Replacing Recurrence with Self-Attention for Weight Evolution in Dynamic Graph Networks

Victoria Yang, Veronica Wang, Kaci Morris
Stanford University
{vicyang,vwang2,kacim}@stanford.edu

1 Introduction

Many real-world systems such as financial transaction ledgers and online communication networks are naturally modeled as dynamic graphs: a sequence of graph snapshots $G_1, \ldots, G_T$ whose nodes, edges, and attributes change over time. Reasoning over such data requires capturing both the relational structure within each snapshot and the temporal dynamics across snapshots.

Two broad families of models address this. The first evolves node embeddings over time, typically by feeding the embeddings produced by a graph neural network (GNN) into a recurrent or attention module. The second, exemplified by EvolveGCN (Pareja et al., 2020), instead evolves the model parameters. It keeps a GCN (Kipf and Welling, 2017) and uses an RNN to roll its convolutional weights forward in time. The latter is appealing because the temporal dynamics are carried entirely by the parameters, so the method does not require a fixed node set and handles nodes that appear or disappear.

Input and output. The input to our algorithm is a sequence of graph snapshots, each described by an adjacency matrix A_t and a node-feature matrix X_t. The output depends on the task: (i) a per-node class label (predicting whether a transaction is illicit on the Elliptic dataset); (ii) a per-edge class label (predicting an edge's rating on Bitcoin-OTC); or (iii) a score for whether edge (u, v) exists at time $t+1$ (link prediction on SBM). In all cases the predictor is a GNN whose convolutional weights are evolved over time by a learned temporal module.

Motivation and research question. EvolveGCN's recurrence summarizes the entire weight history into a single fixed-size hidden state You et al. (2022) which may discard explicit historical information. Transformers (Vaswani et al., 2017) remove exactly this bottleneck by allowing direct attention to any position in a sequence. This motivates our central question: does replacing EvolveGCN's recurrent weight evolution with self-attention over the weight history improve dynamic-graph learning? We implement such a variant, reproduce the original baselines to validate our pipeline, and evaluate the variant in controlled comparisons across three tasks.

2 Related Work

Static graph neural networks. GCN (Kipf and Welling, 2017) popularized spectral-motivated message passing, and Graph Attention Networks introduced attention over neighbors. These operate on a single static graph and are the building block our temporal models repeatedly instantiate, but because they run on a single static graph, their use is limited to unchanging networks, making them insufficient for temporal modeling.

Dynamic GNNs that evolve embeddings. DySAT (Sankar et al., 2020) stacks structural self-attention (over neighbors) with temporal self-attention (over a node's embeddings across snapshots). TGAT (Xu et al., 2020) attends over time-stamped neighbors with a continuous-time encoding. Their strength is expressive temporal modeling at the embedding level while their weakness is that they require a node to persist across the window and scale with the number of nodes attended.

Dynamic GNNs that evolve the model. EvolveGCN (Pareja et al., 2020) is the representative method: a GRU evolves the GCN weight matrices, in either an "-O" variant (weights only, no node embeddings) or an "-H" variant (the recurrent cell additionally ingests an embedding summary). ROLAND (You et al., 2022) recurrently updates hierarchical node embeddings and explicitly critiques EvolveGCN for losing historical information in the weight recurrence. None of these methods apply a Transformer to the sequence of evolving weight matrices–existing attention is applied to embeddings or neighbors.

Critiques of dynamic-graph evaluation. Poursafaei et al. (2022) show that standard dynamic link-prediction evaluation relies on easy (random) negative edges and that many edges reoccur over time, so a pure memorization baseline rivals models that actually learn. High absolute MAP/MRR therefore overstates temporal-modeling skill. On Elliptic, Weber et al. (2019) find that Random Forest outperforms graph models, because the dataset's hand-engineered features make graph structure largely redundant. We use both observations to interpret our results.

3 Dataset and Features

We use three datasets from the EvolveGCN benchmark, obtained from the original release (Pareja et al., 2020) and SNAP (Kumar et al., 2016).

