 (54%)	1.15e−3 (1985%)	47 (1175%)	1.59 (36%)	**6 (24%)**	12 (300%)	3.13e−2 (3%)
	DG-Gen	1024	5.66e−2 (5%)	6.70e−3 (31%)	5.32e−5 (92%)	**3 (75%)**	2.50 (56%)	9 (36%)	**1 (25%)**	5.32e−3 (1%)
	DyGDi	1024	**4.41e−2 (4%)**	**5.69e−3 (26%)**	**4.12e−5 (71%)**	4 (100%)	**7.00e−1 (16%)**	8 (32%)	**1 (25%)**	**4.29e−3 (0%)**
	Real Data	2048	1.11 -	1.41e−2 -	4.72e−5 -	14 -	3.92 -	40 -	5 -	9.89e−1 -
	Tigger	2048	2.30e−1 (21%)	1.94e−2 (137%)	1.16e−3 (2458%)	38 (271%)	9.68e−1 (25%)	17 (43%)	10 (200%)	2.68e−2 (3%)
	DG-Gen	2048	8.89e−2 (8%)	6.22e−3 (44%)	4.58e−5 (97%)	**11 (79%)**	1.81 (46%)	26 (65%)	**2 (40%)**	9.11e−3 (1%)
	DyGDi	2048	**6.33e−2 (6%)**	**3.61e−3 (26%)**	**2.87e−5 (61%)**	14 (100%)	**5.17e−1 (13%)**	**12 (30%)**	**2 (40%)**	**4.89e−3 (0%)**
	Real Data	4096	1.17 -	9.45e−3 -	4.23e−5 -	41 -	3.55 -	64 -	8 -	9.85e−1 -
	Tigger	4096	2.02e−1 (17%)	2.52e−2 (266%)	1.49e−3 (3516%)	36 (88%)	**6.80e−1 (19%)**	40 (63%)	9 (113%)	2.54e−2 (3%)
	DG-Gen	4096	**1.28e−1 (11%)**	5.31e−3 (56%)	**4.03e−5 (95%)**	**28 (68%)**	1.31 (37%)	60 (94%)	**4 (50%)**	1.18e−2 (1%)
	DyGDi	4096	1.60e−1 (14%)	**2.01e−3 (21%)**	4.16e−5 (98%)	49 (120%)	7.22e−1 (20%)	**14 (22%)**	**4 (50%)**	**6.52e−3 (1%)**
MOOC	Real Data	512	1.97 -	1.28e−1 -	2.25e−2 -	133 -	2.09 -	6 -	36 -	9.54e−1 -
	Tigger	512	8.15 (414%)	2.45e−1 (192%)	3.53e−2 (157%)	634 991 (477 437%)	**4.68e−1 (22%)**	**5 (83%)**	1052 (2922%)	1.03e−1 (11%)
	DG-Gen	512	**3.99e−1 (20%)**	**7.49e−2 (59%)**	**1.88e−2 (84%)**	203 (153%)	6.33e−1 (30%)	8 (133%)	**27 (75%)**	4.96e−2 (5%)
	DyGDi	512	5.00e−1 (25%)	1.06e−1 (83%)	2.05e−2 (91%)	**129 (97%)**	6.17e−1 (30%)	45 (750%)	**27 (75%)**	**2.03e−2 (2%)**
	Real Data	1024	2.52 -	1.78e−1 -	2.62e−2 -	389 -	1.86 -	3 -	89 -	9.48e−1 -
	Tigger	1024	7.57 (300%)	2.48e−1 (139%)	2.18e−2 (83%)	634 564 (163 127%)	**2.38e−1 (13%)**	**3 (100%)**	1024 (1151%)	1.22e−1 (13%)
	DG-Gen	1024	7.24e−1 (29%)	**8.43e−2 (47%)**	**1.55e−2 (59%)**	725 (186%)	7.78e−1 (42%)	6 (200%)	**56 (63%)**	6.89e−2 (7%)
	DyGDi	1024	**6.96e−1 (28%)**	1.22e−1 (68%)	2.34e−2 (89%)	**456 (117%)**	4.98e−1 (27%)	38 (1267%)	64 (72%)	**1.37e−2 (1%)**
	Real Data	2048	3.52 -	2.17e−1 -	2.01e−2 -	1196 -	1.67 -	2 -	136 -	9.43e−1 -
	Tigger	2048	6.26 (178%)	1.89e−1 (87%)	2.11e−2 (105%)	633 043 (52 930%)	**9.92e−2 (6%)**	**1 (50%)**	977 (718%)	1.21e−1 (13%)
	DG-Gen	2048	1.46 (42%)	**6.18e−2 (28%)**	**9.14e−3 (46%)**	2816 (235%)	8.85e−1 (53%)	3 (150%)	**125 (92%)**	9.22e−2 (10%)
