

# Shape-Aware, Scale-Agnostic Representation of Dynamic DAGs

Jennifer Neumann  
University of Augsburg  
Augsburg, Germany  
jennifer.neumann@uni-a.de

Peter M. Fischer  
University of Augsburg  
Augsburg, Germany  
peter.m.fischer@uni-a.de

## ABSTRACT

In this work, we present a vector representation that is built around the notion of interpretable, domain-specific shapes that describe the structure of a whole graph regardless of its size. This representation overcomes common restrictions of popular graph embedding approaches and can be computed efficiently on static and dynamic graphs. We apply this representation to classify the graph structures in information cascades and trace their changes over time. Our experimental study on real-life and synthetic datasets shows both performance results and empirical observations. Although we currently focus on append-only directed acyclic graphs (DAGs), the underlying ideas may be generalized to a wider range of graphs.

### VLDB Workshop Reference Format:

Jennifer Neumann and Peter M. Fischer. Shape-Aware, Scale-Agnostic Representation of Dynamic DAGs. VLDB 2025 Workshop: LSGDA.

## 1 INTRODUCTION

Graph embeddings have become the mainstay in representing graph characteristics for classification, clustering, and other ML tasks. In addition to the compact representation, their main appeal is capturing essential graph characteristics while avoiding the effort of manual feature engineering. In domains such as social media analysis or causal graphs [22], important limitations become apparent but are rarely addressed - especially not in combination:

- Domain-specific **meaningful structures** are not expressed well by generic embeddings, providing coarse information.
- The **positions** of specific **parts** are not (or rigidly) described, limiting the means to capture the overall structure.
- Edge **directions** are typically not used, depriving the representation of important aspects.
- **Black-box** features obstruct **interpretable** analyses.
- Variances in **graph sizes** are prevalent due to power-law distributions but not supported well, as many embeddings assume almost the same size for graph instances.
- **Dynamic** graphs are captured in a rudimentary fashion, typically as **static embeddings on snapshots**.

Our main showcase for these issues are Information Diffusion Cascades (see Figure 1) that capture the spread of information [19, 34] (similar to epidemiological models). Starting from one or a few original sources, the content spreads gradually (**dynamic graph**) to new nodes (**direction**). The diffusion may form **structures** such

as dense superspreading groups or long chains, while the **relative position** provides insight into the effects. The **sizes** of the overall cascades and the structures vary significantly.

So for a given set of graphs  $\mathcal{G}$  containing DAGs of various sizes, our goal is to generate vector representations of every graph  $G \in \mathcal{G}$  that preserve direction, temporal information, and the similarity of graphs with respect to their complete structure. They must encode positional information for certain structural subpatterns and their relevance to the overall structure of  $G$ , but ignore size differences (compared to other graphs in  $\mathcal{G}$ ).

In this work, we present this **whole-graph vector representation** in Section 3) and introduce algorithms for both **static** and **incremental** computation in Section 4 that are fast enough on typical graph sizes to produce representations for **every time point**, enabling fine-grained graph-temporal analyses. As a proof-of-concept, we **group** complex **real-life** graphs of **varying sizes** into **interpretable categories** and present first steps towards change point and temporal evolution studies in Section 5. An experimental study in Section 6 shows both performance results and the empirical observations on the applications.

## 2 RELATED WORK

The purpose of graph vector representations is to provide a notion of distance or similarity between pairs of graph instances, which is the foundation of many ML applications. This is particularly challenging, as traditional **graph matching algorithms** such as (sub)graph isomorphism and associated **Graph Distance Metrics** like Graph Edit Distance (GED) or Maximum Common Subgraph (MCSG) are mostly NP-Hard ([5, 17]).

### 2.1 Graph embeddings

Graph embeddings are general approaches that do not rely on domain-specific knowledge to generate low-dimensional vectors that preserve similarity properties. In this work, we focus on whole-graph embeddings, not individual node or subgraph embeddings.

Many complementary approaches for graph embedding have been proposed [6]: **Matrix factorization** methods generate embeddings using the adjacency matrix of the graph, yet their computational complexity limits them to small graphs; e.g., MagNet [33] encodes directed graphs using a complex Hermitian matrix formulation. **Random walk**-based methods construct embeddings by sampling a set of paths from the graph, with the quality strongly dependent on starting nodes and walk lengths, both of which are difficult to adapt to the specific structure and size [11]. **NN**-based methods employ various neural architectures and a mechanism to encode neighborhood information, but suffer from similar scaling limitations. Examples include GCN and GCKM, which rely on feature transformation [31], as well as message-passing approaches,

such as GNNs [25] and UGraphEmbed [2] using sampled node embeddings to generate whole-graph representations. **Graph kernels** count occurrences of elementary fixed substructures; these can be based on random walks or random subgraphs of fixed size [14], neither can capture structures that scale with the size of the graph.

