FuDGE: Modeling Full Dynamic Graph Evolution

Angelica Liguori[1*†], Simone Mungari[1,2,3†], Ettore Ritacco[4], Edoardo Serra[5], Giuseppe Manco[1]

[1]Institute for High Performance Computing and Networking, Italian National Research Council, Via P. Bucci, 8/9C, Rende, 87036,Italy.
[2]University of Calabria, Via P. Bucci, Rende, 87036, Italy.
[3]Revelis S.r.l, Viale della Resistenza, 19/C, Rende, 87036, Italy.
[4]University of Udine, Via Palladio, 8, Udine, 33100, Italy.
[5]Boise State University, 1910 W University Dr, Boise, 83725, USA.

*Corresponding author(s). E-mail(s): angelica.liguori@icar.cnr.it;
Contributing authors: simone.mungari@icar.cnr.it;
ettore.ritacco@uniud.it; edoardoserra@boisestate.edu;
giuseppe.manco@icar.cnr.it;
†These authors contributed equally to this work.

Abstract

Research in neural generative models for dynamic networks is constantly evolving, and sophisticated solutions have been exploited to characterize the long-term evolution of temporal graphs. Despite the efforts in the literature, state-of-the-art models face the problem of handling changes in the graph structure by relying on prior knowledge, compromising the model's flexibility. In this paper, we propose a graph-size invariant probabilistic generative model, named **FuDGE**, Fully Dynamic Graph Evolution, for predicting the graph evolution through step-wise changes in the graph structure. **FuDGE** can generate evolving graphs by exploring the whole node space, thus ensuring fast and effective generation. We evaluate **FuDGE** on real and synthetic benchmark datasets and compare its performance against state-of-the-art competitors. The results demonstrate that our approach offers a competitive advantage in generation and prediction quality compared to existing literature. The code is publicly available at https://anonymous.4open.science/r/fudge-3C25.

Keywords: Graph Generation, Graph Convolutional Network, Temporal Graph, Deep Learning, Graph Evolution

1 Introduction

Dynamic networks (Gupta and Bedathur 2022), also known as *temporal networks*, are graph-based data structures exhibiting variations in both content and topology over time. Predicting their evolution remains a challenging task encompassing various research domains. In biology, dynamic graphs are utilized to model intricate interactions between molecules (Gainza et al. 2020; Zitnik et al. 2018) or predict virus evolution (Han et al. 2019; Zeng et al. 2008). They are also widely employed in social network analysis to analyze social interactions and user behavior (Granovetter 1973; Held et al. 2013). In the field of complex systems, dynamic graphs are applied to study human-object interactions (Qi et al. 2018) or the movement of objects in physical systems (Battaglia et al. 2016). In cybersecurity, they find significant utility for multiple purposes. They are used for malware detection (Anderson et al. 2011), where the flow of APIs calls is analyzed, as well as for identifying ransomware attacks (Chen et al. 2017). Intrusion detection systems (Duan et al. 2023) leverage dynamic graphs to model network traffic, while fraud and anomaly detection methods (Ranshous et al. 2015; Zheng et al. 2019; Zhou et al. 2021) incorporate dynamic graphs to identify irregular patterns. By effectively modeling and predicting the evolution of graphs, it becomes possible to detect polymorphic malware and cyber attacks (Avllazagaj et al. 2021; Tam et al. 2017) characterized by malicious entities that mutate and evolve to evade detection. Therefore, accurate graph evolution prediction can enhance the detection of such threats in the cybersecurity domain. Beyond predictive and generative models, dynamic graphs have also been studied to track the temporal evolution of communities (Li et al. 2025a). Related research has also investigated temporal knowledge graphs for reasoning and querying over time-evolving relational data (Du et al. 2024; Ma et al. 2023).

Current research in graph evolution modeling and prediction has primarily focused on neural architectures (Rossi et al. 2020; Luo and Li 2022; Zhou et al. 2022a; Lim et al. 2020; Sankar et al. 2020; Trivedi et al. 2019; Wang et al. 2021a). These approaches mainly concentrate on predicting basic operations such as adding/removing edges. However, they overlook the possibility of more comprehensive generative approaches that consider structural changes in graphs, including node dynamics, i.e., adding new unseen nodes. This oversight can have a significant impact on various domains, including cybersecurity, where unexpected new attacks like malware variants can emerge, or in social networks, where new users can join the network. While modeling the dynamics of edges between existing nodes is relatively easier, accurately representing the addition or deletion of nodes requires continuous adjustments in the underlying representations, which poses significant challenges. Practical modeling of fully dynamic networks requires flexible graph feature representations that can dynamically adjust their dimensions over time. In contrast, existing approaches are mostly rigid, as they rely on prior knowledge of the underlying graph structure to make predictions.

One approach to address this issue is selecting an arbitrarily large graph size that encompasses dummy (inactive) nodes. This allows for the formulation of a generative process within the constraints of the maximum anticipated size. However, this approach has a significant limitation: it requires an accurate estimation of the underlying graph dynamics, which is unknown beforehand. If the final size is estimated

incorrectly, it leads to inefficient training and inference phases, failing to reflect the natural evolution of dynamic networks accurately. Furthermore, the arbitrarily large node structure can only incorporate partial information because future nodes can only be associated with meaningful representations once their suitable topological features are progressively revealed during the inference steps. This limitation is particularly evident in some approaches proposed in the literature (Rossi et al. 2020; Luo and Li 2022; Zhou et al. 2022a), which unrealistically exploit topology features statically. These approaches also possess limited generative capabilities as they are primarily tailored for link prediction tasks. Consequently, they tend to consider only a limited set of potential edges during the generation process to avoid exploring the full range of possibilities. This limitation renders them ill-suited for real-world scenarios where a comprehensive search is necessary instead of the constrained approach they employ. Further, focusing on specific next-step link prediction may overlook the general evolution of the underlying graph in terms of network structures and patterns.

In this work, we define **FuDGE**, a **Fu**ll **D**ynamic **G**raph **E**volution model that addresses the aforementioned challenges in generating dynamic networks, by decomposing the graph evolution into probabilistic atomic events. At each time step, given a partial history of the temporal graph, FuDGE predicts the type of evolution (addition or deletion) as well as the nodes involved in the event, enabling step-wise modeling of the most likely changes. To cope with varying network size, FuDGE leverages Graph Convolutional Networks (GCNs) (Kipf and Welling 2017) to process graph snapshots of different sizes and generate latent graph representations, independently of the number of nodes. These representations capture the entire adjacency matrix of a graph and are condensed into fixed-size embeddings. By combining such representations from different graph histories through recurrent modeling, FuDGE produces a compact representation of the graph history that can be exploited for prediction purposes. This history representation is then refined by dedicated networks that predict the event type and derive the source and destination embeddings for the nodes involved in the evolution. To explicitly account for the possibility that an event may involve previously unseen nodes, acting as either source or destination nodes, FuDGE introduces auxiliary graph networks operating on augmented graph snapshots with a virtual node. These networks generate node-level embeddings, which are independently compared with the source and destination representations to identify the nodes involved in the event. This decoupling modeling of source and destination represents a key aspect of the proposed framework, which avoids pairwise node comparisons. This design allows FuDGE to scale linearly with the number of nodes, eliminating the need for sampling strategies, while still considering the entire node space. To assess the capabilities of the model, we establish a fair generation protocol that simulates long-term growth scenarios, where the topological information about new nodes and edges is initially unknown and progressively reconstructed. Notably, the output generated from each prediction step can be used to enhance the input for the subsequent step. The entire generated graph history is evaluated using: (*i*) the Weisfeiler-Lehman similarity (Shervashidze et al. 2011), which measures the level of isomorphism between the actual and generated graph histories, and (*ii*) graph structure metrics.

In summary, our main contributions are:

- We introduce a novel generative model for dynamic networks that leverages convolution and graph history embedding. The model is invariant to the graph size, allowing to consistently predict both edge and node dynamics, demonstrating its effectiveness in both learning and inference stages.
- We conduct an extensive benchmarking study using multiple datasets and comparing our model against other competitors by exploiting a fair evaluation protocol. The latter focuses on long-term graph generation and utilizes graph isomorphism to compare the actual and generated graph histories as well as other statistics based on the graph structure.

The rest of the work is organized as follows. In section 2 we discuss the state of the art; Section 3 introduces the notation that we will use in the rest of the paper and provides a formal description of the problem statement; Section 4 describes the FuDGE architecture; Section 5 shows the evaluation of the proposed model through an extensive analysis on benchmark datasets; final remarks and pointers for future work are discussed in Section 6.

2 Related Work

Modeling graph dynamics involves devising potential evolutions of a graph based on its current snapshot. This can be achieved through two primary approaches: either by looking at general methods that utilize generative models for graphs or by focusing on more specific methods that concentrate on predicting short-term link formation.
Graph Generation. Traditional generative models for synthetic graphs, such as Barabási-Albert (BA) (Barabási and Albert 1999), POWER (Holme and Kim 2002), or Erdős-Rényi (ER) (Erdős et al. 1960), are specifically designed to capture certain distributional properties exhibited by different families of graphs. Alternatively, a more general approach was proposed in (You et al. 2018), in which a deep autoregressive model is employed for learning the distribution of underlying static graphs and generating synthetic graphs with diverse characteristics. Another approach for static graph generation, based on variational autoencoding, was proposed in (Grover et al. 2019). Recently, diffusion-based approaches have been employed for the generation of static graphs (Kong et al. 2023; Yan et al. 2024; Evdaimon et al. 2024; Zhao et al. 2024; Cai et al. 2024). However, to the best of our knowledge, no probabilistic generative approaches have been defined for dynamic graphs. To address the evolving nature of dynamic graphs, researchers have developed Temporal Graph Neural Networks (TGNNs) (Wu et al. 2020; Dai et al. 2020; Lim et al. 2020). TGNNs are capable of simultaneously capturing temporal and structural relationships within dynamic graphs. In (Du et al. 2022), the authors propose a probabilistic model for dynamic spatio-temporal graphs. This framework considers spatial, temporal, and topological information individually and in combination, optimizing a variational objective function. Zeno et al. (2021) take a different approach by using the temporal activity of motifs to represent temporal changes. Motifs represent graph substructures beyond pairwise connections, capturing longer-range correlations in connections and activity. A more recent approach was proposed in (Li et al. 2025c), where the authors use a variational recurrent architecture for capturing co-evolution patterns of structural

information and node attributes used for generating new attributed graphs. A significant limitation of these approaches is their reliance on prior knowledge of the number of nodes to generate. This limitation restricts their applicability in highly dynamic scenarios where such knowledge may not be available. Furthermore, in contrast to approaches that focus on learning the underlying distribution of dynamic graphs to generate new synthesized graphs, our objective is to learn how the graph structure evolves over discrete time steps in order to predict its future evolution.

