DEMO: Heterogeneous Multilayer Density infused Entropy-Modularity Optimization for Unsupervised Social Event Detection

Anonymous ACL submission

Abstract

Event detection from social streams is an essential component of monitoring real-world incidents with applications in disaster monitoring, health surveillance and public opinion analysis, among others. Social media generates information streams containing heterogeneous attributes, such as names, places, and times, which often exhibit noise as the same entities may belong to different events, making detection challenging. The present paper introduces an unsupervised event detection model DEMO (Heterogeneous Multilayer **D**ensity infused **E**ntropy-**M**odularity **O**ptimization). DEMO judiciously optimize both entropy and modularity to deal with the noise arising from multiple heterogeneous interactions. This allows better classification of events from social streams. The method is aided by a community detection algorithm, mCOMM, which infuses heterogeneous multilayered density-based community participation information into the optimization pipeline. Extensive experiments support our model's superior performance, surpassing SOTA methodologies with a maximum gain of 110.7% in ARI, 20.2% in AMI and 19.7% in NMI for publicly available datasets.[1]

1 Introduction

Social events represent occurrences of real-world noteworthy happenings involving specific times, locations, people, and contexts. **Palisades Forest fire**[2] and **2021 Indian farmers' protest**[3], are recent examples. These real-world events triggered an overwhelming number of messages on social media platforms. Event detection identifies any such real-world activities by observing the patterns in the messages. Event detection can be used for disaster monitoring (Saini et al., 2024) and public opinion analysis (Mao et al., 2024) among others in the domain of healthcare (Paganelli et al., 2022), sentiment analysis (PETRESCU et al., 2024), enterprise risk management (Zhang et al., 2022) and political agenda fake news (Rajora et al., 2025).

Detecting social events presents unique challenges. Social platforms continuously generate a stream of messages, ensuing events to evolve with new developments and old events to be phased out. The method should adopt this dynamic environment. Messages contain heterogeneous elements or attributes such as locations, tagged users, organizations, dates, and times. Moreover, text with similar entities can belong to different events. These introduce noises that a text clustering model cannot effectively discern to distinguish events. The existing methods (Liu et al., 2020; Cao et al., 2021; Ren et al., 2024; Peng et al., 2023) require supervision. However, manual annotation of social messages is very costly, hence, an unsupervised event detection model addressing all the aforementioned challenges is a necessity. In this paper, we propose DEMO, an unsupervised heterogeneous multilayer density-infused entropy-modularity optimization event detection model. We construct a time-ordered sequence of heterogeneous multilayered blocks that retains recent events for handling evolving messages. We design a fast heterogeneous multilayer algorithm, mCOMM, for detecting communities using density from heterogeneous multilayer blocks. Further, DEMO uses simultaneous optimization of entropy and modularity to deal with the noise arising from multiple heterogeneous interactions. Entropy is known to identify the knowledge from noisy data, making it suitable to identify important interactions. Lower entropy indicates more homogeneous event groups, while higher entropy reflects greater disorder, signalling the need for further refinement or separation. This endows it with the requisite faculties to be used for identifying

[1] Code repository: https://anonymous.4open.science/r/DEMO-3B2F/README.md

[2] https://en.wikipedia.org/wiki/2023_Odisha_train_collision

[3] https://en.wikipedia.org/wiki/2020-2021_Indian_farmers%27_protest

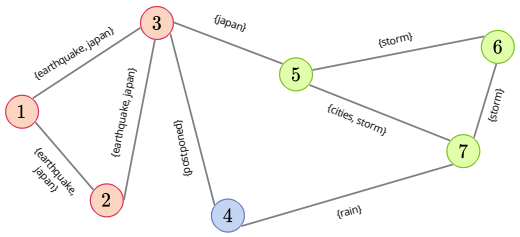


Figure 1: An example showing the importance of both modularity and entropy in event detection.

clusters that signify real-world events. Modularity evaluates the strength of division in networks by comparing the density of edges within an event to the expected density in a random distribution (Newman, 2006b). High modularity signifies well-defined event groups with strong internal connections and minimal overlap with other event groups. When used together, entropy and modularity provide complementary insights. For instance, Figure 1 illustrates that connecting tweets based on shared words can introduce noisy edges e.g., tweets 4 and 7 both mention "rain" but discuss different events. The colored nodes denote ground truth communities in the figure. Using only modularity or entropy can lead to suboptimal community assignments. However, jointly minimizing entropy and maximizing modularity helps distinguish such cases better separating weakly connected tweets like 4, while grouping strongly connected ones like 5, 6, and 7. Entropy manages intra-cluster diversity, while modularity ensures inter-cluster separation, resulting in more robust and scalable event detection models. Our method leverages community participation information from proposed heterogeneous multilayer community detection mCOMM for better feature generation. Our contributions are as follows

1. We create a time-ordered heterogeneous multilayered graph (HMG) from tweets with layers representing heterogeneous interactions.

2. We propose a novel density-based algorithm, namely, mCOMM, that finds communities in HMG.

3. We propose a GNN-based model with an entropy and modularity-driven objective function, leveraging cluster information from mCOMM for event detection.

4. We conduct extensive experiments on two publicly available datasets consisting of two languages with a detailed ablation study.

2 Heterogeneous Multilayer Streaming Community Detection (mCOMM)

Inspired by Gupta and Kundu (2025), we propose a community detection algorithm for heterogeneous multilayer graph streams. Please note that all notations used hereafter are listed in Table 2 in the Appendix.

Heterogeneous Multilayer Graph (HMG):(Chatterjee et al., 2024) An HMG is defined as $H = (V, E, E_T, L, \{R_{EE_T}, R_{E_TL}, R_{VL}\})$ where V, $E \subset V \times V \times L$, E_T, and L is the set of nodes, edges, edge types and layers respectively. Here, each layer corresponds to one edge type. The functions are defined as $R_{EE_T} : E \rightarrow E_T$, $R_{E_TL} : E_T \rightarrow L$ and $R_{VL} : V \rightarrow 2^L \setminus \phi$.

Formally, the task of a streaming community detection for a heterogeneous multilayer graph stream H is to mine a set of communities $\{c_1, \ldots, c_j\}$ based on its heterogeneous interactions without storing it in memory. Before presenting the community detection algorithm, let us define:

Density in a HMG: Given two communities c_1, c_2, with nodes n_1, n_2 and edges m_1, m_2 spread across L layers with m_{12} edges connecting nodes from c_1 to c_2, inner density (ρ_{in}) is the sum of fractions of number of edges of a community to the number of possible edges within that community across all layers. On the other hand, outer density (ρ_{out}) is the sum of fractions of the number of outer edges connecting two communities to the number of possible outer edges connecting two communities across all layers. Mathematically,

$$\rho_{in}(c_1) = \frac{1}{|L|} * \sum_{l \in L, w} w_l * \frac{2 \times m_{1_l}}{n_{1_l}.(n_{1_l} - 1)}$$
$$\rho_{out}(c_1, c_2) = \frac{1}{|L|} * \sum_{l \in L, w} w_l * \frac{m_{12_l}}{n_{1_l}.n_{2_l}} \quad (1)$$

Here, $w_1, w_2, \ldots, w_k$ represent constant values assigned according to the weightage of each layer. The sum of all weights must be equal to 1. The

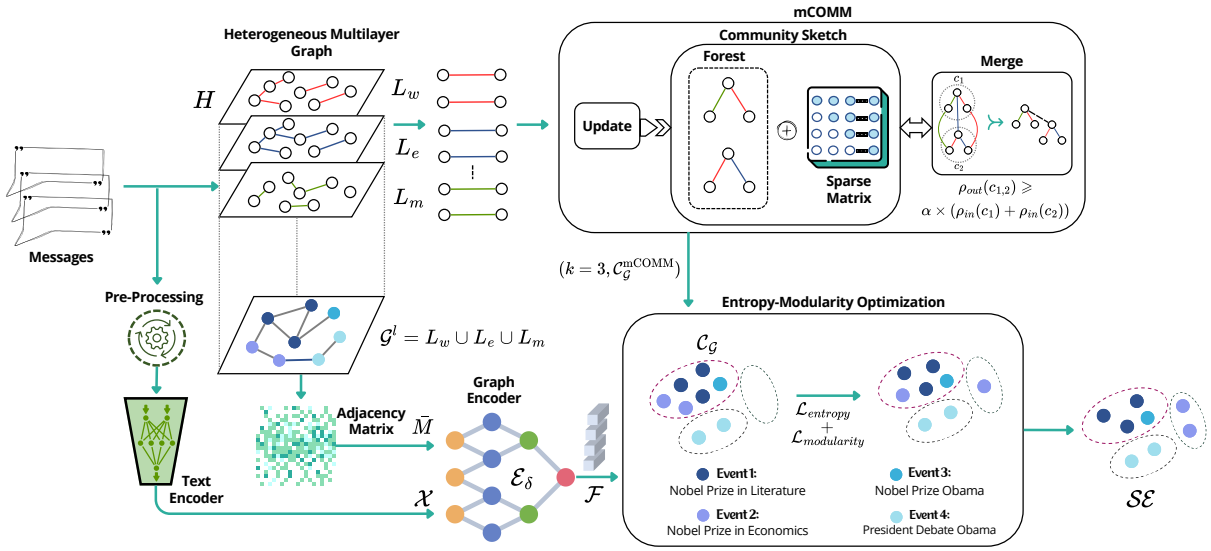


Figure 2: Block diagram showing the methodology of DEMO. Here, a heterogeneous multilayer graph H is constructed where L_w, L_e, L_m represent layers with each showing interactions between nodes based on common words, entities, and mentions, respectively. This H is passed to mCOMM to determine the community participation information (number of clusters, k and cluster assignment matrix, $\mathcal{C}_G^{\text{mCOMM}}$). Then, all layers of H are projected to $\mathcal{G}^l$, and the messages are encoded using a text encoder represented as $\mathcal{X}$. $\mathcal{G}^l$, learnable matrix $\bar{M}$ and $\mathcal{X}$ are then passed to the graph encoder $\mathcal{E}_\delta$, which optimizes cluster assignments through entropy and modularity using information from mCOMM.

terms $m_{j_{l_i}}$ and $n_{j_{l_i}}$ represent the number of edges and nodes belonging to the layer l_i in the community c_j.

2.1 Algorithm

We initialize a community sketch $\mathcal{C}_k$ using the *makeSketch()* function. It observes an HMG stream and any new edge $e(u, v, l)$ triggers *onReceive()* function. The sketch is first updated by *updateSketch()* function to ensure the nodes are added to appropriate communities before any community merger take place by *mergeCommunity()* function. The communities are merged when the outer density of the communities becomes larger or equal than the α times the cumulative inner density of the communities nodes u and v are added with. The function *onQuery()* returns all communities observed in the stream up to the time of the query. The detailed algorithm is shown in Algorithm 1. Kindly note that our algorithm considers all nodes to be in all layers.

mCOMM is designed to extract communities solely based on the graph structure, without considering node features. In the next Section, we identify event partitions by incorporating both node features and the graph structure, with mCOMM playing a crucial role in providing community information. A detailed analysis of mCOMM can be found in the ablation study and the Appendix.

3 Entropy and Modularity based Unsupervised Event Detection

First, we define a social message, a social message stream, an event, and the problem of event detection as follows:

Social message: A social message s is a set $\{text, users, stamp\}$. Here $text$, $users$, $stamp$ denote the text message, the sender of the message with the mentioned users and the timestamp of the message.

Social Message Stream: A social message stream (SS) is a time ordered sequence of message blocks $B_1, \ldots, B_t, B_{t+1}, \ldots$, where block B_t is a collection of social messages that arrive within the time interval $[t, t + 1)$.

Event: An event is a set of social messages that are about the same real-world phenomenon. A social message can only belong to one event.

Event Detection: Given a social message block B_t, the objective of event detection is to learn a model $f_\theta : B_t \rightarrow \mathcal{SE}_t$ where θ is the model parameter. Here $\mathcal{SE}_t$ is the set of all events in a block.

3.1 Methodology

We convert a block B_t to an HMG H_t using the definition in Section 2. A node in H_t represents

Algorithm 1 Streaming Community Detection for HMG

Input: **A HMG edge stream, merge threshold** α

// Intialize Community Sketch
CommunitySketch $\mathcal{C}_k$ = makeSketch()

// Receive Function
def onReceive($e \in E$, α):
  $\mathcal{C}_k$.updateSketch($e = (u, v, l)$)
  c_u = $\mathcal{C}_k$.community(u)
  c_v = $\mathcal{C}_k$.community(v)

  Calculate outer density $\rho_{out}(c_u, c_v)$ and inner density $\rho_{in}(c_u)$ & $\rho_{in}(c_v)$ of c_u and c_v using Eq. 1.

  if $\rho_{out}(c_u, c_v) \geqslant \alpha \times (\rho_{in}(c_u) + \rho_{in}(c_v))$ **then**
    $\mathcal{C}_k$.mergeCommunity(c_u, c_v)
  end if

// Query Function
def onQuery():
  return $\mathcal{C}_k.f$

def community($node$):
  while $node$!= $node$.parent:
    $node$.parent, $node$:= $node$.parent.parent, $node$.parent
  return $node$

def makeSketch($node$):
  $\mathcal{C}_k$ = forest f, sparseMatrix mat
  return $\mathcal{C}_k$

def updateSketch($e = (node_1, node_2, l)$):
  n_1, n_2 = $\mathcal{C}_k$.community($node_1$), $\mathcal{C}_k$.community($node_2$)
  if (n_1 && n_2): {#when both nodes exist}
    $\mathcal{C}_k$.mat(n_1,n_2,l).e += 1
  elif (!n_1 && !n_2): {# when both nodes do not exist}
    n = n_1 if $node_1 < node_2$ else n_2
    n_1, n_2 = $\mathcal{C}_k.f$.add($node_1$), $\mathcal{C}_k.f$.add($node_2$)
    n_1.parent, n_2.parent = n
    $\mathcal{C}_k$.mat(n,n,·).n, $\mathcal{C}_k$.mat(n,n,l).e= 2, 1
  else: {# when one of them exists}
    n, n_e = n_1, n_2 if !n_1 else n_2, n_1
    n = $\mathcal{C}_k.f$.add(n), n.parent = n
    $\mathcal{C}_k$.mat(n,n,·).n, $\mathcal{C}_k$.mat(n,n,l).e= 1, 0
    $\mathcal{C}_k$.mat(n_e,n,l).e = 1

def mergeCommunity(P_{n_1},P_{n_2}):
  P_x, P_y = P_{n_1}, P_{n_2} if $\mathcal{C}_k$.mat(P_{n_1},P_{n_1},l).n > $\mathcal{C}_k$.mat(P_{n_2},P_{n_2},l).n else P_{n_2}, P_{n_1}
  for each l in L:
    $\mathcal{C}_k$.mat(P_x,P_x,l).n += $\mathcal{C}_k$.mat(P_y,P_y,l).n
    $\mathcal{C}_k$.mat(P_x,P_x,l).e += $\mathcal{C}_k$.mat(P_y,P_y,l).e + $\mathcal{C}_k$.mat(P_x,P_y,l).e
  $\mathcal{C}_k$.mat.remove(P_y), P_y.parent = P_x
  $\mathcal{C}_k$.mat.replaceRef(P_y) ← P_x

a social message s and is present in all the layers. Two nodes in a layer can either be connected based on common entities, words or mentions. This H_t is passed to mCOMM in order to get the number of communities used later. Further, we generated a projection graph where nodes are connected if they have connections in any of the layers in H_t. We represent this as $\mathcal{G}_t^l(V_t, E_t^l)$ and use it for our method described next. A visual representation of the methodology is shown in Figure 2.

