engineering proceedings

MDPI

*Proceeding Paper*

# Experimental Evaluation of DyGED for Event Detection in Dynamic Graphs †

**Konstantinos Christopoulos *, Nikolaos Katseris and Christos Konstantopoulos**

Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece; up1070921@upnet.gr (N.K.); chris.konstanto@upatras.gr (C.K.)

* Correspondence: kchristopou@ac.upatras.gr

† Presented at the 2nd International Conference on Engineering, Technology, and Applied Science Innovations (ICETASI 2026), Zakynthos, Greece, 24–26 September 2026.

**Abstract**

Event detection in dynamic graphs focuses on recognizing time instants at which the structure or attributes of an evolving graph differ meaningfully from their ordinary patterns. Dynamic Graph Event Detection (DyGED) is a graph-level deep learning framework that integrates graph convolution, graph pooling, recurrent temporal processing, and temporal attention to capture the relationships between graph-level dynamics and annotated events. This study presents a comprehensive evaluation of DyGED and three associated variants, DyGED-CT, DyGED-NL, and DyGED-NA. The assessment follows the supervised DyGED formulation introduced with DyGED, while expanding the empirical analysis through reporting runtime performance, evaluating threshold-based F1 scores, and examining additional synthetic dynamic graphs generated with RDyn. Experiments are conducted on NYC Cab and Twitter Weather datasets, using three RDyn configurations that included controlled community merge and split events. The findings indicate that the complete DyGED architecture is a reliable and competitive model, particularly for Twitter Weather and several RDyn scenarios. However, the threshold-based evaluation reveals that a high Area Under the ROC Curve (AUC) score does not necessarily translate into strong precision–recall performance when positive events are infrequent. Additionally, the synthetic experiments suggest that DyGED-style models are more effective at identifying community-level structural changes when the event pattern is clearly defined and controlled experimentally.

**Keywords:** dynamic graphs; event detection; anomaly detection; graph neural networks; DyGED; RDyn; synthetic dynamic graphs

## 1. Introduction

Dynamic graphs describe systems in which nodes, edges, or attributes vary across time. Typical examples include transportation networks, communication systems, social media activity, and communities with changing membership. Within these environments, an event can be expressed as a temporary departure from the graph's usual behavior. For instance, a sports match may alter taxi demand patterns, severe weather may change word co-occurrences in weather-related tweets, and a synthetic community event may produce an abrupt merge or split in a generated network.

Detecting events in dynamic graphs is difficult because the label of interest is usually assigned to a complete graph snapshot or a short time interval, rather than to one specific node or edge. Consequently, the model has to condense every graph snapshot into a graph-level embedding and then learn the temporal evolution of these embeddings. This makes

Check for updates

Academic Editor: Konstantinos Poirazidis

Published: 23 November 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

the problem distinct from standard graph-learning tasks such as node classification or link prediction. Global pooling, recurrent temporal modeling, and temporal attention are therefore valuable components, since the relevant signal may be spread across multiple regions of the graph.

Dynamic Graph Event Detection (DyGED) was introduced as an architecture for event detection in dynamic graphs with emphasis on learning graph macro-dynamics [1]. The model combines graph convolution, pooling, recurrent sequence modeling, and attention to associate changes in graph structure with event labels. The present work continues in this direction by examining DyGED together with three related variants: DyGED-CT, DyGED-NL and DyGED-NA. The experiments were conducted on the real-world NYC Cab and Twitter Weather datasets, and were extended with RDyn-generated dynamic graphs where the target events are community merges and splits [2].

This study offers a concise experimental examination of the DyGED model family, focusing on practical evaluation metrics, threshold behavior, and additional controlled synthetic graph scenarios.

The main contributions of the study are as follows:

- It provides a unified empirical comparison of DyGED-CT, DyGED-NL, DyGED-NA and the complete DyGED model on real DyGED datasets.
- It presents both Area Under the ROC Curve (AUC) and runtime results for the NYC Cab and Twitter Weather datasets, highlighting the balance between detection quality and computational demand.
- It includes threshold-based F1, precision, and recall measurements, demonstrating that event-level classification can remain challenging even under high AUC values.
- It broadens the experimental setting through RDyn synthetic dynamic graphs, where community merge and split events serve as controlled anomalous cases.
- It analyzes how model behavior differs between noisy real-world event streams and synthetic graphs with explicit structural event signals.