Model	*Dynamic*	*Dynamic Node Set*	*Fully generative*	*GPU support*
GraphRNN (You et al. 2018)	✗	✗	✓	✓
GRAPHARM (Kong et al. 2023)	✗	✓	✗	-
SwinGNN (Yan et al. 2024)	✗	✗	✓	✓
NGG (Evdaimon et al. 2024)	✗	✗	✓	✓
PARD (Zhao et al. 2024)	✗	✗	✓	✓
LGD (Cai et al. 2024)	✗	✗	✓	✓
EvoNet (Wu et al. 2020)	✓	✗	✓	-
BiGG (Dai et al. 2020)	✗	✗	✗	✓
TASTE (Fan et al. 2022)	✓	✗	✓	-
Graphite (Grover et al. 2019)	✗	✗	✓	-
S-b molecule (Lim et al. 2020)	✓	✓	✓	✗
STGD (Du et al. 2022)	✓	✗	✓	-
DySAT (Sankar et al. 2020)	✓	✗	✓	✓
GraphSAGE (Hamilton et al. 2017)	✗	✗	✗	✓
CoPE (Zhang et al. 2021)	✓	✗	✓	✓
JODIE (Kumar et al. 2019)	✓	✗	✓	✓
TGN (Rossi et al. 2020)	✓	✓	✗	✓
APAN (Wang et al. 2021a)	✓	✗	✗	✓
NAT (Luo and Li 2022)	✓	✓	✗	✓
TGL (Zhou et al. 2022a)	✓	✓	✗	✓
TagGen (Zhou et al. 2020)	✓	✗	✓	✓
FiGTNE (Liu et al. 2020)	✓	✓	✗	-
CAW (Wang et al. 2021b)	✓	✗	✗	✓
DyRep (Trivedi et al. 2019)	✓	✗	✓	-
NPPCTNE (Zhou et al. 2022b)	✓	✗	✗	-
DHGEEP (Chen et al. 2022)	✓	✗	✗	-
TIGGER (Gupta et al. 2022)	✓	✓	✗	✓
DYMOND (Zeno et al. 2021)	✓	✗	✗	✓
VRDAG (Li et al. 2025c)	✓	✗	✓	✓
FuDGE	✓	✓	✓	✓

Table 1: Comparison of FuDGE with the state-of-the-art.

Link Prediction. Link prediction can serve as a valuable tool for modeling the evolution of a graph, assuming that the number of nodes remains constant. In this scenario, the focus is on predicting edge dynamics, specifically the likelihood of two nodes forming a connection over time. Various approaches for link prediction have been proposed, utilizing techniques such as learning dynamic node embeddings (Sankar et al. 2020; Hamilton et al. 2017; Zhang et al. 2021; Kumar et al. 2019). Dynamic node embeddings capture the evolving nature of nodes by encoding events as messages that contribute to the embedding of each node (Longa et al. 2023). TGN (Rossi et al. 2020) is an encoder-decoder architecture that leverages this capability to model connectivity changes over time. By utilizing TGNNs and dynamic node embeddings, TGN effectively tracks events and captures the evolving relationships within dynamic graphs. NAT (Luo and Li 2022) utilizes dictionary-type representations for nodes, enabling

efficient tracking of node neighborhoods in large graphs. To manage these representations, specific caching structures are employed, which facilitate parallel processing. In a similar vein, in (Zhou et al. 2022a) TGL, a comprehensive framework designed for training TGNNs on large graphs, is introduced. TGL offers the ability to train various TGNN variants, including those proposed in (Kumar et al. 2019; Sankar et al. 2020; Xu et al. 2020; Rossi et al. 2020; Wang et al. 2021a). By leveraging TGL, significant speedups can be achieved in both training and evaluation processes. Dynamic representations can also be obtained using temporal walks (Zhou et al. 2020; Liu et al. 2020; Wang et al. 2021b) or by leveraging temporal point processes to model dynamic graphs (Trivedi et al. 2019; Zhou et al. 2022b; Chen et al. 2022). Hybrid approaches (Gupta et al. 2022) that combine both random walks and temporal point processes have been explored as well. In other approaches (Fan et al. 2022), temporal association rules are combined with adversarial learning (Zhou et al. 2020) to predict the likelihood of an edge with a specific label appearing within a certain time.

Long-term prediction Several recent works have investigated long-horizon modeling on temporal graphs, including dynamic graph forecasting and representation learning approaches (Wu et al. 2024; Trivedi et al. 2019; Li et al. 2025d; Zhang et al. 2026). While these methods have demonstrated strong performance in capturing long-term temporal dependencies, they typically focus on predicting node-level signals or edge probabilities over fixed node sets. Other works, such as DyG-Mamba (Li et al. 2025b), handle dynamic graph learning as a long sequence modeling problem. Specifically, DyG-Mamba introduces a continuous-time state space model that exploits irregular time intervals to enable robust learning of long-term temporal dependencies. As a result, it learns time-aware node embeddings for downstream tasks such as dynamic link prediction and node classification. In contrast, FuDGE adopts a generative, discrete-time formulation that autoregressively models the graph evolution by learning how the graph structure evolves over time through step-wise predictions of the most likely changes in the network.

Summary. We analyze the approaches in the literature according to four key properties: suitability for graph dynamics, focus on node dynamics, enabling event generation (as opposed to simply computing their probability, which would require a full exploration of all possible events for generation), and support GPU processing both in training and inference. Compared to the listed methods, our approach supports the whole set of these features, which will be further elaborated in the subsequent sections. Table 1 provides a concise summary comparison.

3 Preliminaries and Problem Statement

Let $G = (V, E)$ be a non-attributed directed graph, where V and E are the sets of nodes and edges. We denote by $n = |V|$ and $m = |E|$ the number of vertices and edges. For generality, we refer to the adjacency matrix $A \in \{0, 1\}^{n \times n}$ denoting, for each entry $A_{i,j}$, whether an edge exists between the node i and j.

A dynamic graph is a graph whose topology and size change over time. Specifically, it can be modeled as a sequence of temporally-sorted graphs $\{G_1, G_2, ..., G_T\}$ in which each graph $G_t = (V_t, E_t)$ may have a non-empty intersection with G_{t-1} and represents

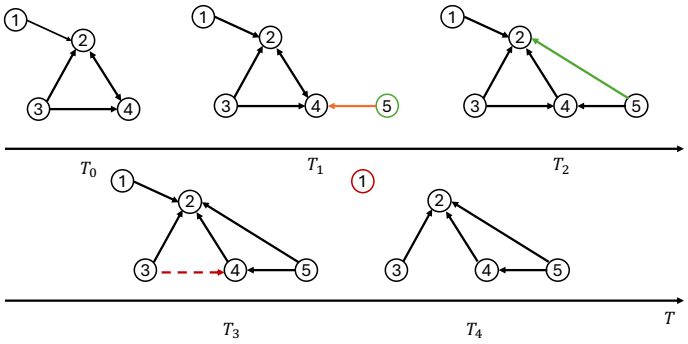

Fig. 1: A graph evolution example. Initial graph G at time T_0. A new node (labeled as 5 in the example) appears in G, and a new edge between the new node and an existing node (4) is added at time T_1. At T_2 an edge between two existing nodes (5 and 2, respectively) is added. T_3 involves edge removal: the edge between the nodes 3 and 4 is removed. Also, in T_4 the edge between nodes 1 and 2 is removed. As a consequence, Node 1 is disconnected from the rest of the network, thus becoming redundant and hence potentially deleted.

the state of the graph at time t. We assume discrete-time evolution where, at each time step, only one operation can occur. This assumption can be easily relaxed to include multiple operations in a single time step, by assuming multiple hierarchical granularities of time (as we will discuss later in the paper, see subsection 5.3).

Formally and without loss of generality, we consider a set of atomic operations that can involve G_{t-1} and G_t. These are: *addition of a new node* (together with an edge to connect it to an existing node); *addition of a new edge* between two existing or brand new nodes; *deletion of an edge* (together with the connected vertices that get disconnected from the rest of the network). The operations are illustrated in Figure 1. We do not explicitly consider the *deletion of a node*, but instead, we correlate this event with the deletion of an edge. The intuition behind this assumption lies in considering that, in real-world situations, the disappearing of a node can only occur due to exogenous causes (thus making the event unpredictable) or because the node becomes disconnected from the network and consequently extraneous to the network activities.

We are interested in learning how the structure of the graph can evolve with time. Given a partial history $H_{G_{<t}} = \{G_1, G_2, ..., G_{t-1}\}$ of the temporal graph, the main question is whether we can predict its evolution $\{G_t, G_{t+1}, ..., G_T\}$. In a first approximation, this corresponds to recursively predicting G_t given $H_{G<t}$, namely the probability $p(G_t|H_{G_{<t}})$, for each t. The evolution can be hence devised by decomposing the probability for the full remaining history:

$$p(\{G_t, G_{t+1}, ..., G_T\}|H_{G<t}) = \prod_{s=t}^{T} p(G_s|H_{G_{<s}}) \tag{1}$$

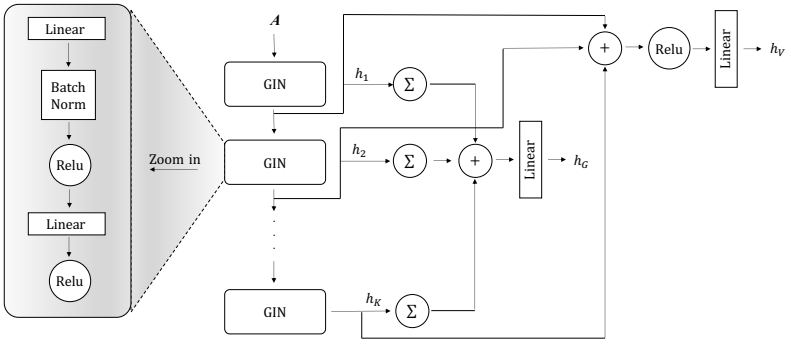

Fig. 2: The GNN module. Σ represents a global sum pooling used for collapsing node embeddings into a graph embedding. The pooling is applied for all hidden node embeddings $h_1, ..., h_K$. The + operator represents concatenation. The grey block zooms in the internal structure of each GIN layer.

In practice, we assume that the history is limited to a window of fixed size w, so that, rather than the whole $H_{G_{<t}}$, we only focus on a windowed portion $\{G_{t-w}, G_{t-w+1}, ..., G_{t-2}, G_{t-1}\}$ of the history, thus reformulating the problem into devising a model for $p(G_t|H_{G_{<t}}, w)$. With an abuse of notation, we shall omit the w when its explicit specification won't affect the context of the discussion.

In this context, we encounter two main challenges. The first challenge arises when dealing with large graphs that contain several nodes, making it highly inefficient to predict the entire graph. The second challenge involves the relationship between the dynamics of the graph and the number of nodes, which causes variations in the structure of the adjacency matrix across timesteps. To address these challenges, we need to develop a model for $p(G_t|H_{G_{<t}})$ that can accurately predict the likelihood of matrices with different sizes. Specifically, accommodating new nodes necessitates continuous adjustments to the matrix size, which requires a flexible representation. Existing literature typically adopts an approach where a maximum size is assumed for V, allowing for a single representation of the adjacency matrix. However, this approach is only a partial solution that significantly compromises the flexibility of the model: Indeed, the maximum size may prove insufficient in continuously evolving contexts characterized by continuous network expansions. As an example, consider a graph where nodes represent information content (news) and edges represent references.

4 Proposed Approach

To cope with the above issues, we approach the first problem from a different perspective. Instead of generating the whole new instance G_t, we aim at predicting the changes resulting from G_{t-1}. For this, we want to build a generative model that, given the partial history $H_{G_{<t}}$ and the window w size of interest, generates two signals: whether the change represents an addition or a deletion, and which components are affected. In practice, at each step, we model three random variables: $Y_t \in \{0, 1\}$ denoting whether

the evolution is represented by a deletion or an addition; $S_t, D_t \in V_t \cup \{v^*\}$, representing the source and destination nodes involved in the evolution. Here, v^* represents a new node not occurring in V_t. Thus, $Y_t = 1$ and $D_t = v^*$ (resp., $S_t = v^*$) denote that a new node has been added. Since no new nodes can be added if disconnected from other nodes, the S_t (resp., D_t) component denotes the peer (either source or destination) connected to the new node. The case where $Y_t = 0$ and both $D_t \in V_t$ and $S_t \in V_t$ denotes the situation where an edge is removed. Notice that, with this model, no explicit node deletion is expressed. In fact, the latter only occurs with disconnected nodes, as explained in Section 3. In the example in Figure 1, the deletion of node 1 occurs at T_4, as a consequence that its last edge with node 2 has been deleted.