A simplified definition (Liu et al., 2019) of entropy ($\Diamond$) for a given graph $\mathcal{G}_t^l$ with partitions $\{c_1, \ldots, c_j\}$ is

$$\Diamond = \sum_{c_i} \frac{intra_i}{2|E_t^l|} log_2 \frac{intra_i + inter_i}{2|E_t^l|} \tag{2}$$

Here $intra_i$ and $inter_i$ refers to the edges with both the endpoints and either endpoint in partition c_i respectively. Minimizing entropy helps in identifying homogeneous groups in a graph.

The definition of entropy mandates partition information. So our first objective is to design an encoder that finds a partition assignment matrix for $\mathcal{G}_t^l$ with node features $\mathcal{X}$ as,

$$\mathcal{E}_\delta(\mathcal{G}_t^l, \mathcal{X}) \rightarrow \{P_1, P_2, P_3, ..., P_k\} \in \mathcal{C}_\mathcal{G} \tag{3}$$

Given that input matrix contains k partitions the assignment matrix is of $|V_t| * k$ dimension where $(\mathcal{C}_\mathcal{G})_{ij} = 1$ when node i belongs to partition P_j. Each row of the matrix is a one-hot encoded vector, and we consider disjoint node partitioning. In our solution we use a GCN encoder (Kipf and Welling, 2016) with residual skip connections and SeLU nonlinearity denoted by $\mathcal{S}_\mathcal{U}$ (Eq. 4). θ_* are learnable weight matrices, D is diagonal degree matrix and M is the undirected adjacency matrix.

$$\begin{aligned} \mathcal{F} &= \mathcal{S}_\mathcal{U}(\bar{M} F_1^{\ell-1} \theta_1 + F_1^{\ell-1} \theta_2), \\ F_1^0 &= \mathcal{X}, \bar{M} = D^{\frac{-1}{2}} M D^{\frac{-1}{2}} + \dot{M} \end{aligned} \tag{4}$$

We add the normalized adjacency matrix $\bar{M}$ with a learnable adjacency matrix $\dot{M}$ and use it as input for the GCN. This addresses the randomness present in real word graphs (Jin et al., 2020) thus helping in event separation. Now with node features $\mathcal{F}$, θ_3 as a learnable matrix, cluster assignment matrix from mCOMM ($\mathcal{C}_\mathcal{G}^{\text{mCOMM}}$), and $\mathcal{S}_\sigma$ as softmax, we construct cluster assignment matrix $\mathcal{C}_\mathcal{G}$ as

$$\mathcal{C}_\mathcal{G} = \mathcal{S}_\sigma(\mathcal{F}.\theta_3 + \mathcal{C}_\mathcal{G}^{\text{mCOMM}}) \tag{5}$$

Finally, after getting $\mathcal{C}_\mathcal{G}$, we can write the entropy as a loss function (using the earlier definition) with $\mathcal{T}$ representing matrix trace and $\odot$ as Hadamard product as

$$L_\Diamond = \mathcal{T}\Big(\frac{\mathcal{C}_\mathcal{G}^T \bar{M} \mathcal{C}_\mathcal{G}}{\sum_{i=1}^{|V_t|} \sum_{j=1}^{|V_t|} \bar{M}_{ij}} \odot \log_2 \Big(\frac{\mathbb{1}_{k*|V_t|} \bar{M} \mathcal{C}_\mathcal{G}}{\sum_{i=1}^{|V_t|} \sum_{j=1}^{|V_t|} \bar{M}_{ij}} \Big) \Big) \tag{6}$$

We use $L_\Diamond$ instead of $\Diamond$ to represent entropy loss. To further increase the feature similarity within the same event group and separability among event groups, we use the DBI index (Davies and Bouldin, 1979) defined as

$$L_I = \frac{1}{k}\sum_{i<k} \mathcal{M}ax_{i\neq j}\left(\frac{\xi_i + \xi_j}{\zeta_{ij}}\right) \tag{7}$$

ξ_i is the mean distance between the feature vector of each member in partition P_i from the mean feature vector of all the nodes in the partition ($\bar{\mathcal{F}}_i$) and ζ_{ij} is the difference between the mean embeddings of cluster P_i and P_j.

$$\bar{\mathcal{F}}_i = \frac{\sum_{\mathcal{C}_{\mathcal{G}_{pi}}=1}\mathcal{F}_p}{|P_i|}, \xi_i = \left(\frac{1}{|P_i|}\sum_{\mathcal{C}_{\mathcal{G}_{pi}}=1}|\mathcal{F}_p - \mathcal{F}_i|^2\right)^{\frac{1}{2}}$$
$$\zeta_{ij} = \left(|\bar{\mathcal{F}}_i - \bar{\mathcal{F}}_j|^2\right)^{\frac{1}{2}} \tag{8}$$

For the robustness of the generated features, we use $\mathcal{F}$ to predict the adjacency matrix of the input graph $\mathcal{G}_t^l$ and include this in our final entropy loss as

$$\mathcal{L}_\Diamond(\mathcal{G}_t^l, \mathcal{F}, \mathcal{C}_\mathcal{G}, M, \bar{M}) = L_\Diamond + L_I + ||\mathcal{S}_e(\mathcal{F}.\mathcal{F}^T) - M||_F^2 \tag{9}$$

Here $\mathcal{S}_e$ represents sigmoid activation. Now, that we have defined the loss function for minimizing the entropy, let us design a loss function for maximizing modularity. Modularity maximization improves clustering by optimizing the division of a network into communities that have dense internal connections and sparse external ones. We know that modularity optimization is an NP-hard problem (Brandes et al., 2008), and it is non-trivial to incorporate it as an optimization objective for our model. We have used the spectral definition of (Newman, 2006a) where the definition of modularity loss can be converted to a series of matrix operations as

$$L_Q = \frac{1}{2|E_t^l|}\mathcal{T}(\mathcal{C}_\mathcal{G}^T \mathcal{B}\mathcal{C}_\mathcal{G}) \tag{10}$$

where $\mathcal{B} = M - \frac{\mathcal{D}\mathcal{D}^T}{2|E_t^l|}$ represents the modularity matrix; $\mathcal{D}$ is the degree vector that contains the degree of every node identified by its index. As modularity also requires partition information we use the partition assignment matrix from Eq. 5. Thus,

$$\mathcal{T}(\mathcal{C}_\mathcal{G}^T \mathcal{B}\mathcal{C}_\mathcal{G}) \approx \mathcal{T}(\mathcal{C}_\mathcal{G}^T M\mathcal{C}_\mathcal{G} - \mathcal{C}_\mathcal{G}^T \mathcal{D}\mathcal{D}^T \mathcal{C}_\mathcal{G}) \tag{11}$$

A trivial solution to the problem is to assign all nodes in a single cluster and must be avoided. There are methods (Bansal et al., 2018) that use an orthogonality regularizer, however, these are seen to dominate the modularity objective function which is again not desirable (Tsitsulin et al., 2024). We use the collapse regularizer from (Tsitsulin et al., 2024) to avoid the trivial solution as

$$\mathcal{L}_Q(\mathcal{C}_\mathcal{G}, M) = -L_Q + \frac{\sqrt{k}}{|V_k|}\|\sum_i (\mathcal{C}_\mathcal{G})_i^T\| \tag{12}$$

Here, the regularizer adds a penalty of $\sqrt{k}$ if all nodes are assigned to a single cluster. Now that we have defined the modularity and entropy loss separately, we combine both of these objective functions together as

$$\mathcal{L} = \beta_1 \mathcal{L}_\Diamond + \beta_2 \mathcal{L}_Q \tag{13}$$

Here all the learnable parameters $\theta_* \in \theta$. We train the encoder $\mathcal{E}_\theta$ and update all model parameters based on $\mathcal{L}$. The final events $\mathcal{SE}_t$ are extracted from trained $\mathcal{E}_\theta$. Note that variable k used in different equations of the loss function is an hyperparameter and is extracted along with matrix $\mathcal{C}_\mathcal{G}^{\text{mCOMM}}$ from *onQuery()* function in proposed mCOMM.

4 Experiments and Results

Data Sets: Our experiments leverage two publicly available large-scale datasets Event2012 (English) and Event2018 (French) (Peng et al., 2023), designed to evaluate streaming social event detection. Event2012 comprises $68,841$ manually labeled tweets spanning 503 event classes over 29 days. Event2018 with $64,516$ labeled tweets across 257 event classes, covering 23 days, is used for cross-lingual experiments. Three relations common entities, common words, and user mentions are used as the schema for constructing a heterogeneous multilayer graph. The evolving social messages are split into blocks by date, according to the paper (Peng et al., 2023). We evaluate our model using a set of metrics, including ARI (Adjusted Rand Index), Normalized Mutual Information (NMI), and Adjusted Mutual Information (AMI).

Baselines: We evaluate our model with various other models specifically used for message representation learning, and similarity measuring. The baselines for comparison are **BERT** (Devlin et al., 2019), **BiLSTM** (Graves and Schmidhuber, 2005), **EventX** (Liu et al., 2020), **KPGNN** (Cao et al.,

Table 1: Event Detection performance for various methods in terms of ARI across different blocks.

