



ARTICLE

RDMG: Role-Decomposed Memory for Event-Aware Dynamic Community Detection

Yuchen Liu^{1,*}, Hongchang Chen² and Meng Du¹

¹Institute of Information Technology, Information Engineering University, Zhengzhou, China

²National Digital Switching System Engineering and Technological R&D Center, Zhengzhou, China

*Corresponding Author: Yuchen Liu. Email: liuyuchen2202@163.com

Received: 24 August 2026; Accepted: 20 September 2026; Published: 22 October 2026

ABSTRACT: Dynamic community detection must preserve community continuity while responding to abrupt structural changes. Temporal smoothing can improve stability but can also suppress the signals of community formation, splitting, or dissolution. To address this stability–plasticity tension, we propose Role-Decomposed Memory for Dynamic Graphs (RDMG), a self-supervised framework that separates persistent, appearing, and disappearing edges before temporal modelling. Role-specific graph convolutional branches supply an identity representation from persistent edges and separate recurrent memories for edge births and deaths. An event-aware assignment head adds a gated birth-minus-death residual to the identity representation, with its contribution controlled by local edge turnover. An optional community-state gated recurrent unit updates soft community representations across snapshots and combines their temporal-state affinities with node-level assignment logits. Training combines differentiable modularity, role-specific reconstruction, and gate-entropy regularisation without community-label supervision. Experiments on four real-world dynamic networks and three controlled synthetic benchmarks show competitive clustering and event responsiveness. RDMG obtains the best or second-best result in eleven of twelve real-world metric–dataset combinations and the highest event-restricted adjusted Rand index on all three synthetic variants. These findings support separating persistent structure from event-related edge changes when learning dynamic community assignments.

KEYWORDS: Dynamic community detection; role decomposition; event detection; graph neural network

1 Introduction

Dynamic community detection aims to reveal how groups in a network change over time. Community structure is a basic mesoscale pattern in complex networks, where nodes within the same group interact more densely or share similar functions. Graph representation learning has improved static community detection through graph convolutional encoders, self-supervised objectives, and differentiable clustering losses [1–3]. However, many real networks are not fixed objects. Social contacts, software dependencies, communication records, and citation links change as edges appear and disappear, causing communities to persist, merge, split, emerge, or vanish [4,5]. If a static detector is applied to each snapshot independently, the detected communities may be inconsistent across time and the evolution of a group cannot be tracked reliably.

The central modelling challenge is a stability–plasticity tension: a detector should retain useful community information through noisy fluctuations while adapting when the underlying groups change. Coupled-graph and two-stage methods organise temporal information through cross-time graphs or post-hoc matching [4–6]. Temporal-smoothing, matrix-factorisation, recurrent graph neural network, and

contrastive-learning methods also use historical information to stabilise assignments [6–10]. Smoothing is useful when adjacent-snapshot differences are noise, but a community split or birth can also appear as an abrupt difference. The problem is therefore how to preserve a persistent structural reference while allowing evidence of changing relations to affect the current partition.

We approach this problem through temporal edge roles. Edges present in both adjacent snapshots provide a reference for continuity; edge births and deaths supply distinct evidence about changing relations. A large number of either can reflect a community event or noise, so the edge split alone does not identify semantic events. Its purpose is to preserve the roles of these observations until the model learns how they should influence community assignment. Separate representations and update rules then allow persistent structure and recent changes to contribute differently.

This paper proposes Role-Decomposed Memory for Dynamic Graphs (RDMG), a self-supervised framework that connects this role separation to memory updates and community assignment. A persistent-edge branch supplies the stable identity channel directly, while birth and death branches maintain recurrent event memories. The assignment head adds their difference as a residual whose strength depends on incident edge turnover. When enabled, a shared community-state gated recurrent unit aggregates soft community representations over time and refines the assignment logits. A modularity-based objective and role-specific reconstruction losses provide graph-derived supervision, while a gate-entropy penalty encourages more selective birth-memory updates.

The main contributions of this paper are summarised as follows:

- We develop a role-decomposed architecture that combines persistent/birth/death graph encoders with distinct state updates. Persistent relations provide the identity channel, while recurrent birth and death memories preserve separate evidence about changing relations.
- We introduce an event-aware readout that injects a turnover-modulated birth-minus-death residual into community assignment. An optional shared community-state GRU supplies a complementary temporal reference at the community level.
- We evaluate clustering quality and event responsiveness on four real-world networks and three dynamic-SBM benchmarks. Snapshot metrics, event-restricted ARI, capacity analysis, and independent component ablations characterise the performance and the dataset-dependent benefit of the community-state option.

The remainder of the paper is organised as follows. [Section 2](#) reviews static and dynamic community detection methods. [Section 3](#) presents the role-decomposed memory architecture and training objective. [Section 4](#) reports the main results, parameter analysis, ablation study, and visual analysis. [Section 5](#) concludes the paper.

2 Related Work

2.1 Static Community Detection Methods

Static community detection identifies communities from a single graph and is often organised by the information source used for partitioning: local neighbourhood structure, global connectivity objectives, or contextual node representations [6,11,12]. Representation-learning methods such as DeepWalk learn node context from random walks [13], while graph neural clustering methods couple representation learning with clustering losses. DAEGC uses graph attention for attributed graph clustering [14], SDCN jointly preserves auto-encoder and GCN representations [15], and DMoN optimises a differentiable modularity objective [3]. Self-supervised graph methods such as DGI and GMI further improve embeddings by maximising mutual information between local nodes, graph summaries, and structural views [2,16]. Neural embeddings also

support community detection at scale [17] and overlapping community assignments [18]. Targeted edge additions and removals can alter detected community structure [19].

These methods provide useful building blocks for RDMG, including GCN encoders [1], modularity-aware objectives, and self-supervised structural losses. However, they assume that the graph is observed as one fixed object. Even relation-aware GNNs, which assign different transformations to different edge types [20], do not define edge roles from temporal change. Static graph clustering therefore cannot by itself track the same community across snapshots or separate stable edges from newly appearing and disappearing edges.

2.2 Dynamic Community Detection Methods

Dynamic community detection adds a temporal dimension and is commonly grouped into coupled-graph, two-stage tracking, and temporal-smoothing families [4–6]. Coupled-graph methods construct a single cross-time graph by linking node instances or snapshots and then apply a static community detector [4,6]. This formulation reuses mature static objectives, but it represents community evolution only through the coupled graph, so the timing of a specific change is less explicit than in event-tracking formulations [4,10]. Two-stage methods first detect communities in each snapshot and then match them over time. ICEM maps snapshot-level communities into evolutionary sequences [21], while DynaMo incrementally updates modularity-based communities as the graph evolves [22]. These methods describe evolution explicitly, but their final result depends strongly on the quality of snapshot-level partitions and matching rules [4,5].

Temporal-smoothing methods model adjacent snapshots jointly so that community assignments do not fluctuate unnecessarily. PisCES performs global spectral clustering over all snapshots [23], while local smoothing and joint-learning methods use neighbouring snapshots or recurrent states. jLDEC jointly learns graph representation and nonnegative-matrix-factorisation-based clustering [24]. MFC+Topo preserves persistent structure through topological regularisation [9]. Recent factorisation variants, including DyG-MF, GrSrNMF, and Evo-NMTF, improve scalability or temporal regularisation through low-rank and bi-regularised formulations [25–27]. These methods provide useful temporal constraints, but they do not explicitly distinguish whether an observed edge is persistent, newly appearing, or disappearing before community assignment.

2.3 Contrastive and Role-Aware Dynamic Graph Learning

Deep dynamic methods learn temporal representations jointly with community assignments. Dynamic GCNs propagate graph-convolutional parameters or embeddings over time [7,28]. ODDGI combines recurrent propagation with an optimised Deep Graph Infomax objective [8]. Graph-convolutional contrastive learning aligns complementary structural views [29], and jNCDC combines non-negative matrix factorisation with node-role-guided contrastive sampling [10]. MSGCL integrates neighbour-overlap and topology-similarity views with multi-scale contrastive learning and recurrent temporal smoothing [6]. An enhanced GraphSAGE model further couples temporal representation learning with fast label matching between adjacent snapshots [30].

This contrastive line addresses noise and view inconsistency in graph representations. Local–local objectives encourage agreement among nearby or multi-view node representations, while local–global objectives align node embeddings with graph-level summaries [2,16,31]. Stability- and role-aware methods add another perspective. STR-GNN suppresses spurious temporal variation [32], while CoDÆN highlights the stability–responsiveness trade-off under controlled events [33]. Other studies show that sparse temporal observations can distort dynamic communities [34], and that node roles can guide dynamic graph clustering

with large language models [35]. These methods motivate RDMG, but most of them either pass temporal information through one representation or define roles for nodes rather than edges [6–8,10]. RDMG instead assigns each changed edge to a temporal role and keeps stable structure separate from event-related changes during memory updates.

3 Methodology

3.1 Problem Definition

Let $\mathcal{D} = \{(A^t, X^t)\}_{t=0}^{T-1}$ denote a discrete-time dynamic graph with T snapshots and N aligned node slots. At snapshot t , $A^t \in \{0,1\}^{N \times N}$ is an undirected binary adjacency matrix without self-edges and $X^t \in \mathbb{R}^{N \times F}$ is the node feature matrix; when attributes are unavailable, X^t is the row-normalised adjacency. Node identities are aligned between consecutive snapshots; if a slot is reassigned to a new node, its previous memory is reset. The community count K is supplied as an input; automatic estimation of K is outside the current framework. RDMG learns the stateful mapping

$$f_{\Theta} : (A^{t-1}, A^t, X^t, \mathcal{M}^{t-1}) \mapsto (S^t, \mathcal{M}^t), \quad (1)$$

for $t = 1, \dots, T-1$, where $\mathcal{M}^t = (s^t, e^t, d^t, C^t)$ contains the three node memories and, when enabled, the community state. The soft assignment $S^t \in [0,1]^{N \times K}$ gives the membership probabilities, and hard labels are $\hat{y}_i^t = \arg \max_k S_{ik}^t$. Snapshot 0 supplies the initial graph for edge comparison and is excluded from the evaluation window. Training is self-supervised: community labels are used for NMI and ARI evaluation, while modularity Q depends only on the graph and predicted partition.