We can finally decompose $p(G_t|H_{G_{<t}})$ as the probability of observing Y_t, S_t, D_t:

$$\begin{aligned} p(Y_t, S_t, D_t|H_{G_{<t}}) = \\ p(Y_t|H_{G_{<t}})\, p(S_t|H_{G_{<t}}, Y_t)\, p(D_t|H_{G_{<t}}, S_t, Y_t) \end{aligned} \tag{2}$$

and model these components individually, assuming dependence between S_t and Y_t, and among S_t, D_t and Y_t. A feature shared by all these components is the adoption of a framework combining Graph Convolutional Neural Network (GCN) (Kipf and Welling 2017) and Recurrent Neural Network (RNN) (Graves 2012) to capture both structural and temporal evolution patterns used to predict the future changes. The adoption of GCNs allows us to face the second main challenge in predicting temporal networks. In addition to their suitability for modeling graph properties, they can be exploited to guarantee invariance with respect to the size of the graph and, hence, to model source/destination probabilities in an effective way. Given the adjacency matrix A, we devise the convolutional layer with the following layer-wise propagation rule:

$$h^{l+1} = f\big\{[A + (1+\epsilon)\cdot I]\cdot h^l \cdot W^{l+1}\big\} \quad \in \mathbb{R}^{n\times d_{l+1}}, \tag{3}$$

where f is an activation function (for example, a $ReLU$), ϵ is a fixed scalar, $W^{l+1} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a matrix of learnable weights, $h^0 = \{1\}^{n\times d_0}$, with d_l arbitrarily chosen. This convolutional layer operates with input matrices of variable row size, denoted as n, since A is provided as input. The resulting network architecture ensures output consistency without requiring prior knowledge of the maximum size of the evolution. Additionally, this layer serves as the foundation for the Graph Isomorphism Network (GIN) (Xu et al. 2019), which is considered the most effective among the variants of Graph Convolutional Networks in capturing graph structures. GIN achieves this by generating distinct embeddings for non-isomorphic graphs (Shervashidze et al. 2011).

Neural architecture modules. We use the aforementioned concept layer to develop a generalized Graph Neural Network (GNN) architecture. It takes a graph G as input and produces two embeddings. Firstly, it returns an embedding $h_G \in \mathbb{R}^{d_1}$ for the entire graph. Secondly, it provides a set of embeddings $h_V \in \mathbb{R}^{d_2 \times n}$ for each node $v \in V$. The architecture, depicted in Figure 2, consists of K hierarchical GIN layers that generate hidden node embeddings $h_1, h_2, \ldots h_K$. Additionally, two fully-connected linear layers are employed to generate the graph embedding and the node embeddings, respectively. The dimensions of these layers are represented by d_1 and d_2.

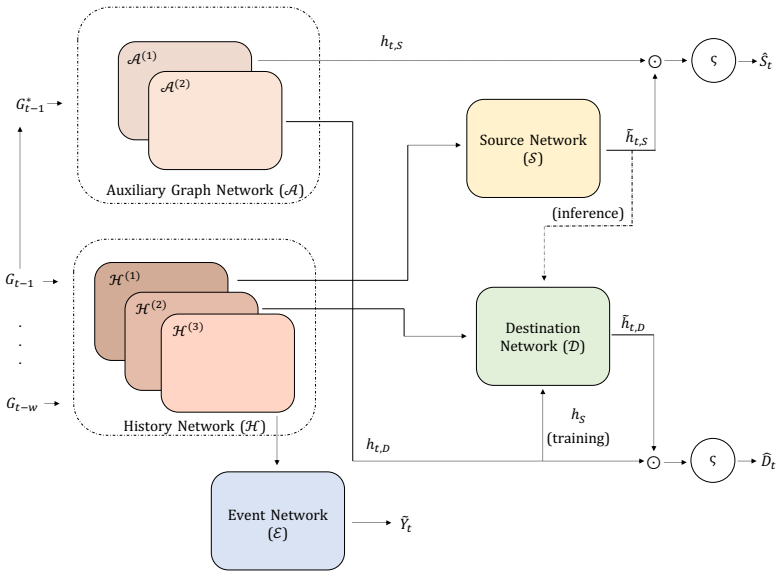

Fig. 3: Overall architecture of FuDGE. History networks are fed with an observation window of size w, $\{G_{t-w}, \ldots, G_{t-1}\}$. G^*_{t-1} is an augmented version of G_{t-1}, where at most 2 brand new nodes are added, that feeds $\mathcal{A}$. $\mathcal{S}$ and $\mathcal{D}$ map the output of $\mathcal{H}$ to embeddings that, combined with the output of $\mathcal{A}$, provide the distributions for sampling the new edge to add to G_{t-1} to generate G_t. $\odot$ represents the element-wise product, while ς stands for the `SoftMax` function.

The *GNN* module is instantiated within an extended network architecture, that we called **Fu**ll **D**ynamic **G**raph **E**volution (FuDGE), shown in Figure 3. FuDGE comprises the following components: (1) An *event network* $\mathcal{E}$ employed for predicting the event type; (2) A *source network* $\mathcal{S}$ and a *destination network* $\mathcal{D}$ utilized to derive embeddings for both source and destination nodes; (3) A set of *history networks* $\{\mathcal{H}^{(j)}\}_{j\in\{1,2,3\}}$ responsible for generating a representation of the entire graph history $H_{G_{<t}}$, exploited respectively by the source, the destination, and the event networks; (4) A set of *auxiliary graph networks* $\{\mathcal{A}^{(j)}\}_{j\in\{1,2\}}$ that generate node embeddings used for matching with source and destination embeddings and generating probability scores. The redundancy on $\{\mathcal{H}^{(j)}\}_{j\in\{1,2,3\}}$ and $\{\mathcal{A}^{(j)}\}_{j\in\{1,2\}}$ allows for the implementation of parallel training strategies. In the following, a dedicated ablation study will evaluate the effectiveness of this architectural choice.

Any $\mathcal{H} \in \{\mathcal{H}^{(j)}\}_{j\in\{1,2,3\}}$ combines the *GNN* module with a recurrent neural network to produce an own global embedding of the history $H_{G_{<t}}$, i.e., $s_{t-1} \in \mathbb{R}^{d_h}$:

$$\left.\begin{aligned} h_{G_i} &= GNN(G_i) \\ s_i &= RNN(h_{G_i}, s_{i-1}) \end{aligned}\right\}, \quad t-w \le i \le t-1$$
$$\mathcal{H}(H_{G_{<t}}) = s_{t-1}\,. \tag{4}$$

Both $\mathcal{S}$ and $\mathcal{D}$ represent simple additional multilayer perceptrons (MLP) that specialize the graph embedding resulting from $\mathcal{H}(H_{G_{<t}})$, yielding the embeddings for the source and destination nodes, respectively. However, while $\mathcal{S}$ is only meant to gather information from the partial evolution, the destination network is also modeled to exploit the additional information coming from a source embedding $h_S \in \mathbb{R}^{d_2}$:

$$\begin{aligned} \mathcal{S}(H_{G_{<t}}) =& \texttt{mlp}\left(\mathcal{H}^{(1)}\left(H_{G_{<t}}\right)\right) \\ \mathcal{D}(H_{G_{<t}}, h_S) =& \texttt{mlp}\left(\mathcal{H}^{(2)}\left(H_{G_{<t}}\right), h_S\right) \end{aligned} \tag{5}$$

These layers produce the final source and destination embeddings to be used jointly with the related auxiliary embeddings provided by $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$. The core intuition here is the following. A source (destination) node can be either one of the nodes existing within V_{t-1} or a new node v^* not existing in V_{t-1}. Thus, in principle, the source (destination) network should produce a multinomial probability among $n+1$ possible choices. We realize this by building an extension graph G^*_{t-1}, which is built from G_{t-1} by including an additional node v^* connected to all possible nodes in V_{t-1} and associated with default feature values. Such a graph is then passed through an additional *GNN* layer, i.e., the auxiliary network $\mathcal{A}$, to compute node embeddings:

$$\mathcal{A}(G^*_{t-1}) = GNN\left(G^*_{t-1}\right) = h_{V^*_{t-1}}, \tag{6}$$

with $\mathcal{A} \in \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$. The final probability scores can be obtained by matching the resulting node embeddings with the one derived from the source (destination) network. Formally:

$$\begin{aligned} h_{t,S} &= \mathcal{A}^{(1)}\left(G^*_{t-1}\right) \qquad \tilde{h}_{t,S} = \mathcal{S}\left(H_{G_{<t}}\right) \\ h_{t,D} &= \mathcal{A}^{(2)}\left(G^*_{t-1}\right) \qquad \tilde{h}_{t,D} = \mathcal{D}\left(H_{G_{<t}}, h_S\right) \\ \hat{S}_t &= \texttt{SoftMax}\left(\tilde{h}_{t,S} \cdot h_{t,S}\right) \qquad \hat{D}_t = \texttt{SoftMax}\left(\tilde{h}_{t,D} \cdot h_{t,D}\right), \end{aligned} \tag{7}$$

where $h_{t,S} \in \mathbb{R}^{n \times d_2}$, $h_{t,D} \in \mathbb{R}^{n \times d_2}$, $\tilde{h}_{t,S} \in \mathbb{R}^{d_2}$ and $\tilde{h}_{t,D} \in \mathbb{R}^{d_2}$. As shown in Equation 7, rather than generating the probability of a link between the two nodes as a whole, which requires pairwise comparisons between all node pairs, FuDGE independently models the selection of the source and destination nodes. Indeed, each node can be independently evaluated as a potential source (destination, resp.) by comparing its embedding with those produced by the source (destination, resp.) network. As a result, identifying the most likely source and destination nodes reduces to linear comparisons between $\tilde{h}_{t,S}$ and $h_{t,S}$, and between $\tilde{h}_{t,D}$ and $h_{t,D}$. This leads to linear computational complexity with respect to the number of nodes.

Finally, we define $p(S_t|H_{G_{<t}}, Y_t) = \hat{S}_t$ and $p(D_t|H_{G_{<t}}, S_t, Y_t) = \hat{D}_t$. For the modeling of Y_t, a similar schema applies:

$$\mathcal{E}(H_{G_{<t}}) = \mathcal{H}^{(3)}\left(H_{G_{<t}}\right); \hat{Y}_t = \texttt{SoftMax}\left(\mathcal{E}(H_{G_{<t}})\right), \tag{8}$$

and $p(Y_t|H_{G_{<t}}) = \hat{Y}_t$.

The model is trained on pairs of training examples $H_{G_{<t}}, G_t$. In particular, under the assumption of single updates at each step, it is possible to infer from G_t the actual values of Y_t, S_t, and D_t. As a consequence, the final loss is a combination of component-wise cross-entropy losses:

$$\ell(H_{G_{<t}}, G_t) = BCE(\hat{Y}_t, Y_t) + CE(\hat{S}_t, S_t) + CE(\hat{D}_t, D_t)\,.$$

It is important to note that in Equation 7, the term h_S serves as a placeholder for the embedding of the hypothetical source node. During training, we replace this component with the embedding obtained from $h_{t,S}$ representing the actual source node S_t. However, during the inference phase, we instantiate it with the component $\tilde{h}_{t,S}$ derived from the source network.