Block	BERT	BiLSTM	EventX	KPGNN	KPGNN$_t$	FinEvent$_k$	CLKD	HISEvent	HyperSED	**DEMO**
					Event 2012					
B_1	$0.03 \pm .00$	$0.15 \pm .00$	$0.02 \pm .00$	$0.02 \pm .01$	$0.03 \pm .02$	$0.05 \pm .02$	$0.04 \pm .00$	$0.10 \pm .00$	$0.25 \pm .01$	$\mathbf{0.43 \pm .03}$
B_2	$0.24 \pm .00$	$0.02 \pm .00$	$0.00 \pm .01$	$0.60 \pm .02$	$0.65 \pm .03$	$0.68 \pm .01$	$0.69 \pm .01$	$0.69 \pm .00$	$0.08 \pm .02$	$\mathbf{0.81 \pm .03}$
B_3	$0.11 \pm .00$	$0.08 \pm .00$	$0.42 \pm .00$	$0.44 \pm .03$	$0.50 \pm .01$	$0.52 \pm .00$	$0.56 \pm .02$	$\mathbf{0.88 \pm .00}$	$0.07 \pm .02$	$0.69 \pm .02$
B_4	$0.02 \pm .00$	$0.01 \pm .00$	$0.04 \pm .00$	$0.22 \pm .02$	$0.24 \pm .02$	$0.27 \pm .01$	$0.22 \pm .01$	$\mathbf{0.45 \pm .00}$	$0.16 \pm .01$	$0.40 \pm .04$
B_5	$0.02 \pm .00$	$0.06 \pm .00$	$0.03 \pm .00$	$0.32 \pm .01$	$0.36 \pm .02$	$0.49 \pm .00$	$0.32 \pm .03$	$\mathbf{0.73 \pm .00}$	$0.11 \pm .02$	$0.57 \pm .02$
B_6	$0.02 \pm .00$	$0.02 \pm .00$	$0.09 \pm .00$	$0.46 \pm .01$	$0.61 \pm .04$	$0.51 \pm .01$	$0.62 \pm .01$	$0.60 \pm .00$	$0.26 \pm .03$	$\mathbf{0.74 \pm .02}$
B_7	$-0.02 \pm .00$	$0.14 \pm .00$	$0.10 \pm .01$	$0.06 \pm .01$	$0.06 \pm .02$	$0.08 \pm .01$	$0.07 \pm .01$	$0.28 \pm .00$	$0.18 \pm .02$	$\mathbf{0.59 \pm .02}$
B_8	$0.09 \pm .00$	$0.02 \pm .00$	$0.00 \pm .00$	$0.46 \pm .04$	$0.50 \pm .01$	$0.56 \pm .02$	$0.46 \pm .01$	$0.40 \pm .00$	$0.14 \pm .03$	$\mathbf{0.68 \pm .03}$
B_9	$0.01 \pm .00$	$0.18 \pm .00$	$0.09 \pm .01$	$0.28 \pm .02$	$0.30 \pm .03$	$0.39 \pm .01$	$0.59 \pm .01$	$0.70 \pm .00$	$0.16 \pm .02$	$\mathbf{0.73 \pm .05}$
B_{10}	$0.04 \pm .00$	$0.01 \pm .00$	$0.06 \pm .00$	$0.57 \pm .02$	$0.50 \pm .01$	$0.56 \pm .00$	$0.52 \pm .02$	$0.67 \pm .00$	$0.21 \pm .02$	$\mathbf{0.83 \pm .01}$
B_{11}	$-0.02 \pm .00$	$0.04 \pm .00$	$0.07 \pm .00$	$0.43 \pm .01$	$0.37 \pm .02$	$0.34 \pm .01$	$0.36 \pm .01$	$0.60 \pm .00$	$0.28 \pm .02$	$\mathbf{0.84 \pm .02}$
B_{12}	$0.10 \pm .00$	$0.03 \pm .00$	$0.04 \pm .00$	$0.26 \pm .01$	$0.32 \pm .03$	$0.38 \pm .01$	$0.34 \pm .01$	$\mathbf{0.80 \pm .00}$	$0.03 \pm .02$	$0.57 \pm .02$
B_{13}	$0.01 \pm .00$	$0.04 \pm .00$	$0.03 \pm .00$	$0.30 \pm .02$	$0.22 \pm .02$	$0.17 \pm .00$	$0.33 \pm .00$	$0.52 \pm .00$	$0.30 \pm .01$	$\mathbf{0.75 \pm .02}$
B_{14}	$0.08 \pm .00$	$0.02 \pm .00$	$0.01 \pm .00$	$0.22 \pm .02$	$0.21 \pm .01$	$0.35 \pm .01$	$0.22 \pm .01$	$\mathbf{0.75 \pm .00}$	$0.07 \pm .03$	$0.64 \pm .02$
B_{15}	$0.01 \pm .00$	$0.03 \pm .00$	$0.03 \pm .00$	$0.10 \pm .01$	$0.07 \pm .03$	$0.16 \pm .01$	$0.41 \pm .02$	$0.19 \pm .00$	$0.31 \pm .02$	$\mathbf{0.64 \pm .03}$
B_{16}	$0.00 \pm .00$	$0.03 \pm .00$	$0.01 \pm .00$	$0.44 \pm .01$	$0.45 \pm .01$	$0.48 \pm .00$	$0.57 \pm .02$	$0.75 \pm .00$	$0.22 \pm .03$	$\mathbf{0.89 \pm .02}$
B_{17}	$0.00 \pm .00$	$0.03 \pm .00$	$0.04 \pm .01$	$0.31 \pm .02$	$0.31 \pm .02$	$0.32 \pm .02$	$0.36 \pm .02$	$0.61 \pm .00$	$0.10 \pm .02$	$\mathbf{0.78 \pm .02}$
B_{18}	$0.01 \pm .00$	$0.03 \pm .00$	$0.02 \pm .00$	$0.20 \pm .03$	$0.22 \pm .01$	$0.35 \pm .01$	$0.38 \pm .01$	$0.72 \pm .00$	$0.07 \pm .02$	$0.54 \pm .02$
B_{19}	$0.01 \pm .00$	$0.02 \pm .00$	$0.03 \pm .00$	$0.26 \pm .02$	$0.24 \pm .01$	$0.48 \pm .01$	$0.35 \pm .02$	$0.62 \pm .00$	$0.14 \pm .02$	$\mathbf{0.79 \pm .02}$
B_{20}	$0.02 \pm .00$	$0.02 \pm .00$	$0.03 \pm .00$	$0.37 \pm .02$	$0.34 \pm .03$	$0.40 \pm .00$	$0.34 \pm .02$	$\mathbf{0.52 \pm .00}$	$0.25 \pm .02$	$0.51 \pm .01$
B_{21}	$0.03 \pm .00$	$0.03 \pm .00$	$0.04 \pm .00$	$0.11 \pm .01$	$0.10 \pm .02$	$0.22 \pm .02$	$0.37 \pm .01$	$0.33 \pm .00$	$0.09 \pm .03$	$\mathbf{0.38 \pm .01}$
					Event 2018					
B_1	0.01 ± 0.00	0.02 ± 0.01	0.01 ± 0.00	0.28 ± 0.02	0.28 ± 0.02	0.33 ± 0.01	0.29 ± 0.04	$0.55 \pm .00$	0.02 ± 0.02	$\mathbf{0.82 \pm 0.00}$
B_2	0.04 ± 0.00	0.03 ± 0.00	0.00 ± 0.00	0.31 ± 0.01	0.30 ± 0.03	0.34 ± 0.04	0.33 ± 0.03	$0.61 \pm .00$	0.04 ± 0.01	$\mathbf{0.74 \pm 0.04}$
B_3	0.02 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.35 ± 0.01	0.34 ± 0.01	0.37 ± 0.01	0.49 ± 0.02	$0.49 \pm .00$	0.04 ± 0.03	$\mathbf{0.79 \pm 0.02}$
B_4	0.03 ± 0.00	0.01 ± 0.01	0.01 ± 0.00	0.29 ± 0.02	0.43 ± 0.02	0.23 ± 0.02	0.29 ± 0.03	$0.47 \pm .00$	0.06 ± 0.01	$\mathbf{0.69 \pm 0.01}$
B_5	0.08 ± 0.00	0.02 ± 0.00	0.01 ± 0.01	0.37 ± 0.02	0.30 ± 0.01	0.34 ± 0.01	0.38 ± 0.01	$0.51 \pm .00$	0.04 ± 0.02	$\mathbf{0.58 \pm 0.04}$
B_6	0.03 ± 0.00	0.02 ± 0.01	0.03 ± 0.00	0.17 ± 0.02	0.20 ± 0.03	0.18 ± 0.00	0.40 ± 0.03	$0.61 \pm .00$	0.04 ± 0.03	$\mathbf{0.80 \pm 0.04}$
B_7	0.05 ± 0.00	0.08 ± 0.01	0.01 ± 0.01	0.29 ± 0.02	0.22 ± 0.03	0.23 ± 0.01	0.35 ± 0.01	$0.62 \pm .00$	0.00 ± 0.00	$\mathbf{0.81 \pm 0.03}$
B_8	0.05 ± 0.00	0.10 ± 0.00	0.01 ± 0.01	0.22 ± 0.02	0.22 ± 0.01	0.32 ± 0.03	0.38 ± 0.03	$\mathbf{0.79 \pm .00}$	0.04 ± 0.01	0.58 ± 0.02
B_9	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.01	0.22 ± 0.02	0.12 ± 0.04	0.18 ± 0.01	0.27 ± 0.02	$0.43 \pm .00$	0.08 ± 0.02	$\mathbf{0.61 \pm 0.00}$
B_{10}	0.07 ± 0.00	0.01 ± 0.01	0.01 ± 0.00	0.18 ± 0.01	0.19 ± 0.01	0.27 ± 0.01	0.40 ± 0.04	$0.53 \pm .00$	0.07 ± 0.01	$\mathbf{0.55 \pm 0.01}$
B_{11}	0.06 ± 0.00	0.06 ± 0.00	0.02 ± 0.01	0.16 ± 0.02	0.23 ± 0.02	0.18 ± 0.03	0.25 ± 0.01	$0.56 \pm .00$	0.07 ± 0.03	$\mathbf{0.61 \pm 0.03}$
B_{12}	0.08 ± 0.00	0.02 ± 0.00	0.01 ± 0.01	0.26 ± 0.03	0.23 ± 0.04	0.28 ± 0.02	0.57 ± 0.02	$\mathbf{0.77 \pm .00}$	0.01 ± 0.02	0.71 ± 0.02
B_{13}	0.02 ± 0.00	0.05 ± 0.00	0.02 ± 0.00	0.17 ± 0.01	0.22 ± 0.01	0.20 ± 0.03	0.37 ± 0.03	$0.74 \pm .00$	0.03 ± 0.01	$\mathbf{0.75 \pm 0.01}$
B_{14}	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.16 ± 0.02	0.33 ± 0.02	0.31 ± 0.04	0.54 ± 0.02	$0.78 \pm .00$	0.03 ± 0.02	$\mathbf{0.79 \pm 0.02}$
B_{15}	0.01 ± 0.00	0.02 ± 0.01	0.03 ± 0.00	0.24 ± 0.04	0.28 ± 0.03	0.36 ± 0.02	0.54 ± 0.03	$0.69 \pm .00$	0.02 ± 0.03	$\mathbf{0.74 \pm 0.00}$

2021), **FinEvent** (Peng et al., 2023), and **CLKD** (Ren et al., 2024). For more details on the dataset, metrics, and reproducibility of the baselines and our model we request the reader to refer to the Appendix.

Results: We show the results of our experiments in Tables 1, 3 and 4 for the metrics ARI, NMI and AMI respectively (Tables 3 and 4 are present in the Appendix). The standard deviations are reported after running each model 5 times. It is evident from the tables that our method outperforms other models in 14 out of 21 blocks for the Event2012 dataset and in 13 out of 15 blocks for the Event2018 dataset. We have an average improvement of 15.48% when compared to the second-best results for Event2012. We see an average improvement of 18% when compared to the second-best results for the Event2018 dataset. For NMI, we see an improvement in 8 blocks for the Event2012 dataset with a mean increase of 6% in these blocks. For the Event2018 dataset we are 14% behind the highest result on average. Now, moving to AMI, we see an improvement in 6 blocks with an average increase of 7% for the Event2012 dataset. For the Event2018 dataset we are 16% behind on average from the highest result.

Insights: We are underperforming in NMI due to the presence of inherent class imbalance in the datasets, especially in the Event2018 dataset. In the case of class imbalance, supervised models and models minimizing higher order entropy (like HISEvent) are shown to perform better (Das et al., 2022; Cao et al., 2024). It must be noted that we have incorporated a simpler 1-D entropy for our loss function, whereas HISEvent uses 2-D entropy, which is better at capturing higher-order structures, thus enhancing cluster separation. In spite of a simpler version of entropy, we have outperformed HISEvent in many Blocks for various metrics across the datasets. On further inspection, we found that we obtain a lower AMI and NMI as our method cannot always match the ground truth clusters with the actual number of clusters, and AMI is sensitive to the number of clusters. The number of clusters is provided by mCOMM in our method, which does not take higher-order entropy into account. As we have already mentioned, higher order entropy is already a better representative of the clusters, which is empirically found to be true, especially in the case of the Event2018 dataset, thus explaining the better results of HISEvent.

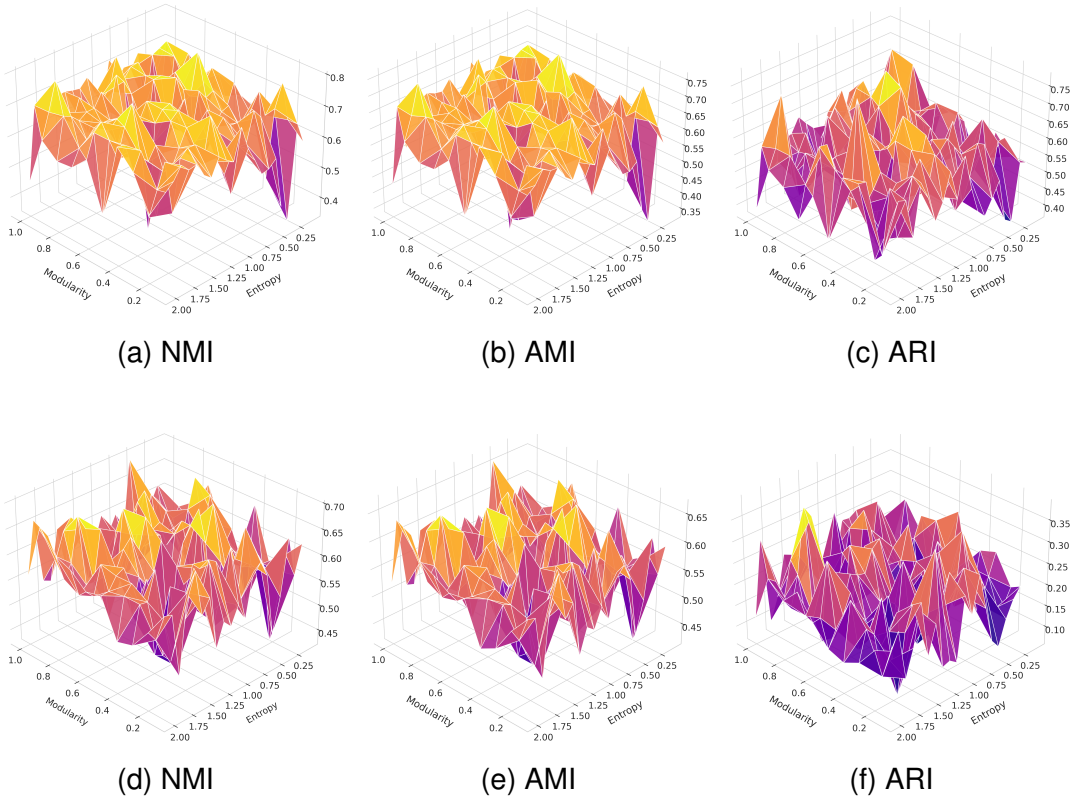


Figure 3: The above figures indicate the change in NMI, AMI and ARI for a change in entropy and modularity in the loss. Rows 1 and 2 are for Blocks B_2 and B_4 respectively.

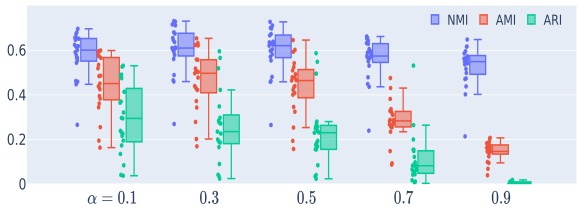


Figure 4: mCOMM with different parameter values of merge threshold (α)

4.1 Ablation Study

Modularity vs Entropy: We vary and adjust the weights assigned to modularity and entropy in the loss function, observing the resulting impact on ARI. This is done by changing the parameters β_1 and β_2 of Eq. 13. By analyzing the interplay between modularity, we aim to understand their individual and combined effects on the model's performance. The results for two random blocks of Event2012 are shown in Figure 3. We can determine that row 1 benefits from a high modularity with low entropy whereas the row 2 benefits from high entropy even in scenarios where the modularity is comparatively lower. This shows the advantages of both entropy and modularity in certain conditions.

Analysis of mCOMM with varying α: We conduct an analysis by varying the parameter α and evaluating its impact on the performance of **mCOMM** for event detection using the **Event2012** dataset across all blocks, as illustrated in Figure 4. From the figure, it is evident that a lower value of α leads to improved performance across all evaluation metrics. However, as α increases, a gradual decline in performance is observed. This suggests that lower values of α contribute to a more effective identification of events, whereas higher values may reduce the discriminative capability of the model. We have provided a more detailed explanation of the parameter α in the Appendix H.4.

Importance of mCOMM: What if mCOMM is directly used for event detection? What if we use any other clustering algorithm? In this experiment, we compare the results of mCOMM with DBSCAN and with the proposed model DEMO. We use DBSCAN for comparison with mCOMM, as both of

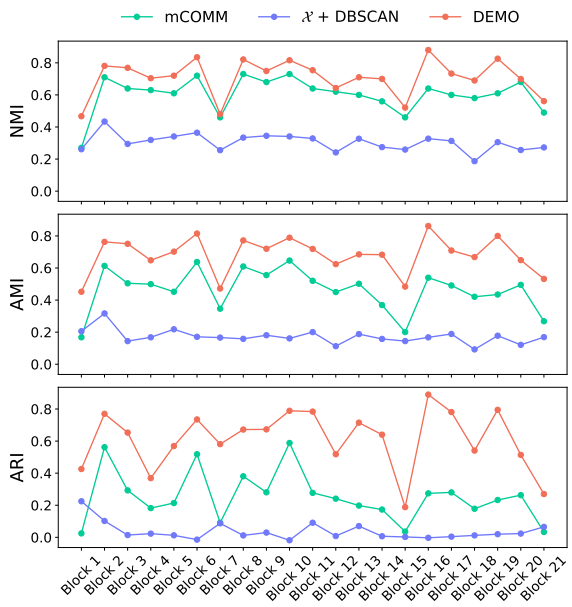


Figure 5: Comparison of DEMO with mCOMM and DBSCAN with $\mathcal{X}$ Features

them do not require the number of clusters to be known beforehand. Figure 5 shows the obtained result across the blocks for Event2012 data. It is evident that mCOMM without the use of features can produce better results than DBSCAN. It shows the effectiveness of using mCOMM with HMG. This motivated us to use the proposed mCOMM for adding cluster information to DEMO, which obtained the best results here. When we use DBSCAN with DEMO, we obtain lower results as the clusters identified by DBSCAN are not up to the mark (as Figure 5 already highlights).

4.2 Time Analysis

mCOMM processes each edge only once, i.e., time complexity is linear to the size of the stream, where density calculation and checking merging condition take $O(|L|)$ time. Thus, the overall time complexity of **onReceive**(e) function is $O(|E| \times |L|)$, where E and L are the edge set and layers in the whole stream, respectively. Complexity of **onQuery()** function is constant $O(1)$, which returns the pointer to all communities. The time complexity for creating each block H is of the order $\mathcal{O}(|E|)$.

The time complexity of GNN for each block H is $\mathcal{O}(\mathcal{K}(d|E| + d^2|V|))$ where $\mathcal{K}$ is the number of layers in the GNN, d is the feature dimension (we consider the input and output dimension to be the same for convenience), $|V|$ is the number of nodes in a block and $|E|$ is the number of edges.