## 2. Related Work

### *2.1. Dynamic Graph Event Detection (DyGED)s*

Dynamic graphs provide an effective representation of systems that evolve. In supervised event detection, every time step is paired with a binary label that specifies whether an event is present. The objective is to train a model that uses recent graph snapshots and their features to estimate the probability of an event. This setting requires reasoning at the graph level, because the label describes the entire snapshot rather than a single node or edge.

Earlier approaches to anomaly and event detection commonly used statistical indicators, manually designed graph descriptors, or time-series techniques [3,4]. Such methods can be effective when abnormal behavior has a simple form, but they may be less successful when the event pattern is distributed throughout the graph. Graph neural networks address this limitation by learning structural representations directly from adjacency information and node attributes, while temporal models describe how these representations change across snapshots.

### *2.2. Graph Neural Networks*

Graph convolutional networks produce node embeddings by aggregating information from neighboring nodes. The GCN formulation of Kipf and Welling remains a common foundation for graph representation learning [5]. In graph-level tasks, node embeddings are usually combined through pooling operations to construct an embedding for the complete graph snapshot. This representation can then support classification, regression, or event-detection objectives.

https://doi.org/10.3390/engproc2026167025

Dynamic graph learning builds on this idea by incorporating a temporal dimension. Rather than treating each graph snapshot independently, the model processes a sequence of snapshots. Recurrent neural networks, including LSTMs, can capture dependencies across time [6]. Attention mechanisms can further support temporal modeling by assigning more weight to the time steps that contain the most informative evidence.

### *2.3. DyGED Model Family*

DyGED targets graph-level event detection in dynamic graphs. Its objective is to learn macro-dynamic patterns by combining graph-structured processing with temporal sequence modeling [1]. The model family examined in this study consists of four architectures. DyGED-CT uses graph convolution, pooling, temporal concatenation, and an MLP classifier. DyGED-NL incorporates temporal attention. DyGED-NA applies LSTM-based temporal modeling without using temporal attention. The full DyGED architecture combines graph convolution, pooling, LSTM temporal modeling, temporal attention, and an MLP classifier.

These variants are useful because they allow the contribution of individual architectural components to be studied separately. A simpler version may be faster or adequate for certain datasets, whereas the full architecture may be more effective when temporal dependencies are richer. This distinction is important in DyGED, since the most helpful component can vary with the graph domain and the type of event labels.

### *2.4. Synthetic Dynamic Graphs*

Synthetic dynamic graphs are valuable because they provide control over the event-generation process. RDyn creates dynamic networks with planted community events, such as community merges and splits [2]. Although these graphs cannot replace real-world datasets, they help test whether a model can identify structural changes when the signal is known and well-defined. They also allow experiments across different graph sizes and temporal lengths while preserving explicit event definitions.

## 3. Definition and Description

### *3.1. Problem Definition*

Let a dynamic graph be expressed as the sequence:

$$G = \{G_1, G_2, \ldots, G_T\}, \quad (1)$$

where every snapshot $G_t = (V_t, E_t)$ represents the graph at time step t. Here, $V_t$ denotes the set of vertices (nodes) present at time step t, while $E_t$ denotes the set of edges connecting these vertices at that time step. In the supervised event-detection setting, each time step has a label $y_t \in \{0, 1\}$, where $y_t = 1$ indicates an event and $y_t = 0$ indicates a normal snapshot. The goal is to learn a function that predicts the event probability at time t by using graph structure, node features, and a temporal window of earlier snapshots.

This is a graph-level event-detection task because the prediction refers to an entire snapshot. Thus, the model must convert node-level and edge-level information into a compact representation of the graph as a whole. The resulting representation should retain the structural and temporal cues that are informative for event occurrence.

### *3.2. Datasets*

The experimental analysis uses two real-world datasets and three synthetic datasets generated with RDyn. For NYC Cab models, nodes denote taxi zones, edges denote trips between zones, and event labels correspond to baseball games involving the Yankees or the Mets. Twitter Weather is constructed as a word co-occurrence graph from weather-related tweets, with event labels representing extreme weather phenomena. Here, nodes represent

English words related to weather. Edges represent the co-occurrence of two words in tweets, while edge weights indicate how many times the word pair co-occurs.