Fig. 1 illustrates how edge births and deaths reshape community structure between consecutive snapshots. Because f_{Θ} conditions on both A^{t-1} and A^t , it observes which edges persist, appear, or disappear—the temporal structural signal that RDMG exploits.

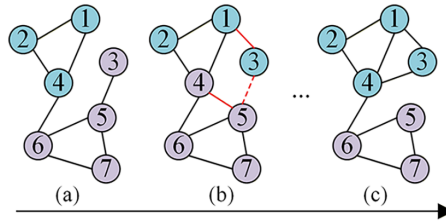


Figure 1: Dynamic graph evolution from (a) G^1 through (b) G^2 to (c) G^T . Solid and dashed red edges denote births and deaths; node colours indicate communities. The schematic uses indices $1, \dots, T$, while the formulation uses $0, \dots, T-1$.

3.2 The Proposed RDMG Method

Fig. 2 outlines three stages of RDMG. Edge-role decomposition first separates persistent, birth, and death edges by comparing consecutive snapshots. A shared base GCN and three role-specific branches then encode snapshot structure and each edge role. The assignment head combines the stable representation with recurrent birth/death evidence to produce community assignments. An optional community-state GRU refines the assignment logits through community-level temporal states (Section 3.2.3).

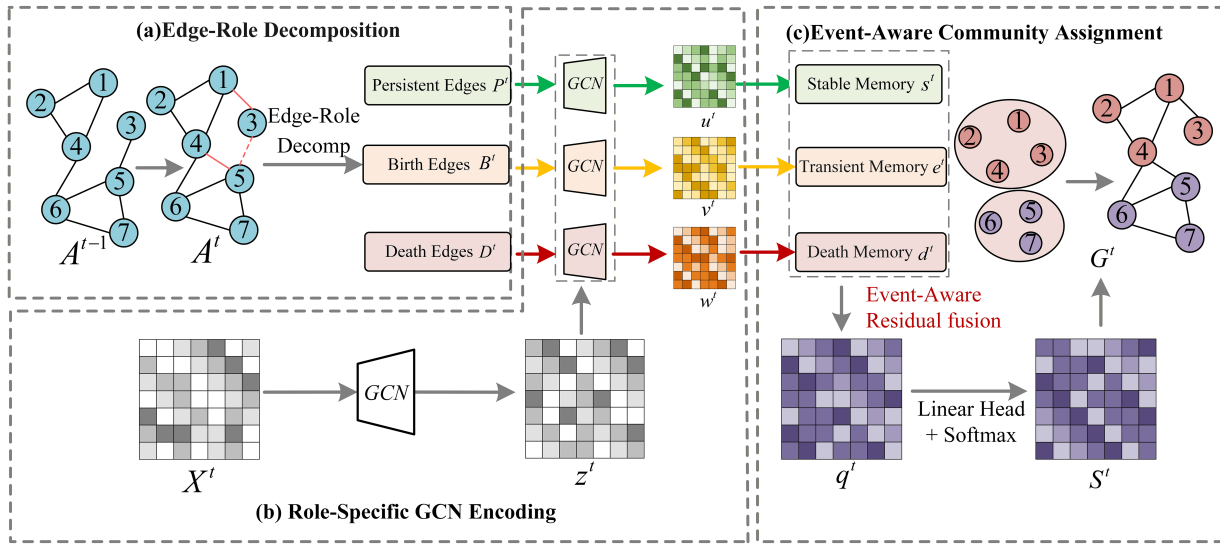


Figure 2: RDMG architecture: (a) Edge-role decomposition; (b) Base and role-specific GCN encoders; (c) Memory updates and community assignment. Green, yellow, and red paths represent persistent, birth, and death edges. The stable state is recomputed; birth/death states are recurrent. The optional community-state (CS) branch is described in [Section 3.2.3](#).

3.2.1 Edge-Role Decomposition

Given consecutive adjacency matrices A^{t-1} and A^t , we partition their edge union into three disjoint roles:

$$P^t = A^{t-1} \odot A^t \quad (\text{persistent edges}), \quad (2)$$

$$B^t = A^t - P^t \quad (\text{birth edges}), \quad (3)$$

$$D^t = A^{t-1} - P^t \quad (\text{death edges}). \quad (4)$$

By construction, $P^t + B^t = A^t$ and $P^t + D^t = A^{t-1}$, so each edge in either snapshot is assigned to exactly one role. Event intensity measures incident edge turnover relative to the current degree:

$$a_i^t = \frac{\sum_j (B_{ij}^t + D_{ij}^t)}{\deg^t(v_i) + 1}, \quad (5)$$

$$\eta_i^t = \begin{cases} a_i^t / \max_j a_j^t, & \max_j a_j^t > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

Thus $\eta_i^t \in [0, 1]$. Both edge additions and removals contribute, including rewiring that leaves a node's degree unchanged. The degree offset avoids division by zero, and the snapshot-wise maximum puts the signal on a common scale for the memory gates and assignment residual.

3.2.2 Role-Specific GCN Encoder and Temporal Memory Updates

A shared two-layer GCN base encoder produces snapshot-level node features:

$$z^t = \text{GCN}_{\text{base}}(X^t, \tilde{A}^t), \quad (7)$$

where $\tilde{A}^t = \Lambda^{t,-1/2}(A^t + I)\Lambda^{t,-1/2}$ is the symmetrically normalised adjacency and Λ^t denotes the diagonal degree matrix of $A^t + I$. Each role adjacency is normalised in the same way after adding self-loops for message passing; these self-loops are excluded from the edge-role targets. We reserve D^t for the death edge subset. For featureless datasets, $X_{ij}^t = A_{ij}^t / \max(\deg^t(v_i), 1)$; for Cora, X^t is the supplied static feature matrix.

Three role-specific single-layer GCN branches consume the base encoding z^t along with their respective role adjacency:

$$u^t = \text{GCN}_{\text{stable}}(z^t, \tilde{P}^t), \quad (8)$$

$$v^t = \text{GCN}_{\text{transient}}(z^t, \tilde{B}^t), \quad (9)$$

$$w^t = \text{GCN}_{\text{death}}(z^t, \tilde{D}^t). \quad (10)$$

RDMG then maintains three node-level memories aligned with the three edge roles:

$$s^t = u^t, \quad (11)$$

$$\lambda^t = \min(\exp(-c \cdot \eta^t), \lambda_{\max}), \quad (12)$$

$$g_e^t = \sigma(W_e[v^t; \eta^t] + b_e), \quad g_d^t = \sigma(W_d[w^t; \eta^t] + b_d), \quad (13)$$

$$e^t = \lambda^t \odot (g_e^t \odot v^t + (1 - g_e^t) \odot e^{t-1}), \quad (14)$$

$$d^t = g_d^t \odot w^t + (1 - g_d^t) \odot d^{t-1}. \quad (15)$$

Here $\lambda_{\max} = 0.95$ and $c = 1.0$. The gates are scalar values per node and are broadcast across the h memory dimensions. The factor λ^t attenuates the entire mixture of current and previous birth information; its cap also causes decay when structural change is small. The death memory uses its update gate without this additional decay. Persistent edges supply the identity channel directly through $s^t = u^t$; this channel has no recurrent averaging of its previous state. Temporal recurrence occurs in e^t , d^t , and the optional community state.

3.2.3 Event-Aware Community Assignment

Community assignment reads the stable memory s^t as the primary identity signal and adds an event-aware residual:

$$q^t = s^t + \sigma(\alpha) \cdot \eta^t \cdot (e^t - d^t), \quad (16)$$

where $\alpha \in \mathbb{R}$ is a learnable scalar gate and σ is the sigmoid. The coordinate-wise difference $(e^t - d^t)$ contrasts learned birth and death representations; the assignment head learns how this signed latent signal affects community membership. The per-node factor η^t makes this residual small for nodes whose neighbourhood changes little. In stable snapshots ($\eta^t \rightarrow 0$), $q^t \approx s^t$, so the assignment mainly follows the stable memory. The community logits and soft assignments are produced by a linear head:

$$\ell^t = q^t W_h, \quad W_h \in \mathbb{R}^{h \times K}. \quad (17)$$

The linear head is bias-free. An optional community-state module refines its logits by maintaining a $K \times h$ state matrix. Let $\tilde{S}^t = \text{softmax}(\ell^t)$ denote the preliminary soft assignment. The representation of community k and the updated community state are

$$\mu^{t,k} = \frac{\sum_i \tilde{S}^{t,i,k} \cdot s^{t,i}}{\sum_i \tilde{S}^{t,i,k} + \varepsilon}, \quad C^t = \text{GRUCell}(\text{stopgrad}(\mu^t), C^{t-1}). \quad (18)$$

The centroid calculation is detached during training: the GRU learns from these summary inputs, while gradients do not pass through their soft-assignment weighting. The final logits mix the node-level linear head with the community-state affinity:

$$\text{logits}^t = \begin{cases} \ell^t, & \text{without community state,} \\ \beta \ell^t + (1 - \beta) s^t \widehat{C}^{t,\top}, & \text{with community state,} \end{cases} \quad (19)$$

where $\widehat{C}^t$ is the row-normalised community state and $\beta = 0.5$. The soft community assignment and hard prediction are then

$$S^t = \text{softmax}(\text{logits}^t), \quad \hat{y}_i^t = \arg \max_k S_{ik}^t. \quad (20)$$

The community state $C^t \in \mathbb{R}^{K \times h}$ is a runtime state buffer requiring $\mathcal{O}(Kh)$ storage, rather than a set of trainable parameters. A single GRUCell(h, h) is shared across all K communities. With two $3h \times h$ weight matrices and two length- $3h$ bias vectors, this cell has $6h^2 + 6h$ trainable parameters, or $\mathcal{O}(h^2)$, independent of K . The module supplies a community-level temporal signal complementary to the node-level memories.