5 Related Work

Social event detection methods can be broadly classified into three areas of work, namely term-commonness, topic modelling and online event detection-based approaches. In term-commonness, the top-most occurring words are detected within a time frame. Some notable works using commonness are (Li et al., 2012; Marcus et al., 2011). Topic modelling-based approaches for event detection operate by assigning each tweet a probabilistic distribution over multiple latent topics. Some relevant works are (Xie et al., 2013; Zhou et al., 2015; You et al., 2013). These approaches are primarily designed to work in an offline setting and struggle with event detection in real-time Twitter data due to the need to predefine the number of clusters.

GNNs have been used in the context of event detection quite frequently (Cao et al., 2021; Peng et al., 2023; Ren et al., 2024), but these methods require supervision. Entropy (Kenley and Cho, 2011) and modularity (Weng and Lee, 2021) are commonly used metrics for graph clustering (Weng and Lee, 2021), each offering unique advantages in capturing the structure and organization of complex networks. However, these methods have not addressed unsupervised event detection. Here, we propose a novel unsupervised method that combines modularity and entropy for event detection, harnessing the strengths of both metrics. For a detailed related work, kindly refer to Appendix B.

6 Conclusion

We have introduced DEMO, a novel heterogeneous multilayer density-infused entropy-modularity optimization event detection model, designed to effectively identify events while handling noise from heterogeneous attributes. From this study, it is evident that the trade-off between minimizing entropy and maximizing modularity helps our model distinguish between different event groups' features more effectively. Additionally, we also propose a streaming community detection algorithm for HMG (mCOMM), capable of processing millions of messages efficiently and providing the community participation information using density calculation. One can note that our study not only provides a solution for event detection but also has broader implications for various NLP applications. For example, the use of the proposed DEMO is not limited to event detection only. One may use wherever better clusters among different classes are required.

7 Limitations

We have mentioned the advantages of the proposed method in the previous sections. Here we acknowledge and discuss some of its limitations.

mCOMM: In the current form of the algorithm, we have given equal weightage to edges in each layer for deciding communities. We did not achieve significant improvements with different weight values for each layer, though we have not exhaustively searched every possible weight configuration. We acknowledge that this does not take into account the variable contributions of each type of relation, and whether each edge should be given equal weightage. In the future, we would like to improve upon this by figuring out a way to use learnable (or dynamically updatable) weights for each type of relation.

DEMO: We have already shown how we have improved upon existing works with our proposed model. There are some cases where our model does not perform better than the existing unsupervised method, HISEvent. We hypothesize that this is primarily due to the lack of a higher-order entropy in our 1-D entropy-based optimization function. This affects our performance, especially in metrics NMI and AMI, as already discussed in our results section. In our future works, we would like to improve upon this by incorporating a higher-order entropy in our model that captures hierarchical relationships, resulting in better cluster separation.

References

Foteini Alvanaki, Michel Sebastian, Krithi Ramamritham, and Gerhard Weikum. 2011. Enblogue: emergent topic detection in web 2.0 streams. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 1271–1274, New York, NY, USA. Association for Computing Machinery.

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. 2018. Can we gain more from orthogonality regularizations in training deep cnns? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4266–4276, Red Hook, NY, USA. Curran Associates Inc.

Hila Becker, Mor Naaman, and Luis Gravano. 2021. Beyond trending topics: Real-world event identification on twitter. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):438–441.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2008. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188.

Hongyun Cai, Yang Yang, Xuefei Li, and Zi Huang. 2015. What are popular: Exploring twitter features for event detection, tracking and visualization. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 89–98, New York, NY, USA. Association for Computing Machinery.

Yuwei Cao, Hao Peng, Jia Wu, Yingtong Dou, Jianxin Li, and Philip S. Yu. 2021. Knowledge-preserving incremental social event detection via heterogeneous gnns. In *Proceedings of the Web Conference 2021*, WWW '21, page 3383–3395, New York, NY, USA. Association for Computing Machinery.

Yuwei Cao, Hao Peng, Zhengtao Yu, and Philip S Yu. 2024. Hierarchical and incremental structural entropy minimization for unsupervised social event detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8255–8264.

Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. 2010. Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, MDMKDD '10, New York, NY, USA. Association for Computing Machinery.

Shraban Chatterjee, Suman Kundu, and Suman Kundu. 2024. Hmn: Generalization of heterogeneous and multi-layered network. *Preprint*, arXiv:2310.11534.

Swagatam Das, Sankha Subhra Mullick, and Ivan Zelinka. 2022. On supervised class-imbalanced learning: An updated perspective and some key challenges. *IEEE Transactions on Artificial Intelligence*, 3(6):973–993.

David L. Davies and Donald W. Bouldin. 1979. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Pablo A. Estevez, Michel Tesmer, Claudio A. Perez, and Jacek M. Zurada. 2009. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201.

Salvatore Gaglio, Giuseppe Lo Re, and Marco Morana. 2016. A framework for real-time twitter data analysis. *Comput. Commun.*, 73(PB):236–242.

A. Graves and J. Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4.

Shubham Gupta and Suman Kundu. 2023. Interaction graph, topical communities, and efficient local event detection from social streams. *Expert Systems with Applications*, 232:120890.

Shubham Gupta and Suman Kundu. 2025. Communities in streaming graphs: Small space data structure, benchmark data generation, and linear algorithm. *ACM Trans. Knowl. Discov. Data*. Just Accepted.

Obaida Hanteer and Matteo Magnani. 2020. Unspoken assumptions in multi-layer modularity maximization. *Scientific Reports*, 10.

Mahmud Hasan, Mehmet A. Orgun, and Rolf Schwitter. 2016. Twitternews+: A framework for real time event detection from the twitter data stream. In *Social Informatics*, pages 224–239, Cham. Springer International Publishing.

Alexandre Hollocou, Julien Maudet, Thomas Bonald, and Marc Lelarge. 2017. A linear streaming algorithm for community detection in very large networks. *Preprint*, arXiv:1703.02955.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 66–74, New York, NY, USA. Association for Computing Machinery.

Edward Casey Kenley and Young-Rae Cho. 2011. Entropy-based graph clustering: Application to biological and social networks. In *2011 IEEE 11th International Conference on Data Mining*, pages 1116–1121.

Junghoon Kim, Siqiang Luo, Gao Cong, and Wenyuan Yu. 2022. Dmcs: Density modularity based community search. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 889–903, New York, NY, USA. Association for Computing Machinery.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Chenliang Li, Aixin Sun, and Anwitaman Datta. 2012. Twevent: segment-based event detection from tweets. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, page 155–164, New York, NY, USA. Association for Computing Machinery.

Hui Li, Fucai Chen, and Jianpeng Zhang. 2020. A streaming-based algorithm for overlapping community detection. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 1925–1930.

Bang Liu, Fred X. Han, Di Niu, Linglong Kong, Kunfeng Lai, and Yu Xu. 2020. Story forest: Extracting events and telling stories from breaking news. *ACM Trans. Knowl. Discov. Data*, 14(3).

Yiwei Liu, Jiamou Liu, Zijian Zhang, Liehuang Zhu, and Angsheng Li. 2019. Rem: From structural entropy to community structure deception. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Amina Madani, Omar Boussaid, and Djamel Eddine Zegour. 2015. Real-time trending topics detection and description from twitter content. *Social Network Analysis and Mining*, 5(1):59.

Hangyin Mao, Ying Jiang, Xiaohan Lai, Yehua Zhang, and Xiaoxiao Huang. 2024. Enhancing public opinion monitoring for social hot events with a time series neural network-based logic map. *Measurement and Control*, 0(0):00202940241284017.

Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, and Robert C. Miller. 2011. Twitinfo: aggregating and visualizing microblogs for event exploration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 227–236, New York, NY, USA. Association for Computing Machinery.

Michael Mathioudakis and Nick Koudas. 2010. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 1155–1158, New York, NY, USA. Association for Computing Machinery.

Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

M. E. J. Newman. 2006a. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104.

M. E. J. Newman. 2006b. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

Miles Osborne, Sean Moran, Richard McCreadie, Alexander Von Lunen, Martin Sykora, Elizabeth Cano, Neil Ireson, Craig Macdonald, Iadh Ounis, Yulan He, Tom Jackson, Fabio Ciravegna, and Ann O'Brien. 2014. Real-time detection, tracking, and monitoring of automatically discovered events in social media. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Baltimore, Maryland. Association for Computational Linguistics.

Antonio Iyda Paganelli, Abel González Mondéjar, Abner Cardoso da Silva, Greis Silva-Calpa, Mateus F. Teixeira, Felipe Carvalho, Alberto Raposo, and Markus Endler. 2022. Real-time data analysis in health monitoring systems: A comprehensive systematic literature review. *Journal of Biomedical Informatics*, 127:104009.

Ruchi Parikh and Kamalakar Karlapalem. 2013. Et: events from tweets. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 613–620, New York, NY, USA. Association for Computing Machinery.

Hao Peng, Ruitong Zhang, Shaoning Li, Yuwei Cao, Shirui Pan, and Philip S. Yu. 2023. Reinforced, incremental and cross-lingual event detection from social messages. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):980–998.

Alexandru PETRESCU, Ciprian-Octavian TRUICA, Elena-Simona APOSTOL, and Adrian PASCHKE. 2024. Edsa-ensemble: an event detection sentiment analysis ensemble architecture. *IEEE Transactions on Affective Computing*, pages 1–18.

Swit Phuvipadawat and Tsuyoshi Murata. 2010. Breaking news detection and tracking in twitter. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 120–123.

Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. In *Computer and Information Sciences - ISCIS 2005*, pages 284–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Abhishek Rajora, Shubham Gupta, and Suman Kundu. 2025. EA2N: Evidence-based AMR Attention Network for Fake News Detection . *IEEE Transactions on Knowledge & Data Engineering*, (01):1–12.

Jiaqian Ren, Lei Jiang, Hao Peng, Zhiwei Liu, Jia Wu, and Philip S. Yu. 2022. Evidential temporal-aware graph-based social event detection via dempster-shafer theory. In *2022 IEEE International Conference on Web Services (ICWS)*, pages 331–336.

Jiaqian Ren, Hao Peng, Lei Jiang, Zhifeng Hao, Jia Wu, Shengxiang Gao, Zhengtao Yu, and Qiang Yang. 2024. Toward cross-lingual social event detection with hybrid knowledge distillation. *ACM Trans. Knowl. Discov. Data*, 18(9).

Alan Ritter, Mausam, Oren Etzioni, and Sam Clark. 2012. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, page 1104–1112, New York, NY, USA. Association for Computing Machinery.

Martin Rosvall and Carl T. Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123.

Nandini Saini, Shubham Gupta, Suman Kundu, and Debasis Das. 2024. Crisiskan: Knowledge-infused and explainable multimodal attention network for crisis event classification. In *Advances in Information Retrieval: 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24–28, 2024, Proceedings, Part II*, page 18–33, Berlin, Heidelberg. Springer-Verlag.

David Shepard. 2014. Nonparametric bayes pachinko allocation for super-event detection in twitter. In *TENCON 2014 - 2014 IEEE Region 10 Conference*, pages 1–5.

Giovanni Stilo and Paola Velardi. 2016. Efficient temporal mining of micro-blog texts and its application to event discovery. *Data Mining and Knowledge Discovery*, 30(2):372–402.

Chaodong Tong, Huailiang Peng, Xu Bai, Qiong Dai, Ruitong Zhang, Yangyang Li, Hanjie Xu, and Xian-Ming Gu. 2023. Learning discriminative text representation for streaming social event detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12295–12309.

V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1).

Vincent A. Traag and Lovro Šubelj. 2023. Large network community detection by fast label propagation. *Scientific Reports*, 13(1).

Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2024. Graph clustering with graph neural networks. *J. Mach. Learn. Res.*, 24(1).

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. *Preprint*, arXiv:1710.10903.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11:2837–2854.

Yifei Wang, Yupan Wang, Zeyu Zhang, Song Yang, Kaiqi Zhao, and Jiamou Liu. 2023. User: unsupervised structural entropy-based robust graph neural network. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*

and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Jianshu Weng and Bu-Sung Lee. 2021. Event detection in twitter. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):401–408.

Runquan Xie, Feida Zhu, Hui Ma, Wei Xie, and Chen Lin. 2014. Clear: a real-time online observatory for bursty and viral events. *Proc. VLDB Endow.*, 7(13):1637–1640.

Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. 2013. Topicsketch: Real-time bursty topic detection from twitter. In *2013 IEEE 13th International Conference on Data Mining*, pages 837–846.

Yue You, Guangyan Huang, Jian Cao, Enhong Chen, Jing He, Yanchun Zhang, and Liang Hu. 2013. Geam: A general and event-related aspects model for twitter event detection. In *Web Information Systems Engineering – WISE 2013*, pages 319–332, Berlin, Heidelberg. Springer Berlin Heidelberg.

Xiaoyan Yu, Yifan Wei, Shuaishuai Zhou, Zhiwei Yang, Li Sun, Hao Peng, Liehuang Zhu, and Philip S Yu. 2024. Towards effective, efficient and unsupervised social event detection in the hyperbolic space. *arXiv*.

Chuanhui Zhang, Junxiao Chen, Tao Shu, and Jinghua Tan. 2022. Enterprise event risk detection based on supply chain contagion. In *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10.

Xiaoming Zhang, Xiaoming Chen, Yan Chen, Senzhang Wang, Zhoujun Li, and Jiali Xia. 2015. Event detection and popularity prediction in microblogging. *Neurocomputing*, 149:1469–1480.

Deyu Zhou, Liangyu Chen, and Yulan He. 2015. An unsupervised framework of exploring events on twitter: Filtering, extraction and categorization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1).

A Implementation Details

A.1 Initial Feature Generation

We extract word embeddings for each word of a tweet after removal of all rare and common words from the tweet using a pre-trained word embedding model (Mikolov et al., 2013). In addition, each tweet contains time stamp information, which we encode into a vector representation (Cao et al., 2021). The encoded vector representation of the timestamp is concatenated with the textual representation of the tweet to get the final feature $\mathcal{X}$ of a tweet.

A.2 Reproducibility

In this Section, we provide the detailed configuration for the proposed method and all the baselines used in the experiments. We use identical features (including timestamp encoding) for all methods. Also, we use the same hyperparameters across the models for comparison. In case a hyperparameter is unique to a model, we use the default settings for that model.

BERT: The results for BERT are obtained after clustering the BERT encodings of the tweets with DBSCAN clustering method. We use the respective language encoders for each data set (English for Event2012 and French for Event2018).

BiLSTM: BiLSTM model is trained using Triplet Loss with learning rate of 0.001, batch size of 1000, dropout of 0.8, output dimension of 64, one LSTM bidirectional layer, 20 epochs and KMeans with the actual number of clusters. We use KMeans here as it gives better results.

EventX: We train this with a minimum number of co-occurrences threshold set to 2, conditional probability threshold for occurrence of words set to 0.15 and the minimum number of node threshold to stop graph splitting set to 3 as suggested in (Cao et al., 2021) and (Liu et al., 2020).