	DyGDi	2048	**1.20 (34%)**	8.38e−2 (39%)	1.22e−2 (61%)	**1576 (132%)**	5.00e−1 (30%)	10 (500%)	297 (218%)	**1.71e−2 (2%)**
	Real Data	4096	5.17 -	2.59e−1 -	1.45e−2 -	3569 -	1.55 -	1 -	182 -	9.39e−1 -
	Tigger	4096	4.1 (79%)	1.01e−1 (39%)	1.13e−2 (78%)	625 800 (17 534%)	**4.28e−2 (3%)**	**0 (0%)**	890 (489%)	1.21e−1 (13%)
	DG-Gen	4096	2.70 (52%)	5.14e−2 (20%)	**6.98e−3 (48%)**	10 698 (300%)	8.66e−1 (56%)	1 (100%)	**279 (153%)**	1.14e−1 (12%)
	DyGDi	4096	**2.15 (42%)**	**3.45e−2 (13%)**	9.74e−3 (67%)	**6007 (168%)**	3.26e−1 (21%)	**0 (0%)**	407 (224%)	**3.20e−2 (3%)**
LastFM	Real Data	512	1.05 -	1.43e−1 -	3.42e−4 -	1 -	4.65 -	5 -	3 -	9.94e−1 -
	Tigger	512	1.37 (130%)	1.49e−1 (104%)	2.37e−2 (6931%)	419 (41 900%)	2.32 (50%)	**2 (40%)**	63 (2100%)	8.92e−2 (9%)
	DG-Gen	512	5.88e−2 (6%)	**8.89e−2 (62%)**	4.90e−4 (143%)	1 (100%)	9.03e1 (1941%)	**2 (40%)**	1 (33%)	7.00e−3 (1%)
	DyGDi	512	**0.0 (0%)**	2.67e−1 (187%)	**0.0 (0%)**	**0 (0%)**	**0.0 (0%)**	3 (60%)	**0 (0%)**	**1.13e−4 (0%)**
	Real Data	1024	1.14 -	8.06e−2 -	8.17e−4 -	3 -	3.60 -	7 -	3 -	9.83e−1 -
	Tigger	1024	1.29 (113%)	1.78e−1 (221%)	2.45e−2 (3004%)	412 (13 733%)	**1.41 (39%)**	5 (71%)	61 (2033%)	8.15e−2 (8%)
	DG-Gen	1024	**1.23e−1 (11%)**	**3.72e−2 (46%)**	**8.14e−4 (100%)**	3 (100%)	9.06e1 (2519%)	**4 (57%)**	**1 (33%)**	**1.48e−2 (2%)**
	DyGDi	1024	1.25e−1 (11%)	1.90e−1 (236%)	8.92e−4 (109%)	**2 (67%)**	9.12e1 (2533%)	5 (71%)	**1 (33%)**	1.50e−2 (2%)
	Real Data	2048	1.24 -	5.40e−2 -	7.42e−4 -	11 -	3.21 -	12 -	5 -	9.71e−1 -
	Tigger	2048	1.21 (97%)	1.87e−1 (347%)	2.47e−2 (3334%)	421 (3827%)	**1.10 (34%)**	10 (83%)	59 (1180%)	6.80e−2 (7%)
	DG-Gen	2048	**1.88e−1 (15%)**	**2.23e−2 (41%)**	**6.38e−4 (86%)**	**9 (82%)**	1.69 (53%)	**6 (50%)**	**2 (40%)**	**2.37e−2 (2%)**
	DyGDi	2048	2.1 e−1 (17%)	1.03e−1 (191%)	7.43e−4 (100%)	10 (91%)	9.13e1 (2850%)	8 (67%)	**2 (40%)**	2.57e−2 (3%)
	Real Data	4096	1.35 -	3.76e−2 -	6.03e−4 -	41 -	2.96 -	18 -	8 -	9.59e−1 -
	Tigger	4096	1.08 (80%)	1.97e−1 (524%)	2.40e−2 (3986%)	371 (905%)	8.91e−1 (30%)	19 (106%)	55 (688%)	5.77e−2 (6%)
	DG-Gen	4096	2.38e−1 (18%)	**1.87e−2 (50%)**	**5.01e−4 (83%)**	**32 (78%)**	9.93e−1 (33%)	**11 (61%)**	**3 (38%)**	3.18e−2 (3%)
	DyGDi	4096	**2.31e−1 (17%)**	6.66e−2 (177%)	1.04e−3 (173%)	37 (90%)	**8.90e−1 (30%)**	12 (67%)	4 (50%)	**2.98e−2 (3%)**