Methods tailored to **directed** graphs are mainly based on adjacency matrices or targeted on attributed or knowledge graphs. **Position-aware** approaches are usually limited to (single) node embeddings, such as [32], or rely on random walks within undirected graphs [7]. Positional and structural encoding (PSE) approaches incorporate the positions of the nodes as input features [27], but in a very rigid and overly detailed manner (e.g., 'the second child of the first child of the node  $x$ '). Almost all embeddings of **dynamic graphs** use techniques designed for static graphs applied to snapshots, mostly due to cost [3].

## 2.2 Graph Summarization

The large size and complex structure of graphs often prohibit directly computing NP-Hard distance measures. Reducing the graph to 'interesting' parts may improve the scalability, but the definition is subjective, usually requiring both domain knowledge and user preferences [16]. [13] comes closest to our needs by capturing typical shapes, but is restricted to undirected edges and only captures single graphs, thus hindering comparison.

## 2.3 Structures in Large Graphs

A key element for graph summarization is the choice of 'interesting' shapes: specific enough to capture domain semantics, but not over-specialized. They should be cheap to compute, yet allow composition into larger patterns. At the one end of the resolution spectrum, **Motifs** [15] provide a detailed local view: they are predefined, very small (typically 3-5 nodes), connected subgraphs that can be computed efficiently and approximate the overall structure by counting or sampling. Due to their fixed size, they cannot effectively capture structural patterns that scale with the size of the graph. On the other end of the spectrum, **community detection** provides a wider coverage, but usually with limited internal structure. This is particularly apparent for popular density-based methods, such as Louvain [29] or Label Propagation [9], while pattern-based [18] approaches uncover motif-like structures with mechanisms such as random walks [26]. In recent years, the field has seen a shift towards encoding strategies [12], leading to the same limitations outlined in the previous section.

## 2.4 Analyses of Dynamic Graphs

Understanding how graphs change over time provides valuable insight into the underlying processes (e.g., Protein-Protein Interactions [8], mass mobility [30], information cascades [34]).

**Change points** correspond to moments at which the representation of the graph of the evolving structure changes significantly [35]. Generally, change point detection is applied to a sequence of coarse graph snapshots, and depending on the underlying dynamic graph model, such as Markov models, various detection methods have been developed. In our approach, all the change point detection techniques applicable to (relational) time series [1] data can be employed on the fine-grained sequence of vector representations.

The detection of **change paths** extends the analysis to the entire data set, mining the sequence of labels and detecting meaningful migration patterns. Often, determining this sequence of labels is expressed as a cluster evolution analysis, such as [24]. Similar approaches are used in analyzing and predicting human behaviors over time, like students [23] and customers [20].

## 3 ENCODING APPROACH

Considering that existing approaches do not fit our requirements, we introduce a vector representation for domain-specific shapes over directed graphs, which are assembled into larger, scale-free patterns with positional information. For dynamic graphs, the same definitions of shapes and patterns can be reused, so that for any given time point  $t$ , we create them on  $G_t$  as the subgraph that contains all nodes and edges that were active at time  $t$ . For now, we focus on *single-rooted directed acyclic graphs* (DAGs) and point timestamps in an append-only model.

### 3.1 Structural Shapes: Stars and Chains

Like motif-based approaches, we keep the concept of a predefined shape but relax the fixed size towards coverage of larger graph sections, using the shape properties and the relative positions as features. Our choice of shapes for social media and interaction graphs is influenced by [13], although the graph model makes **Stars** and **Chains** the most suitable **shapes**: Stars express the influence of a node in its (dense) neighborhood, while chains connect sections of the graph. Given its wider range of graph types, [13] also consider cliques (too expensive, limited applicability in our scope) and bipartite graphs (future work). Our shapes greedily "cover" (and thus summarize) their respective graph parts, avoiding over-weighting minor shapes and reducing computational effort. **Coverage** of a node is **unique**, unless it is at the **boundary** of shapes, where it is **fractional**. To **assemble complex structures** succinctly, we express the relationships between the shapes up to a certain level of detail, while [13] uses (random) subgraphs.