3.2.4 Self-Supervised Objective

RDMG is trained without community labels. The objective combines a modularity-based clustering term, role-specific reconstruction terms, and an entropy penalty on the transient gate:

$$\mathcal{L} = w_c \mathcal{L}_{\text{comm}} + w_p \mathcal{L}_{\text{persist}} + w_t \mathcal{L}_{\text{trans}} + w_d \mathcal{L}_{\text{death}} + w_g \mathcal{L}_{\text{gate}}, \quad (21)$$

with weights $(w_c, w_p, w_t, w_d, w_g) = (2.0, 1.0, 1.0, 1.0, 0.1)$ unless otherwise stated. Inspired by differentiable modularity-based clustering [3], the clustering term promotes structurally coherent communities while discouraging collapsed assignments:

$$\mathcal{L}_{\text{comm}} = -\frac{1}{2m^t} \text{tr}(S^{t,\top} M^t S^t) + \gamma \left\| \frac{S^{t,\top} S^t}{N} - \frac{I_K}{K} \right\|_F, \quad (22)$$

where $\gamma = 1$, $M^t = A^t - \frac{\mathbf{r}^t \mathbf{r}^{t,\top}}{2m^t}$ is the modularity matrix, $\mathbf{r}^t$ is the degree vector, $m^t = \frac{1}{2} \sum_i r_i^t$, and I_K is the K -dimensional identity matrix. A small numerical offset protects the edge-count denominator. The modularity term uses the observed adjacency as its structural signal. The second term penalises deviations of the assignment Gram matrix from I_K/K , favouring distinct, approximately balanced communities and discouraging allocation of all nodes to one cluster. It introduces a balance preference rather than supervision from community labels.

For a representation H and role adjacency E , let $\mathcal{B}(H, E)$ be binary cross-entropy over sampled unordered pairs Ω_E :

$$\mathcal{B}(H, E) = -\frac{1}{|\Omega_E|} \sum_{(i,j) \in \Omega_E} [E_{ij} \log \sigma(H_i^\top H_j) + (1 - E_{ij}) \log (1 - \sigma(H_i^\top H_j))]. \quad (23)$$

The implementation evaluates BCE directly from the logits for numerical stability. Each call independently samples without replacement up to 1000 positive and 1000 negative pairs with $i < j$. Each pool is capped separately at the number available; if either pool is empty, that call contributes zero. When both

pools contain at least 1000 pairs, the sample is balanced; otherwise their sampled counts may differ. The reconstruction terms are

$$\mathcal{L}_{\text{persist}} = \mathcal{B}(s^t, P^t) + \frac{1}{2}\mathcal{B}(u^t, P^t), \quad (24)$$

$$\mathcal{L}_{\text{trans}} = \mathcal{B}(e^t, B^t) + \frac{1}{2}\mathcal{B}(v^t, B^t), \quad (25)$$

$$\mathcal{L}_{\text{death}} = \mathcal{B}(d^t, D^t). \quad (26)$$

These targets are constructed from adjacent observed graphs: P^t supervises persistent relations, B^t supervises new relations, and D^t supervises removed relations. The auxiliary terms act on the branch outputs before memory fusion. Because $s^t = u^t$ in the reported configuration, the two persistent terms use the same representation with separate pair samples; they do not introduce a second persistent-memory mechanism. The birth auxiliary term also constrains v^t when its gated contribution to e^t is small. No ground-truth community or event label is used by these losses. Additional contrastive, orthogonality, and temporal-consistency objectives are inactive in the reported configuration.

Gate-entropy regularisation. The gate penalty acts on the birth/transient update gate:

$$\mathcal{L}_{\text{gate}} = \frac{1}{N} \sum_{i=1}^N \mathcal{H}(g_{e,i}^t), \quad \mathcal{H}(p) = -p \log(p + \varepsilon) - (1 - p) \log(1 - p + \varepsilon). \quad (27)$$

For the ideal binary entropy, ignoring the small numerical stabiliser, $\mathcal{H}(0.5) = \log 2$, while $\mathcal{H}(p) \rightarrow 0$ as $p \rightarrow 0$ or 1, and

$$\frac{d\mathcal{H}}{dp} = \log \frac{1-p}{p}. \quad (28)$$

With positive weight $w_g = 0.1$, minimisation pushes gates below 0.5 towards retaining birth history and gates above 0.5 towards writing the current representation. The decay factor λ^t then attenuates this mixture. This preference discourages prolonged near-equal mixing, which can blur changes by combining old and new birth evidence.

Clustering and reconstruction losses determine which update is useful for each node. Entropy minimisation promotes decisive updates; it does not prevent saturation or require different gate choices across nodes. At exactly $p = 0.5$, the ideal entropy gradient is zero. Only the birth gate receives this penalty; the stable channel writes directly without a trainable gate.

3.2.5 Training and Inference

Algorithm 1 gives the complete training loop for one model seed. Parameters and Adam statistics are initialised once; node and community states are reset at the start of each epoch. Each transition contributes one optimiser update. After that update, all recurrent states, including C^t , are detached before the next transition, limiting back-propagation through time to one step while preserving the state values. If a node ID changes at an aligned slot, its node-memory rows are cleared before the forward pass. Training runs for 100 epochs with the settings in [Tables 1](#) and [2](#); there is no validation-based early stopping, and the final epoch supplies the parameters for evaluation.

Algorithm 1: RDMG training for one model seed**Require:** Snapshots $\{(A^t, X^t)\}_{t=0}^{T-1}$ and aligned node IDs; K ; model seed; settings in [Tables 1 and 2](#)**Ensure:** Final parameters Θ^* after $E_{\max} = 100$ epochs

```

1: Set random seeds; initialise  $\Theta$  and the Adam optimiser
2: for  $e_{\text{epoch}} = 1$  to  $E_{\max}$  do
3:   Enable training mode; set  $s^0, e^0, d^0 \leftarrow 0$  and  $C^0 \leftarrow 0$  if CS is enabled
4:   for  $t = 1$  to  $T - 1$  do
5:     Clear previous node-memory rows whose node IDs have changed
6:     Compute  $P^t, B^t, D^t$  and  $\eta^t$  from the adjacent graphs
7:      $z^t \leftarrow \text{GCN}_{\text{base}}(X^t, \tilde{A}^t)$ 
8:      $u^t, v^t, w^t \leftarrow \text{GCN}_{\text{stable/transient/death}}(z^t, \tilde{P}^t/\tilde{B}^t/\tilde{D}^t)$ 
9:      $s^t \leftarrow u^t$ ;  $\lambda^t \leftarrow \min(\exp(-c \cdot \eta^t), \lambda_{\max})$ 
10:    Compute gates  $g_e^t \leftarrow \sigma(W_e[v^t; \eta^t] + b_e)$  and  $g_d^t \leftarrow \sigma(W_d[w^t; \eta^t] + b_d)$ 
11:     $e^t \leftarrow \lambda^t \odot (g_e^t \odot v^t + (1 - g_e^t) \odot e^{t-1})$ 
12:     $d^t \leftarrow g_d^t \odot w^t + (1 - g_d^t) \odot d^{t-1}$ 
13:     $q^t \leftarrow s^t + \sigma(\alpha) \cdot \eta^t \cdot (e^t - d^t)$ 
14:     $\ell^t \leftarrow q^t W_h$ 
15:    if community-state GRU is enabled then
16:      Compute  $\tilde{S}^t$  and centroids  $\mu^t$ ;  $C^t \leftarrow \text{GRUCell}(\text{stopgrad}(\mu^t), C^{t-1})$ 
17:       $\text{logits}^t \leftarrow \beta \ell^t + (1 - \beta) s^t \tilde{C}^{t,\top}$ 
18:    else
19:       $\text{logits}^t \leftarrow \ell^t$ 
20:    end if
21:     $S^t \leftarrow \text{softmax}(\text{logits}^t)$ 
22:    Sample node pairs for each reconstruction call; compute  $\mathcal{L}$  in Eq. \(21\)
23:    Zero gradients; back-propagate  $\mathcal{L}$ ; clip gradient norm to 1.0; take one Adam step
24:    Detach  $s^t, e^t, d^t$  and, if enabled,  $C^t$ 
25:  end for
26: end for
27: return  $\Theta^* \leftarrow \Theta$ 

```

Table 1: Dataset statistics and dataset-specific RDMG settings.

Dataset	N	$\sum_t E_t $	$ E_t $ Range	T	K	F	h	CS
HighSchool	327	13,435	1179–2242	9	7	327	128	On
Java	376	12,190	609–610	20	20	376	512	On
CellphoneCall	400	5124	498–530	10	20	400	256	Off
Cora	2708	15,988	1082–5278	5	7	1433	256	Off
DynBench-A	350	200,569	4892–7861	30	5	16	64	Off
DynBench-B	500	428,752	7283–13,736	40	6	16	64	Off
DynBench-C	500	482,267	9045–13,263	40	6	16	64	Off

Note: N , T , K , and F denote nodes, snapshots, communities, and input features. $|E_t|$ counts undirected edges once per snapshot. GCN and memory widths are tied at h ; the optional community-state (CS) GRU also has input and hidden width h .

Table 2: Shared RDMG hyperparameters; loss weights correspond to [Eq. \(21\)](#).

Parameter	Setting
GCN depth	Two base layers; one layer per role branch
Activation and dropout	ReLU and dropout 0.1 after the first base layer; other encoder outputs are linear
Optimiser	Adam; learning rate 10^{-3} ; weight decay 10^{-5} ; moment coefficients $(0.9, 0.999)$; $\epsilon = 10^{-8}$
Training and checkpoint	100 epochs; gradient-norm clipping 1.0; final-epoch parameters, with no early stopping
Loss weights	$(w_c, w_p, w_t, w_d, w_g) = (2, 1, 1, 1, 0.1)$; $\gamma = 1$; auxiliary persistent and birth terms each weighted 0.5
Event decay	$c = 1.0$; $\lambda_{\max} = 0.95$
Event residual and CS mix	Learnable $\sigma(\alpha)$ initialised to 0.5; CS coefficient $\beta = 0.5$ when enabled
Edge sampling per BCE	Up to 1000 positive and 1000 negative pairs

Algorithm 2 separates evaluation from fitting. Parameters are fixed, dropout is disabled, and memories are reset before traversing the sequence. The forward update uses only the previous graph, current graph/features, and carried state; it does not use event annotations, community labels, reconstruction samples, or optimiser updates. NMI, ARI, and Q are averaged over snapshots $1, \dots, T-1$ within a run and then over model seeds. event-ARI averages ARI only over annotated event snapshots in that same window. A sequence without an annotated event has no defined event-ARI and should not be assigned a score of zero by default.