For the synthetic experiments, RDyn is used to generate dynamic graphs with evolving community structure. The generator outputs graph snapshots, community assignments, and event files. In this study, community merge and split events are considered anomalous events. Three configurations are examined: 30 nodes with 60 snapshots, 150 nodes with 100 snapshots, and 500 nodes with 60 snapshots. Table 1 presents the real dynamic graph datasets used in the first experimental phase.

**Table 1.** Real dynamic graph datasets used in the first experimental phase.

| Property | NYC Cab Dataset | Twitter Weather Dataset |
|---|---|---|
| Average nodes | 263 | 300 |
| Average edges | 3717 | 1142 |
| Static features | 6 | 300 |
| Dynamic features | 3 | 3 |
| Snapshots | 4464 | 2557 |
| Events | 162 | 287 |
| Event type | Baseball games | Extreme weather |

### *3.3. Model Variants*

Four architectures from the DyGED family are evaluated. DyGED-CT is the compact variant based on temporal concatenation. DyGED-NL adds a temporal attention mechanism. DyGED-NA uses LSTM temporal modeling but omits the attention layer. DyGED is the complete model, combining graph convolution, pooling, LSTM temporal modeling, temporal attention, and an MLP classifier. The inclusion of these variants makes it possible to observe how performance changes when recurrent modeling or attention is added or removed. Since all comparisons remain within the DyGED model family, the analysis emphasizes the behavior of the architectural components under a consistent experimental setup.

### *3.4. Evaluation Metrics*

AUC measures the ability of a model to rank event snapshots above normal snapshots. Because it is independent of a specific threshold, it is suitable for comparing the quality of the produced event scores. In practical event-detection systems, however, a binary alert decision is usually required. For this reason, F1, precision, and recall are also reported after threshold selection. Runtime is also included as a practical evaluation measure. Dynamic-graph models may be computationally demanding because they process graph structure over many time steps. A small gain in AUC may not justify a large increase in computational cost, especially if threshold-based predictions remain weak.

## 4. The Extended Experiments

### *4.1. Experimental Protocol*

For the real datasets, the pipeline processes graph tensors, dynamic node attributes, and event labels. The data is split chronologically without shuffling, meaning that earlier snapshots are used for training while later snapshots are used for testing. For the RDyn datasets, the raw graph files are converted into adjacency matrices, community files are represented as node features, and event files are transformed into binary labels. A snapshot is labeled as positive when a merge or split event occurs at the corresponding time step.

Unless otherwise specified, training employs the Adam optimizer with a learning rate of 0.005, a dropout rate of 0.02, and a hidden size of 64. Binary cross-entropy is utilized as the loss function for event classification. For the real datasets, the experiments report AUC

and runtime metrics. Additionally, threshold-based evaluations report F1 score, precision, and recall. For the RDyn datasets, both AUC and F1 score are reported.

### *4.2. AUC and Runtime on Real Datasets*

Table 2 presents the AUC and runtime results for the NYC Cab and Twitter Weather datasets. For the NYC Cab dataset, DyGED-NA achieves the highest AUC, while the full DyGED model attains a similar score. In the case of the Twitter Weather dataset, the complete DyGED architecture achieves the best AUC. Across both datasets, DyGED has the longest runtime, which is expected due to its incorporation of both recurrent temporal processing and temporal attention.

**Table 2.** AUC and runtime on NYC Cab and Twitter Weather.

| Model | NYC Cab Dataset AUC | NYC Cab Dataset Time (s) | Twitter Dataset AUC | Twitter Dataset Time (s) |
|---|---|---|---|---|
| DyGED-CT | 0.8696 | 135.28 | 0.6926 | 46.99 |
| DyGED-NL | 0.8560 | 142.84 | 0.6203 | 46.83 |
| DyGED-NA | 0.8873 | 147.14 | 0.6245 | 47.10 |
| DyGED | 0.8857 | 149.08 | 0.7257 | 49.63 |

These results indicate that the benefit of each architectural component depends on the dataset. LSTM-based temporal modeling without attention is especially effective for the NYC Cab dataset, whereas combining recurrent modeling with temporal attention is more beneficial for the Twitter Weather dataset. This interpretation is reasonable because the Twitter Weather dataset is derived from word co-occurrence dynamics, where relevant evidence may be concentrated in particular time steps and therefore may benefit from attention.