On Wikipedia and LastFM the picture is reversed, as at the ablation steps where PLE collapses, the JL similarity stays similar to the full-model value. On Wikipedia the collapse coincides with removing the event-class information $\boldsymbol{c}$ ($-\boldsymbol{c} - \xi$ row), confirming that $\boldsymbol{c}$ is what mitigates memory matching collapse by routing each generated embedding to the correct pool, an already-seen node or a new one. On LastFM the same collapse appears only once the graph-level summary $\boldsymbol{z}_{\text{sum}}$ is removed, suggesting that on this dataset most nodes are matched to identifiers outside the recent

Table 21: **Ablation study on DyGDi.** Feature quality (Jensen–Shannon distance), per-snapshot topology, and the global JL similarity as the components of DyGDi are removed one at a time. Each row omits every component listed in its label, namely the random embedding ξ, the event-class information $\boldsymbol{c}$, the graph-level summary $\boldsymbol{z}_{\text{sum}}$, and the past context $\boldsymbol{z}_{\text{past}}$. Topology cells report the median absolute error to the real per-snapshot statistic, with the (percentage) rescaling it by the magnitude of that statistic. The *Real Data* rows give the median of each statistic over windows.

	Model	Pairwise JS↓	ND ↓	CLO ↓	BC ↓	WC ↓	PLE ↓	NCC ↓	LCC ↓	REL ↓	JL↑
Reddit	Real Data	- ± -	1.32	1.39e−2	9.62e−5	178	3.18	58	14	9.57e−1	
	DyGDi	0.19 ± 0.12	1.49e−1 (11%)	3.90e−3 (28%)	7.21e−5 (75%)	146 (82%)	2.99e−1 (9%)	15 (26%)	7 (50%)	2.74e−2 (3%)	0.26
	DyGDi- ξ	0.20 ± 0.12	7.97e−2 (6%)	4.73e−3 (34%)	6.69e−5 (70%)	129 (72%)	1.70e−1 (5%)	16 (28%)	7 (50%)	2.07e−2 (2%)	0.19
	DyGDi- $\boldsymbol{c}$ - ξ	0.19 ± 0.13	2.25e−1 (17%)	7.13e−3 (51%)	8.44e−5 (88%)	150 (84%)	9.20e−1 (29%)	27 (47%)	7 (50%)	3.41e−2 (4%)	0.20
	DyGDi- $\boldsymbol{c}$ - ξ - $\boldsymbol{z}_{\text{sum}}$	0.19 ± 0.12	1.90e−1 (14%)	5.51e−3 (40%)	6.83e−5 (71%)	145 (81%)	5.64e−1 (18%)	13 (22%)	7 (50%)	3.19e−2 (3%)	0.15
	DyGDi- $\boldsymbol{c}$ - ξ - $\boldsymbol{z}_{\text{sum}}$ - $\boldsymbol{z}_{\text{past}}$	0.19 ± 0.13	1.39e−1 (11%)	6.17e−3 (44%)	6.58e−5 (68%)	126 (71%)	2.88e−1 (9%)	22 (38%)	6 (43%)	2.93e−2 (3%)	0.13
Wiki.	Real Data	- ± -	1.04	3.51e−2	6.17e−5	1	5.09	15	3	9.96e−1	
	DyGDi	0.17 ± 0.18	3.68e−2 (4%)	1.13e−2 (32%)	8.52e−5 (138%)	1 (100%)	1.26 (25%)	5 (33%)	1 (33%)	3.83e−3 (0%)	0.13
	DyGDi- ξ	0.20 ± 0.20	9.52e−2 (9%)	1.03e−2 (29%)	2.01e−4 (326%)	5 (500%)	1.56 (31%)	5 (33%)	1 (33%)	8.91e−3 (1%)	0.06
	DyGDi- $\boldsymbol{c}$ - ξ	0.18 ± 0.18	3.23e−2 (3%)	1.13e−2 (32%)	6.11e−5 (99%)	1 (100%)	8.95e1 (1759%)	6 (40%)	1 (33%)	3.37e−3 (0%)	0.13