First, we establish some common definitions. For a given directed graph  $G=(V, E)$  with  $n := |V|$  and  $m := |E|$ , let  $\text{sp}(s \rightarrow x)$  the shortest (directed) path from  $s$  to  $x$ , while  $\text{len}(p)$  expresses the length of a path  $p$ . *Leaves* ( $L$ ) are nodes without outgoing edges, *roots* without ingoing. *Neighbors* of a node  $v$  are all nodes connected to  $v$  via an outgoing edge:  $\text{neighbors}(v) := \{w \in V | (v, w) \in E\}$ .  $\text{Indegree}(v)/\text{outdegree}(v)$  are the number of inbound/outbound edges to a node  $v$ .

Based on these definitions, we now establish our shapes. **Chains** are paths that contain only nodes with a limited outdegree (except when crossing a star). A **maximum chain** of a graph is a shortest path from the root to a leaf with maximum length:  $C_{\max} := \arg \max_{l \in L} \text{len}(\text{sp}(\text{root} \rightarrow l))$ . We only consider chains  $C_{\tau_{\text{chain}}}$  that reach a significant ratio ( $\tau_{\text{chain}}$ ) of  $C_{\max}$ :  $c = (c_{\text{start}} \rightarrow c_{\text{end}})$ , where  $\text{len}(\text{sp}(c)) \geq \text{len}(C_{\max}) * \tau_{\text{chain}}$ . The nodes **covered** by  $C_{\tau_{\text{chain}}}$  are exactly those that are **connected by the edges** of these chains.

Our second pattern, the **star**, consists of a **center**  $s$  **directly reaching a significant share** ( $\tau_{\text{star}}$ ) of the entire graph:

$$\text{out}(s) := \sum_{i \in \text{neighbors}(s)} \frac{1}{\text{indegree}(i)} \geq \max(\tau_{\text{star}} * |V|, 3) \quad (\star)$$

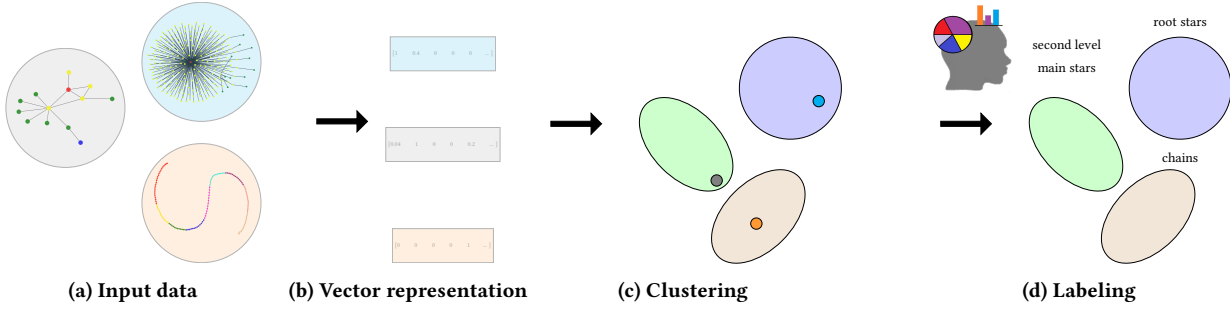


Figure 1: Overview: Use Static Shape Detection for Graph Classification

Stars have a more **complex coverage**: From the center  $s$ , the outgoing edges are considered **recursively** until a **leaf** or another star center is reached. To minimize **outliers**, all nodes that have a **distance greater** than the **average** distance of those considered are **excluded**:  $\text{len}(\text{sp}(s \rightarrow n)) > \text{avg}_{l \in R(s)} \text{len}(\text{sp}(s \rightarrow l))$  with  $R(s)$  are the leaves and star centers reached by  $s$  without visiting any other star centers or chains on the path (if  $R(s)$  is empty we use all paths to leaves and star centers covered by chains). If multiple stars reach a leaf or star centers without visiting other stars, we assign it to multiple stars.

Let  $SC_{\tau_{\text{star}}}$  be all nodes covered by the stars  $S$ . Using this approach, some nodes  $O$  are **not covered** by chains or stars, formally  $O = V - SC_{\tau_{\text{star}}} - C_{\tau_{\text{chain}}}$ . Depending on the nearest pattern, they are classified as **star tentacles** or **chain tentacles**. Let  $ST_{\tau_{\text{star}}} = \{o \in O \mid \arg \min_{i \in SC_{\tau_{\text{star}}} \cup C_{\tau_{\text{chain}}}} \text{sp}(i \rightarrow o) \in SC_{\tau_{\text{star}}}\}$  be star tentacles, and  $CT_{\tau_{\text{chain}}} = O - ST_{\tau_{\text{star}}}$  be chain tentacles.