3.1 Elliptic (node classification)

A Bitcoin transaction graph of $\sim$203k nodes and $\sim$234k edges spanning 49 weekly time steps. Each node is labeled illicit ($\sim$2%), licit ($\sim$21%), or unknown ($\sim$77%). Preprocessed by removing unknown nodes and making edges bidirectional–only labeled nodes contribute to the loss. Each node carries 166 unknown precomputed features. We adopt the paper's temporal split of 31/5/13 train/validation/test time steps and use class-weighted loss due to the class imbalance. A documented regime shift occurs at time step 43 following a market shutdown, making later performance decrease.

3.2 SBM (link prediction)

A synthetic stochastic block model containing 1,000 nodes over 50 time steps creating 4,870,863 total directed edges. The data is designed to display temporal dynamics over attributes as node features are one-hot degree encodings. It was preprocessed by binarizing edge weights. We follow the original 70/10/20 temporal split, corresponding to 35 training, 5 validation, and 10 test snapshots. We use a history window of 5 steps. Link prediction is trained with negative sampling (50$\times$ negatives during training, 100$\times$ at test). The abundance of negatives makes the task imbalanced toward non-edges.

3.3 Bitcoin-OTC (edge classification)

Bitcoin-OTC is a signed, weighted trust network in which users assign ratings from -10 to $+10$ to one another. It contains approximately 5.9k nodes and 35.6k edges. We formulate edge classification as predicting the label of an interaction using one-hot degree encodings as node features and a 10-step history window. Following the original temporal splits, we use 85/14/29 timesteps for train/validation/test on OTC after reserving the history window. Because the positive class is a small minority, model selection is based on minority-class F1, while performance is reported using micro-averaged F1.

4 Methods

4.1 Graph Convolution

A graph convolutional network updates each node representation by aggregating information from its neighbors. The normalized adjacency matrix $\hat{A}_t$ determines how messages are passed between connected nodes, while the learnable weight matrix $W_t^{(l)}$ transforms the aggregated features. Within snapshot t, layer l produces node embeddings by one round of normalized message passing,

$$H_t^{(l+1)} = \sigma\big(\hat{A}_t\, H_t^{(l)}\, W_t^{(l)}\big), \qquad \hat{A}_t = \tilde{D}_t^{-1/2}\tilde{A}_t\tilde{D}_t^{-1/2}, \quad \tilde{A}_t = A_t + I, \tag{1}$$

where $H_t^{(0)} = X_t$, $\tilde{D}_t$ is the degree matrix of $\tilde{A}_t$, and σ is a nonlinearity (we use RReLU). We stack two such layers throughout.

4.2 EvolveGCN and recurrent baselines

Rather than learning a single time-invariant weight $W^{(l)}$, EvolveGCN evolves the weight matrix across snapshots. The EvolveGCN-O variant is a lighter model where the GRU only sees the previous weight matrix. It does not receive node embeddings as input. The weights are the hidden state:

$$W_t^{(l)} = \mathrm{matGRU}\big(W_{t-1}^{(l)}\big), \tag{2}$$

where the GRU computes update and reset gates from $W_{t-1}^{(l)}$ and produces $W_t^{(l)}$. The EvolveGCN-H variant additionally feeds a summary of the current node embeddings $H_t^{(l)}$ into the cell. It therefore carries more information but more parameters. Both variants share the structural limitation that the entire weight history is compressed into the single matrix $W_{t-1}^{(l)}$.