	DyGDi- $\boldsymbol{c}$ - ξ - $\boldsymbol{z}_{\text{sum}}$	0.20 ± 0.20	3.23e−2 (3%)	1.44e−2 (41%)	6.11e−5 (99%)	1 (100%)	8.94e1 (1758%)	9 (60%)	1 (33%)	3.37e−3 (0%)	0.15
	DyGDi- $\boldsymbol{c}$ - ξ - $\boldsymbol{z}_{\text{sum}}$ -$\boldsymbol{z}_{\text{past}}$	0.19 ± 0.20	3.44e−2 (3%)	1.17e−2 (33%)	6.11e−5 (99%)	1 (100%)	8.97e1 (1762%)	6 (40%)	1 (33%)	3.37e−3 (0%)	0.15
MOOC	Real Data	- ± -	1.97	1.28e−1	2.25e−2	133	2.09	6	36	9.54e−1	
	DyGDi	0.11 ± 0.04	5.00e−1 (25%)	1.06e−1 (83%)	2.05e−2 (91%)	129 (97%)	6.17e−1 (30%)	45 (750%)	27 (75%)	2.03e−2 (2%)	0.32
	DyGDi- ξ	0.12 ± 0.04	7.03e−1 (36%)	1.06e−1 (83%)	2.22e−2 (99%)	148 (111%)	8.35e−1 (40%)	55 (917%)	40 (111%)	2.56e−2 (3%)	0.24
	DyGDi- $\boldsymbol{c}$ - ξ	0.09 ± 0.04	6.23e−1 (32%)	1.19e−1 (93%)	2.27e−2 (101%)	110 (83%)	7.96e−1 (38%)	67 (1117%)	27 (75%)	2.50e−2 (3%)	0.15
	DyGDi- $\boldsymbol{c}$ - ξ -$\boldsymbol{z}_{\text{sum}}$	0.12 ± 0.03	6.67e−1 (34%)	1.15e−1 (90%)	2.23e−2 (99%)	102 (77%)	8.61e−1 (41%)	63 (1050%)	27 (75%)	2.66e−2 (3%)	0.18
	DyGDi- $\boldsymbol{c}$ - ξ -$\boldsymbol{z}_{\text{sum}}$ -$\boldsymbol{z}_{\text{past}}$	0.09 ± 0.02	6.71e−1 (34%)	1.17e−1 (92%)	2.24e−2 (100%)	103 (77%)	8.52e−1 (41%)	62 (1033%)	28 (78%)	2.68e−2 (3%)	0.14
LastFM	Real Data	- ± -	1.05	1.43e−1	3.42e−4	1	4.65	5	3	9.94e−1	
	DyGDi	- ± -	0.0 (0%)	2.67e−1 (187%)	0.0 (0%)	0 (0%)	0.0 (0%)	3 (60%)	0 (0%)	1.13e−4 (0%)	0.17
	DyGDi- ξ	- ± -	4.35e−2 (4%)	6.67e−1 (467%)	2.22e−4 (65%)	0 (0%)	6.64e−1 (14%)	3 (60%)	0 (0%)	5.29e−3 (1%)	0.09
	DyGDi- $\boldsymbol{c}$ - ξ	- ± -	0.0 (0%)	6.67e−1 (467%)	0.0 (0%)	0 (0%)	0.0 (0%)	3 (60%)	0 (0%)	1.04e−4 (0%)	0.19
	DyGDi- $\boldsymbol{c}$ - ξ -$\boldsymbol{z}_{\text{sum}}$	- ± -	5.26e−2 (5%)	6.67e−1 (467%)	2.80e−4 (82%)	1 (100%)	9.03e1 (1941%)	3 (60%)	1 (33%)	6.01e−3 (1%)	0.19
	DyGDi- $\boldsymbol{c}$ - ξ -$\boldsymbol{z}_{\text{sum}}$ -$\boldsymbol{z}_{\text{past}}$	- ± -	3.70e−2 (4%)	6.67e−1 (467%)	1.14e−4 (33%)	1 (100%)	8.98e1 (1930%)	3 (60%)	1 (33%)	3.96e−3 (0%)	0.17

past-event context, so the model must fall back on the global summary to resolve them. Because PLE collapses while the JL similarity barely moves, JL alone does not capture this failure mode, a limitation of the metric and a reminder that the global and per-window statistics are complementary rather than interchangeable.

Author Contributions

If you'd like to, you may include a section for author contributions as is done in many journals. This is optional and at the discretion of the authors.