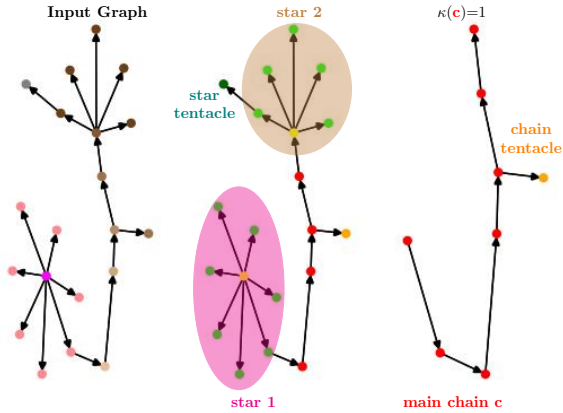


Figure 2: Graph shape detection using  $\tau_{\text{chain}} = 0.3$ ,  $\tau_{\text{star}} = 0.05$

In Figure 2 we show the result of this approach: All centers of stars  $s$  are marked **yellow**, nodes covered by stars **bright green**, nodes belonging to chains **red**, chain tentacles **orange**, and star tentacles **dark green**.

### 3.2 Vector Representation

Interpretable, structure-preserving features for graphs of different sizes are difficult to express in a dense, low-dimensional vector, so we aim for a compromise between accuracy, flexibility, and compactness. The detected shapes (chains, stars) of a graph are *each* explicitly taken into account in the representation, whereas tentacles are *summarized* because of their small overall impact.

Since the number of shapes and star levels varies among graph instances or within a dynamic graph, a *global* static number of dimensions is not feasible. Instead, we generate the representation specifically for a dataset, which also matches the use case. After detecting the shapes in each graph, we can determine the maximum number and sizes needed. For fully incremental evaluations, a local (e.g., window) scope is typically sufficient. Instance data are then filled in, while unneeded dimensions are padded with 0, leading to some sparse vectors. The number of dimensions can be controlled by  $\tau_{\text{star}}$  and  $\tau_{\text{chain}}$ , allowing for datasets with high structural complexity or variance. In our experiments, the number of dimensions was similar to state-of-the-art embeddings (128-256).

Expressing the structure *among shapes* requires similar trade-offs. Given our use cases and graph structures, we place more emphasis on the relationship *among stars*, as they represent stronger activity, while the chains lead to sparse areas with little structure. Instead of fully encoding a summary graph among star centers with full positions and distances, we only represent the *star positions* relative to the root(s) using *breadth-first* order. At each level, the stars are then *sorted by their size*. If the relation of stars is highly relevant, a possible direction is to use e.g. GED to compute the distance between star graphs (a reduced but weighted version of the given graph only containing the star centers and connecting edges between them). Special consideration is given to the interaction of stars, which may occur in triangle structures in DAGs with overlapping stars. The fractional coverage is resolved by assigning the nodes fully to the star closer to the root but retaining count of the reassigned nodes in the further nodes.

After the stars, we place the *chains* sorted by descending length. The vector is completed by the *tentacle* information. Although this order of components does not express the structure completely, it ensures that the component-wise comparisons in distance functions match the importance of the respective shapes.

To cater for size variance both in overall graphs and shape, we *normalize shape sizes* inside a graph *mostly*, but not fully: for stars,

we consider *not the full coverage*, but the number of *nodes directly connected to the center*, since this balances between stars of the *same node distribution*, but *different coverage size* vs. stars with the *same node distribution*, but *different coverage size*.

The normalization is bounded by the size of the largest shape  $s$ , which can be a star (from  $S$ ) or a chain (from  $C$ ):

$$d := \max(\max_{s \in S} \text{out}(s), \max_{c \in C} \text{len}(\text{sp}(c)))$$

with  $\text{out}(s)$  as defined in  $\star$ .

The scaling function  $\kappa : S \cup C \mapsto [0, 1]$  is defined as

$$\kappa(v) = \begin{cases} \frac{\text{out}(v)}{d} & \text{if } v \in S \\ \frac{\text{len}(\text{sp}(v))}{d} & \text{if } v \in C \end{cases}$$

The (scaled) individual shape types are represented as follows:

- For **chains** we use the (scaled) length.
- For **stars** we use the (scaled) number of nodes assigned to a closer star and nodes at every level off the center.
- **Tentacles** are summarized using the  $d$ -normalized average length and fraction of nodes for each tentacle type.