Algorithm 2: RDMG inference with fixed parameters

Require: Learned parameters Θ^* ; snapshots $\{(A^t, X^t)\}_{t=0}^{T-1}$ with aligned node IDs; the training configuration

Ensure: Soft assignments and hard partitions $\{(S^t, \hat{y}^t)\}_{t=1}^{T-1}$

- 1: Load Θ^* ; enable evaluation mode and disable gradient recording
 - 2: Initialise $s^0, e^0, d^0 \leftarrow 0$; set $C^0 \leftarrow 0$ if CS is enabled
 - 3: **for** $t = 1$ to $T - 1$ **do**
 - 4: Clear node-memory rows whose node IDs have changed
 - 5: Compute P^t, B^t, D^t and η^t from A^{t-1}, A^t
 - 6: Encode X^t and role graphs; update s^t, e^t, d^t as in [Section 3.2.2](#)
 - 7: Compute q^t and linear logits $\ell^t = q^t W_h$
 - 8: **if** CS is enabled **then**
 - 9: Compute centroids μ^t ; update C^t using the shared GRU
 - 10: logits ^{t} $\leftarrow \beta \ell^t + (1 - \beta) s^t \hat{C}^{t,\top}$
 - 11: **else**
 - 12: logits ^{t} $\leftarrow \ell^t$
 - 13: **end if**
 - 14: $S^t \leftarrow \text{softmax}(\text{logits}^t)$; $\hat{y}_i^t \leftarrow \arg \max_k S_{ik}^t$
-

(Continued)

Algorithm 2 (continued)

```

15: Store  $(S^t, \hat{y}^t)$  and carry the updated memories to the next snapshot
16: end for
17: return  $\{(S^t, \hat{y}^t)\}_{t=1}^{T-1}$ 

```

3.3 Computational Complexity Analysis

We analyse one snapshot transition, with N nodes, input dimension F , tied GCN/memory width h , K communities, and b sampled pairs in total across the reconstruction calls. Let $\delta_{CS} \in \{0, 1\}$ indicate whether the community-state GRU is enabled. The implementation used by the current configuration materialises dense adjacency, role, and modularity matrices; sparse-graph bounds alone therefore do not describe its execution cost.

Time complexity. Dense role decomposition and normalisation cost $\mathcal{O}(N^2)$. The encoders cost $\mathcal{O}(NFh + Nh^2 + N^2h)$, including the input projection and dense graph aggregation. Memory updates and assignment add $\mathcal{O}(NhK)$; CS adds $\mathcal{O}(NhK + Kh^2)$. The modularity product costs $\mathcal{O}(N^2K + NK^2)$, and the assignment Gram penalty costs $\mathcal{O}(NK^2)$. Although sampled pair scoring costs $\mathcal{O}(bh)$, the current sampler enumerates eligible pairs in dense matrices, adding $\mathcal{O}(N^2)$ work. The per-transition training cost is therefore

$$\mathcal{O}(N^2h + NFh + Nh^2 + N^2K + NK^2 + NhK + \delta_{CS}Kh^2 + bh). \quad (29)$$

The quadratic dependence arises from both graph aggregation and the training objective. For $E_{\max}$ epochs and T snapshots, the training cost scales by $E_{\max}(T - 1)$. Fixed-parameter inference omits the loss and reconstruction sampling, leaving $\mathcal{O}(N^2h + NFh + Nh^2 + NhK + \delta_{CS}Kh^2)$ per transition.

Space complexity. Active graph tensors, features, node states, assignments, and loss intermediates require $\mathcal{O}(N^2 + NF + Nh + NK + K^2 + \delta_{CS}Kh)$ storage per transition. The three node memories contain $3Nh$ values. The CS buffer contains Kh values, whereas its shared GRU has $6h^2 + 6h$ trainable parameters. Model weights, gradients, and Adam statistics contribute an additional $\mathcal{O}(P_{\Theta})$ for P_{Θ} total parameters. Detaching states bounds the autograd history to one transition, but does not remove data storage: the current loader retains all snapshots on the compute device, with an upper bound of $\mathcal{O}(TN^2 + TNF)$ for the resident sequence. Section 4.11 discusses the practical memory implications and corresponding implementation improvements.

4 Experiments**4.1 Datasets**

The experiments use four real-world graphs and three synthetic graphs. Table 1 summarises their statistics and dataset-specific RDMG settings.

Real-world datasets. We use four real-world dynamic graphs: HighSchool,¹ Java,² CellphoneCall,³ and Cora.⁴ They cover contact, software, communication, and citation networks, providing a compact but diverse testbed for dynamic community detection.

¹<http://www.sociopatterns.org/datasets>

²<https://github.com/gephi/gephi/wiki/Datasets>

³<http://www.cs.umd.edu/hcil/VASTchallenge08/download/Download.htm>

⁴<https://linqs.so.e.ucsc.edu/data>

HighSchool: This dataset records proximity interactions among students in a French high school over five days in December 2013. Nodes denote students, and edges denote sensor-detected close contacts within the corresponding time window.

Java: This dataset describes the structural evolution of an open-source Java software project. Nodes correspond to source-code entities, and edges encode co-change or dependency relations observed during project evolution.

CellphoneCall: This dataset is constructed from cellular communication records in the VAST 2008 Challenge. Nodes represent mobile devices, while edges correspond to calls observed within each temporal window.

Cora: Following Gao et al. [7], we construct five cumulative snapshots by progressively adding edges from this static citation network. The sequence represents constructed edge accumulation, not naturally observed temporal evolution. Each node carries a 1433-dimensional bag-of-words feature vector, and topic labels are reserved for evaluation.

Synthetic datasets. We generate three synthetic dynamic graphs from a dynamic stochastic block model with controlled merge, split, birth, and death events. Event annotations are used only for event-restricted evaluation.

DynBench-A/B/C: The three variants provide controlled settings with different scales and noise levels. DynBench-A is a small and difficult setting with 350 nodes, 5 communities, and 30 snapshots. DynBench-B is the default mid-scale setting with 500 nodes, 6 communities, and 40 snapshots. DynBench-C keeps the same scale as DynBench-B but adopts a lower community-to-noise contrast, making community transitions harder to separate from random edge noise.

4.2 Evaluation Metrics and Experimental Setup

Evaluation Metrics: We report Normalised Mutual Information (NMI), Adjusted Rand Index (ARI), and modularity (Q), following the standard evaluation protocol in dynamic community detection [6–8]. NMI and ARI measure agreement between detected communities and ground-truth labels, while Q evaluates the structural quality of the detected partition. For DynBench-A/B/C, we additionally report event-ARI, defined as ARI averaged only over snapshots with annotated merge, split, birth, or death events. Clustering scores are percentages; higher values indicate better performance. Score differences are reported in percentage points (pp). In comparison tables, **bold** and underlining mark the best and second-best distinct values, including ties.

Experimental Environment: The experiments use an NVIDIA GeForce RTX 4060 Laptop GPU with 8 GiB of memory and an Intel Core i9-13900HX CPU. The software environment is Python 3.9.7, PyTorch 2.4.0, and CUDA 12.4. Resource measurements use FP32 GPU arithmetic with TF32 disabled and four CPU numerical-library threads.

Parameter Settings: The number of communities K is treated as known. Table 1 gives the dataset-specific model settings, Table 2 lists the shared hyperparameters, and Table 3 reports the synthetic generator parameters.

With the tied widths in Table 1, the base encoder maps $F \rightarrow h \rightarrow h$, and each role encoder maps $h \rightarrow h$. For HighSchool, Java, and CellphoneCall, the input is the row-normalised adjacency matrix, so $F = N$; Cora uses its supplied 1433-dimensional attributes. Synthetic features are 16-dimensional Gaussian vectors conditioned on community membership, with within-community noise standard deviation 0.3. This construction introduces target-related information into the inputs even though labels are not used in the

training objective. The synthetic benchmark therefore evaluates community-informative attributes and does not establish performance with attributes independent of community labels.

Table 3: Dynamic stochastic block model generator settings. Merge, split, birth, and death probabilities apply per snapshot.

Parameter	DynBench-A	DynBench-B	DynBench-C
Within-community edge probability	0.12	0.15	0.12
Between-community edge probability	0.03	0.02	0.03
Edge noise rate	0.03	0.02	0.03
Node churn rate	0.12	0.10	0.12
Merge/split probability	0.07/0.07	0.05/0.05	0.07/0.07
Birth/death probability	0.04/0.04	0.03/0.03	0.04/0.04

The dataset-specific widths and CS switches in Table 1 are fixed for the main comparisons, with one configuration used for all metrics of a dataset. The hidden-width candidate set is $\{64, 128, 256, 512\}$. Unless stated otherwise, scores average transitions $1, \dots, T - 1$ and model seeds $\{42, 43, 44\}$; synthetic graph seed 42 is fixed. These repetitions vary model randomness on one generated graph sequence per variant, rather than testing robustness across independent graph realisations. The event-ARI metric uses only annotated event snapshots within this window. Section 4.7.2 varies memory width and CS while fixing the intermediate GCN width. The component ablations in Section 4.8 retain the dataset-specific widths and change one mechanism at a time from the reported RDMG configuration.

4.3 Baselines

The real-world comparison uses five dynamic community detection baselines that cover recurrent temporal smoothing, dynamic GCN parameter evolution, matrix-factorisation regularisation, joint NMF-contrastive learning, and multi-scale contrastive learning: ODDGI [8], DGCN [7], MFC+Topo [9], jNCDC [10], and MSGCL [6]. The synthetic benchmark further includes PerSnap, SimplifiedNMF, and SingleMem as diagnostic baselines for snapshot-only detection, simple temporal smoothing, and memory without edge-role decomposition.

ODDGI. ODDGI extends Deep Graph Infomax to evolving networks by combining graph convolution with recurrent temporal propagation. Its aggregation mechanism encourages neighbouring snapshots to share compatible embedding information.

DGCN. DGCN learns dynamic graph representations by coupling GCN aggregation with a recurrent update of model parameters. The method uses structural similarity to guide aggregation and a matrix-valued LSTM to transfer information across time.

MFC+Topo. MFC+Topo combines deep matrix factorisation with a topological constraint on temporal evolution. It is designed to retain recurring community structure while allowing snapshot-level clustering to change.

jNCDC. jNCDC formulates dynamic community tracking as a joint factorisation and contrastive-learning problem. Historical features are used to build an evolutionary graph, while node-role information guides the construction of contrastive pairs.

MSGCL. MSGCL learns complementary structural views and aligns them through multi-scale contrastive objectives. It combines local and global graph representations with recurrent temporal modelling and serves as a strong recent neural baseline in our real-world comparison.