4.1 Extensions

In our proposed approach, we initially focus on a non-attributed directed graph, where we define $h^0 = \{1\}^{n \times d_0}$ for all nodes (Cui et al. 2022), including newly generated ones. However, this formulation can be readily extended to accommodate an attributed directed graph $G = (V, E, X)$, where X is the set of node features. For generality, we assume that $X \in \mathbb{R}^{n \times k}$ is a matrix associating each node u with a feature vector x_u of size k. It is worth mentioning that when a new node occurs, generating its features becomes necessary. In our framework this can be demanded to the networks $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$, that can produce also the required features. Moving forward, to maintain alignment with competitors that exclude feature generation, we proceed with the formulation that does not consider the feature evolution.

4.2 Model complexity

To analyze the complexity of our framework, we first evaluate the cost of each component. For simplicity, K denotes the number of GIN layers used in our architecture (both in the history and auxiliary networks), $L_{\texttt{lin}}$ represents the number of fully-connected linear layers used in each GIN layer, $L_{\texttt{fc}}$ is the number of fully-connected linear layers used in the history network for computing the node and graph embeddings, d is the dimension of the node and graph embeddings, h the number of hidden units in the recurrent layer, w the window size. The overall cost of the history network is:

$$O\left(w\left(K \cdot L_{\texttt{lin}} \cdot (n^2 \cdot d + n \cdot d^2 + n) + L_{\texttt{fc}} \cdot n \cdot d^2 + h \cdot d + h^2\right)\right) \tag{9}$$

The cost of the auxiliary network is $O\left(K \cdot L_{\texttt{lin}} \cdot (n^2 \cdot d + n \cdot d^2 + n) + L_{\texttt{fc}} \cdot n \cdot d^2\right)$. The cost of the source, destination and event networks is $O\left(d \cdot h\right)$, while the cost of computing the dot product within the softmax operation (as in Eq. 7) is $O\left(n \cdot d^2\right)$. Summarizing, the overall complexity of the model simplifies to the cost of the history network given in Eq. 9.

5 Experimental Assessment

In this section, we thoroughly evaluate the proposed model through empirical analysis of both real-world and synthetic datasets. The aim is to determine the effectiveness and validity of the modeling choices put forth in this study. Our goal is to answer the following research questions:

- **RQ1**. Can FuDGE be used to track and predict graph evolution in real-world scenarios? How does it compare to state-of-the-art approaches? (Section 5.4.1).
- **RQ2**. Can FuDGE generate realistic graphs? That is, in a generative setting, when sampling from the underlying probability distributions, does the resulting graph evolution produce meaningful, realistic, and non-trivial events? (Section 5.4.1).
- **RQ3**. Which components of the model contribute to the overall quality? How do the architectural and learning choices affect the accuracy of the model performance? (Section 5.4.2).
- **RQ4**. In terms of efficiency, how do the learning and inference processes of FuDGE compare to other state-of-the-art approaches? (Section 5.4.3).

Next, we provide an overview of the datasets, baselines, and competitors used for evaluation, as well as the evaluation protocol adopted. We will address the aforementioned questions in the subsequent subsections.

5.1 Datasets

The experiments are based on four real-world benchmark datasets, described below.

1. **Bitcoin-Alpha**[1] and **Bitcoin-OTC**[2]. The datasets consist of graphs derived from two online platforms (Bitcoin Alpha and Bitcoin OTC) for Bitcoin trading. These graphs are generated from the transaction network, where nodes represent Bitcoin users and edges represent the ratings assigned by one user to another, indicating the level of trust in their interactions.
2. **UCI-Forum**[3]. The dataset represents the exchange of messages in an online social network at the University of California, Irvine. Users are modeled as nodes, and messages as edges.
3. **EU-Core**[4]. The graphs in the dataset represent the temporal evolution of exchanged emails in a certain time frame between users. Nodes represent users, and edges are the pairs (sender, receiver) of each e-mail.

Some datasets (e.g., EU-Core and UCI-Forum) contain duplicate edges, i.e., an edge that occurs multiple times in the graph. While our framework can generate such edges, we opt to concentrate solely on non-duplicated ones for the sake of comparison with the baselines and to streamline the evaluation protocol. Consequently, all datasets undergo preprocessing to eliminate duplicates. Table 2 displays the statistics of the datasets after the preprocessing stage.

[1] https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html
[2] https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html
[3] https://archive.ics.uci.edu/ml/datasets/Labeled+Text+Forum+Threads+Dataset
[4] https://snap.stanford.edu/data/email-Eu-core-temporal.html

Dataset	$\|V\|$	$\|E\|$	D	CC_{avg}	Density	Timestamps
Synthetic	2,002	2,593	19	0.0022	0.0006	3,388
Bitcoin-Alpha	3,783	24,186	10	0.1583	0.0017	1,647
Bitcoin-OTC	5,881	35,592	11	0.1511	0.0010	35,592
UCI-Forum	899	7,089	9	0.0366	0.0088	7,076
EU-Core	986	24,929	7	0.3727	0.0256	20,414

Table 2: Dataset Description. D is used to indicate the graph diameter, and CC_{avg} for the average clustering coefficient.

The real-world datasets in our study solely focus on graph evolution with growth and do not involve any edge or node removal events. To explore the effects of complete dynamics, we introduce a synthetic data generation approach that allows for both insertions and deletions. The synthetic data generation strategy aims to devise a process that governs the occurrence of insertion and deletion events based on certain structural properties of the graph. Throughout the experiments, we will assess the ability of FuDGE and its competitors to capture and replicate this process. The synthetic graph generation is described in the Appendix.

5.2 Competitors and Baselines

For comparative analysis, we adapt models devised for link prediction task which achieve state-of-the-art quality compared to other approaches described in section 2.

- Temporal Graph Network (TGN) (Rossi et al. 2020). It proposes four variants concerning the embedding relative to node information: *identity*, which simply returns the unaltered memory for a given node; *time*, which computes an embedding based on temporal aspects, *attention*, which exploits attention networks; and *sum*, which aggregates neighbor embeddings. In our experiments, we used all these variants for our comparisons.
- Neighborhood-Aware-Temporal-Network (NAT) (Luo and Li 2022). We tuned two parameters relative exploiting neighborhood size from the range $\{16, 32\}$ and the embedding size from the range $\{2, 4\}$.
- TGL, a general framework for training TGNN on billion-scale graphs (Zhou et al. 2022a). This model implements a general strategy to enhance the capabilities of SOTA models such as TGN, JODIE (Kumar et al. 2019), DySAT (Sankar et al. 2020), TGAT (Xu et al. 2020), and APAN (Wang et al. 2021a). We used all variants except for the latter.

Other potential competitors, namely (Wu et al. 2020) and (Lim et al. 2020), present approaches that bear the closest resemblance to FuDGE. Nevertheless, our initial experiments indicated that both these approaches face challenges when applied to large graphs. This is primarily due to their underlying modeling decisions, particularly in the generation of the complete adjacency matrix and/or the utilization of internal representations that do not efficiently support GPU processing. These constraints make these approaches impractical or infeasible for handling large-scale graph data.

In addition to the above competitors, we also use some baseline graph generation models from network science. These models are useful to compare the properties of the generated graph evolution.

- Erdős-Rényi (ER) (Erdős et al. 1960) generates random graphs given a fixed number of nodes, adding an edge between two nodes at each step with probability p.
- Barabási-Albert (BA) (Barabási and Albert 1999) generates graphs according to the preferential attachment principle: new edges are likely to be linked to nodes with a higher degree;
- Power-law (POWER) (Holme and Kim 2002), based on the Barabási-Albert model: each step a new node x is connected according to PA with an existing node y; moreover, with a probability p a new edge is created between x and a neighbor of y.

For the dynamic generation, we adopt TIGGER (Gupta et al. 2022) and a recent approach, named Variational Recurrent Dynamic Attributed Graph Generator (VRDAG) (Li et al. 2025c). TIGGER learns the joint structural and temporal distributions of dynamic graphs to generate new temporal interaction graphs, while VRDAG captures the underlying distribution of dynamic graphs to generate new instances accordingly. Following the original VRDAG work, we tuned two parameters: the latent embedding dimensionality, selected from the range $\{4, 32\}$, and the number of Bernoulli mixtures, chosen from $\{2, 3\}$.

5.3 Evaluation Protocol

Training details and model settings. Each dataset exhibits a dynamic graph as a sequence of temporally-sorted graphs $\{G_1, G_2, ..., G_T\}$. In this sequence, we apply a chronological split to obtain training D_{tr}, validation D_{va}, and test D_{te} subsequences. We perform three sets of experiments with different split percentages: 50/25/25, 60/20/20, and 70/15/15 ratio for training/validation/test, respectively. To identify $H_{G<t}$ at each step t, we apply a sliding window procedure to obtain a set of fixed-size observation windows. In the training phase, we set the window size to 30 and select samples using a window shift of 10.

Our implementation of FuDGE is based on the PyTorch framework (Paszke et al. 2019). To foster reproducibility, we publicly release all the data and code necessary to replicate our experiments[5]. As described in Section 4, FuDGE is a composite architecture including several modules. $\mathcal{H}$ combines graph and recurrent modules. The graph module, shown in Figure 2, consists of two GIN layers producing hidden embedding h_1 and h_2.We use the graph module also for $\mathcal{A}$. In the recurrent module, a GRU (Cho et al. 2014) with three layers is instantiated. The $\mathcal{S}$ and $\mathcal{D}$ modules are composed of three fully-connected linear layers. The first two fully-connected linear layers are equipped with a Rectified Linear Unit (ReLU) activation function. In addition, two Dropout layers are also used with a dropout rate equal to 0.3. Finally, $\mathcal{E}$ consists of two fully-connected linear layers, where the first one is equipped with the ReLU.

As shown in Figure 3, the reference architecture of FuDGE instantiates three $\mathcal{H}$ and two $\mathcal{A}$ modules, with no weight sharing. For these modules, tuning of the hyperparameters has been performed to select the best configuration. In particular, the

[5]The code to reproduce our experiments is available at: https://anonymous.4open.science/r/fudge-3C25

node/graph embedding size is selected in the range $\{16, 32, 64\}$, and the optimal value is reported in the results. Specifically, we perform three sets of experiments: (*i*) In the graph module, we instantiate the GIN layer and the two linear layers with 16 neurons, whereas, the GRU layers with 8 neurons. Linear layers in the source, destination, and event networks include 16, 16, and 8 neurons, respectively; (*ii*) alternatively, GIN and linear layers include 32 neurons, and GRU layers with 16 neurons. Linear layers in the source, destination, and event networks include 32, 32, and 16 neurons, respectively; (*iii*) Finally, GIN and linear layers are configured with 64 neurons, while the GRU layers with 32 neurons. Linear layers in the source, destination, and event networks include 64, 64, and 32 neurons, respectively.

We utilize the Adam optimizer (Kingma and Ba 2015). For the Bitcoin datasets, a learning rate of 10^{-4} is employed, whereas for all other datasets, the learning rate is 10^{-3}. Each model undergoes training for 200 epochs.