Using the approach on Figure 2 yields the following vector:

$$\text{VecFig2} = [0, 1.0, 0, 0.714, 1.0, 0.29, 0.05, 0.29, 0.05]$$

with  $\text{out}(\text{star } 1) = 7$ ,  $\text{out}(\text{star } 2) = 5$ , and  $\text{len}(\text{main chain}) = 7$  and therefore,  $d = 7$ ,  $\kappa(\text{star } 1) = 1$ ,  $\kappa(\text{star } 2) = 0.714$  and  $\kappa(\text{main chain}) = 1$ . The first two values represent **star 1** (reassigned, level 1), value 3 and 4 represent **star 2**, the fifth the single **chain** (length), and the last four represent the **star** and **chain tentacles** (avg. length, fraction).

## 4 SHAPE DETECTION ALGORITHMS

Although shape and pattern definitions carry a significant amount of complexity in their definition and interaction, they lend themselves well to efficient implementation without erasing finer structures or fracturing larger patterns. Our goal is to ensure that the resulting vector representation is **deterministic** while leaving sufficient flexibility during computation. In particular, we aim to make decisions about shapes in a **local** scope with minimized coordination, allowing for **greedy** computation of the cover.

To achieve these design goals, our algorithms perform a number of **well-defined steps** that capture the interaction of shapes so that none of them is overrepresented. Each step handles a **main type** and its remainder class, while the **order of steps** ensures that the interaction between shapes is consistent.

In general, the proposed algorithms achieve complete shape detection close to  $O(m)$ , which is optimal for an approach that is not based on sampling. To simplify the explanation, we start with a **single root node**, no deletions, and no shortcut edges with higher timestamp, but will lift these restrictions later.

### 4.1 Static Shape Detection

The first algorithm (Alg 1) covers the static case in which the **full graph** with no natural order of operations. This results in four steps, which we explain using the example in Figure 2. Since both chains and stars extend their respective cover greedily to the limit of the graph, the **first** step detects the **chains**, as otherwise the connection between the two star centers would be covered by **star 1** or become a star tentacle. By definition, chains fork off longer

chains, so we prepare our graph by computing the single-source shortest path from the root, which costs  $O(m)$  (BFS on unweighted graphs). Starting from the **longest** path (main chain), we assign chains as long as their length (from the last fork) is at least  $\tau_{\text{chain}}$ , which also costs  $O(m)$ . In practice, this total order can be relaxed, allowing for optimizations.

In the **second** step, **star** centers are detected by checking  $\star$  for each node. For each detected center, **coverage** is computed **independently** until either leafs, other star centers, or chains are reached. If two stars **overlap** (e.g. in a diamond structure), common nodes are labeled **fractionally**. In total, this step visits each edge at most as often as there is fractional cover, so we also achieve  $O(m)$ .

The **third** step performs a **clean-up** between chains and stars: If a chain runs through a cluster center, we retain it only if the **remaining length** is at least  $\tau_{\text{chain}}$ . In Figure 2, the main chain before this step ran to the node labeled "star tentacle", but after the "brown" star center it is too short. This step covers only a small section of the graph and can be run independently for each chain.

The **last** step ensures that all the **uncovered** nodes become **chain tentacles**, as any other shape would have consumed them.

---

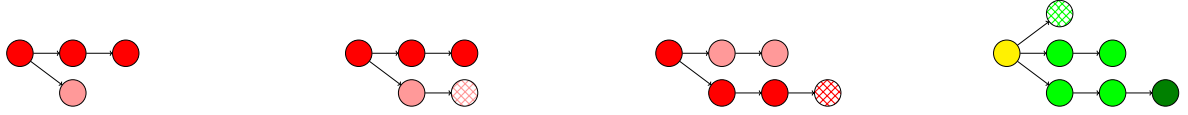
#### Algorithm 1: Shape detection

---

**Input:** Graph  $G = (V, E)$ , parameters  $\tau_{\text{chain}}, \tau_{\text{star}}$   
**Output:** Chains  $C^{\tau_{\text{chain}}}$ , Star Covers  $SC^{\tau_{\text{star}}}$ , Star tentacles  $ST^{\tau_{\text{star}}}$ , Chain Tentacles  $CT^{\tau_{\text{chain}}}$