MSGCL uses its official implementation on real-world data and a compact shared-framework implementation (width 64, 100 epochs) in the synthetic diagnostic comparison.

PerSnap. PerSnap applies spectral clustering independently to each snapshot’s adjacency matrix, using precomputed affinities and k -means assignment with five initialisations. Because it ignores temporal history, it provides a reference for responsiveness without temporal smoothing.

SimplifiedNMF. SimplifiedNMF factorises each snapshot while penalising large changes between consecutive factor matrices. It represents a compact non-neural baseline for smooth temporal evolution. We use rank K , temporal weight 0.5, 30 multiplicative iterations per snapshot, and selective-update threshold 0.3.

SingleMem. SingleMem encodes the full current adjacency and updates one gated memory without the persistent/birth/death split. Its GCN hidden width is 96 and its memory width is 128; it uses a prototype-based assignment head. It provides a diagnostic reference for unified temporal memory. Its capacity and readout differ from RDMG; [Section 4.8](#) separately isolates the role mechanisms.

4.4 Experiments on Real-World Datasets

[Table 4](#) reports the real-world comparison. RDMG obtains the best or second-best value in eleven of twelve metric–dataset combinations, spanning contact, software, communication, and citation networks. The gains span label agreement and modular structure, with their magnitude varying across datasets.

Table 4: Community detection results on four real-world datasets (%).

Dataset	Metric	ODDGI	DGCN	MFC+Topo	jNCDC	MSGCL	RDMG
HighSchool	NMI	67.27	65.47	64.72	66.71	70.14	<u>68.71</u>
	ARI	56.19	53.71	47.67	54.79	<u>56.56</u>	56.80
	Q	59.56	67.02	67.31	67.63	<u>70.49</u>	70.80
Java	NMI	75.97	76.21	70.14	76.82	<u>77.69</u>	77.88
	ARI	49.23	52.13	42.32	52.65	56.14	<u>53.30</u>
	Q	70.57	64.92	61.47	65.92	67.58	<u>70.22</u>
CellphoneCall	NMI	37.45	37.18	34.13	36.76	39.05	<u>37.66</u>
	ARI	11.81	13.39	8.42	<u>14.23</u>	15.21	14.07
	Q	60.45	61.15	60.52	<u>62.47</u>	63.25	<u>62.83</u>
Cora	NMI	15.61	27.79	20.78	26.99	<u>30.01</u>	37.56
	ARI	5.44	16.87	12.89	15.73	<u>18.22</u>	30.04
	Q	24.84	62.57	39.65	63.26	<u>66.77</u>	71.09

Note: Bold and underlined values indicate the best and second-best distinct results, respectively, for each metric on each dataset, including ties.

The clearest gains occur on Cora, where RDMG achieves the best NMI, ARI, and Q, exceeding MSGCL by 7.55, 11.82, and 4.32 pp, respectively. Cora contains informative node attributes and evolves by progressively adding citation edges. The persistent-edge channel supplies a structural reference, complemented by recurrent event memories; the component analysis in [Section 4.8](#) tests these contributions directly. On

HighSchool, RDMG achieves the best ARI (56.80) and Q (70.80), while ranking second in NMI (68.71), indicating that the model captures strong contact-based communities while remaining in agreement with the student-class labels.

Java exhibits a different pattern. It has more snapshots and a larger number of communities, so community assignments can become unstable if each snapshot is treated too locally. RDMG obtains the best NMI (77.88), the second-best ARI (53.30), and the second-best Q (70.22), with modularity only 0.35 pp below ODDGI. The CS ablation in Section 4.8 confirms improvements in all three Java metrics. CellphoneCall is a more difficult setting because phone-call edges are sparse and noisy indicators of community membership. With CS disabled, RDMG obtains the second-best Q (62.83), 0.42 pp below MSGCL, the second-best NMI (37.66), and the third-best ARI (14.07), 0.16 pp below jNCDC. These results indicate competitive structural quality and label agreement on noisy communication data.

4.5 Experiments on Synthetic Datasets

The real-world datasets used here have static community labels, so they cannot directly evaluate whether a method reacts to community-level events. DynBench-A/B/C address this gap by providing annotated merge, split, birth, and death snapshots. Table 5 and Fig. 3 therefore evaluate not only average clustering accuracy, but also event-restricted ARI.

Table 5: Clustering results on DynBench-A/B/C (%). event-ARI denotes ARI averaged over annotated community-event snapshots.

Method	DynBench-A			DynBench-B			DynBench-C		
	ARI	NMI	event-ARI	ARI	NMI	event-ARI	ARI	NMI	event-ARI
ODDGI	17.40	28.10	13.60	24.30	37.50	19.40	19.20	31.00	13.80
DGCN	20.52	<u>31.56</u>	17.79	21.05	34.66	20.47	21.76	36.14	15.51
MFC+Topo	13.50	24.70	8.50	47.90	62.30	51.20	24.60	39.10	26.80
jNCDC	19.70	29.40	15.90	29.80	42.60	25.30	22.40	34.50	17.20
MSGCL	12.92	15.33	15.30	21.39	22.70	0.00	8.96	9.81	9.61
SingleMem	5.92	6.31	1.82	33.73	39.77	28.18	13.90	16.50	9.31
SimplifiedNMF	14.09	26.03	9.72	50.15	65.60	55.39	27.03	42.09	29.03
PerSnap	<u>27.32</u>	32.78	<u>20.12</u>	<u>72.88</u>	<u>73.45</u>	<u>74.72</u>	<u>36.44</u>	<u>45.60</u>	<u>42.00</u>
RDMG	31.48	30.50	25.83	76.87	79.00	77.67	46.08	47.77	45.40

Note: Bold and underlined values indicate the best and second-best distinct results, respectively, for each metric on each dataset, including ties.

The synthetic results show the main advantage of RDMG: it reacts to events while keeping good overall clustering quality. Across the nine metric-variant combinations, RDMG obtains the best value in eight. It achieves the highest event-ARI on all three variants, with scores of 25.83, 77.67, and 45.40 on DynBench-A/B/C, and it also obtains the best ARI on every variant. For NMI, RDMG ranks first on DynBench-B and DynBench-C, while PerSnap and DGCN are higher on DynBench-A. This pattern indicates that the proposed model is especially strong when event responsiveness and overall partition accuracy must be considered together.

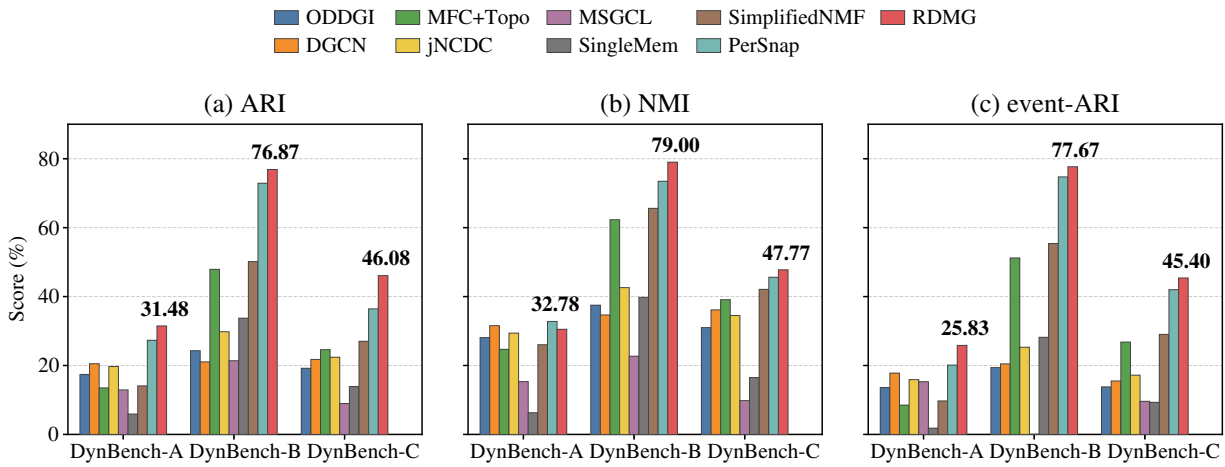


Figure 3: Overall and event-restricted clustering performance on DynBench-A/B/C (%). Bar colours identify methods, as in the shared legend.

The diagnostic baselines compare snapshot-level detection with temporal approaches. Relative to PerSnap, RDMG improves ARI by 4.16, 3.99, and 9.64 pp on DynBench-A/B/C, respectively. The corresponding event-ARI gains are 5.71, 2.95, and 3.40 pp. RDMG thus combines strong event responsiveness with higher partition quality than snapshot-level detection. RDMG also exceeds SingleMem on every reported synthetic score, supporting the effectiveness of the complete role-decomposed design relative to unified memory with a prototype readout. SimplifiedNMF and MFC+Topo remain competitive on DynBench-B/C, while RDMG's higher event-ARI demonstrates its advantage on annotated community transitions.

4.6 Statistical Comparison

RDMG exceeds all eight synthetic baselines in ARI and event-ARI on every scenario. Each of these 16 baseline-metric comparisons has positive and negative rank sums of 6 and 0; median gains over PerSnap are 4.16 and 3.40 pp, respectively.

We apply two-sided exact Wilcoxon signed-rank tests [36] to four real-world dataset pairs or three synthetic scenario pairs per metric. Each pair uses one reported method mean; snapshots, seeds, and metrics are not pooled. Nonzero absolute differences receive average ranks for ties; exact p -values enumerate sign assignments. We use the displayed score precision and cross-check results against SciPy 1.13.1. Holm correction covers all 39 comparisons (five real-world and eight synthetic baselines, each on three metrics), at $\alpha = 0.05$.

Table 6 reports unadjusted $p = 0.125$ –1.000 and Holm-adjusted $p = 1.000$. For four or three nonzero dataset pairs, the minimum attainable two-sided exact value, $2/2^n$, is 0.125 or 0.250, respectively. We therefore interpret the tests primarily as evidence of consistency across the evaluated datasets. The measured gains and uniformly positive signed ranks for synthetic ARI and event-ARI highlight RDMG's consistent performance advantage across all three scenarios.

Table 6: Exact two-sided Wilcoxon p -values for RDMG vs. each baseline. All 39 Holm-adjusted values are 1.000; dashes denote unreported metrics.