### *4.3. F1 and Threshold Behavior*

AUC evaluates how well model scores rank event and non-event snapshots without choosing a decision threshold. In an applied event-detection setting, however, the system must determine whether an event is present at each time step. Therefore, F1, precision, and recall provide important complementary information. Table 3 presents the best threshold-based scores for the two real datasets.

**Table 3.** Best threshold-based F1 scores on real datasets.

| Model | Dataset | F1 | Threshold | Precision | Recall |
|---|---|---|---|---|---|
| DyGED-CT | NYC Cab | 0.27 | 0.45 | 0.21 | 0.36 |
| DyGED-CT | Twitter | 0.33 | 0.20 | 0.29 | 0.38 |
| DyGED-NL | NYC Cab | 0.38 | 0.30 | 0.36 | 0.40 |
| DyGED-NL | Twitter | 0.32 | 0.50 | 0.25 | 0.43 |
| DyGED-NA | NYC Cab | 0.26 | 0.50 | 0.21 | 0.36 |
| DyGED-NA | Twitter | 0.21 | 0.10 | 0.12 | 1.00 |
| DyGED | NYC Cab | 0.23 | 0.20 | 0.28 | 0.20 |
| DyGED | Twitter | 0.37 | 0.15 | 0.26 | 0.64 |

The key observation is that F1 scores remain limited even when AUC is comparatively high. This suggests that the models often rank event snapshots reasonably well, but their binary decisions are highly affected by the selected threshold and by the imbalance between event and non-event cases. A model may assign relatively high scores to many true events and still achieve poor precision or recall once a fixed threshold is imposed.

This finding has practical implications. When the objective is to prioritize suspicious snapshots for further inspection, AUC may be an adequate measure. When the objective

is to generate event alerts, threshold calibration becomes a key part of the system. Potential improvements include cost-sensitive objectives, resampling strategies for rare events, adaptive thresholding, and probability calibration.

### 4.4. RDyn Synthetic Graph Evaluation

Table 4 presents the results for the RDyn datasets. All four models achieved excellent performance on the synthetic merge/split task, with AUC scores above 0.895. The F1 scores are also considerably higher than those obtained on the real datasets. The full DyGED model achieves the best AUC in the 30-node and 150-node configurations and the best F1 in the 500-node configuration. DyGED-CT obtains the highest AUC in the 500-node configuration, showing that a simpler architecture can remain competitive when the synthetic structural signal is strong.

**Table 4.** AUC and F1 on RDyn-generated dynamic graphs.

| Dataset | Model | AUC | F1 | Epochs |
|---|---|---|---|---|
| 30 nodes | DyGED-CT | 0.973 | 0.800 | 50 |
| 30 nodes | DyGED-NL | 0.971 | 0.800 | 50 |
| 30 nodes | DyGED-NA | 0.973 | 0.800 | 50 |
| 30 nodes | DyGED | 0.975 | 0.800 | 50 |
| 150 nodes | DyGED-CT | 0.936 | 0.909 | 100 |
| 150 nodes | DyGED-NL | 0.895 | 0.909 | 100 |
| 150 nodes | DyGED-NA | 0.902 | 0.910 | 100 |
| 150 nodes | DyGED | 0.945 | 0.909 | 100 |
| 500 nodes | DyGED-CT | 0.981 | 0.863 | 150 |
| 500 nodes | DyGED-NL | 0.962 | 0.754 | 150 |
| 500 nodes | DyGED-NA | 0.963 | 0.861 | 150 |
| 500 nodes | DyGED | 0.967 | 0.868 | 150 |

The difference between the real and synthetic outcomes is meaningful. RDyn generates explicit community merge and split events, which form coherent structural patterns and are easier to learn than noisy real-world events. Therefore, the RDyn results should not be viewed as evidence that the models solve every real event-detection setting. Rather, they show that DyGED-like architectures can recognize community-level structural changes when the event mechanism is clear and controlled.