---

- 1 **Step 1: Detect chain candidates;**
  - 2 initialized  $\leftarrow$  false;
  - 3 Sort all root-to-leaf paths in decreasing order of length;
  - 4 **foreach** path  $p$  **in sorted order** **do**
  - 5     **if not initialized** **then**
  - 6          $C_{\text{max}} = \text{len}(p)$ ; initialized  $\leftarrow$  true;
  - 7     **if**  $p$  **reaches an existing chain** **then**
  - 8         Truncate  $p$  at the (last) connection point;
  - 9     **if**  $\text{length}(p) \geq C_{\text{max}} \cdot \tau_{\text{chain}}$  **then**
  - 10         Add  $p$  to  $\hat{C}^{\tau_{\text{chain}}}$ ;
  - 11 **Step 2: Identify star centers and their cover;**
  - 12 Identify star centers  $S^{\tau_{\text{star}}}$  using  $\star$ ;
  - 13 Compute star covers  $SC^{\tau_{\text{star}}}$ , excluding paths in  $C^{\tau_{\text{chain}}}$ ;
  - 14 Add outliers shorter than  $C_{\text{max}} \cdot \tau_{\text{chain}}$  to  $ST^{\tau_{\text{star}}}$ ;
  - 15 **Step 3: Refine chains;**
  - 16 **foreach** chain candidate  $c$  in  $\hat{C}^{\tau_{\text{chain}}}$  **do**
  - 17     **if** ( $c$  satisfies  $\tau_{\text{chain}}$  condition from star center to leaf **or** between star centers) **and** covers at least one node **exclusively** **then**
  - 18         Keep  $c$  (if nodes at the end of  $c$  are already covered by stars truncate  $c$  and keep the remaining);
  - 19     **else**
  - 20         Discard  $c$ ;
  - 21 Merge chains at star centers if both types meet;
  - 22 **Step 4: Add chain tentacles;**
  - 23 Add remaining uncovered nodes to  $CT^{\tau_{\text{chain}}}$ ;
-



(a) Temporal graph with 4 nodes, containing a chain with 3 nodes and a sub chain with 1 node (b) Add a new node: Extend sub chain (c) Add a new node: After redirection split into a sub chain with 2 nodes and chain with 4 nodes (d) Add a new node: Creation of a new star (avg length 2) and a new star tentacle

Figure 3: Incremental shape detection: Recombination of chains and star creation.

## 4.2 Incremental Shape Detection

---

### Algorithm 2: Incremental shape detection

---

**Data:** Detected shapes for  $G_t = (V_t, E_t)$ , a new node  $n_{t+1} \notin V_t$ , and new edges  $e_{t+1}$ ; node depth (*depth*), graph depth ( $D_t$ ), and graph size ( $S_t$ ) of  $G_t$ ; parameters  $\tau_{star}, \tau_{chain}$

**Result:** Detected shapes for  $G_{t+1} = (V_t \cup \{n_{t+1}\}, E_t \cup e_{t+1})$

```

1 Step 1: Update metadata;
2  $D_{t+1} \leftarrow \max(D_t, \min_{(x, n_{t+1}) \in e_{t+1}} (\text{depth}(x) + 1))$ ;
3  $S_{t+1} \leftarrow S_t + 1$ ;

4 Step 2: Update old shapes;
5 for every star  $s$  of  $G_t$  do
6   if  $s$  does not satisfy  $\star$  for  $G_{t+1}$  then
7     Delete star  $s$  and update predecessor stars
8   or generate chains if no predecessor stars exists;

9 if  $\lfloor \tau_{chain} * D_t \rfloor \neq \lfloor \tau_{chain} * D_{t+1} \rfloor$  then
10  for every star  $s$  of  $G_t$  do
11    Recompute star cover of  $s$ ;
12  for chain  $c$  in  $G_t$  do
13    if  $\text{length}(c) < \lfloor \tau_{chain} * D_{t+1} \rfloor$  then
14      Relabel  $c$  to chain tentacle;

15 Step 3: Create new shapes;
16 for  $(s, n_{t+1}) \in e_{t+1}$  do
17   if  $s$  satisfies  $\star$  for  $G_{t+1}$  but not for  $G_t$  then
18     Create new star with center  $s$ 
19   and update predecessor stars of  $s$ ;

20 Step 4: Add new node;
21 Attach  $n_{t+1}$  to all possible stars and change the star cover if
   needed;
22 Attach  $n_{t+1}$  to longest possible chain;

23 Step 5: Recombination of chains;
24 if  $n_{t+1}$  is attached to a chain then
25   Recombine chains if needed ;

```

---

When computing the representation for dynamic graphs, we adopt an **incremental** strategy that utilizes the previously detected shapes of the graph. Each newly added node, along with all its incoming edges, is encoded to reflect its integration into the current graph state. Because most of the graph structure remains