Baseline	NMI	ARI	Q	event-ARI
<i>Real-world datasets ($n = 4$)</i>				
ODDGI	0.125	0.125	0.250	–
DGCN	0.125	0.125	0.125	–
MFC+Topo	0.125	0.125	0.125	–
jNCDC	0.125	0.250	0.125	–
MSGCL	1.000	1.000	0.375	–
<i>Synthetic scenarios ($n = 3$)</i>				
ODDGI	0.250	0.250	–	0.250
DGCN	0.500	0.250	–	0.250
MFC+Topo	0.250	0.250	–	0.250
jNCDC	0.250	0.250	–	0.250
MSGCL	0.250	0.250	–	0.250
SingleMem	0.250	0.250	–	0.250
SimplifiedNMF	0.250	0.250	–	0.250
PerSnap	0.750	0.250	–	0.250

4.7 Parameter Analysis

4.7.1 Event-Decay Parameters

We test the event-decay coefficient $c \in \{0, 0.5, 1, 2, 4\}$ and maximum retention $\lambda_{\max} \in \{0.50, 0.70, 0.85, 0.95, 0.99\}$ on DynBench-B/C, fixing all other settings. Three model seeds per combination give 150 fits. The control $c = 0$ makes $\lambda_i^t = \lambda_{\max}$ independent of event intensity. Fig. 4 summarises the results.

All four panels peak at the default $(c, \lambda_{\max}) = (1, 0.95)$, reproducing the DynBench-B/C scores in Table 5. At fixed $\lambda_{\max} = 0.95$, removing event modulation ($c = 0$) lowers event-ARI by 2.07 and 2.70 pp; aggressive attenuation ($c = 4$) lowers it by 2.67 and 1.90 pp. At $c = 1$, a low retention cap of 0.50 lowers ARI by about 4.28 pp and event-ARI by about 5.69 pp on both benchmarks. These controls support retaining useful birth-edge evidence while adjusting its persistence to local change.

The optimum lies within a broad plateau: for $c \in \{0.5, 1, 2\}$ and $\lambda_{\max} \in \{0.85, 0.95, 0.99\}$, ARI ranges are 76.10%–76.87% on DynBench-B and 45.20%–46.08% on DynBench-C; event-ARI ranges are 76.10%–77.67% and 44.00%–45.40%. This pattern supports the shared decay setting across the evaluated configurations.

4.7.2 Memory and Community-State Capacity

We vary memory width $h \in \{64, 128, 256, 512\}$ with CS on and off, fixing the intermediate GCN width at 512 on Java and 256 on CellphoneCall. Encoder outputs and role representations use width h ; the optional CS GRU also uses input and hidden width h . Remaining settings follow Section 4.2. Fig. 5 presents Java and CellphoneCall; Cora is omitted after inspection showed comparatively small variation.

Java benefits from increasing width throughout the tested range: ARI rises from 46.52% to 53.30% with CS and from 44.85% to 50.24% without it. CS improves ARI at every width by 1.67–3.06 pp, supporting the combination of node-representation capacity and community-level temporal state.

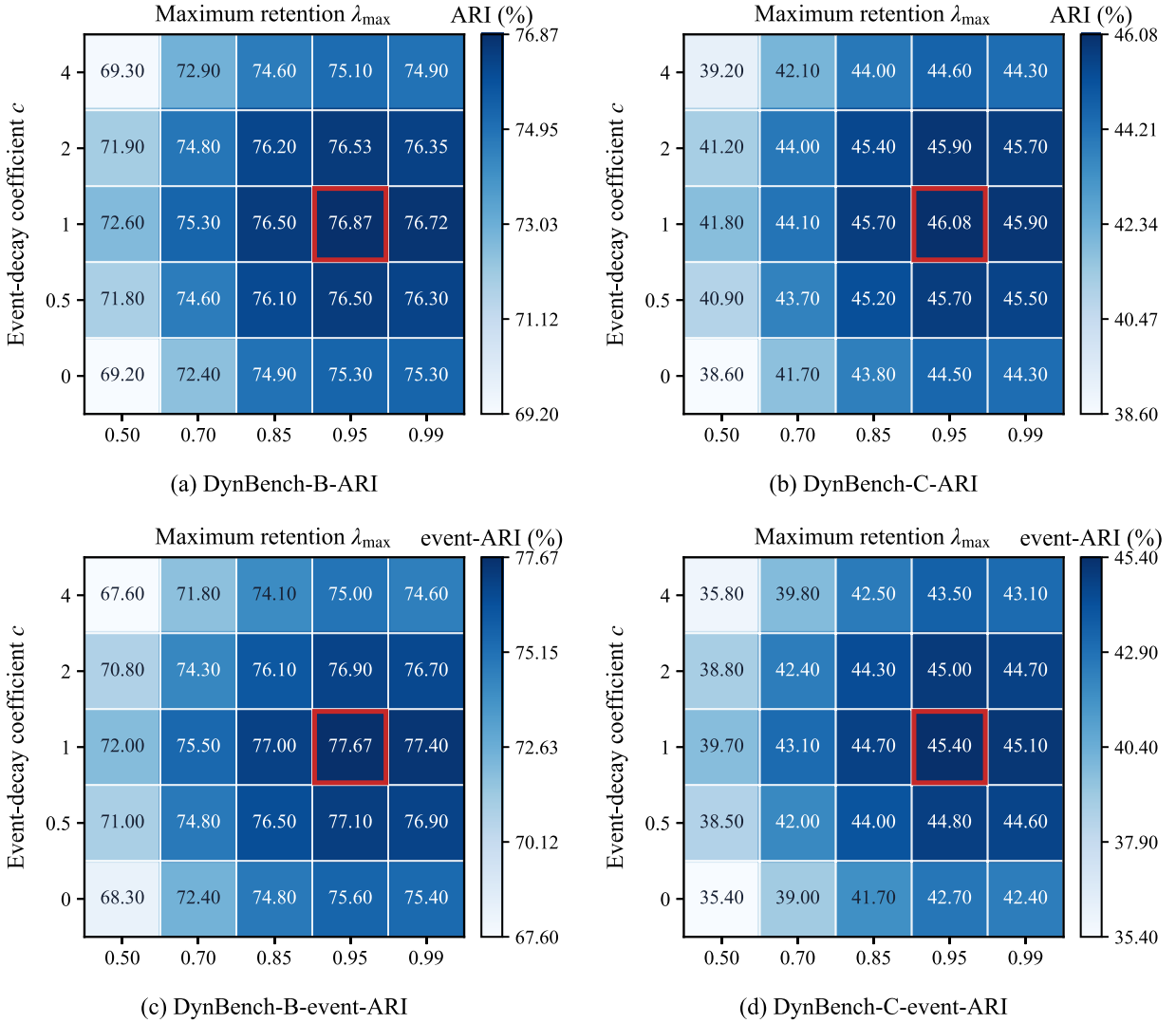


Figure 4: Sensitivity to event decay c and retention cap $\lambda_{\max}$ on DynBench-B/C. Cells report ARI or event-ARI (%); red frames mark the default (1, 0.95).

CellphoneCall peaks at $h = 256$: ARI is 14.07% without CS and 13.51% with it, decreasing to 13.72% and 13.40% at $h = 512$. CS off remains higher by 0.32–0.64 pp across widths. These results support the reported choices of $h = 512$ with CS on Java and $h = 256$ without CS on CellphoneCall.

4.8 Ablation Study

Table 7 tests independent interventions on Java, CellphoneCall, and Cora at widths 512, 256, and 256. Each changes one mechanism while retaining the remaining dataset-specific settings. Alternative CS disables community state on Java and enables it on CellphoneCall and Cora. Shared history averages the previous birth/death states but retains separate current branch outputs. Without role routing, all three encoders use $\text{clip}(P^t + B^t + D^t, 0, 1)$, retaining role-specific reconstruction targets. Without the event residual, $q^t = s^t$, retaining event gates, decay, and auxiliary losses. Without gate entropy, $w_g = 0$ removes the birth-gate penalty while retaining update gates.

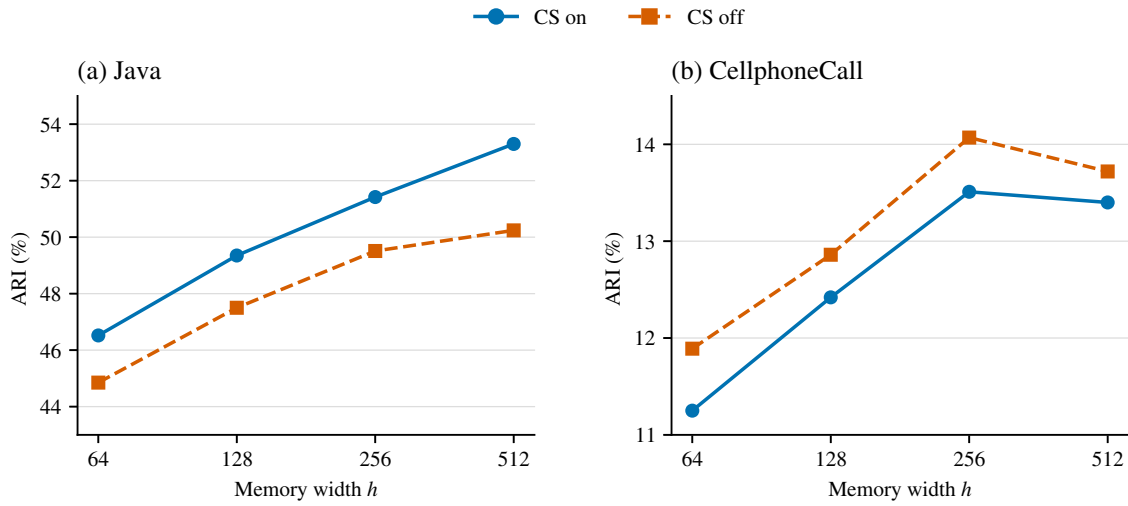


Figure 5: ARI vs. memory width with community state (CS) enabled or disabled on Java and CellphoneCall. Panels use different vertical ranges to show within-dataset changes.

Table 7: Independent component ablations on Java, CellphoneCall, and Cora (%). Each variant changes one mechanism from the reported RDMG configuration.