### 4.5. Overall Interpretation

The complete DyGED model appears to be a strong and stable choice across the evaluated scenarios. It does not achieve the best score for every metric, but it consistently remains among the leading alternatives. DyGED-NA performs particularly well on NYC Cab, which suggests that recurrent temporal modeling is useful for mobility-based graph sequences. DyGED performs best on Twitter Weather, indicating that temporal attention can be valuable when event-related information is concentrated in specific time steps.

The RDyn results further show that the models improve when events correspond to clear structural transformations. This supports the view that graph-level neural architectures are appropriate for community-level event detection. Nevertheless, the F1 results on the real datasets emphasize that class imbalance and threshold selection remain important obstacles.

## 5. Conclusions

This study presents an expanded experimental evaluation of Dynamic Graph Event Detection (DyGED) for event detection in dynamic graphs. The analysis includes DyGED and three variants tested on NYC Cab, Twitter Weather, and synthetic dynamic graphs generated

with RDyn. The results indicate that the full DyGED model consistently performs well, particularly on Twitter Weather and several synthetic configurations. Notably, DyGED-NA achieves the highest Area Under the ROC Curve (AUC) score on NYC Cab, while DyGED-CT remains competitive in RDyn scenarios where the structural signal is pronounced.

The experiments also reveal that AUC and F1 scores measure different aspects of model behavior. A model may effectively rank event snapshots yet still produce weak threshold-based alerts due to class imbalance. This distinction is crucial for practical event-detection systems, as the chosen operating threshold directly influences the balance between precision and recall.

Additionally, the RDyn evaluation provides further evidence of DyGED-like architectures' ability to detect community merge and split events when structural changes are clear. Future research should consider conducting repeated experiments with fixed random seeds, exploring a wider range of hyperparameters, testing more synthetic graph configurations, implementing calibration techniques, and performing a more detailed assessment of operating thresholds for real event streams.

**Author Contributions:** Conceptualization: K.C.; methodology: K.C.; software: N.K.; validation: N.K.; formal analysis: N.K., K.C. and C.K.; investigation: N.K., K.C. and C.K.; data curation: N.K.; draft preparation: N.K., K.C. and C.K.; writing—review and editing: N.K., K.C. and C.K.; visualization: N.K., K.C. and C.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The NYC Cab and Twitter Weather datasets are available through the public DyGED repository at https://github.com/NikolaosKatseris/Anomaly-Detection-in-Dynamic-Graphs-Using-Deep-Machine-Learning-Methods (accessed on 25 November 2025). The synthetic datasets were generated using RDyn.

**Acknowledgments:** This manuscript is based on the diploma thesis Anomaly Detection in Dynamic Graphs Using Deep Machine Learning Methods. The author acknowledges the supervision and academic support provided during the thesis work.

**Conflicts of Interest:** Author Christos Konstantopoulos is a member of the Technical Committee of the conference and serves as a reviewer. The authors declare no other conflicts of interest.

## Abbreviations

| | |
|---|---|
| AUC | Area Under the ROC Curve |
| F1 | Harmonic mean of precision and recall |
| GCN | Graph Convolutional Network |
| LSTM | Long Short-Term Memory |
| MLP | Multilayer Perceptron |
| RDyn | Dynamic network generation with planted community events |

## References

1. Kosan, M.; Silva, A.; Medya, S.; Uzzi, B.; Singh, A. Event Detection on Dynamic Graphs. *arXiv* **2021**. [CrossRef]
2. Rossetti, G. RDyn: Graph Benchmark Handling Community Dynamics. GitHub. Available online: https://github.com/GiulioRossetti/RDyn (accessed on 25 November 2025).
3. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection: A Survey. *ACM Comput. Surv.* **2009**, *41*, 1–58. [CrossRef]
4. Akoglu, L.; Tong, H.; Koutra, D. Graph Based Anomaly Detection and Description: A Survey. *Data Min. Knowl. Disc* **2015**, *29*, 626–688. [CrossRef]

5. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv* **2016**. [CrossRef]
6. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

https://doi.org/10.3390/engproc2026167025