**unchanged** after the addition of a new node, existing shapes can typically be reused. Furthermore, this method simplifies the identification of the necessary structural changes, as existing shapes retain their identity. Minimal updates by extending existing shapes are not always possible, causing both **deletion** of existing shapes and **creation** of new shapes. As the definitions of the *stars* and *chain tentacle* structures depend on both the size and depth of the graph, the addition of a new node can **retroactively** affect the validity of the shapes detected earlier. For example, previously identified stars may no longer satisfy the condition ( $\star$ ), necessitating their removal which may also affect chains. In turn, the addition of a node may prompt the formation of a **new** star structure. If a node is added to an existing chain, it must be ensured that this addition does not allow for the reconstruction of longer chains by recombining segments of previously identified chains.

The incremental approach begins with an initial chain between the first two arriving nodes. Algorithm 2 describes the steps when a new node and its incoming edges arrive: Step 1 updates the graphs metadata needed to compute thresholds. In Step 2, previously detected structures in the graph  $G_t$  are revised. Step 3 involves creating new stars if required, and Step 4 integrates the new node into the graph. Finally, Step 5 identifies and, if necessary, constructs maximal chains within existing chain structures.

Figure 3 shows the approach in a mini-example (using  $\tau_{chain} = 0.3$ ,  $\tau_{star} = 0.2$ ). The new node is marked with a cross-hatch. All nodes are color coded as in Figure 2, while nodes belonging to the same chain structure are marked with the same red tone.

In most cases, only Step 1 and 4 of the update procedure are executed, clearly leading to  $O(|e_{t+1}|)$  cost for the whole graph. The probability of invoking more expensive operations—Steps 2, 3, and 5—decreases as the graph size increases. Typically, all steps other than the ‘snowballing’ operation (Step 5) involve modifications with limited local scope and can be parallelized.

Since each node is assigned to very few shapes, computational complexity is not primarily determined by the number of nodes  $n$  or edges  $m$ , but rather by the number of shapes (usually  $\ll n$ ) because the update is performed by altering the given shapes. Specifically, let  $n_{star}$  denote the number of star shapes,  $n_{starten}(s)$  the number of tentacles in star  $s$ ,  $n_{chain}$  the number of chain shapes, and  $n_{chainten}$  the number of chain tentacles. The creation of a new star structure incurs a cost proportional to the size of the new star. If the center of this new star was previously assigned to another star  $s$ , an additional cost of  $O(n_{starten}(s))$  to alter the old star by adjusting or deleting its star tentacles if needed. Finally, the ‘snowballing’ step (5) has a worst-case complexity of  $O(n_{chain} + n_{chainten})$ .

### 4.3 Generalizing the Graph Model

The algorithms described are limited to append-only single-rooted graphs without the addition of new edges between existing nodes. We now sketch several approaches to overcome these limitations.

If we encounter a graph with **more than one root**, we introduce a **shadow root** as soon as a second node without incoming edges appears. From this shadow root, edges point to all *real* roots that are ignored during encoding. If the shadow root forms a star, care must be taken to ensure that it only covers the actual roots.

Removing a **leaf** node typically has only a **local** impact but may result in (multiple) structure changes. Deleting an edge or a non-leaf node or adding edges between existing nodes requires **relabeling** the entire graph using the static approach in order to obtain a new start encoding for subsequent steps (since it might destroy a shortest paths used to generate the old shapes).

## 5 APPLYING REPRESENTATIONS FOR GRAPH ANALYSIS

The interpretable vector representation serves as a promising basis for more elaborate downstream ML tasks. Continuous computation provides the means to detect **changes** at every modification, opening the way to state-of-the-art time series analysis techniques. We will provide brief proof-of-concept descriptions in three directions.

### 5.1 Determining Graph Structure Classes

Detecting **relevant groups** by clustering and classifying is a main driver of graph representations. Since the values of the absolute value of the individual dimensions have a measurable impact on discerning graphs, we chose the Euclidean distance. **Hierarchical clustering** with Ward linkage proved to be most effective and provides a good insight into the clustering decision. To assess quality and determine the best cluster count, we choose to use the Distance-based Separability Index (DSI) [10], as it can deal well with skewed cluster sizes and non-convex shapes at low computational cost.

The actual **class detection** requires a systematic human investigation, for which our interpretable and meaningful dimensions are useful. The path of feature “decisions” in hierarchical clustering highlights the properties of these groups and thus provides clues for descriptive **cluster labels**. Despite the inherent meaning of the components, the high dimensionality makes them hard to interpret for humans. We presented an **excerpt** of our vector representation to the expert, mainly summarizing the number, position, and score of the main shapes ( $\kappa > 0.85$ ) and the non-main stars.