Table 2: Notation Table

Notations	Meaning
s	A social message
B_t	A homogeneous message block
SS	Social Stream
c_i	Event Class
$\mathcal{SE}_t$	Set of event classes in a block t
H_t	Heterogeneous Multilayered Block
ρ_{in}	Inner Density
ρ_{out}	Outer Density
α	Merge Threshold
$\mathcal{C}_k$	Community Sketch
$\mathcal{G}_t^l$	Homogeneous projection of H_t
$\mathcal{X}$	Initial Features
f_θ	Function for social event detection
$\Diamond$	Entropy
Q	Modularity
$\mathcal{S}_\mathcal{U}$	Activation Fn SeLU
$\mathcal{F}, F_1^l$	Final node features from GCN, features in intermediate layers
$\mathcal{S}_e$	Activation Fn Sigmoid
$\mathcal{S}_\sigma$	Softmax
$M, \dot{M}, \bar{M}$	Raw, learnable and normalized Adjacency Matrix
$\mathcal{T}$	Matrix Trace
$\mathcal{C}_\mathcal{G}$	Partition assignment matrix
$\mathcal{E}_\delta$	Encoder with parameter δ
$\mathcal{B}$	Modularity Matrix
β_1	Weight to control entropy loss
β_2	Weight to control modularity loss
k	Number of partitions

KPGNN: We use a window size of 3, batch size of 2000, learning rate of 0.001, latest message strategy, output dimension of 32, GAT model with 4 attention heads and residual connections. The number of epochs are 15. All these parameters are the default parameters used in the KPGNN implementation mentioned in the paper (Cao et al., 2021). We use KMeans with the actual number of clusters as this produces best results and is the default choice for the default KPGNN implementation. The KPGNN$_t$ uses the global-local pair loss mentioned in the paper (Cao et al., 2021). The rest of the parameters are the same as KPGNN.

FinEvent: We use a window size of 3, batch size of 100, learning rate of 0.001, GAT model with 4 attention heads and residual connections and output dimension of 64. The step size of RL-0 for state1 and state3 are 0.02. The initial value of epsilon for state2 is set as 0.001 with a step size of 0.02. All the parameters are taken from the authors implementation with the paper (Peng et al., 2023). In the case of Event2018 dataset we use the FinEvent$_g$ setting reported in the paper.

CLKD: We train for 15 epochs using a window size of 3, batch size of 2000, learning rate of 0.001, latest message strategy, output dimension of 32, GAT model with 4 attention heads and residual connections. For the Event2012 dataset we use mode 1 as the teacher and student are the same language. For Event2018 we report the results on the mode 2 with linear cross-lingual knowledge distillation with English as the teacher model and French as the student model.

DEMO: In case of our model, we use GCN encoder with one hidden layer with a dimension of 1024, number of epochs to 200, learning rate to 0.001, $\alpha = 0.5$ and $\beta = 0.5$. We initialize all the learnable parameters θ and weights $\dot{M}$ for the adjacency matrix from a standard normal distribution. We set $\beta_1 = \beta_2 = 1$. In the case of DEMO, we use the weights of the k-1th block GNN for initializing the weights of the kth layer GNN.

HISEvent: We use the minimum group n = 10 for smaller initial clusters that have possibility to merge into bigger clusters. The choice of n does not affect the performance of HISEvent significantly as shown in the paper but taking a lower value of n avoids the deadlock situation as suggested by (Yu et al., 2024).

HyperSED: We use the default settings of HyperSED as provided in the repository of the paper.

A.3 Metrics

In clustering tasks, evaluating the quality of the resulting partitions is crucial for comparing models and ensuring reliable results. Three widely used metrics for this purpose are Normalized Mutual Information (NMI) (Estevez et al., 2009), Adjusted Mutual Information (AMI) (Vinh et al., 2010), and Adjusted Rand Index (ARI) (Vinh et al., 2010). NMI measures the amount of information shared between the predicted and true clusters, normalized to ensure values between 0 and 1, where 1 indicates perfect clustering. AMI improves upon NMI by adjusting for chance, penalizing random assignments, and ensuring that the metric remains unbiased regardless of the number of clusters. ARI, on the other hand, assesses the similarity between predicted and ground-truth labels by computing the ratio of correctly paired samples, adjusting for chance to mitigate the impact of random cluster assignments. Together, these metrics provide a robust framework for evaluating clustering performance, capturing different aspects of cluster alignment and ensuring a comprehensive assessment of model effectiveness. In this work we use all the three metrics for a rigorous analysis as suggested by some earlier works (Cao et al., 2021; Peng et al., 2023; Ren et al., 2024), ensuring that clustering quality is assessed from multiple perspectives, reinforcing the robustness and credibility of the results.

B Related Work

Social event detection can be broadly classified in three areas of work namely term commonness, topic modeling and online event detection based approaches. In term commonness the top most occurring words are detected within a time frame, clusters of messages containing such words are detected and the clusters are ranked. Some notable works following this approach are (Marcus et al., 2011; Li et al., 2012; Gaglio et al., 2016; Mathioudakis and Koudas, 2010; Alvanaki et al., 2011; Cataldi et al., 2010; Parikh and Karlapalem, 2013; Weng and Lee, 2021; Zhang et al., 2015; Stilo and Velardi, 2016; Gupta and Kundu, 2023). Topic modeling-based approaches for event detection operate by assigning each tweet a probabilistic distribution over multiple latent topics, enabling the extraction of hidden semantic structures from large

Table 3: Event detection performance for various methods in terms of NMI across different blocks.

Block	BERT	BiLSTM	EventX	KPGNN	KPGNN$_t$	FinEvent$_k$	CLKD	HISEvent	HyperSED	**DEMO**
					Event 2012					
B_1	0.34 ± .00	0.27 ± .00	**0.62 ± .00**	0.24 ± .01	0.25 ± .00	0.40 ± .00	0.28 ± .04	0.40 ± 0.00	0.30 ± 0.01	0.47 ± .01
B_2	0.28 ± .00	0.40 ± .00	0.22 ± .00	0.67 ± .02	0.70 ± .01	0.80 ± .00	0.69 ± .01	0.78 ± 0.00	0.43 ± 0.02	**0.83 ± .02**
B_3	0.44 ± .00	0.32 ± .00	0.63 ± .00	0.62 ± .01	0.66 ± .01	0.80 ± .00	0.76 ± .00	**0.86 ± 0.00**	0.42 ± 0.04	0.81 ± .02
B_4	0.31 ± .00	0.36 ± .00	0.59 ± .00	0.58 ± .01	0.57 ± .00	0.68 ± .01	0.57 ± .02	**0.77 ± 0.00**	0.51 ± 0.00	0.74 ± .02
B_5	0.33 ± .00	0.28 ± .00	0.58 ± .00	0.57 ± .01	0.60 ± .00	0.73 ± .01	0.58 ± .01	**0.81 ± 0.00**	0.47 ± 0.03	0.74 ± .02
B_6	0.36 ± .00	0.23 ± .00	0.58 ± .00	0.72 ± .00	0.76 ± .00	0.81 ± .00	**0.86 ± .00**	0.74 ± 0.00	0.71 ± 0.05	0.83 ± .03
B_7	0.39 ± .00	0.30 ± .00	**0.66 ± .00**	0.40 ± .00	0.41 ± .02	0.52 ± .02	0.38 ± .02	0.59 ± 0.00	0.41 ± 0.01	0.50 ± .04
B_8	0.27 ± .00	0.23 ± .00	0.42 ± .00	0.70 ± .01	0.71 ± .00	0.82 ± .01	0.69 ± .00	0.64 ± 0.00	0.68 ± 0.02	**0.85 ± .01**
B_9	0.36 ± .00	0.30 ± .00	0.67 ± .00	0.60 ± .02	0.61 ± .01	0.73 ± .00	0.78 ± .01	0.79 ± 0.00	0.62 ± 0.00	**0.82 ± .01**
B_{10}	0.36 ± .00	0.30 ± .00	0.63 ± .00	0.71 ± .00	0.71 ± .00	0.81 ± .00	0.70 ± .01	0.76 ± 0.00	0.65 ± 0.03	**0.86 ± .00**
B_{11}	0.36 ± .00	0.24 ± .00	0.62 ± .00	0.62 ± .01	0.61 ± .01	0.69 ± .02	0.60 ± .02	0.78 ± 0.00	0.60 ± 0.05	**0.81 ± .01**
B_{12}	0.35 ± .00	0.22 ± .00	0.58 ± .00	0.49 ± .01	0.51 ± .00	0.67 ± .00	0.63 ± .00	**0.84 ± 0.00**	0.31 ± 0.04	0.72 ± .02
B_{13}	0.26 ± .00	0.28 ± .00	0.59 ± .00	0.62 ± .00	0.59 ± .00	0.67 ± .00	0.63 ± .00	**0.77 ± 0.00**	0.53 ± 0.02	0.75 ± .00
B_{14}	0.34 ± .00	0.33 ± .00	0.49 ± .00	0.48 ± .01	0.48 ± .00	0.70 ± .01	0.47 ± .02	**0.80 ± 0.00**	0.35 ± 0.01	0.72 ± .02
B_{15}	0.28 ± .00	0.20 ± .00	0.54 ± .00	0.40 ± .01	0.39 ± .01	0.59 ± .02	0.64 ± .00	**0.66 ± 0.00**	0.50 ± 0.03	0.63 ± .02
B_{16}	0.26 ± .00	0.28 ± .00	0.46 ± .00	0.68 ± .01	0.67 ± .01	0.75 ± .01	0.73 ± .03	0.76 ± 0.00	0.64 ± 0.04	**0.91 ± .01**
B_{17}	0.34 ± .00	0.28 ± .00	0.55 ± .00	0.54 ± .01	0.54 ± .01	0.70 ± .00	0.56 ± .00	**0.80 ± 0.00**	0.37 ± 0.00	0.79 ± .01
B_{18}	0.32 ± .00	0.28 ± .00	0.51 ± .00	0.46 ± .00	0.46 ± .00	0.64 ± .02	0.64 ± .02	**0.79 ± 0.00**	0.40 ± 0.05	0.70 ± .00
B_{19}	0.20 ± .00	0.27 ± .00	0.54 ± .00	0.54 ± .00	0.52 ± .01	0.74 ± .02	0.56 ± .02	0.83 ± 0.00	0.50 ± 0.02	**0.87 ± .01**
B_{20}	0.32 ± .00	0.28 ± .00	0.57 ± .00	0.60 ± .01	0.62 ± .01	0.71 ± .01	0.61 ± .01	0.66 ± 0.00	0.71 ± 0.01	**0.73 ± .02**
B_{21}	0.28 ± .00	0.27 ± .00	**0.63 ± .00**	0.38 ± .00	0.41 ± .01	0.61 ± .01	0.59 ± .02	0.59 ± 0.00	0.34 ± 0.03	0.57 ± .02
					Event 2018					
B_1	0.16 ± 0.00	0.11 ± 0.01	0.34 ± 0.00	0.45 ± 0.02	0.42 ± 0.02	0.56 ± 0.01	0.56 ± 0.04	**0.78 ± 0.00**	0.11 ± 0.01	0.72 ± 0.00
B_2	0.16 ± 0.00	0.09 ± 0.00	0.37 ± 0.00	0.49 ± 0.01	0.46 ± 0.03	0.57 ± 0.04	0.55 ± 0.03	**0.77 ± 0.00**	0.19 ± 0.04	0.69 ± 0.04
B_3	0.19 ± 0.00	0.09 ± 0.01	0.37 ± 0.01	0.46 ± 0.01	0.45 ± 0.01	0.59 ± 0.01	0.64 ± 0.02	**0.75 ± 0.00**	0.19 ± 0.02	0.71 ± 0.02
B_4	0.23 ± 0.00	0.12 ± 0.01	0.39 ± 0.00	0.42 ± 0.02	0.48 ± 0.02	0.48 ± 0.02	0.52 ± 0.03	**0.72 ± 0.00**	0.19 ± 0.03	0.65 ± 0.01
B_5	0.29 ± 0.00	0.22 ± 0.00	0.53 ± 0.01	0.55 ± 0.02	0.50 ± 0.01	0.57 ± 0.01	0.61 ± 0.01	**0.77 ± 0.00**	0.25 ± 0.01	0.68 ± 0.04
B_6	0.26 ± 0.00	0.17 ± 0.01	0.44 ± 0.00	0.35 ± 0.02	0.40 ± 0.03	0.51 ± 0.00	0.65 ± 0.03	**0.81 ± 0.00**	0.18 ± 0.02	0.73 ± 0.04
B_7	0.21 ± 0.00	0.16 ± 0.01	0.41 ± 0.01	0.45 ± 0.02	0.37 ± 0.03	0.48 ± 0.01	0.66 ± 0.01	**0.80 ± 0.00**	0.18 ± 0.05	0.77 ± 0.03
B_8	0.26 ± 0.00	0.20 ± 0.00	0.53 ± 0.01	0.39 ± 0.02	0.40 ± 0.01	0.54 ± 0.03	0.61 ± 0.03	**0.86 ± 0.00**	0.34 ± 0.01	0.72 ± 0.02
B_9	0.26 ± 0.00	0.16 ± 0.00	0.45 ± 0.01	0.34 ± 0.02	0.27 ± 0.04	0.43 ± 0.01	0.54 ± 0.02	**0.72 ± 0.00**	0.29 ± 0.03	0.58 ± 0.00
B_{10}	0.30 ± 0.00	0.25 ± 0.01	0.52 ± 0.00	0.39 ± 0.01	0.43 ± 0.01	0.60 ± 0.01	0.63 ± 0.04	**0.80 ± 0.00**	0.34 ± 0.02	0.67 ± 0.01
B_{11}	0.28 ± 0.00	0.19 ± 0.00	0.48 ± 0.01	0.38 ± 0.02	0.38 ± 0.02	0.51 ± 0.03	0.59 ± 0.01	**0.83 ± 0.00**	0.29 ± 0.04	0.66 ± 0.03
B_{12}	0.25 ± 0.00	0.22 ± 0.00	0.51 ± 0.01	0.41 ± 0.03	0.46 ± 0.04	0.52 ± 0.02	0.72 ± 0.02	**0.85 ± 0.00**	0.21 ± 0.01	0.72 ± 0.02
B_{13}	0.17 ± 0.00	0.13 ± 0.00	0.44 ± 0.00	0.34 ± 0.01	0.37 ± 0.01	0.47 ± 0.03	0.64 ± 0.03	**0.86 ± 0.00**	0.19 ± 0.05	0.73 ± 0.02
B_{14}	0.24 ± 0.00	0.16 ± 0.00	0.52 ± 0.00	0.40 ± 0.02	0.47 ± 0.02	0.53 ± 0.04	0.72 ± 0.02	**0.88 ± 0.00**	0.24 ± 0.03	0.80 ± 0.03
B_{15}	0.26 ± 0.00	0.20 ± 0.01	0.49 ± 0.00	0.45 ± 0.04	0.44 ± 0.03	0.58 ± 0.02	0.75 ± 0.03	**0.83 ± 0.00**	0.22 ± 0.02	0.73 ± 0.00

tweet corpora. These methods leverage advanced probabilistic models to infer latent topics, which serve as the foundational framework for identifying and characterizing events within the data. For example TwiCal (Ritter et al., 2012) constructs an open-domain calendar by extracting and categorizing significant events from Twitter data using Conditional Random Fields (Lafferty et al., 2001) and latent variable models. Events are ranked by entity-date association, though the method struggles with unexpected or low-significance events. Some other notable works in this area are (Xie et al., 2013; Zhou et al., 2015; You et al., 2013; Xie et al., 2014; Cai et al., 2015; Shepard, 2014; Madani et al., 2015). The aforementioned approaches are primarily designed to work in an offline setting and they struggle with event detection in real-time Twitter data due to the need to predefine the number of clusters. This is challenging because the volume and variety of topics are unpredictable. Incremental clustering strategies are used to dynamically detect events without requiring a fixed number of clusters. For example, TwitterNews+ (Hasan et al., 2016) detects events by first identifying bursts of similar tweets and then clustering them incrementally using tf-idf and cosine similarity. Some other works that focus on incremental event detection are (Osborne et al., 2014; Becker et al., 2021; Phuvipadawat and Murata, 2010).