Variant	Java			CellphoneCall			Cora		
	NMI	ARI	Q	NMI	ARI	Q	NMI	ARI	Q
RDMG	77.88	53.30	70.22	<u>37.66</u>	14.07	<u>62.83</u>	37.56	30.04	71.09
Alternative CS	75.02	50.24	69.23	36.92	<u>13.51</u>	64.03	37.56	30.04	71.09
Shared history	75.83	51.57	<u>69.66</u>	37.28	13.46	61.96	36.50	28.43	68.67
w/o role routing	67.95	40.59	66.56	32.89	11.06	58.22	34.89	28.08	66.98
w/o event residual	69.73	42.56	67.13	33.06	11.69	59.07	35.64	28.23	67.95
w/o gate entropy	<u>76.50</u>	<u>52.13</u>	66.36	37.89	13.13	61.36	<u>36.88</u>	<u>28.59</u>	<u>69.35</u>

Note: Bold and underlined values indicate the best and second-best distinct results, respectively, for each metric on each dataset, including ties.

RDMG is highest or tied-highest in seven of nine metric–dataset combinations. Removing role routing lowers all nine scores, with ARI losses of 12.71, 3.01, and 1.96 pp on Java, CellphoneCall, and Cora, respectively. Removing the event residual also lowers every score, with ARI losses of 10.74, 2.38, and 1.81 pp. These are the two largest ARI reductions among the four mechanism ablations on each dataset. Both role-specific propagation and the direct birth-minus-death assignment path therefore contribute strongly to partition quality.

Sharing birth/death history also lowers all nine scores, with ARI losses of 1.73, 0.61, and 1.61 pp, supporting separate recurrent histories. Removing gate entropy reduces ARI on all datasets and Q by 3.86, 1.47, and 1.74 pp. Its CellphoneCall NMI rises slightly (37.66% to 37.89%), alongside lower ARI and Q. Thus, the regulariser improves structural quality and partition agreement with a small NMI trade-off on that dataset.

CS improves all three Java scores. On CellphoneCall, enabling it raises Q by 1.20 pp but lowers NMI and ARI; Cora’s scores are unchanged. These differences reinforce the dataset-dependent CS choices identified by the capacity study.

4.9 Visualization Analysis

Fig. 6 compares raw Cora attributes with stable representations s^t using separately fitted t-SNE projections [37] (perplexity 30, seed 42, 750 iterations). The learned representations form more distinct topic-related groups. This seed-42 visualisation run includes a $t = 0$ forward pass with $P^0 = A^0$, $B^0 = D^0 = 0$. The main evaluation instead resets memories before transitions $1, \dots, 4$ and averages three seeds. Separately fitted panels do not share a temporal coordinate scale.

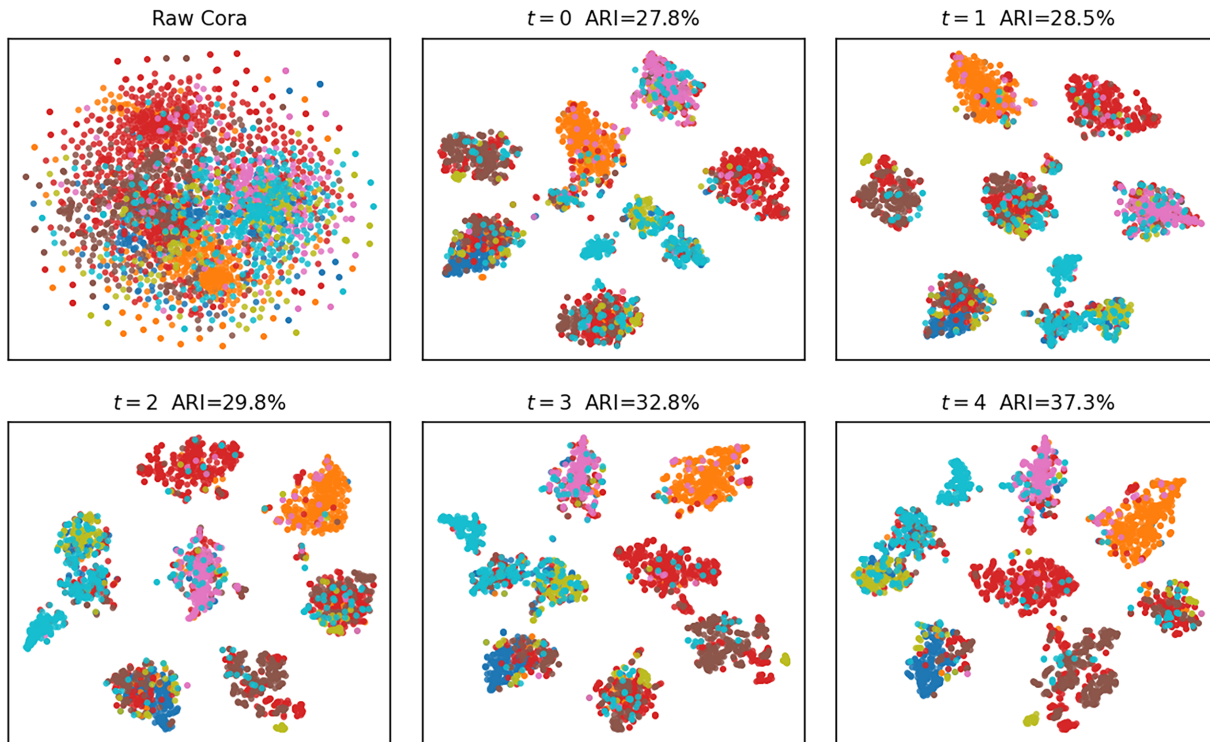


Figure 6: t-SNE projections of raw Cora attributes and RDMG stable states at $t = 0, \dots, 4$. Colours denote research topics; ARI labels report assignments from the visualisation run.

4.10 Computational Efficiency and Resource Usage

We compare RDMG, DGCN, and MSGCL on Java and Cora using the environment in Section 4.2 and three separate-process fits per configuration (seeds 42–44). Java represents a smaller, longer sequence with CS; Cora represents a larger attributed graph without CS. RDMG follows Tables 1 and 2; Java without CS isolates its cost at fixed width.

Official baseline implementations are used. DGCN uses row-normalised adjacency features, sparse propagation, and width 128; MSGCL ties structural feature and hidden widths to N . Both use dropout 0.5, Adam (10^{-3} learning rate, zero weight decay), and training-loss patience 30. DGCN/MSGCL allow 200/500 epochs per snapshot, retaining the stopping parameters. Baseline fits include snapshot 0; RDMG updates once per transition per epoch. Table 8 reports the observed optimiser-step range.

Training time covers a sequence fit, including initialisation and baseline history extraction, but excludes input preparation, transfer, and quality metrics. Peak allocated GPU memory includes resident snapshots and training tensors; CPU memory and the CUDA context are excluded. Parameter counts use all trainable weights of RDMG or the largest baseline snapshot model.

Table 8: Resource costs on Java and Cora. Times are mean $\pm$ sample SD over three fits; peak memory is the maximum across fits.

Method	Params (M)	Updates	Train (s)	Infer (ms)	Peak (MiB)
<i>Java</i> ($N = 376$, $T = 20$)					
DGCN	1.106	3160–3808	11.2 ± 1.1	51.8 ± 2.1	35.3
MSGCL	3.395	3333–3555	12.8 ± 0.2	154.4 ± 6.2	78.4
RDMG	2.829	1900	286.3 ± 78.9	121.5 ± 1.8	118.5
RDMG w/o CS	1.253	1900	258.2 ± 27.9	142.0 ± 20.6	98.6
<i>Cora</i> ($N = 2708$, $T = 5$)					
DGCN	45.752	874–928	6.0 ± 0.1	179.3 ± 3.7	430.6
MSGCL	176.009	1323–1550	220.6 ± 20.5	2355.2 ± 21.7	1948.6
RDMG	0.632	400	235.5 ± 1.0	731.5 ± 109.0	807.2

CUDA-synchronised inference covers transitions $1, \dots, T - 1$, assignment readout, and CPU output transfer. RDMG resets memories; baselines use fitted snapshot models and k -means with ten initialisations. After three warm-ups, the median of five sequence timings is divided by $T - 1$; baseline sequence latency sums snapshot latencies.

RDMG’s mean inference is $1.27\times$ and $3.22\times$ faster than MSGCL on Java and Cora. On Cora, it combines higher NMI, ARI, and Q with 58.6% lower peak training memory and 0.632 million parameters vs. 176.009 million. DGCN has the lowest training time, inference latency, and peak memory on both datasets. RDMG’s accuracy and inference gains accompany higher fitting costs, particularly on Java, where its peak memory also exceeds MSGCL’s.

MSGCL’s largest snapshot model contains $24N^2 + 4N + 2$ parameters. Increasing N from 376 to 2708 raises this count by $51.85\times$; dense transformations and N -dimensional clustering embeddings also enlarge computation and storage. This explains much of Cora’s greater resource demand despite fewer snapshots, with timings additionally affected by stopping and device utilisation.

RDMG’s $3Nh$ FP32 node states occupy 2.20 MiB on Java and 7.93 MiB on Cora. At Java’s $h = 512$, CS adds 40 KiB of community state and 1,575,936 shared GRU parameters. Removing CS lowers peak memory from 118.5 to 98.6 MiB without a consistent inference reduction across fits, quantifying its incremental cost alongside the accuracy gains in [Section 4.8](#).

4.11 Limitations and Future Work

The resource measurements in [Section 4.10](#) cover graphs up to 2708 nodes. Scaling to larger graphs is primarily constrained by the dense graph tensors and full-sequence GPU residency analysed in [Section 3.3](#). One dense $N \times N$ FP32 modularity matrix occupies $4N^2$ bytes: approximately 0.4 GB at $N = 10,000$, 10 GB at $N = 50,000$, and 40 GB at $N = 100,000$ (decimal GB). These analytic single-matrix sizes exclude adjacency and role matrices, pair indices, activations, resident snapshots, and optimiser state.

An exact reformulation could avoid explicitly storing the modularity matrix:

$$\text{tr}(S^\top MS) = \text{tr}(S^\top AS) - \frac{\|S^\top \mathbf{r}\|_2^2}{2m}. \quad (30)$$

With sparse A , this term requires sparse matrix products and a degree-vector reduction; the separate Gram penalty still costs $\mathcal{O}(NK^2)$. Sparse GCN propagation, pair sampling without dense enumeration, and

streamed snapshot loading would address the remaining costs without changing the one-step gradient policy. These are implementation directions; the reported experiments use the dense configuration.