The best models that maximize the given metric, i.e., the WL, on the validation set are used in the inference phase. All the link prediction-based competitors were trained for $1,000$ epochs with early-stopping criteria to avoid overfitting. Furthermore, because VRDAG requires excessive training time due to the need to generate the entire graph at each timestamp, we limited the maximum training time to 12 hours. For all the other parameters (except the ones used for the hyper-tuning), we follow the authors' suggestions. Baseline models need no training but a simple estimation of the underlying parameters (such as probability for triangle closure in POWER or edge likelihood in ER). For these, we use the last graph of the training set. The parameters resulting from this estimation are then used in the inference phase. All experiments were performed on an NVIDIA DGX equipped with 4 V100 GPUs.

Inference and graph generation. We aim to assess the predictive accuracy of FuDGE in the context of predicting the full graph evolution, as outlined in Section 3. This evaluation differs from the traditional predictive setting where the objective is to predict G_t given a partial history $H_{G<t}$. Instead, we are interested in evaluating the model's capability to predict the entire remaining history $\{G_t, G_{t+1}, \ldots, G_T\}$. This autoregressive scenario is more complex as predicting G_s for $s > t$ requires multiple generative steps in a situation of high uncertainty. In other words, predicting G_{t+1} relies on knowledge about the event at step t, which cannot be assumed in a realistic scenario. Consequently, our approach is to evaluate $p(G_{t+1}|H_{G<t} \cup \hat{G}_t)$, where $\hat{G}_t$ is sampled from $p(G_t|H_{G<t})$. This recursive scheme applies to all components of the remaining history. Thus, we initiate the process with the most recent history window H_G extracted from the validation set. Then, for each time step t in the test set, we sample the next event through $\hat{Y}_t$, $\hat{S}_t$ and $\hat{D}_t$ and use it to generate $\hat{G}_t$ which is subsequently added to the history window.

For the link prediction-based competitors, the inference phase requires special attention. Indeed, at a step t, the generation of $\hat{G}_t$ relies on selecting a candidate link, which in turn requires inferring the probability for all candidate edges. Within realistic graphs, the number of candidates is intractably large (i.e., quadratic in the number of nodes), so we need to resort to sampling. At each time step t, a subset of edges is selected from which the edge likely to appear at time $t+1$ is chosen. Sampling poses a potential issue for competing approaches, as the choice of sample size can significantly

affect efficiency and algorithm performance. We chose a fixed-size sample set (equal to $4,096$) for our comparison. Notably, in FuDGE, the generation phase is distinctive in that it does not require exhaustive exploration of the entire space of potential edges. This is achieved by decoupling the selection of source and destination nodes, resulting in a linear complexity with respect to the number of nodes. We further tailor the link-prediction task of the competitors to our generation protocol. In particular, additional information concerning neighborhood, edge features, and node features relative to unknown edges and nodes is only considered based on the generated history, rather than the true history.

For the VRDAG competitor, it is important to note that it is specifically designed to learn the underlying distribution of a dynamic graph and subsequently synthesize new graph instances. Therefore, unlike its original evaluation protocol, where the inference phase generates the same graph observed during the learning phase, we assess the performance of VRDAG using our own evaluation protocol. In our setting, the model is required to generate $\hat{G}_t$ for each time step t in the test set.

Lastly, it is worth mentioning that some datasets exhibit multiple events in a single time step. To ensure a fair evaluation on these datasets, a post-processing strategy is employed to group individual generation steps together. This requires an additional inference step to determine the size of each group. We utilize a simple probabilistic model that estimates the size distributions based on their frequencies in the training set. While more advanced models can be developed, their exploration falls outside the scope of this paper.

Evaluation metric. Due to the objective of characterizing long-term evolution, metrics that assess the accuracy of step-size predictions are not appropriate. As mentioned earlier, predicting the complete evolution involves sequential graph generation steps. Therefore, if an erroneous generation occurs at any point, it can have a cascading effect on subsequent steps, compromising the overall process and leading to a lower accuracy score. Moreover, accuracy may not adequately capture situations where the likelihood of subsequent events is reversed, but the overall predictions are correct when viewed holistically. To mitigate the aforementioned issues, we employ graph kernels, specifically the Weisfeiler-Lehman (WL) subtree kernel (Shervashidze et al. 2011; Weisfeiler and Leman 1968), as proposed in (Wu et al. 2020). This kernel utilizes graph isomorphism to evaluate the quality of generated graphs. Graph isomorphism refers to a one-to-one mapping, denoted by π, between the vertices of two graphs, preserving their adjacency relationships. For instance, if there is an edge (i, j) in graph G, then there should be a corresponding edge $(\pi(i), \pi(j))$ in graph G'. The WL test is an isomorphism test that employs color refinement. It represents a necessary but insufficient condition for stating that two graphs are isomorphic; nonetheless, the test can distinguish almost all non-isomorphic graphs (Babai et al. 1980). In the WL test, each node in the graphs is assigned a color based on certain criteria, such as uniform or degree-based assignments. The algorithm then iteratively updates the colors of nodes based on those of their neighboring nodes for a specified number of steps. This process results in two graph colorings. If the graphs have the same color distribution, they will likely be isomorphic. We utilized the WL test to compute a similarity score between two graphs in our evaluation. To perform this computation,

we leveraged the GraKeL Python library[6]. In our experiments, we initialized the node labels based on their degrees. We employed two distinct stopping conditions for the WL score computation: "Same Col" and "Max Diam". The "Same Col" condition terminates the computation when the coloring process reaches a stable state, where node colors no longer change. On the other hand, the "Max Diam" condition sets the number of computation steps equal to the diameter of the last graph in the test set.

In addition to the WL test, we have incorporated several graph structural metrics into our evaluation framework to facilitate a comprehensive assessment of the generated graphs. In this sense, a comparative analysis between the actual and generated final graphs is involved, leveraging key properties, including the distributions of in-degree, out-degree and clustering coefficients, wedge count, number of components, and the size of the largest connected components. For the evaluation, we follow the approach described in (Li et al. 2025c). This rigorous evaluation methodology ensures a more robust and nuanced evaluation of the generated graphs.

5.4 Results

5.4.1 Comparative analysis

Table 3 summarizes the experiments to evaluate FuDGE against the state-of-the-art competitors over the chosen data sets, with the split 70/15/15. All the experiments are performed on five runs, and the average values are reported, with pairwise (FuDGE vs. competitor) statistical significance computed at 95% confidence. Bold, gray, and unaltered values denote a win, loss, and tie, respectively. As discussed in section 5.3, we train and evaluate the model on different configurations and report the configuration guaranteeing optimal performance. The same protocol is adopted for the competitors by analyzing all possible variants discussed in section 5.2. The reported results, therefore, reflect the best parameter combinations for all models.

The results show that FuDGE achieves competitive performance across all datasets when compared to state-of-the-art approaches. Specifically, on the Synthetic dataset, FuDGE achieves the highest WL similarity scores considering the Max Diam strategy and remains competitive with TGN under the Same Col strategy. On Bitcoin-Alpha, it achieves superior or competitive WL similarity, while also performing well on structural metrics. For Bitcoin-OTC, although TIGGER achieves the best WL score under the Max Diam strategy, FuDGE remains competitive across nearly all metrics, whereas some models (e.g., VRDAG) suffer from scalability issues. On UCI-Forum, TGL and TIGGER outperform FuDGE in WL similarity, though FuDGE still delivers strong results on the remaining metrics. A similar trend appears in EU-Core, where TIGGER surpasses FuDGE in WL similarity but fails to capture key structural properties.

To comparatively study the robustness of FuDGE and the competitors in predicting long-term evolution, we also investigate how the accuracy, in terms of WL similarity, degrades according to the length of the generation phase. Figure 4 shows on the X axis the generation step and the relative WL similarity on the Y axis. The graphs show that FuDGE is more resilient in its ability to predict long-term evolution while