4.3 EvolveGCN-T (proposed)

We replace the recurrent cell with a Transformer encoder (Vaswani et al., 2017) that operates on the explicit history of weight states. Unlike most dynamic graph Transformers, which attend over node embeddings, EvolveGCN-T applies attention directly to the sequence of evolving GCN weight matrices. We instantiate EvolveGCN-T as the Transformer analog of EvolveGCN-O so that the comparison isolates the temporal module. Let $\mathcal{S}_t^{(l)} = [\, W_{t-k}^{(l)}, \ldots, W_{t-1}^{(l)} \,]$ be the most recent k weight matrices, where each $W \in \mathbb{R}^{r \times c}$ is viewed as c tokens of dimension r stacked along a learned positional encoding P over the time axis. The cell computes

$$Z = \mathrm{TransformerEncoder}\big(\mathcal{S}_t^{(l)} + P\big), \qquad W_t^{(l)} = Z_{[-1]} + W_{t-1}^{(l)}, \tag{3}$$

where the encoder applies multi-head self-attention which allows the model to directly compare every weight matrix in the history window and assign higher importance to past states that are most relevant for predicting the next weight matrix. Using multiple attention heads allows the model to attend to different aspects of the weight-history sequence simultaneously, improving its ability to capture complex dependencies.

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\Big(\tfrac{QK^\top}{\sqrt{d}}\Big)V, \tag{4}$$

followed by a position-wise feed-forward block with residual connections and layer normalization. The next weight matrix is obtained from the Transformer's output, with a residual connection from the previous weight matrix to improve training stability. Since the attention window contains only past weight matrices, the model cannot access future information and therefore avoids temporal leakage.

4.4 Libraries

Experiments were implemented in PyTorch Paszke et al. (2019) using NumPy Harris et al. (2020), SciPy Virtanen et al. (2020), and Pandas Team (2020). Evaluation metrics were computed with Scikit-learn Pedregosa et al. (2011), visualizations were generated with Matplotlib Hunter (2007), and training was tracked with Weights & Biases Biewald (2020).

5 Experiments / Results / Discussion

5.1 Metrics

For node and edge classification we report precision $P = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$, recall $R = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, and $F1 = 2PR/(P + R)$. For Elliptic, we report the F1 score of the minority illicit class because detecting illicit transactions is the primary objective. For Bitcoin, we report micro-F1, which aggregates true positives, false positives, and false negatives across all classes before computing a single overall F1 score. For SBM link prediction we report Mean Average Precision, $\mathrm{MAP} = \frac{1}{|Q|}\sum_q \mathrm{AP}_q$ with $\mathrm{AP} = \sum_k (R_k - R_{k-1})P_k$ over the ranked candidate edges.

5.2 Hyperparameter selection.

For node and edge classification, predictions are produced by task-specific MLP heads. Classification heads are trained with class-weighted cross-entropy $\mathcal{L} = -\sum_i w_{y_i} \log p_{i,y_i}$ using weights $[0.15, 0.85]$ on the imbalanced tasks. Optimization uses Adam with learning rate 5×10^{-3}. We early-stop on the validation metric and report test performance at the best-validation epoch.

EvolveGCN-O and EvolveGCN-T were configured identically wherever possible. Both use two GCN layers, a learning rate of 510^{-3}, the same history window, and the same class weights. As a result, the two models differ only in their temporal implementation (GRU versus Transformer), making the comparison controlled. For EvolveGCN-T, we use 4 attention heads and a single Transformer encoder layer. These were the smallest settings that trained reliably and were chosen to keep the model size comparable to EvolveGCN-O. In contrast, EvolveGCN-H uses the authors' separately tuned hyperparameters and additionally incorporates node-embedding information during weight evolution. We therefore treat EvolveGCN-H as a reference baseline rather than something directly comparable. All hyperparameter selection, early stopping, and epoch selection were performed using the validation set only.

5.3 Main Results

Our baselines reproduce the published numbers: Elliptic EvolveGCN-O reaches illicit-F1 0.578 (paper $\approx$ 0.51); SBM EvolveGCN-O reaches MAP 0.194 (paper 0.199); Bitcoin-OTC EvolveGCN-H reaches micro-F1 0.892 (paper-level). These reproductions establish the correctness of our implementation before comparing against the proposed model.

Table 1: SBM link prediction, MAP (mean$\pm$std over 3 seeds). EvolveGCN-O and -T are architecture-matched; -H uses its own tuned config. The Transformer variant underperforms recurrence in this short-history regime.