GNNs have been used in the context of event detection quite frequently (Ren et al., 2022; Tong et al., 2023; Cao et al., 2021; Peng et al., 2023; Ren et al., 2024). Some of these works focus on the offline setting where the number of events and messages are predefined. For example in the case of (Ren et al., 2022) they consider long tailed nature of event distribution by adding temporal information in the message passing of the GNN. Similarly, (Tong et al., 2023) uses a GAT (Veličković et al., 2018) with contrastive learning. But (Tong et al., 2023) does not consider the feature pivots between text messages thus misplacing messages into wrong and sometimes isolated clusters. To address online event clustering there are several supervised methods such as (Cao et al., 2021; Peng et al., 2023; Ren et al., 2024). In (Cao et al., 2021) the authors construct time temporal message graphs based on common feature pivots such as common entities

Table 4: Event detection performance for various methods in terms of AMI across different blocks.

Block	BERT	BiLSTM	EventX	KPGNN	KPGNN$_t$	FinEvent$_k$	CLKD	HISEvent	HyperSED	**DEMO**
					EVENT 2012					
B_1	0.17 ± .00	0.16 ± .00	0.29 ± .00	0.22 ± .01	0.23 ± .00	0.39 ± .00	0.26 ± .00	0.39 ± .00	0.28 ± .02	**0.45 ± 0.02**
B_2	0.22 ± .00	0.11 ± .00	0.06 ± .00	0.64 ± .02	0.68 ± .01	0.78 ± .00	0.67 ± .00	0.78 ± .00	0.38 ± .01	**0.80 ± 0.02**
B_3	0.33 ± .00	0.10 ± .00	0.40 ± .00	0.59 ± .01	0.63 ± .01	0.78 ± .00	0.65 ± .00	**0.85 ± .00**	0.36 ± .03	0.79 ± 0.03
B_4	0.15 ± .00	0.11 ± .00	0.26 ± .00	0.54 ± .01	0.52 ± .00	0.67 ± .00	0.53 ± .00	**0.75 ± .00**	0.45 ± .02	0.68 ± 0.03
B_5	0.17 ± .00	0.09 ± .00	0.25 ± .01	0.55 ± .01	0.57 ± .00	0.73 ± .00	0.55 ± .00	**0.80 ± .00**	0.42 ± .01	0.70 ± 0.02
B_6	0.23 ± .00	0.08 ± .00	0.30 ± .00	0.68 ± .00	0.72 ± .00	0.78 ± .00	0.75 ± .00	0.72 ± .00	0.65 ± .03	**0.81 ± 0.01**
B_7	0.18 ± .00	0.15 ± .00	0.38 ± .00	0.36 ± .02	0.37 ± .02	0.52 ± .01	0.34 ± .00	**0.57 ± .00**	0.37 ± .02	0.48 ± 0.02
B_8	0.18 ± .00	0.09 ± .00	0.14 ± .00	0.64 ± .01	0.65 ± .00	**0.81 ± .00**	0.64 ± .00	0.62 ± .00	0.61 ± .01	0.78 ± 0.03
B_9	0.17 ± .00	0.18 ± .00	0.32 ± .01	0.55 ± .02	0.56 ± .01	0.69 ± .03	0.63 ± .00	0.77 ± .00	0.56 ± .03	**0.78 ± 0.02**
B_{10}	0.20 ± .00	0.07 ± .00	0.31 ± .00	0.68 ± .00	0.67 ± .00	0.78 ± .00	0.66 ± .00	0.75 ± .00	0.60 ± .02	**0.83 ± 0.03**
B_{11}	0.18 ± .00	0.10 ± .00	0.30 ± .00	0.57 ± .01	0.57 ± .01	0.65 ± .00	0.56 ± .00	**0.77 ± .00**	0.55 ± .01	0.78 ± 0.02
B_{12}	0.22 ± .00	0.07 ± .00	0.27 ± .00	0.45 ± .01	0.47 ± .00	0.65 ± .00	0.48 ± .00	**0.83 ± .00**	0.26 ± .03	0.68 ± 0.04
B_{13}	0.12 ± .00	0.12 ± .00	0.23 ± .00	0.58 ± .00	0.55 ± .00	0.64 ± .00	0.59 ± .00	**0.75 ± .00**	0.48 ± .01	0.72 ± 0.04
B_{14}	0.19 ± .00	0.05 ± .00	0.21 ± .00	0.45 ± .01	0.44 ± .00	0.67 ± .00	0.44 ± .00	**0.79 ± .00**	0.31 ± .02	0.70 ± 0.02
B_{15}	0.16 ± .00	0.08 ± .00	0.22 ± .00	0.35 ± .01	0.34 ± .01	0.57 ± .00	0.51 ± .00	**0.63 ± .00**	0.45 ± .03	0.58 ± 0.02
B_{16}	0.15 ± .00	0.12 ± .00	0.17 ± .00	0.65 ± .01	0.63 ± .01	0.74 ± .00	0.70 ± .00	0.74 ± .00	0.60 ± .01	**0.89 ± 0.02**
B_{17}	0.18 ± .00	0.12 ± .00	0.24 ± .00	0.51 ± .01	0.51 ± .01	0.68 ± .00	0.53 ± .00	**0.80 ± .00**	0.33 ± .02	0.76 ± 0.02
B_{18}	0.20 ± .00	0.11 ± .00	0.23 ± .00	0.42 ± .00	0.42 ± .00	0.63 ± .00	0.52 ± .00	**0.78 ± .00**	0.35 ± .01	0.67 ± 0.03
B_{19}	0.11 ± .00	0.12 ± .00	0.18 ± .00	0.50 ± .00	0.49 ± .01	0.77 ± .00	0.53 ± .00	**0.82 ± .00**	0.46 ± .03	**0.82 ± 0.02**
B_{20}	0.19 ± .00	0.12 ± .00	0.22 ± .00	0.53 ± .01	0.56 ± .01	**0.68 ± .00**	0.55 ± .00	0.63 ± .00	0.66 ± .02	0.66 ± 0.02
B_{21}	0.13 ± .00	0.12 ± .00	0.24 ± .00	0.35 ± .00	0.37 ± .01	**0.61 ± .00**	0.44 ± .00	0.57 ± .00	0.29 ± .01	0.53 ± 0.02
					EVENT 2018					
B_1	0.11 ± 0.00	0.06 ± 0.01	0.10 ± 0.00	0.44 ± 0.02	0.41 ± 0.02	0.56 ± 0.01	0.55 ± 0.04	**0.77 ± .00**	0.09 ± .01	0.71 ± 0.00
B_2	0.11 ± 0.00	0.03 ± 0.00	0.12 ± 0.00	0.48 ± 0.01	0.45 ± 0.03	0.56 ± 0.04	0.54 ± 0.03	**0.77 ± .00**	0.17 ± .02	0.68 ± 0.04
B_3	0.13 ± 0.00	0.03 ± 0.01	0.11 ± 0.01	0.45 ± 0.01	0.44 ± 0.01	0.58 ± 0.01	0.62 ± 0.02	**0.75 ± .00**	0.17 ± .01	0.70 ± 0.02
B_4	0.16 ± 0.00	0.05 ± 0.01	0.14 ± 0.00	0.41 ± 0.02	0.47 ± 0.02	0.47 ± 0.02	0.51 ± 0.03	**0.71 ± .00**	0.17 ± .03	0.64 ± 0.01
B_5	0.18 ± 0.00	0.08 ± 0.00	0.24 ± 0.01	0.53 ± 0.02	0.48 ± 0.01	0.56 ± 0.01	0.59 ± 0.01	**0.75 ± .00**	0.21 ± .02	0.65 ± 0.04
B_6	0.19 ± 0.00	0.09 ± 0.01	0.15 ± 0.00	0.34 ± 0.02	0.39 ± 0.03	0.49 ± 0.00	0.62 ± 0.03	**0.80 ± .00**	0.16 ± .01	0.72 ± 0.04
B_7	0.14 ± 0.00	0.08 ± 0.01	0.12 ± 0.01	0.43 ± 0.02	0.36 ± 0.03	0.47 ± 0.01	0.65 ± 0.01	**0.80 ± .00**	0.15 ± .03	0.76 ± 0.03
B_8	0.15 ± 0.00	0.06 ± 0.00	0.21 ± 0.01	0.37 ± 0.02	0.37 ± 0.01	0.52 ± 0.03	0.59 ± 0.03	**0.85 ± .00**	0.30 ± .02	0.68 ± 0.02
B_9	0.18 ± 0.00	0.05 ± 0.00	0.16 ± 0.01	0.32 ± 0.02	0.25 ± 0.04	0.41 ± 0.01	0.47 ± 0.02	**0.71 ± .00**	0.26 ± .01	0.54 ± 0.00
B_{10}	0.20 ± 0.00	0.08 ± 0.01	0.19 ± 0.00	0.35 ± 0.01	0.40 ± 0.01	0.58 ± 0.01	0.61 ± 0.04	**0.79 ± .00**	0.30 ± .03	0.65 ± 0.01
B_{11}	0.18 ± 0.00	0.05 ± 0.00	0.18 ± 0.01	0.35 ± 0.02	0.36 ± 0.02	0.49 ± 0.03	0.57 ± 0.01	**0.82 ± .00**	0.25 ± .02	0.63 ± 0.03
B_{12}	0.15 ± 0.00	0.09 ± 0.00	0.20 ± 0.01	0.39 ± 0.03	0.44 ± 0.04	0.50 ± 0.02	0.68 ± 0.02	**0.84 ± .00**	0.17 ± .01	0.71 ± 0.02
B_{13}	0.10 ± 0.00	0.05 ± 0.00	0.15 ± 0.00	0.32 ± 0.01	0.35 ± 0.01	0.45 ± 0.03	0.63 ± 0.03	**0.85 ± .00**	0.16 ± .03	0.71 ± 0.02
B_{14}	0.15 ± 0.00	0.04 ± 0.00	0.22 ± 0.00	0.38 ± 0.02	0.45 ± 0.02	0.51 ± 0.04	0.71 ± 0.02	**0.87 ± .00**	0.20 ± .02	0.78 ± 0.03
B_{15}	0.15 ± 0.00	0.09 ± 0.01	0.22 ± 0.00	0.43 ± 0.04	0.42 ± 0.03	0.56 ± 0.02	0.72 ± 0.03	**0.82 ± .00**	0.18 ± .01	0.72 ± 0.00

and common words, aggregate the node features using GAT and triplet loss followed by Kmeans or DBSCAN clustering. This is followed by (Peng et al., 2023) where the authors use GNN along with RL for assigning weights to each type of edge between two messages. The node features are aggregated similarly to KPGNN, and then an RL-based DBSCAN algorithm, namely DRL-DBSCAN, is used to cluster the node features. Also, FinEvent shows the result of cross-lingual evaluation on the French dataset. Prior to this work, most works were focused on social messages in the English language. Further works like (Ren et al., 2024) extend KPGNN to work for multiple languages (termed as low resource languages in the paper) other than English. For the low resource languages in the paper, the authors use a knowledge distillation (KD) based approach (Ren et al., 2024). In the KD approach, a teacher model is trained for incremental event detection on English tweets, and a student model extracts knowledge from the teacher model and applies it to the target low-resource language. This process improves the clustering results for the tweets in the target language.

Entropy (Kenley and Cho, 2011) and modularity (Weng and Lee, 2021) are commonly used metrics for clustering graphs (Weng and Lee, 2021), each offering unique advantages in capturing the structure and organization of complex networks. Entropy measures the uncertainty or randomness within cluster assignments, promoting diverse and balanced clusters, while modularity evaluates the strength of division by comparing the density of edges within clusters to those between clusters, effectively highlighting community structures. Recently, GNN-based methods for graph clustering have gained attention. For example, the authors of (Wang et al., 2023) introduce an unsupervised approach that learns node representations by modeling graph perturbations and derives an intrinsic graph using entropy. Similarly, (Tsitsulin et al., 2024) shifts focus from node pooling to modularity optimization for clustering. However, these methods have not addressed unsupervised event detection. In this paper, we propose a novel unsupervised method that combines modularity and entropy for incremental event detection, harnessing the strengths of both metrics.

Table 5: Practical runtime (in seconds) comparison of DEMO with other unsupervised event detection models

Blocks	# Tweets	HISEvent	HyperSED	DEMO		
				mCOMM	Mod+Ent	Total
Event2012						
B_1	8722	19025	180	13	734	747
B_2	1491	482	44	1	31	32
B_3	1835	484	51	1	30	31
B_4	2010	696	58	1	30	31
B_5	1834	398	25	1	28	29
B_6	1276	303	41	1	23	24
B_7	5278	5142	166	6	211	217
B_8	1560	447	50	1	26	27
B_9	1363	402	37	1	25	26
B_{10}	1096	268	35	1	23	24
B_{11}	1232	381	37	1	22	23
B_{12}	3237	1100	86	1	54	55
B_{13}	1972	715	35	1	34	35
B_{14}	2956	1060	77	1	48	49
B_{15}	2549	607	66	1	35	36
B_{16}	910	239	25	1	20	21
B_{17}	2676	1298	38	1	44	45
B_{18}	1887	1073	26	1	30	31
B_{19}	1399	331	40	1	24	25
B_{20}	893	218	31	1	21	22
B_{21}	2410	1124	67	1	33	34
Event2018						
B_1	5356	5023	178	7	219	226
B_2	3186	951	90	2	59	61
B_3	2644	546	70	1	42	43
B_4	3179	1552	82	3	66	69
B_5	2662	499	66	1	42	43
B_6	4200	1903	104	3	106	109
B_7	3454	1353	83	2	77	79
B_8	2257	340	57	1	34	35
B_9	3669	1508	93	2	79	81
B_{10}	2385	424	59	1	37	38
B_{11}	2802	725	49	1	46	47
B_{12}	2927	511	72	1	49	50
B_{13}	4884	2604	127	4	165	169
B_{14}	3065	778	74	1	55	56
B_{15}	2411	500	58	1	39	40

C Comparing the runtime of proposed DEMO with unsupervised methods HISEvent and HyperSED

We compare the runtime of individual components our proposed method with existing unsupervised methods HISEvent and HyperSED as shown in Table 5. It is clearly evident from the Table that we are faster (in terms of combined performance) than existing unsupervised methods in 16 of 21 blocks for Event 2012 and 12 of 15 blocks for Event 2018.