RDMG targets undirected binary graphs with aligned node slots and a supplied community count K . Fixed-width, non-overlapping assignments favour applications with known community granularity; adaptive community counts and expanded node states would broaden this scope. Size-adaptive regularisation could accommodate strongly unequal partitions. Relative edge turnover also responds to random rewiring, and maximum-based normalisation couples node intensities. Attributes or higher-order temporal patterns could provide more selective event signals; extending role decomposition to motifs or temporal hyperedges could represent higher-order community changes. The evaluation concerns clustering on fitted sequences, with synthetic annotations measuring event responsiveness.

5 Conclusion

We proposed RDMG for dynamic community detection, separating persistent structure from edge births and deaths through role-specific encoders and recurrent event memories. A gated event residual updates community assignments, while optional community state provides temporal context at the cluster level.

Experiments showed competitive clustering on four real-world networks and consistently leading ARI and event-restricted ARI across three synthetic scenarios. Independent ablations supported the contributions of role-specific propagation, separate event histories, and residual assignment, with community-state benefits varying by dataset. Resource measurements also showed faster inference than MSGCL on Java and Cora. These results support role decomposition as an effective approach to balancing community continuity and event responsiveness in the evaluated dynamic graphs.

Acknowledgement: Not applicable.

Funding Statement: Publication of this article is supported by the Top Talent Training Project of Henan Province (Grant No. 244500510012).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Yuchen Liu and Hongchang Chen; methodology, Yuchen Liu; software, Yuchen Liu and Meng Du; validation, Yuchen Liu and Meng Du; formal analysis, Yuchen Liu; investigation, Yuchen Liu and Meng Du; data curation, Yuchen Liu and Meng Du; writing—original draft preparation, Yuchen Liu; writing—review and editing, Yuchen Liu, Hongchang Chen and Meng Du; visualization, Yuchen Liu and Meng Du; supervision, Hongchang Chen; project administration, Yuchen Liu. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The public datasets analyzed in this study are described in the article and are available from the sources cited therein. The implementation code and processed results are available from the Corresponding Author, Yuchen Liu, upon reasonable request.

Ethics Approval: Not applicable. This study does not involve human participants or animals.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the International Conference on Learning Representations (ICLR); 2017 April 24–26; Toulon, France.
2. Veličković P, Fedus W, Hamilton WL, Liò P, Bengio Y, Hjelm RD. Deep graph infomax. In: Proceedings of the International Conference on Learning Representations (ICLR); 2019 May 6–9; New Orleans, LA, USA.

3. Tsitsulin A, Palowitch J, Perozzi B, Müller E. Graph clustering with graph neural networks. *J Mach Learn Res.* 2023;24(127):1–21.
4. Rossetti G, Cazabet R. Community discovery in dynamic networks: a survey. *ACM Comput Surv.* 2018;51(2):1–37. doi:10.1145/3172867.
5. Saeed HS, Abbood AD. A comprehensive review of dynamic community detection: taxonomy, challenges, and future directions. *Comput Mater Contin.* 2025;85(3):4375–405. doi:10.32604/cmc.2025.067783.
6. Teng M, Gao C, Li X, Wang Z, Fan K, Nekorkin V. Multi-scale graph contrastive learning for community detection in dynamic graphs. *Inf Process Manag.* 2026;63(2):104410. doi:10.1016/j.ipm.2025.104410.
7. Gao C, Zhu J, Zhang F, Wang Z, Li X. A novel representation learning for dynamic graphs based on graph convolutional networks. *IEEE Trans Cybern.* 2023;53(6):3599–612. doi:10.1109/tcyb.2022.3159661.
8. Liu H, He L, Zhang F, Wang Z, Gao C. Dynamic community detection over evolving networks based on the optimized deep graph infomax. *Chaos Interdiscip J Nonlinear Sci.* 2022;32(5):053119. doi:10.1063/5.0086795.
9. Kong D, Zhang A, Li Y. Learning persistent community structures in dynamic networks via topological data analysis. In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence*; 2024 Feb 20–27; Vancouver, BC, Canada. p. 8617–26.
10. Ai Y, Xie X, Ma X. Graph contrastive learning for tracking dynamic communities in temporal networks. *IEEE Trans Emerg Top Comput Intell.* 2024;8(5):3422–35. doi:10.1109/tetci.2024.3386844.
11. Su X, Xue S, Liu F, Wu J, Yang J, Zhou C, et al. A comprehensive survey on community detection with deep learning. *IEEE Trans Neural Netw Learn Syst.* 2024;35(4):4682–702. doi:10.1109/tnnls.2021.3137396.
12. Liu Y, Peng Y, Chen H, Zhang J, Li S. BCDAN: a balanced method for community detection in attributed networks. *Neurocomputing.* 2025;644(2):130411. doi:10.1016/j.neucom.2025.130411.
13. Perozzi B, Al-Rfou R, Skiena S. DeepWalk: online learning of social representations. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; 2014 Aug 24–27; New York, NY, USA. p. 701–10.
14. Wang C, Pan S, Hu R, Long G, Jiang J, Zhang C. Attributed graph clustering: a deep attentional embedding approach. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*; 2019 Aug 10–16; Macao, China. p. 3670–6.
15. Bo D, Wang X, Shi C, Zhu M, Lu E, Cui P. Structural deep clustering network. In: *Proceedings of the Web Conference 2020*; 2020 Apr 20–24; Taipei, Taiwan. p. 1400–10.
16. Peng Z, Huang W, Luo M, Zheng Q, Rong Y, Xu T, et al. Graph representation learning via graphical mutual information maximization. In: *Proceedings of the Web Conference 2020*; 2020 Apr 20–24; Taipei, Taiwan. p. 259–70.
17. Kojaku S, Radicchi F, Ahn YY, Fortunato S. Network community detection via neural embeddings. *Nat Commun.* 2024;15(1):9446.
18. Cheng X, Zhu W, Yan WQ. Centrality-aware collaborative network embedding for overlapping community detection. *IEEE Trans Netw Sci Eng.* 2026;13(12):2236–50. doi:10.1109/tnse.2025.3611500.
19. Cheng X, Zhu W, Yan WQ. MrNEAD: community detection attacks in social networks using modularity-regularized network embedding via adversarial decomposition. *Inf Sci.* 2026;740(1):123225. doi:10.1016/j.ins.2026.123225.
20. Schlichtkrull M, Kipf TN, Bloem P, van den Berg R, Titov I, Welling M. Modeling relational data with graph convolutional networks. In: *The Semantic Web: 15th International Conference, ESWC 2018*; 2018 Jun 3–7; Heraklion, Greece. p. 593–607.
21. Kadkhoda Mohammadmosaferi K, Naderi H. Evolution of communities in dynamic social networks: an efficient map-based approach. *Expert Syst Appl.* 2020;147(4):113221. doi:10.1016/j.eswa.2020.113221.
22. Zhuang D, Chang JM, Li M. DynaMo: dynamic community detection by incrementally maximizing modularity. *IEEE Trans Knowl Data Eng.* 2021;33(5):1934–45. doi:10.1109/tkde.2019.2951419.
23. Liu F, Choi D, Xie L, Roeder K. Global spectral clustering in dynamic networks. *Proc Natl Acad Sci.* 2018;115(5):927–32. doi:10.1073/pnas.1718449115.

24. Li D, Lin Q, Ma X. Identification of dynamic community in temporal network via joint learning graph representation and nonnegative matrix factorization. *Neurocomputing*. 2021;435:77–90.
25. Li D, Kosugi S, Zhang Y, Okumura M, Xia F, Jiang R. Revisiting dynamic graph clustering via matrix factorization. In: *Proceedings of the ACM Web Conference 2025*; 2025 Apr 28–May 2; Sydney, NSW, Australia. p. 1342–52.
26. Yu W, Wu S, Shen S, Li H, Yang W, Li X, et al. GrSrNMF: dynamic community detection with graph and symmetry bi-regularized non-negative matrix factorization. *Sci Rep*. 2025;15(1):26427. doi:10.1038/s41598-025-09996-8.
27. Li X, Liu Y. Evo-NMTF: dynamic community detection via dual-path evolutionary nonnegative matrix tri-factorization. *Knowl Inf Syst*. 2026;68(1):87.
28. Manessi F, Rozza A, Manzo M. Dynamic graph convolutional networks. *Pattern Recognit*. 2020;97(1):107000. doi:10.1016/j.patcog.2019.107000.
29. Li X, Zhen X, Qi X, Han H, Zhang L, Han Z. Dynamic community detection based on graph convolutional networks and contrastive learning. *Chaos Solitons Fractals*. 2023;176(164):114157. doi:10.1016/j.chaos.2023.114157.
30. Shahgholi P, Bouyer A, Arasteh B, Kusetogullari H, Liu X. Dynamic community detection using enhanced GraphSAGE deep model with fast semi-supervised time-step label matching in social networks. *Appl Netw Sci*. 2026;11(1):46.
31. Wang R, Li L, Tao X, Wang P, Liu P. Contrastive and attentive graph learning for multi-view clustering. *Inf. Process Manag*. 2022;59(4):102967. doi:10.1016/j.ipm.2022.102967.
32. Qu D, Ma Y, Wang Y. STR-GNN: stability-regularized graph neural networks for suppressing spurious temporal variations in dynamic community detection. *Sci Rep*. 2026;16:25050.
33. Paoletti G, Gioacchini L, Mellia M, Vassio L, Almeida J. CoDÆN: benchmarks and comparison of evolutionary community detection algorithms for dynamic networks. *ACM Trans Web*. 2025;19(3):1–25. doi:10.1145/3718988.
34. Djurdjevac Conrad N, Tonello E, Zonker J, Siebert H. Detection of dynamic communities in temporal networks with sparse data. *Appl Netw Sci*. 2025;10(1):1. doi:10.1007/s41109-024-00687-3.
35. Li D, Zhang Y, Wu Y, Jiang R. Node role-guided LLMs for dynamic graph clustering. In: *Proceedings of the ACM Web Conference 2026*; 2026 Apr 13–17; Dubai, United Arab Emirates. p. 4418–28.
36. Demšar J. Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res*. 2006;7:1–30.
37. van der Maaten L, Hinton G. Visualizing data using t-SNE. *J Mach Learn Res*. 2008;9:2579–605.