	EvolveGCN-O	EvolveGCN-H	EvolveGCN-T
MAP (ours)	$\mathbf{0.194 \pm 0.001}$	0.176 ± 0.011	0.130 ± 0.029
Paper (Pareja et al., 2020)	0.199	0.195	—

Table 2: Bitcoin edge classification, micro-averaged F1 (best run). On Bitcoin-OTC the Transformer variant reaches a higher ceiling than the matched recurrent EvolveGCN-O, though both trail the -H reference.

Dataset	EvolveGCN-O	EvolveGCN-H	EvolveGCN-T (ours)
Bitcoin-OTC	0.699	**0.892**	0.783

Tables 1 and 2 summarize the primary comparison. On SBM link prediction, EvolveGCN-T underperforms the recurrent EvolveGCN-O baseline (MAP 0.130 vs. 0.194; Table 1), indicating that self-attention does not improve performance in this short-history setting. On Bitcoin-OTC edge classification, however, EvolveGCN-T achieves a higher micro-F1 score than EvolveGCN-O (0.783 vs. 0.699; Table 2). The embedding-augmented EvolveGCN-H remains the strongest model overall, although it is not directly comparable because it uses additional node-embedding information and separately tuned hyperparameters.

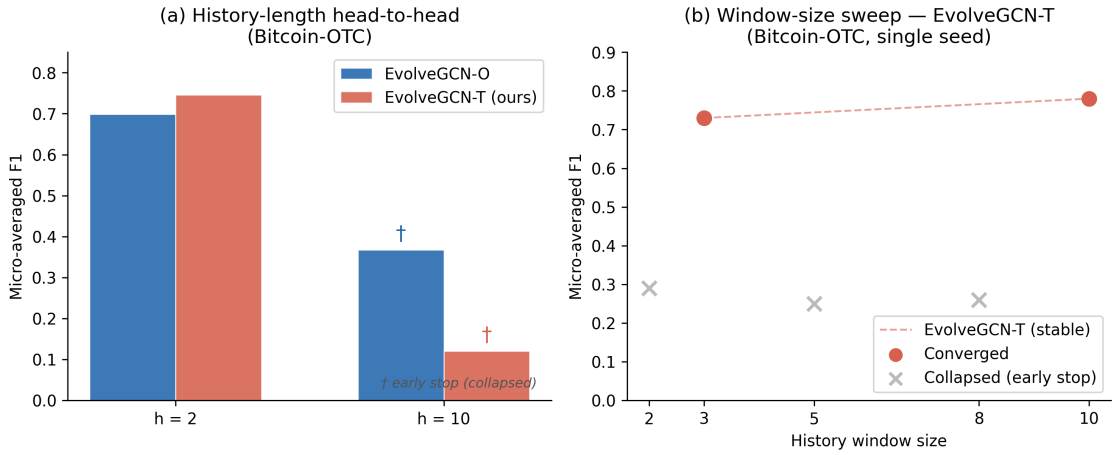
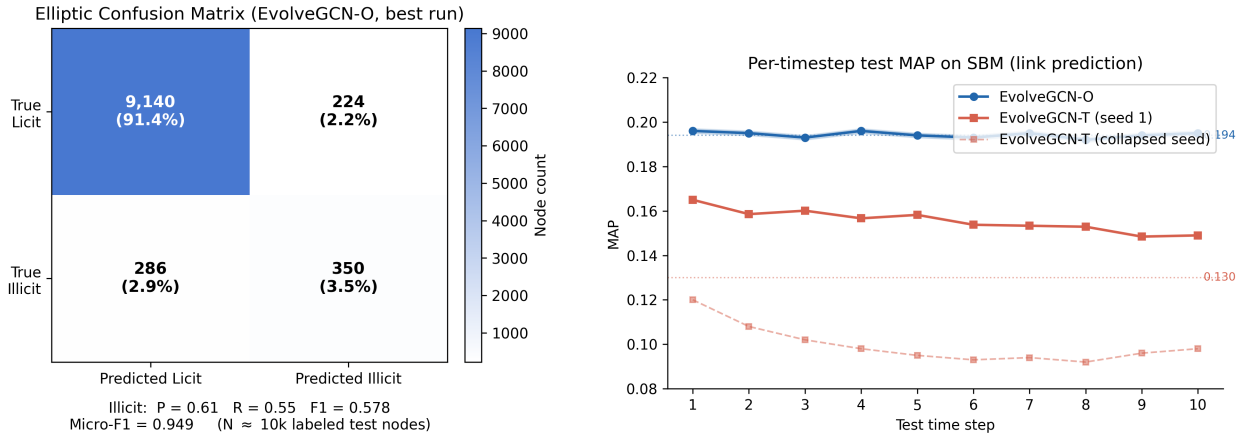