D DEMO under noisy input

We check the robustness of our model by adding noisy edges to the input graph and then running DEMO. We add 20% noisy edges to each block of the Event2012 dataset randomly and see how DEMO performs in these scenarios. We show the variance in the results of DEMO with noisy edges in Figure 6. We set the value of $\alpha = 0.5$ and the value of $\beta_1 = \beta_2 = 0.5$. The results show that the variance is very small, even with the addition of noisy edges, thus showing our model's robustness.

E Qualitative Study

We show the qualitative results of DEMO showing the predicted communities by DEMO for HMG B_{20} of Event2012 dataset (in Figure 7b) and compare our results with ground truth communities (in Figure 7a). We can see that the number of ground truth communities is 34, and our model predicts 30 communities, which are very comparable, as is also evident visually. One of the primary mistakes our model makes is in detecting the very small communities. It merges these small communities with bigger communities.

F Extending DEMO to a streaming scenario

The proposed algorithm mCOMM is a fully streaming algorithm which does not store the graph at any point in time. We have also shown that mCOMM provides comparable results in terms of various metrics for event detection. The remaining part of DEMO requires some form of the graph to be stored at any time instant. Given this, we can easily convert DEMO to a streaming scenario if we construct message graphs hourly or per minute and then extract the latest events from these graphs. We can combine the current events with past events using the merge operation of mCOMM as defined in Algorithm 1. Specifically, we assume that we have the graph and tweets of two time instances t and $t-1$ saved in memory. We also consider that we have the inner and outer densities of communities available from $\mathcal{C}_{\mathcal{G}_{t-1}}$. Now, given that DEMO finds $\mathcal{C}_{\mathcal{G}_t}$ events for a HMG at time step t and we have $\mathcal{C}_{\mathcal{G}_{t-1}}$ events from time step $t-1$ (as well as the graph and tweets from both the time steps), we can merge them using the merge operation of mCOMM. We start by creating inter edges (based on the same criteria as mentioned above) between the tweets in communities $\mathcal{C}_{\mathcal{G}_{t-1}}$ and communities $\mathcal{C}_{\mathcal{G}_t}$, we call these edges E_1. We also include the inter community edges already present in the communities of $\mathcal{C}_{\mathcal{G}_t}$ (call them as E_2) and the inter community edges already present in the communities of $\mathcal{C}_{\mathcal{G}_{t-1}}$ (call them as E_3). We combine all these edges and call them E. We create $\mathcal{C}_k$ by merging the communities in $\mathcal{C}_{\mathcal{G}_{t-1}}$ and $\mathcal{C}_{\mathcal{G}_t}$. Also, based on the edgeset E we update the outer density of every community. Now, with this information available, the edgeset E is then streamed to mCOMM. The mCOMM algorithm creates a final set of communities by merging communities from $\mathcal{C}_{\mathcal{G}_{t-1}}$ and $\mathcal{C}_{\mathcal{G}_t}$. At time step 1,

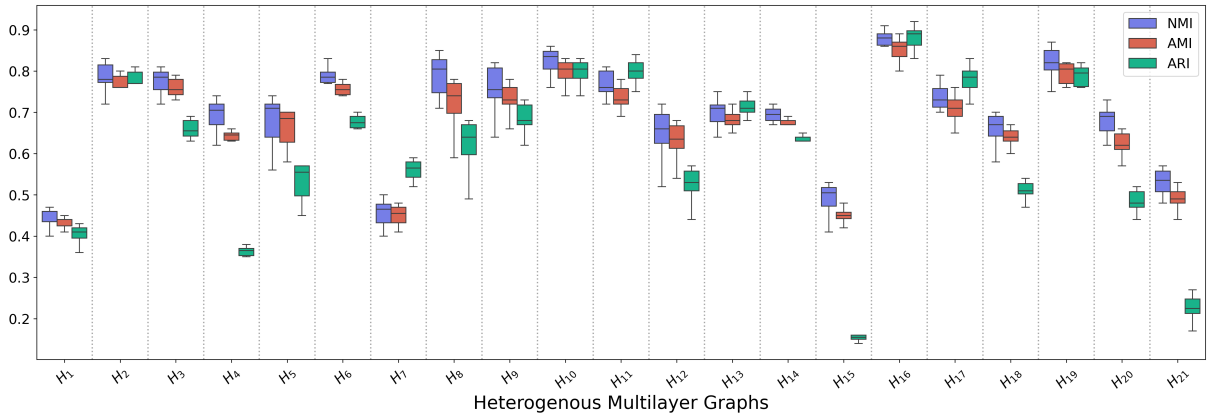
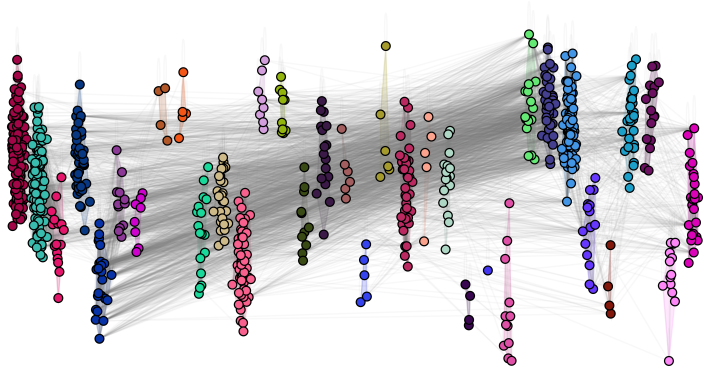
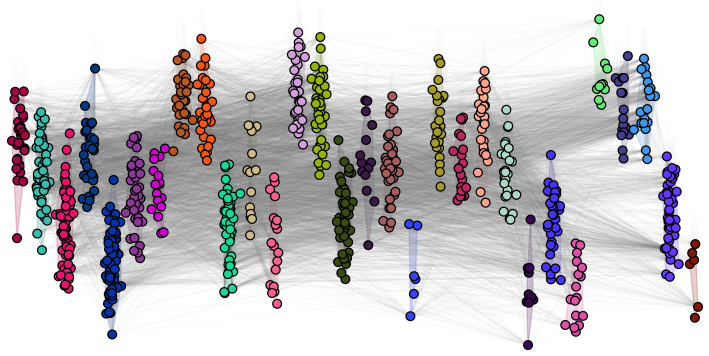


Figure 6: Change in variance of DEMO with the addition of noisy edges. Here each of the boxes represent the variance in NMI, AMI or ARI.



(a) Ground Truth Events



(b) Predicted Events

Figure 7: Qualitative Study showing predicted communities by DEMO for HMG B_{20} compared with ground truth communities.

the merging step is trivial. For time step 2, we will have the inter and intra community densities of communities that evolved in time step 1, and these will be updated in time step 2 and so on. Thus, we will not be required to keep data for more than two consecutive time steps. In all these scenarios, we store a homogeneous graph for time steps t and $t-1$.

G Scaling DEMO to large datasets

We have shown mCOMM to be very fast compared to existing methods (Table 7), along with its scalability to bigger graphs. Our learning pipeline in DEMO can handle large-scale data efficiently by leveraging parallel processing, which is already popular in deep neural networks. Since DEMO primarily does not mandate an incremental setting, we can divide the original message graph into smaller subgraphs, which can then be processed in parallel. In real-world scenarios, such as real-time traffic updates or event detection, social streams are often pre-processed by region or topic and filtered to reduce noise. These filtered message graphs typically stay compact (e.g., covering only the past hour's activity) to support rapid decision-making (Cao et al., 2024). DEMO's architecture makes it well-suited for real-time monitoring tasks like crisis alerts, urban mobility tracking, or social sentiment analysis, offering value to first responders, municipal agencies, and media analysts.

H Further details on mCOMM

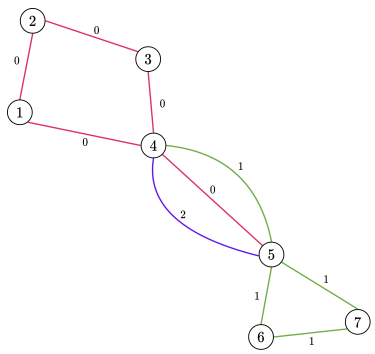
H.1 Detailed Explanation of mCOMM (Algorithm 1)

Algorithm 1 describes the process of community detection for heterogeneous multilayer streaming data. When an edge $e(u, v, l)$ is received, the algorithm first initializes a community sketch data structure, which consists of a forest f and a sparse triangular matrix mat, using the *makeSketch()* function. Next, the *updateSketch()* function is called to check the membership of nodes u and v in existing communities. If both nodes belong to the same community, the edge count within the community is increased. If the nodes belong to different communities, the edge count between the communities is updated. If only one of the nodes is part of an existing community, a new single-node community is created, with the other node serving as its community representative. If neither node belongs to any community, a new community with two nodes is formed, assigning the node with the lower value as the community representative and the other node as its child. The sparse triangular matrix mat is updated with the information of the newly formed or modified community. The algorithm then calculates the outer and inner densities of the communities based on edge and node counts across different layers from mat. If the outer density is greater than the α (user-defined parameter) times sum of inner density of both communities, they are merged using the *mergeCommunity()* function. Finally, the *onQuery()* function retrieves all communities observed in the stream till the query is made. Detailed description of functions namely community$(\cdot)$, makeSketch$(\cdot)$, updateSketch$(\cdot)$, and mergeCommunity$(\cdot)$ is provided below:

community($node$): It returns a community representative of a community where $node$ belongs. In other words, it returns the root node of a tree where a $node$ lies.

makeSketch$()$: It initializes a community sketch $\mathcal{C}_k$ with a forest f and a sparse matrix mat.

updateSketch($e = (node_1, node_2, l)$): The function updates the Community Sketch with a new edge e arrival. It checks if both nodes belong to existing communities. If they belong to



Edge	$\alpha = 0.1, L = 3, w = [0.33, 0.33, 0.33]$	ρ
$(1, 2, 0)$	$\mathcal{C}_1 = \{1, 2\}$	
$(2, 3, 0)$	$\mathcal{C}_1 = \{1, 2\}, \mathcal{C}_3 = \{3\} \xrightarrow{\text{merge}(\mathcal{C}_1, \mathcal{C}_3)} \mathcal{C}_1 = \{1, 2, 3\}$	$\rho_{in}(\mathcal{C}_1) = 0.11, \rho_{in}(\mathcal{C}_3) = 0.0, \rho_{out}(\mathcal{C}_1, \mathcal{C}_3) = 0.06$
$(3, 4, 0)$	$\mathcal{C}_1 = \{1, 2, 3\}, \mathcal{C}_4 = \{4\} \xrightarrow{\text{merge}(\mathcal{C}_1, \mathcal{C}_4)} \mathcal{C}_1 = \{1, 2, 3, 4\}$	$\rho_{in}(\mathcal{C}_1) = 0.07, \rho_{in}(\mathcal{C}_4) = 0.0, \rho_{out}(\mathcal{C}_1, \mathcal{C}_4) = 0.04$
$(4, 1, 0)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}$	
$(5, 6, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6\}$	
$(6, 7, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6\}, \mathcal{C}_7 = \{7\} \xrightarrow{\text{merge}(\mathcal{C}_5, \mathcal{C}_7)} \mathcal{C}_5 = \{5, 6, 7\}$	$\rho_{in}(\mathcal{C}_5) = 0.11, \rho_{in}(\mathcal{C}_7) = 0.0, \rho_{out}(\mathcal{C}_5, \mathcal{C}_6) = 0.06$
$(7, 5, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	
$(4, 5, 0)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	
$(4, 5, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\} \xrightarrow{\text{merge}(\mathcal{C}_1, \mathcal{C}_5)} \mathcal{C}_1 = \{1, 2, 3, 4, 5, 6, 7\}$	$\rho_{in}(\mathcal{C}_1) = 0.07, \rho_{in}(\mathcal{C}_5) = 0.11, \rho_{out}(\mathcal{C}_1, \mathcal{C}_5) = 0.02$
$(4, 5, 2)$	$\mathcal{C}_1 = \{1, 2, 3, 4, 5, 6, 7\}$	

Edge	$\alpha = 0.3, L = 3, w = [0.33, 0.33, 0.33]$	ρ
$(1, 2, 0)$	$\mathcal{C}_1 = \{1, 2\}$	
$(2, 3, 0)$	$\mathcal{C}_1 = \{1, 2\}, \mathcal{C}_3 = \{3\} \xrightarrow{\text{merge}(\mathcal{C}_1, \mathcal{C}_3)} \mathcal{C}_1 = \{1, 2, 3\}$	$\rho_{in}(\mathcal{C}_1) = 0.11, \rho_{in}(\mathcal{C}_3) = 0.0, \rho_{out}(\mathcal{C}_1, \mathcal{C}_3) = 0.06$
$(3, 4, 0)$	$\mathcal{C}_1 = \{1, 2, 3\}, \mathcal{C}_4 = \{4\} \xrightarrow{\text{merge}(\mathcal{C}_1, \mathcal{C}_4)} \mathcal{C}_1 = \{1, 2, 3, 4\}$	$\rho_{in}(\mathcal{C}_1) = 0.07, \rho_{in}(\mathcal{C}_4) = 0.0, \rho_{out}(\mathcal{C}_1, \mathcal{C}_4) = 0.04$
$(4, 1, 0)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}$	
$(5, 6, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6\}$	
$(6, 7, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6\}, \mathcal{C}_7 = \{7\} \xrightarrow{\text{merge}(\mathcal{C}_5, \mathcal{C}_7)} \mathcal{C}_5 = \{5, 6, 7\}$	$\rho_{in}(\mathcal{C}_5) = 0.11, \rho_{in}(\mathcal{C}_7) = 0.0, \rho_{out}(\mathcal{C}_5, \mathcal{C}_6) = 0.06$
$(7, 5, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	
$(4, 5, 0)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	
$(4, 5, 1)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	(no merge because α is high.)
$(4, 5, 2)$	$\mathcal{C}_1 = \{1, 2, 3, 4\}, \mathcal{C}_5 = \{5, 6, 7\}$	

Figure 8: Working demo of mCOMM.

Table 6: Statistics of real networks from SNAP datasets.

Graph	Type	Nodes	Edges	Average Degree	Ground Truth Communities
Amazon	Co-purchasing	334,863	925,872	2.76	311,782
DBLP	Co-citation	317,080	1,049,866	3.31	1,449,666
Youtube	Social	1,134,890	2,987,624	2.63	8,455,253
LiveJournal	Social	3,997,962	34,681,189	8.67	137,177
Orkut	Social	3,072,441	117,185,083	38.14	49,732

Table 7: Comparison with state-of-the-art algorithms in terms of 1) NMI 2) F1-Score 3) Weighted Community Clustering (WCC) 4) Execution Time (ET) (in seconds) and 5) Execution Memory (EM) (in GBs).