[6] https://ysig.github.io/GraKeL/

Dataset	Model	Weisfeiler-Lehman ↑		In-degree distribution ↓	Out-degree distribution ↓	Clustering distribution ↓	Wedge count ↓	# Components ↓	Size of Largest Connected Components ↓
		Same Col	Max Diam						
Synthetic	FuDGE	**0.9651 ± 0.0015**	**0.9280 ± 0.0005**	0.0068 ± 0.0006	0.0012 ± 0.0000	0.0013 ± 0.0000	0.1595 ± 0.0007	0.0169 ± 0.0042	0.1082 ± 0.0005
	BA	0.8986 ± 0.0021	0.8704 ± 0.0014	**0.0066 ± 0.0012**	**0.0012 ± 0.0000**	**0.0016 ± 0.0008**	**0.0752 ± 0.0249**	0.0235 ± 0.0000	0.1780 ± 0.0005
	ER	0.0016 ± 0.0004	0.0014 ± 0.0004	**0.0074 ± 0.0006**	0.0332 ± 0.0035	0.0075 ± 0.0008	11.7127 ± 1.8168	0.9953 ± 0.0000	0.3199 ± 0.0000
	POWER	0.9007 ± 0.0037	0.8788 ± 0.0029	**0.0071 ± 0.0006**	**0.0012 ± 0.0000**	**0.0014 ± 0.0006**	**0.1095 ± 0.0074**	0.0235 ± 0.0000	0.1449 ± 0.0061
	TGN	**0.9635 ± 0.0027**	0.9100 ± 0.0078	**0.0071 ± 0.0009**	0.0037 ± 0.0032	0.0043 ± 0.0022	**0.1279 ± 0.0063**	0.2873 ± 0.0422	**0.0440 ± 0.0175**
	NAT	0.9602 ± 0.0043	0.9148 ± 0.0065	**0.0070 ± 0.0003**	**0.0018 ± 0.0008**	0.0080 ± 0.0038	**0.1327 ± 0.0062**	0.2967 ± 0.0562	**0.0540 ± 0.0058**
	TGL	0.9447 ± 0.0048	0.8855 ± 0.0226	**0.0086 ± 0.0027**	0.0052 ± 0.0014	0.0030 ± 0.0008	0.5962 ± 0.0094	0.9944 ± 0.0021	**0.0563 ± 0.0059**
	VRDAG	0.3315 ± 0.1789	0.3353 ± 0.1731	**0.0086 ± 0.0035**	0.0933 ± 0.0964	0.0049 ± 0.0038	**0.4451 ± 0.4587**	0.9304 ± 0.0389	0.1276 ± 0.0160
	TIGGER	0.9461 ± 0.0028	0.8585 ± 0.0016	**0.0031 ± 0.0012**	0.0020 ± 0.0004	0.0115 ± 0.0086	**0.0858 ± 0.0118**	91.4000 ± 83.5123	0.1408 ± 0.0442
Bitcoin-Alpha	FuDGE	**0.9659 ± 0.0016**	0.9392 ± 0.0023	**0.0059 ± 0.0014**	0.0086 ± 0.0004	**0.0115 ± 0.0009**	0.0303 ± 0.0072	0.4800 ± 0.1095	0.0292 ± 0.0004
	BA	0.1452 ± 0.0081	0.1672 ± 0.0105	0.0150 ± 0.0054	**0.0068 ± 0.0000**	0.0260 ± 0.0076	0.4127 ± 0.0387	**0.0000 ± 0.0000**	1.3002 ± 0.0790
	ER	0.0024 ± 0.0004	0.0023 ± 0.0005	0.1638 ± 0.0198	0.1745 ± 0.0164	0.5204 ± 0.0000	10.3725 ± 3.7950	0.8000 ± 0.0000	1.3027 ± 0.0790
	POWER	0.1683 ± 0.0092	0.1791 ± 0.0098	0.0132 ± 0.0062	**0.0068 ± 0.0000**	0.0227 ± 0.0053	**0.0038 ± 0.0013**	**0.0000 ± 0.0000**	1.2970 ± 0.0774
	TGN	0.7215 ± 0.0243	0.6875 ± 0.0267	0.0104 ± 0.0034	0.0148 ± 0.0049	0.0292 ± 0.0048	**0.0157 ± 0.0055**	0.6400 ± 0.0894	**0.0036 ± 0.0032**
	NAT	0.6923 ± 0.0681	0.6499 ± 0.0701	0.0101 ± 0.0020	0.0126 ± 0.0033	0.0266 ± 0.0034	**0.0132 ± 0.0047**	0.8000 ± 0.0000	**0.0014 ± 0.0007**
	TGL	0.8056 ± 0.0203	0.5939 ± 0.0111	0.0099 ± 0.0018	**0.0111 ± 0.0028**	0.0439 ± 0.0064	0.0462 ± 0.0066	0.6800 ± 0.1789	**0.0085 ± 0.0014**
	VRDAG	0.1449 ± 0.1713	0.1223 ± 0.1522	0.1471 ± 0.1680	0.1477 ± 0.1686	0.1193 ± 0.0389	29.8089 ± 34.1246	0.8000 ± 0.0000	**0.0017 ± 0.0008**
	TIGGER	0.9554 ± 0.0021	**0.9402 ± 0.0024**	0.0140 ± 0.0052	**0.0106 ± 0.0027**	0.5342 ± 0.0000	0.1219 ± 0.0350	0.8400 ± 0.3847	0.8161 ± 0.0012
Bitcoin-OTC	FuDGE	**0.8788 ± 0.0196**	**0.8690 ± 0.0198**	**0.0068 ± 0.0003**	0.0128 ± 0.0001	0.0118 ± 0.0001	1.3447 ± 0.0000	0.7500 ± 0.0000	0.1015 ± 0.0000
	BA	0.2935 ± 0.0039	0.3579 ± 0.0002	0.0110 ± 0.0006	**0.0109 ± 0.0000**	0.0200 ± 0.0028	**0.0684 ± 0.0108**	**0.2500 ± 0.0000**	0.7856 ± 0.0001
	ER	0.0041 ± 0.0001	0.0037 ± 0.0001	0.2077 ± 0.0130	0.2114 ± 0.0051	0.5616 ± 0.0000	4.1558 ± 0.0266	0.7500 ± 0.0000	0.7864 ± 0.0000
	POWER	0.3184 ± 0.0027	0.3509 ± 0.0014	0.0100 ± 0.0026	**0.0109 ± 0.0000**	0.0166 ± 0.0028	**0.2111 ± 0.0015**	**0.2500 ± 0.0000**	0.7851 ± 0.0006
	TGN	0.8003 ± 0.0565	**0.8390 ± 0.0761**	0.0127 ± 0.0018	**0.0110 ± 0.0010**	**0.0126 ± 0.0023**	**0.2117 ± 0.0051**	0.4500 ± 0.2092	0.0427 ± 0.0448
	NAT	**0.7894 ± 0.0930**	0.7542 ± 0.1287	0.0105 ± 0.0027	**0.0112 ± 0.0007**	0.0131 ± 0.0041	**0.2149 ± 0.0052**	0.5500 ± 0.2739	**0.0215 ± 0.0285**
	TGL	**0.8664 ± 0.0185**	**0.8464 ± 0.0051**	0.0084 ± 0.0013	**0.0093 ± 0.0017**	0.0183 ± 0.0040	**0.2068 ± 0.0030**	0.6500 ± 0.1369	**0.0348 ± 0.0058**
	VRDAG	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	TIGGER	**0.9078 ± 0.0019**	**0.9267 ± 0.0019**	0.0097 ± 0.0020	0.0156 ± 0.0063	0.5761 ± 0.0000	**0.1184 ± 0.0357**	1.1500 ± 0.4183	0.6139 ± 0.0009
UCI-Forum	FuDGE	**0.6956 ± 0.0050**	**0.5897 ± 0.0031**	0.0081 ± 0.0004	0.0056 ± 0.0002	**0.0041 ± 0.0002**	**0.0389 ± 0.0002**	0.0000 ± 0.0000	0.1001 ± 0.0000
	BA	0.1466 ± 0.0003	0.1445 ± 0.0003	**0.0064 ± 0.0009**	**0.0047 ± 0.0000**	0.0055 ± 0.0014	0.0516 ± 0.0078	**0.0000 ± 0.0000**	1.0812 ± 0.0014
	ER	0.1780 ± 0.0094	0.1650 ± 0.0079	0.2419 ± 0.0060	0.1366 ± 0.0056	0.0570 ± 0.0009	0.9708 ± 0.0011	47.2000 ± 8.9275	1.0249 ± 0.0115
	POWER	0.1563 ± 0.0003	0.1546 ± 0.0003	**0.0089 ± 0.0024**	**0.0047 ± 0.0000**	0.0060 ± 0.0014	0.1849 ± 0.0046	**0.0000 ± 0.0000**	1.0812 ± 0.0014
	TGN	**0.7668 ± 0.0439**	**0.5986 ± 0.0636**	**0.0110 ± 0.0027**	**0.0045 ± 0.0002**	0.0329 ± 0.0195	0.1702 ± 0.0190	**0.0000 ± 0.0000**	**0.0610 ± 0.0311**
	NAT	**0.7106 ± 0.0425**	**0.5272 ± 0.0595**	**0.0121 ± 0.0053**	**0.0044 ± 0.0004**	0.0186 ± 0.0126	0.1949 ± 0.0130	**0.0000 ± 0.0000**	**0.0189 ± 0.0210**
	TGL	**0.7977 ± 0.0153**	**0.6720 ± 0.0099**	**0.0104 ± 0.0044**	0.0095 ± 0.0067	0.0064 ± 0.0009	0.1697 ± 0.0076	**0.0000 ± 0.0000**	**0.0801 ± 0.0057**
	VRDAG	0.1213 ± 0.0364	0.0967 ± 0.0270	0.0332 ± 0.0394	0.0627 ± 0.0272	0.0444 ± 0.0172	2.3906 ± 0.7582	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
	TIGGER	**0.7698 ± 0.0052**	**0.6500 ± 0.0041**	0.0072 ± 0.0011	0.0052 ± 0.0009	0.0234 ± 0.0403	0.1218 ± 0.0121	0.4000 ± 0.8944	0.1880 ± 0.1903
EU-Core	FuDGE	**0.7849 ± 0.0138**	**0.7453 ± 0.0172**	**0.0100 ± 0.0005**	**0.0083 ± 0.0002**	**0.0507 ± 0.0010**	**0.0480 ± 0.0001**	**0.0000 ± 0.0000**	0.0365 ± 0.0000
	BA	0.0236 ± 0.0007	0.0248 ± 0.0008	**0.0128 ± 0.0019**	0.0087 ± 0.0000	**0.0252 ± 0.0041**	0.0615 ± 0.0078	**0.0000 ± 0.0000**	3.8363 ± 0.1318
	ER	0.0068 ± 0.0005	0.0067 ± 0.0005	0.2713 ± 0.0236	0.2811 ± 0.0238	0.7372 ± 0.0003	2.6842 ± 0.5293	**0.0000 ± 0.0000**	3.8363 ± 0.1318
	POWER	0.0254 ± 0.0006	0.0259 ± 0.0007	**0.0111 ± 0.0028**	0.0087 ± 0.0000	**0.0231 ± 0.0061**	0.1689 ± 0.0037	**0.0000 ± 0.0000**	3.8363 ± 0.1318
	TGN	0.4875 ± 0.0465	0.3672 ± 0.0640	**0.0108 ± 0.0029**	0.0109 ± 0.0020	**0.0272 ± 0.0095**	0.1185 ± 0.0112	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
	NAT	0.5222 ± 0.0575	0.3977 ± 0.0659	**0.0101 ± 0.0021**	**0.0099 ± 0.0023**	**0.0214 ± 0.0063**	0.1154 ± 0.0109	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
	TGL	0.7098 ± 0.0224	0.6352 ± 0.0267	**0.0091 ± 0.0027**	**0.0107 ± 0.0034**	**0.0532 ± 0.0038**	0.0950 ± 0.0077	**0.0000 ± 0.0000**	**0.0057 ± 0.0021**
	VRDAG	0.0939 ± 0.0863	0.0539 ± 0.0536	0.1050 ± 0.0984	0.1364 ± 0.1827	0.5299 ± 0.0552	14.7910 ± 18.3215	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
	TIGGER	**0.8796 ± 0.0063**	**0.8584 ± 0.0075**	0.0123 ± 0.0011	0.0093 ± 0.0045	0.7700 ± 0.0000	**0.0447 ± 0.0431**	1.6000 ± 0.5477	0.7489 ± 0.0026

Table 3: Comparison between FuDGE and state of the art methods. The results show the WL score as well as the main network properties computed between the final graphs resulting from the actual and predicted evolution. Bold values denote scores superior to our method, while gray values indicate inferior ones. The metrics should be interpreted as follows: ↓ indicates that lower values are better, while ↑ indicates that higher values are better.

minimizing the generation error. In contrast, for the competitors, except TIGGER, a drop in their performances is already observed from the initial stages of generation.

We also evaluated the competitor's performance with different sample sizes. The results show marginal changes with respect to results shown here, at the cost of significantly slower generation times.

To address **RQ2**, we compare the degree distributions of the final graphs obtained from the real evolution and those generated (predicted) by FuDGE. As shown in Figure 5, the distributions largely overlap, with only minor deviations on datasets containing the highest-degree nodes. This observation is consistent with the results reported in Table 3, where FuDGE demonstrates competitive performance against the competitors in capturing structural properties.

Finally, we note that the performance differences between the baselines and FuDGE can be largely attributed to the density of the graphs (Table 2). Baseline models tend to outperform FuDGE on dense graphs, such as EU-Core (0.02) and UCI-Forum (0.009), whereas FuDGE performs better on sparser graphs, including Synthetic (0.0006) and Bitcoin networks (0.001). This trend can be attributed to the more conservative generation behavior of FuDGE, which assigns relatively lower edge existence probabilities. In sparse regimes this conservative bias helps preventing the generation of spurious connections, resulting in graphs that more faithfully match the underlying topology.

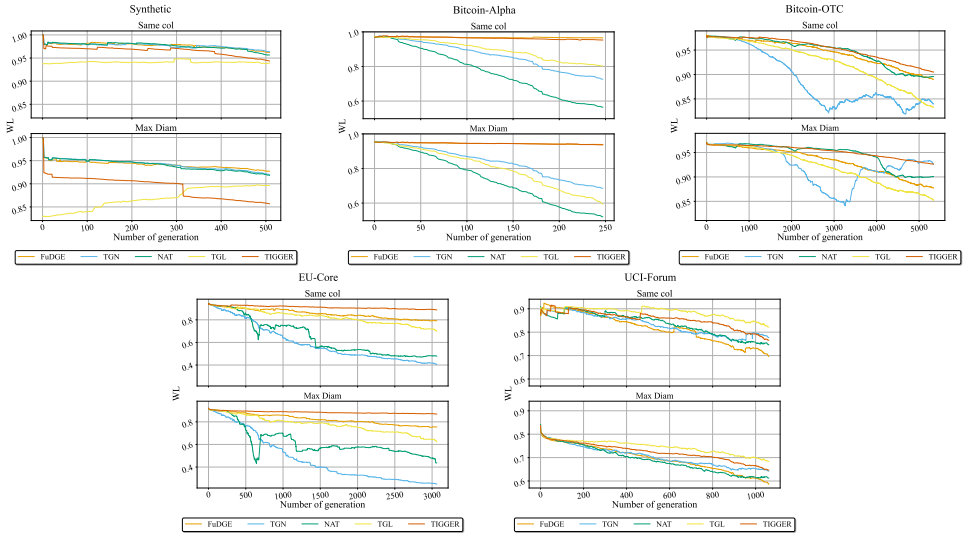

Fig. 4: Long-Term Graph Generation.