Figure 1: History-length ablation on Bitcoin-OTC No consistent improvement is observed with longer histories.

Figure 1 shows the controlled history-length study. On the left we see a comparison between EvolveGCN-O and EvolveGCN-T at history lengths $h = 2$ and $h = 10$. On the right, we see EvolveGCN-T performance across window sizes. At h=2, EvolveGCN-T outperforms EvolveGCN-O, while both models collapse at h=10. No consistent improvement is observed with longer histories. We hypothesized that self-attention would be most beneficial when longer histories were available. Instead, performance varied irregularly across window sizes, and low-scoring runs coincided with early training collapse. These results suggest that stochastic optimization instability had a larger influence on performance than context length.



((a)) Elliptic confusion matrix.

((b)) Per-timestep SBM MAP.

Figure 2: Qualitative error analysis.

Figure 2a shows the Elliptic confusion matrix. The model identifies illicit transactions with reasonable precision (0.61) but misses many positives (recall 0.55), yielding an illicit-class F1 of 0.578. Performance also degrades on later test snapshots following the market-shutdown regime shift. On SBM, Figure 2b shows that EvolveGCN-T consistently achieves lower MAP and exhibits stronger performance decay across the test horizon, suggesting that the learned weight evolution generalizes poorly outside the training regime. On Bitcoin-OTC, runs either converged successfully or collapsed within the first few epochs, indicating that training instability is a major limitation.

6 Conclusion / Future Work

We introduced EvolveGCN-T, a Transformer that evolves GCN weights by self-attending over their explicit history, and evaluated it as a potential improvement over the EvolveGCN's recurrence. The result is task-dependent: EvolveGCN-O is stronger and more stable on SBM link prediction, but EvolveGCN-T attains a higher ceiling on Bitcoin-OTC edge classification and wins a controlled head-to-head with matched context. We did not find the hypothesized context-length advantage. In fact, frequent stochastic training collapse that affected both models was the principal obstacle, and the embedding-augmented EvolveGCN-H remains the strongest model overall. The clearest takeaway is that, for this method family, optimization stability is more decisive than the choice of temporal aggregator. With more time and compute we would: (i) attack instability directly with gradient clipping, a Transformer-specific learning-rate warm-up, and careful initialization, which we believe is the highest-leverage next step, (ii) run multi-seed confirmation of every cell, since our single-seed Bitcoin/Elliptic comparisons are a key limitation, (iii) build a Transformer analog of EvolveGCN-H to test whether the embedding pathway and attention work well together, and (iv) re-evaluate under the harder negative-sampling protocol of Poursafaei et al. (2022) and the leakage-free live-update setting of You et al. (2022).

References

Lukas Biewald. 2020. Experiment tracking with weights and biases. Software available from `https://www.wandb.com`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585:357–362.

John D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *IEEE International Conference on Data Mining (ICDM)*, pages 221–230.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. 2020. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. 2022. Towards better evaluation for dynamic link prediction. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 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 (WSDM)*, pages 519–527.

The Pandas Development Team. 2020. pandas-dev/pandas: Pandas.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272.

Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. In *KDD Workshop on Anomaly Detection in Finance*.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*.

Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: Graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2358–2366.