Graph/Algorithm	**Amazon**					**DBLP**					**Youtube**					**LiveJournal**					**Orkut**				
	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM
Infomap(Rosvall and Bergstrom, 2008)	.16	.31	.00	47.6	.56	.00	.09	.01	45.5	.60	.00	.01	.00	191.4	2.02	.01	.04	.00	2908.3	14.51	.00	.04	.00	4165.2	101.59
Leiden (Traag et al., 2019)	.20	.33	.01	21.8	.43	.11	.17	.00	24.6	.47	.04	.10	.00	77.6	1.49	.01	.18	.00	894.9	13.56	.01	.09	.00	3940.7	42.06
FastLPA (Traag and Šubelj, 2023)	**.28**	**.48**	**.20**	17.6	.29	**.13**	**.31**	**.18**	19.6	.32	.01	.05	.01	94.7	.99	.03	.05	.02	1765.21	7.81	.06	.17	.00	5902.9	22.69
Louvain (Blondel et al., 2008)	.14	.28	.01	93.5	.61	.06	.13	.01	202.9	.67	.00	.00	.00	436.4	2.19	.02	.08	.01	12111.4	15.02	-	-	-	-	-
Walktrap (Pons and Latapy, 2005)	.27	.44	.13	1291.5	.42	.10	.29	.16	2747.6	.47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SCODA (Hollocou et al., 2017)	.11	.37	.09	3.4	.19	.04	.22	.10	3.8	.21	**.06**	**.21**	.01	16.2	**.67**	**.06**	**.23**	.01	190.7	**4.96**	.16	.36	.00	**695.9**	**14.39**
SAOCD (Li et al., 2020)	.14	.38	.10	8.2	.23	.06	.24	.12	9.3	.27	.03	.15	.00	31.3	.80	.02	.13	.01	350.4	8.23	.06	.29	.00	1677.3	27.19
Proposed mCOMM	.16	.40	**.20**	**3.2**	**.18**	.09	.29	.16	**3.67**	**.20**	.05	.20	**.02**	**15.4**	.68	.03	.16	**.03**	**183.2**	6.34	**.28**	**.43**	**.02**	711.9	17.83

the same community, the edge count within the community is increased. If they belong to different communities, the edge count between communities is increased. If only one of the nodes is part of an existing community, a new single-node community is created with the remaining node as its representative. If both nodes are not part of any community, a new community with two nodes is created, with the node having the lower value assigned as the community representative and the other node as its child node. The sparse triangular matrix mat is updated with the information of the newly formed or modified community.

mergeCommunity(P_{n_1}, P_{n_2}): This method merges communities represented by P_{n_1} and P_{n_2} based on the size of the communities, and the smaller community is linked to the larger community representative. The sparse triangular matrix mat is updated after the merge, and entries of the merged community are removed and references are updated to the new community.

H.2 Working Demo of mCOMM

We show the working of mCOMM (Figure 8) in a streaming setting, highlighting the edge merging process under different values of α (merging parameter that controls merging in mCOMM). The figure demonstrates the relation graph where there can be edges between two nodes at different layers. The value on the edges denotes the layer in which it belongs. We consider three layers $L = \{0, 1, 2\}$. The table details the edge merging process under different $\alpha = \{0.1, 0.3\}$ values. For $\alpha = 0.1$, it can be observed that merging occurs at edge $(4, 5, 1)$, but no merging occurs for $\alpha = 0.3$. It is evident from the Figure that α values play a crucial role in the merging of communities, which results in different outcomes for the same edge stream. Specifically, we can see that for $\alpha = 0.1$, merging occurs at edge $(4, 5, 1)$, but there is no merging when we set the alpha value to $\alpha = 0.3$.

H.3 Convergence of mCOMM

We have designed mCOMM for streaming community detection, and it only terminates at the end of the stream. For each incoming edge, we make a deterministic decision to put its nodes in a particular community (which is highlighted in detail in the working example of Figure 8), always ensuring a well-separated community for a query (onQuery() function in the Algorithm 1) at any time step in the stream. The same is empirically visible in terms of modularity in Figure 9. This shows that mCOMM always converges to a set of communities.

H.4 Choice of α for mCOMM

We have already done an ablation study (Section 4.1) showing the change in the results of mCOMM (NMI, AMI, ARI) with the change in α. We had mentioned that α determines when we should merge communities. In the case of the Twitter datasets used here, sometimes two tweets talking about the same events have weak connections with just a single common word. We have used a low α value of 0.5 in the case of our experiments to accommodate such weakly connected nodes in a community. This does not affect nodes that are strongly connected, as they will be merged anyway. This idea works well with the Twitter dataset, as is indicated in Figure 4 of the paper. It must be noted, though, that setting α too low causes a dip in the NMI value. In case we set α to 0, we get connected

components as communities. This is because a 0 value of α concatenates all connected nodes into a single community by eliminating the merging restrictions. We obtain a balanced result across the metrics with α set to 0.5.

H.5 Performance of mCOMM on varied datasets

We show the performance of mCOMM in terms of various metrics for various homogeneous datasets from SNAP (Leskovec and Krevl, 2014) like Amazon, DBLP, Youtube, LiveJournal and Orkut. The dataset statistic is shown in Table 6. We can see from the Table 7 that mCOMM has comparable performance for Amazon in NMI and F1 and best result for WCC (Weighted clustering co-efficient), ET (Execution Time in seconds) and EM (Execution Memory in GBs) for the Amazon dataset. We have the best result for ET and EM in DBLP and the best result for WCC and ET for the LiveJournal dataset. For the Orkut dataset we have the best result for NMI, F1 and WCC. It must be noted that we have comparable results for all the metrics across the datasets in cases when we do not have the best results.

Intuition for using density in mCOMM: Pure modularity-based measures (Louvian) suffer from problems like the resolution limit problem, which density-based approaches avoid (Kim et al., 2022). This also restricts modularity-based methods to detect smaller communities that exist in the dataset, as shown in Fig. 3 in FinEvent (Peng et al., 2023).

H.6 Theoretical Analysis of mCOMM

Lemma H.1. *Given a heterogeneous multilayer graph stream $S = \{e_1, e_2, \ldots, e_t\}$, where each edge $e = (i, j, l)$ arrives in some layer l, the modularity Q is defined based on Equation (2) from (Hanteer and Magnani, 2020) as:*

$$Q = \frac{1}{2\mu}\left[\underbrace{\sum_{C\in\mathcal{C}}\sum_{\substack{i_s,j_s\in C\\ s\in l}}(A^s_{ij}-P^s_{ij})}_{\textit{Intralayer contribution}} + \underbrace{\sum_{C\in\mathcal{C}}\sum_{\substack{i_s,j_r\in C\\ s,r\in l}} W^{sr}_{ij}\cdot\delta(i_s,j_r)}_{\textit{Interlayer contribution}}\right]$$

where, A^s_{ij} is the adjacency matrix in layer s. P^s_{ij} is the null model in layer s, e.g., $\frac{k^s_i k^s_j}{2|\mathcal{E}_s|}$. W^{sr} is the coupling matrix that describes the interlayer edges between layers s and r. The term W^{sr}_{ij} corresponds to the coupling strength ω_{ij}, which is the weight assigned to the interlayer edge connecting node i in layer s and node j in layer r. $\delta(i_s, j_r) = 1$ if i_s and j_r refer to the same actor, otherwise it equals 0. $\mu_t = \sum_{s\in l} 2|E_s| + \sum_{i,s\neq r}\omega^{sr}_i$. $\mathcal{C}$ is the community structure. Find the change in modularity Δ_{Q_t} at time t.

Proof: *Let us derive the change in modularity Δ_{Q_t} incrementally:*

***At** $t = 0$**:** No edges exist.*

$$Q_0 = 0;\ \Delta Q_0 = 0$$

***At** $t = 1$**:** Edge $e_1 = (i, j, s)$ arrives. Edge appears in layer s. No interlayer contribution is present. Only intralayer contribution exists.*

$$\begin{aligned} Q_1 &= \frac{1}{2\mu_1}\left[(A^s_{ij}-P^s_{ij})\right];\\ \Delta_{Q_1} &= Q_1 - Q_0\\ &= \frac{1}{2\mu_1}(A^s_{ij}-P^s_{ij}) - \Delta_{Q_0} \end{aligned}$$

***At** $t = 2$**:** Now we consider different scenarios occur in mCOMM depending on the nature of the second edge e_2.*

Case 1:$e_2 = (i, k, s)$- Only one node exists (Same node, same layer)
Intralayer contribution: $(A^s_{ij} - P^s_{ij}) + (A^s_{ik} - P^s_{ik})$
Interlayer contribution: 0

$$\begin{aligned} Q_2 &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^s_{ik}-P^s_{ik})\right]\\ \Delta_{Q_2} = Q_2 - Q_1 &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^s_{ik}-P^s_{ik})\right] - \frac{1}{2\mu_1}(A^s_{ij}-P^s_{ij})\\ &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^s_{ik}-P^s_{ik})\right] - \Delta_{Q_1} - \Delta_{Q_0} \end{aligned}$$

Case 2: $e_2 = (i, k, r)$- Only one node exists (Same node, different layer). Nodes i spans layers s and r, introducing interlayer contribution for i.
Intralayer contribution: $(A^s_{ij} - P^s_{ij}) + (A^r_{ik} - P^r_{ik})$
Interlayer contribution: $\omega^{sr}_{ii}.\delta(i_s, i_r) = \omega^{sr}_{ii} \because \delta(i_s, i_r) = 1$

$$\begin{aligned} Q_2 &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^r_{ik}-P^r_{ik}) + \omega^{sr}_{ii}\right]\\ \Delta_{Q_2} = Q_2 - Q_1 &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^r_{ik}-P^r_{ik}) + \omega^{sr}_{ii}\right] - \frac{1}{2\mu_1}(A^s_{ij}-P^s_{ij})\\ &= \frac{1}{2\mu_2}\left[(A^s_{ij}-P^s_{ij}) + (A^r_{ik}-P^r_{ik}) + \omega^{sr}_{ii}\right] - \Delta_{Q_1} - \Delta_{Q_0} \end{aligned}$$

Case 3: $e_2 = (p, q, s/r)$- *Both nodes do not exist (New node-pair in same/different layer)*
Intralayer contribution: $(A^s_{ij} - P^s_{ij}) + (A^{s/r}_{pq} - P^{s/r}_{pq})$
Interlayer contribution: 0

$$Q_2 = \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^{s/r}_{pq} - P^{s/r}_{pq})\right]$$

$$\Delta_{Q_2} = Q_2 - Q_1 = \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^{s/r}_{pq} - P^{s/r}_{pq})\right] - \frac{1}{2\mu_1}(A^s_{ij} - P^s_{ij})$$

$$= \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^{s/r}_{pq} - P^{s/r}_{pq})\right] - \Delta_{Q_1} - \Delta_{Q_0}$$

Case 4: $e_2 = (i, j, r)$- *Both nodes exist (Same node-pair, different layer). Now nodes* i *and* j *span layers* s *and* r*, introducing interlayer contribution for* i *and* j*.*
Intralayer contribution: $(A^s_{ij} - P^s_{ij}) + (A^r_{ik} - P^r_{ik})$
Interlayer contribution: $\omega^{sr}_{ii}.\delta(i_s, i_r) + \omega^{sr}_{jj}.\delta(j_s, j_r) = \omega^{sr}_{ii} + \omega^{sr}_{jj} \because \delta(i_s, i_r) = \delta(j_s, j_r) = 1$

$$Q_2 = \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^r_{ij} - P^r_{ij}) + \omega^{sr}_{ii} + \omega^{sr}_{jj}\right]$$

$$\Delta_{Q_2} = Q_2 - Q_1 = \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^r_{ij} - P^r_{ij}) + \omega^{sr}_{ii} + \omega^{sr}_{jj}\right] - \frac{1}{2\mu_1}(A^s_{ij} - P^s_{ij})$$

$$= \frac{1}{2\mu_2}\left[(A^s_{ij} - P^s_{ij}) + (A^r_{ij} - P^r_{ij}) + \omega^{sr}_{ii} + \omega^{sr}_{jj}\right] - \Delta_{Q_1} - \Delta_{Q_0}$$

$\dots$

Furthermore, $\Delta_{Q_3} \cdots \Delta_{Q_{t-1}}$ *will be derived recursively using the previous changes in modularity. Each new edge will fall into one of the cases discussed above for* $t = 2$*.*
$\therefore$ *The modularity change at time* t *is:*

$$\Delta_{Q_t} = \frac{1}{2\mu_t}\left[\sum_{C\in\mathcal{C}}\sum_{\substack{i_s, j_s \in C \\ s\in l}}(A^s_{ij} - P^s_{ij}) + \sum_{\substack{i\in\mathcal{V}_t \\ s\neq r}}\omega^{sr}_{ii}\right] - \sum_{\tau=0}^{t-1}\Delta_{Q_\tau} \tag{14}$$

$$\mu_t = \sum_{s\in l} 2|E^t_s| + \sum_{\substack{i\in\mathcal{V}_t \\ s\neq r}}\omega^{sr}_{ii}$$

Remark H.1. *Considering the mCOMM algorithm, if both nodes from an incoming edge do not exist (irrespective of the layer), then we fall back to the modularity* (Q_1) *at* $t = 1$*. This will always be the condition on the arrival of the first edge. Now there are four possibilities. In the first possibility, one node may already exist (in any layer), and the other may be a new node in the same layer. In that case, the modularity can be calculated from Case 1 at* $(Q_2, t = 2)$*. The second possibility can be an edge between two nodes where one node exists, say in layer* s*, but the new node is in layer* r*. In this case, the modularity can be calculated from Case 2 at* $(Q_2, t = 2)$*. In the third possibility, both nodes can already exist, say in layer* s*, but the edge between them is in another layer, say layer* r*. In this case, the modularity can be calculated from Case 4 at* $(Q_2, t = 2)$*. The final possibility is an edge between two new nodes in the same or different layer, where the modularity can be represented using Case 3 at* $(Q_2, t = 2)$*. Specifically, for possibility 1,* $\Delta_{Q_2} = 0$ *as the new node will always be assigned to a new community, therefore* $(A^s_{ik} - P^s_{ik}) = 0$*. Similarly, in the case of possibility 2,* $(A^r_{ik} - P^r_{ik}) = 0$*, as* i *and* k *will always belong to different communities. So,* $\Delta_{Q_2} = \omega^{sr}_{ii}$*. One must note that* $\omega^{sr}_{ii} = 0$ *in our case, as we do not consider any coupling between the same nodes in two different layers.*

H.7 Practical analysis of change in modularity for mCOMM.

We conduct an empirical study to show the change in modularity of mCOMM for block B_4 of the Event2012 dataset. In the Figure 9, the x-axis is indicative of the number of incoming edges. Initially, there is a decrease in modularity due to the presence of isolated communities. However, we can see that after 20000 incoming edges, the merge operations start, and we get an increase in the modularity of the communities, indicating that the proposed algorithm generates well-separated communities. We can see that the change in modularity is well supported by the Lemma explained above.

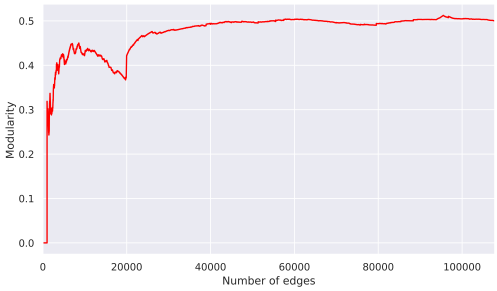


Figure 9: Modularity based analysis of mCOMM on B_4 block of Event2012 dataset.