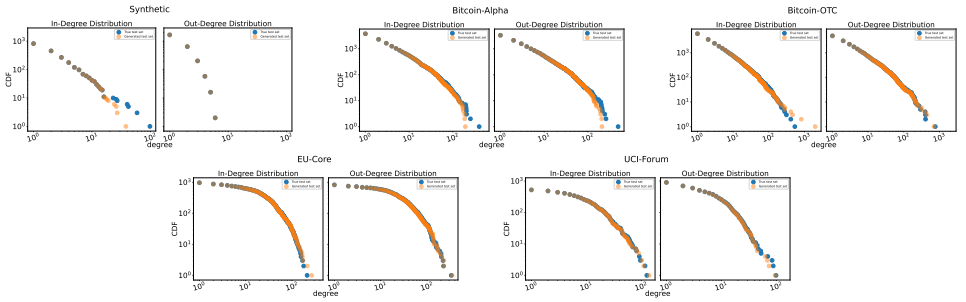

Fig. 5: Cumulative In/Out-Degree Distribution for real and generated graphs.

Conversely, in denser graphs, the abundance of structural signals favors baseline generators that more aggressively instantiate edges, reducing the relative advantage of FuDGE.

5.4.2 Sensitivity analysis and ablation study

In a further set of experiments, we study the contribution of each model component to the quality of the generated graphs. We evaluate mainly the choices that characterize the architecture proposed in Figure 3 and in particular the importance of having different instantiations $\mathcal{H}^{(i)}$. The basic architecture used in the previous experiments is composed of three $\mathcal{H}$ and two $\mathcal{A}$. We study an alternate configuration where the instances of $\mathcal{H}$ are collapsed and shared by the other modules. Within Table 4, FuDGE$_{(\mathcal{H},2\mathcal{A})}$ represent the model with one instance of $\mathcal{H}$. The experiments, performed on the synthetic dataset on the split 70/15/15, show that the reference

Model	Strategy	Configurations		
		(i)	(ii)	(iii)
FuDGE	Same Col	0.9621	0.9559	0.9606
	Max Diam	0.9270	0.9203	0.9252
FuDGE$_{(\mathcal{H},2,\mathcal{A})}$	Same Col	0.9665	0.9476	0.9588
	Max Diam	0.9268	0.9145	0.9222

Table 4: Ablation Study

architecture with redundant instances and no weight sharing offers a competitive advantage, besides being amenable to parallel training. Further, we adopt independent $\mathcal{H}$ modules because each module is specialized to model a distinct aspect of the system: source, destination, and event networks. This modular design allows each $\mathcal{H}$ to focus on learning patterns specific to its corresponding network, improving overall predictive accuracy as reported in Table 4.

We also perform a sensitivity analysis to study how the architectural and learning choices influence the model performances. The results computed on the validation set are shown in Table 5. First, we evaluate the behavior for different chronological splits of the datasets, to measure the robustness to limited amounts of training data. From the table, we can see that the decrease in quality is consistent almost anywhere. In both tables 4 and 5, we also analyze the contribution of configurations to the reconstruction quality. We can see that datasets with few nodes, e.g., EU-Core, tend to have better results with a low-dimensional embedding, whereas larger datasets require larger configurations. By contrast, the number of edges and other features does not seem to affect the model capacity.

5.4.3 Model efficiency

We finally compare the execution times of each model during both the training and generation phases, as shown in Figure 6. We want to stress that, according to our fair protocol, the training times also account for the validation phase, which is needed to select the best configuration. Overall, during the training phase, FuDGE is among the fastest models on most datasets, particularly on Synthetic, Bitcoin-OTC, and UCI-Forum. However, on Bitcoin-Alpha, FuDGE is slower than most competitors, being faster only than TIGGER, while on EU-Core it achieves comparable but not dominant performance. The same pattern can be observed in the generation phase, where FuDGE is even faster than the competitors that resort to sampling, as discussed before.

6 Conclusion

We introduced FuDGE, a deep neural network architecture that combines graph convolution and recurrence to support full graph evolution. With its modular design, FuDGE ensures invariance to different input sizes and generates effective node and graph representations for capturing dynamic graph patterns. Our extensive experiments demonstrate that FuDGE presents several benefits both in terms of efficacy and efficiency over state-of-the-art models, showcasing its competitive advantage.

Dataset	Strategy	70%			60%			50%		
		(*i*)	(*ii*)	(*iii*)	(*i*)	(*ii*)	(*iii*)	(*i*)	(*ii*)	(*iii*)
Synthetic	Same Col	**0.9504**	0.9452	0.9389	**0.9515**	0.9302	0.9323	**0.8934**	0.8615	0.8775
	Max Diam	**0.9237**	0.9182	0.9152	**0.9210**	0.8999	0.8934	0.8438	0.8347	**0.8486**
Bitcoin-Alpha	Same Col	0.6352	0.8381	**0.9589**	0.5216	**0.9579**	0.5457	**0.5757**	0.5339	0.5457
	Max Diam	0.6653	0.8590	**0.9349**	0.4890	**0.9358**	0.5595	0.4187	0.4168	**0.5595**
Bitcoin-OTC	Same Col	0.7643	**0.8454**	0.8369	0.5911	0.2748	**0.8357**	0.3099	0.4001	**0.8072**
	Max Diam	0.7279	0.7670	**0.8035**	0.7306	0.3990	**0.8422**	0.4308	0.5142	**0.8285**
UCI-Forum	Same Col	**0.6781**	0.6167	0.5995	0.4660	0.3639	**0.4865**	0.3330	**0.4078**	0.4071
	Max Diam	**0.5630**	0.5137	0.5000	0.3944	0.1995	**0.4114**	0.1654	**0.3414**	**0.3414**
EU-Core	Same Col	**0.6361**	0.6293	0.5583	0.0906	0.1329	**0.5339**	0.5314	**0.6345**	0.5339
	Max Diam	**0.6052**	0.5978	0.4989	0.1360	0.1972	**0.4472**	**0.6179**	0.5783	0.4453

Table 5: Robustness to decreasing amounts of training data. The results show the WL score computed between the real and generated last graph.

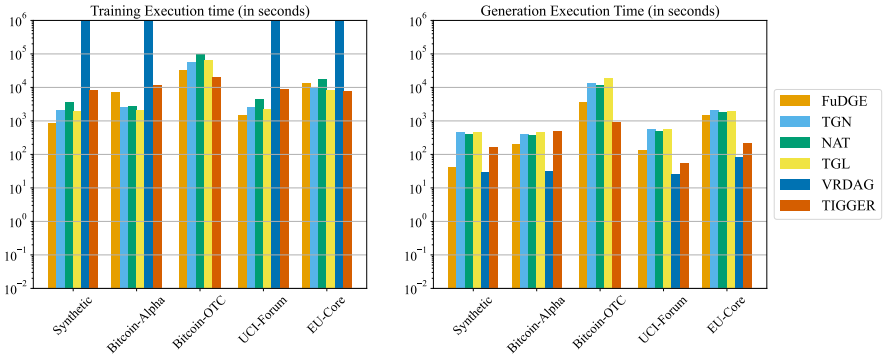

Fig. 6: Training and generation execution times in seconds (log scale)

There are several pointers for further research that characterize the model. A direction is to explore alternative modules, such as those based on message passing, to enhance the efficiency of the training phase, especially in the GNN component. Additionally, extending the model to incorporate feature evolution on both nodes and edges could provide richer representations for capturing dynamic graph properties. Another promising direction is to address the prediction of time to event, which has significant implications in domains like medicine and cybersecurity. This would involve handling continuous-time evolution rather than discrete-time steps.

Acknowledgements. This work was partially supported by Project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU, by MUR on D.M. 352/2022, PNRR Ricerca, CUP H23C22000550005, and by the Moneying-Plus project (CUP J29I24001790005) selected within the framework of the PR FESR – FSE Calabria 2021/2027 and implemented with the support of the Italian State and the Calabria Region – Action 1.1.1: Support for research, development, and innovation projects, including those carried out in collaboration with research organizations, within the priority areas and trajectories of the S3 Strategy.

Declarations

- Funding. Partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU, by MUR on D.M. 352/2022, PNRR Ricerca, CUP H23C22000550005, and by the Moneying-Plus project (CUP J29I24001790005) selected within the framework of the PR FESR – FSE Calabria 2021/2027 and implemented with the support of the Italian State and the Calabria Region – Action 1.1.1: Support for research, development, and innovation projects, including those carried out in collaboration with research organizations, within the priority areas and trajectories of the S3 Strategy.

- Conflict of interest/Competing interests. No conflict of interest.
- Ethics approval and consent to participate. Not applicable.
- Consent for publication.
- Data availability. The data are publicly available at their respective links.
- Code availability. Available at: https://anonymous.4open.science/r/fudge-3C25
- Author contribution. Angelica Liguori and Simone Mungari jointly led the conceptualization, implementation, validation of the results and the writing. The remaining authors contributed equally through discussions of the core ideas, interpretation of the results, and refinement of the overall framework and its presentation.

References

Anderson B, Quist D, Neil J, et al (2011) Graph-based malware detection using dynamic analysis. Journal in Computer Virology 7(4):247–258. https://doi.org/10.1007/s11416-011-0152-x

Avllazagaj E, Zhu Z, Bilge L, et al (2021) When malware changed its mind: an empirical study of variable program behaviors in the real world. In: 30th USENIX Security Symposium, pp 3487–3504

Babai L, Erdos P, Selkow SM (1980) Random graph isomorphism. SIAM Journal on Computing 9(3):628–635. https://doi.org/10.1137/0209047

Barabási AL, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512. https://doi.org/10.1126/science.286.5439.509

Battaglia PW, Pascanu R, Lai M, et al (2016) Interaction networks for learning about objects, relations and physics. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, pp 4509–4517, https://doi.org/10.5555/3157382.3157601

Cai Z, Wang X, Zhang M (2024) Unifying generation and prediction on graphs with latent graph diffusion. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems, pp 61963–61999, https://doi.org/10.48550/arXiv.2402.02518

Chen L, Wang L, Zeng C, et al (2022) DHGEEP: a dynamic heterogeneous graph-embedding method for evolutionary prediction. Mathematics 10(22). https://doi.org/10.3390/math10224193

Chen ZG, Kang HS, Yin SN, et al (2017) Automatic ransomware detection and analysis based on dynamic api calls flow graph. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, pp 196–201, https://doi.org/10.1145/3129676.3129704

Cho K, van Merrienboer B, Bahdanau D, et al (2014) On the properties of neural machine translation: encoder-decoder approaches. In: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, pp 103–111, https://doi.org/10.3115/v1/W14-4012

Cui H, Lu Z, Li P, et al (2022) On positional and structural node features for graph neural networks on non-attributed graphs. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pp 3898–3902, https://doi.org/10.1145/3511808.3557661

Dai H, Nazi A, Li Y, et al (2020) Scalable deep generative modeling for sparse graphs. In: Proceedings of the 37th International Conference on Machine Learning, pp 2302–2312, https://doi.org/10.5555/3524938.3525153

Du C, Li X, Li Z (2024) Semantic-enhanced reasoning question answering over temporal knowledge graphs. Journal of Intelligent Information Systems 62(3):859–881. https://doi.org/10.1007/s10844-024-00840-5

Du Y, Guo X, Cao H, et al (2022) Disentangled spatiotemporal graph generative models. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, pp 6541–6549, https://doi.org/10.48550/arXiv.2203.00411

Duan G, Lv H, Wang H, et al (2023) Application of a dynamic line graph neural network for intrusion detection with semisupervised learning. IEEE Transactions on Information Forensics and Security 18:699–714. https://doi.org/10.1109/TIFS.2022.3228493

Erdős P, Rényi A, et al (1960) On the evolution of random graphs. Publication of the Mathematical Institute of the Hungarian Academy of Sciences 5:17–61

Evdaimon I, Nikolentzos G, Chatzianastasis M, et al (2024) Neural graph generator: feature-conditioned graph generation using latent diffusion models. arXiv 2403.01535

Fan W, Jin R, Lu P, et al (2022) Towards event prediction in temporal graphs. Proceedings of the VLDB Endowment 15(9):1861–1874. https://doi.org/10.14778/3538598.3538608

Gainza P, Sverrisson F, Monti F, et al (2020) Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. Nature Methods 17:184–192. https://doi.org/10.1038/s41592-019-0666-6

Granovetter MS (1973) The strength of weak ties. American Journal of Sociology 78:1360–1380. URL www.jstor.org/stable/2776392

Graves A (2012) Supervised sequence labelling with recurrent neural networks. In: Studies in Computational Intelligence, https://doi.org/10.1007/978-3-642-24797-2

Grover A, Zweig A, Ermon S (2019) Graphite: iterative generative modeling of graphs. In: Proceedings of the 36th International Conference on Machine Learning, https://doi.org/10.48550/arXiv.1803.10459

Gupta S, Bedathur S (2022) A survey on temporal graph representation learning and generative modeling. arXiv 2208.12126

Gupta S, Manchanda S, Bedathur S, et al (2022) TIGGER: scalable generative modelling for temporal interaction graphs. Proceedings of the AAAI Conference on Artificial Intelligence 36(6):6819–6828. https://doi.org/10.1609/aaai.v36i6.20638

Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, p 1025–1035, https://doi.org/10.5555/3294771.3294869

Han L, Li L, Wen F, et al (2019) Graph-guided multi-task sparse learning model: a method for identifying antigenic variants of influenza a(h3n2) virus. Bioinformatics 35:77–87. https://doi.org/10.1093/bioinformatics/bty457

Held P, Moewes C, Braune C, et al (2013) Advanced analysis of dynamic graphs in social and neural networks. In: Towards Advanced Data Analysis by Combining Soft Computing and Statistics. Studies in Fuzziness and Soft Computing, pp 205–222, https://doi.org/10.1007/978-3-642-30278-7_17

Holme P, Kim BJ (2002) Growing scale-free networks with tunable clustering. Physical Review E https://doi.org/10.1103/PhysRevE.65.026107

Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: 3rd International Conference for Learning Representations, https://doi.org/10.48550/arXiv.1412.6980

Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations, https://doi.org/10.48550/arXiv.1609.02907

Kong L, Cui J, Sun H, et al (2023) Autoregressive diffusion model for graph generation. In: Proceedings of the 40th International Conference on Machine Learning, pp

17391–17408, https://doi.org/10.5555/3618408.3619125

Kumar S, Zhang X, Leskovec J (2019) 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, https://doi.org/10.1145/3292500.3330895

Li D, Kosugi S, Zhang Y, et al (2025a) Revisiting dynamic graph clustering via matrix factorization. In: Proceedings of the ACM on Web Conference 2025, pp 1342–1352, https://doi.org/10.1145/3696410.3714646

Li D, Tan S, Zhang Y, et al (2025b) DyG-Mamba: continuous state space modeling on dynamic graphs. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems, https://doi.org/10.48550/arXiv.2408.06966

Li F, Wang X, Cheng D, et al (2025c) Efficient dynamic attributed graph generation. In: IEEE 41st International Conference on Data Engineering, pp 1415–1428, https://doi.org/10.1109/ICDE65448.2025.00110

Li X, Qian Y, Zeng J, et al (2025d) Dynamic graph convolutional recurrent network with temporal self-attention for accurate traffic flow prediction. IET Intelligent Transport Systems 19(1):e70118. https://doi.org/10.1049/itr2.70118

Lim J, Hwang SY, Moon S, et al (2020) Scaffold-based molecular design with a graph generative model. Chemical Science 11:1153–1164. https://doi.org/10.1039/C9SC04503A

Liu Z, Zhou D, Zhu Y, et al (2020) Towards fine-grained temporal network representation via time-reinforced random walk. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 4973–4980, https://doi.org/10.1609/aaai.v34i04.5936

Longa A, Lachi V, Santin G, et al (2023) Graph neural networks for temporal graphs: state of the art, open challenges, and opportunities. Transactions on Machine Learning Research https://doi.org/10.48550/arXiv.2302.01018

Luo Y, Li P (2022) Neighborhood-aware scalable temporal network representation learning. In: Proceedings of the First Learning on Graphs Conference, https://doi.org/10.48550/arXiv.2209.01084

Ma R, Han X, Yan L, et al (2023) Modeling and querying temporal RDF knowledge graphs with relational databases. Journal of Intelligent Information Systems 61(2):569–609. https://doi.org/10.1007/s10844-023-00780-6

Paszke A, Gross S, Massa F, et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp 8026–8037, https://doi.org/10.5555/3454287.3455008

Qi S, Wang W, Jia B, et al (2018) Learning human-object interactions by graph parsing neural networks. In: Computer Vision – ECCV 2018: 15th European Conference, pp 407–423, https://doi.org/10.1007/978-3-030-01240-3_25

Ranshous S, Shen S, Koutra D, et al (2015) Anomaly detection in dynamic networks: a survey. WIREs Computational Statistics 7:223–247. https://doi.org/10.1002/wics.1347

Rossi E, Chamberlain B, Frasca F, et al (2020) Temporal graph networks for deep learning on dynamic graphs. In: International Conference on Machine Learning - Workshop Graph Representation Learning and Beyond, https://doi.org/10.48550/arXiv.2006.10637

Sankar A, Wu Y, Gou L, et al (2020) DySAT: deep neural representation learning on dynamic graphs via self-attention networks. In: Proceedings of the 13th International Conference on Web Search and Data Mining, p 519–527, https://doi.org/10.1145/3336191.3371845

Shervashidze N, Schweitzer P, Van Leeuwen EJ, et al (2011) Weisfeiler-lehman graph kernels. Journal of Machine Learning Research 12:2539–2561. https://doi.org/10.5555/1953048.2078187

Tam K, Feizollah A, Anuar NB, et al (2017) The evolution of android malware and android analysis techniques. ACM Computing Surveys 49(4). https://doi.org/10.1145/3017427

Trivedi R, Farajtabar M, Biswal P, et al (2019) DyRep: learning representations over dynamic graphs. In: International Conference on Learning Representations, https://doi.org/10.48550/arXiv.1803.04051

Wang X, Lyu D, Li M, et al (2021a) APAN: asynchronous propagation attention network for real-time temporal graph embedding. In: Proceedings of the 2021 International Conference on Management of Data, pp 2628–2638, https://doi.org/10.1145/3448016.3457564

Wang Y, Chang Y, Liu Y, et al (2021b) Inductive representation learning in temporal networks via causal anonymous walks. In: International Conference on Learning Representations, https://doi.org/10.48550/arXiv.2101.05974

Weisfeiler B, Leman A (1968) The reduction of a graph to canonical form and the algebra which appears therein. NTI, Series 2(9):12–16

Wu C, Nikolentzos G, Vazirgiannis M (2020) EvoNet: a neural network for predicting the evolution of dynamic graphs. In: Artificial Neural Networks and Machine Learning – ICANN 2020, pp 594–606, https://doi.org/10.1007/978-3-030-61609-0_47

Wu Y, Yang J, Chen X, et al (2024) Long-term airport network performance forecasting with linear diffusion graph networks. IEEE Transactions on Intelligent Transportation Systems 25:18264–18278. https://doi.org/10.1109/TITS.2024.3420423

Xu D, Ruan C, Körpeoglu E, et al (2020) Inductive representation learning on temporal graphs. In: International Conference on Learning Representations, https://doi.org/10.48550/arXiv.2002.07962

Xu K, Hu W, Leskovec J, et al (2019) How powerful are graph neural networks? In: International Conference on Learning Representations, https://doi.org/10.48550/arXiv.1810.00826

Yan Q, Liang Z, Song Y, et al (2024) SwinGNN: rethinking permutation invariance in diffusion models for graph generation. Transactions on Machine Learning Research https://doi.org/10.48550/arXiv.2307.01646

You J, Ying R, Ren X, et al (2018) GraphRNN: generating realistic graphs with deep auto-regressive models. In: Proceedings of the 35th International Conference on Machine Learning, pp 5694–5703, https://doi.org/10.48550/arXiv.1802.08773

Zeng Q, Langereis MA, Van Vliet AL, et al (2008) Structure of coronavirus hemagglutinin-esterase offers insight into corona and influenza virus evolution. Proceedings of the National Academy of Sciences 105(26):9065–9069. https://doi.org/10.1073/pnas.0800502105

Zeno G, La Fond T, Neville J (2021) DYMOND: dynamic motif-nodes network generative model. In: Proceedings of the Web Conference 2021, pp 718–729, https://doi.org/10.1145/3442381.3450102

Zhang H, Qi F, Zhang Y, et al (2026) A traffic flow forecasting model based on dynamic graph learning and temporally adaptive attention. Safety Science 195. https://doi.org/10.1016/j.ssci.2025.107063

Zhang Y, Xiong Y, Li D, et al (2021) CoPE: modeling continuous propagation and evolution on interaction graph. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, p 2627–2636, https://doi.org/10.1145/3459637.3482419

Zhao L, Ding X, Akoglu L (2024) Pard: permutation-invariant autoregressive diffusion for graph generation. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems, https://doi.org/10.48550/arXiv.2402.03687

Zheng L, Li Z, Li J, et al (2019) AddGraph: anomaly detection in dynamic graph using attention-based temporal GCN. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp 4419–4425, https://doi.org/10.24963/ijcai.2019/614

Zhou D, Zheng L, Han J, et al (2020) 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, https://doi.org/10.1145/3394486.3403082

Zhou H, Zheng D, Nisa I, et al (2022a) TGL: a general framework for temporal GNN training on billion-scale graphs. Proceedings of the VLDB Endowment 15(8):1572–1580. https://doi.org/10.14778/3529337.3529342

Zhou R, Zhang Q, Zhang P, et al (2021) Anomaly detection in dynamic attributed networks. Neural Computing and Applications 33:2125–2136. https://doi.org/10.1007/s00521-020-05091-3

Zhou Y, Luo S, Pan L, et al (2022b) Continuous temporal network embedding by modeling neighborhood propagation process. Knowledge-Based Systems 239. https://doi.org/10.1016/j.knosys.2021.107998

Zitnik M, Agrawal M, Leskovec J (2018) Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics 34:i457–i466. https://doi.org/10.1093/bioinformatics/bty294