### 5.2 Change Point Detection

Change point detection in graphs captures the moments in which the **structure** of a graph has changed significantly [35]. Our approach is particularly suitable, as it provides point-in-time resolution (instead of coarse snapshots) and makes the underlying structural modifications, including the **specific type** of change, explicitly accessible. This capability enables the generation of an initial set of labeled change points, which can be used for more elaborate ML tasks, such as training supervised models. As the detection is performed solely on the current graph instance, this lightweight approach is well-suited for timely analysis of dynamic or streaming graphs. To provide a first proof of concept, we utilize

specific change types to define our change points, based on the current **dominant structure** observed. For chain-dominant graphs, change points were identified through the creation of new chains or stars, whereas for star-dominant graphs, they were determined by the creation, deletion, or alteration of star covers.

### 5.3 Change Paths

We extract change paths using vector representations captured after change points or the average between successive change points and perform clustering as defined in Section 5.1. The change path for a graph is then defined as the **sequence of cluster assignments**, ordered chronologically. As a result, the generated change paths may vary in length depending on the number of detected change points for each graph. We analyze the common **evolution paths** of graph shapes using the **entire** dataset, rather than focusing on individual graphs, to compensate for sparsity and provide a uniform, representative model within a dataset.

## 6 EXPERIMENTAL EVALUATION

### 6.1 Setup and Data Sources

For our experimental study, we implemented a prototype as a single-threaded Python program and performed evaluations on an AMD Epyc 7702P running Linux. Following our motivational use case, we investigated information diffusion cascades from **Twitter/X** and **Telegram**, each covering their dominant means of information diffusion. For Twitter, we use *retweet* cascades reconstructed by the approach of [28], considering only cascades with at least 60 % completeness. As this approach relies on a social graph, we used datasets from the Olympics 2012 (London) to 2016 (Rio), where both messages and connections had been crawled.

For **Telegram**, we picked reply cascades, which occur only within groups. This is in contrast to the global Twitter model using hashtags for structure. Other diffusion means exist (even with the same frequency [21]), but are harder to reconstruct. Here, we use the publicly available Pushshift dataset [4] and analyze cascades from 200 randomly sampled channels.

In total, we investigated 18.8k Twitter cascades and 13.9k Telegram cascades (Table 1). The latter are always trees (due to the data model), while the former are mostly DAGs. The cascade sizes vary from min 6 nodes (our cut-off) to 23k nodes. On both platforms, an average cascade contains around 35–40 nodes. The cascades on Telegram are much deeper, whereas the cascades on Twitter contain more edges per node. To avoid the effect of outliers, we capped the depth values at 10 and re-scaled derived metrics accordingly.

As our real-world dataset includes cascades containing (only) up to 23k nodes, we also evaluated the computation time of our approach using **synthetic test data** with up to 20m nodes. These test graphs were specifically constructed to trigger particular types of alterations, such as the deletion of a star of a given size.

We conducted a (limited) parameter sensitivity analysis showing robust and predictable behavior. For  $\tau_{\text{star}}$  values between 0.03 and 0.1 provide good results, for  $\tau_{\text{star}}$  0.3 and 0.6, with a gradual decline outside these ranges. Higher values put more emphasis on larger structures and summarize more, whereas lower values achieve the opposite. We performed our experiments with 0.05 and 0.4, respectively, which was a good fit for all datasets.



	Twitter	Telegram
<b>number of cascades</b>	18817	13906
<b>tree casc</b>	6132	13906
<b>max casc size</b>	23028	896
<b>min casc size</b>	8	6
<b>avg casc size</b>	36.56	39.09
<b>max edges per node</b>	45.48	x
<b>min edges per node</b>	0.875	x
<b>avg edges per node</b>	1.33	x
<b>max depth</b>	10	702
<b>min depth</b>	1	3
<b>avg depth</b>	2.09	16.4

Table 1: Metadata of used real-world datasets

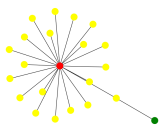
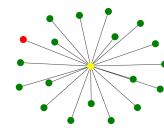
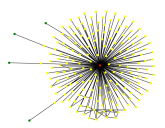
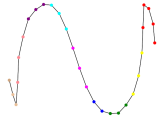
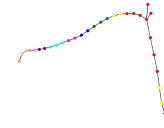

Input Graph	Considered equal	Considered different
	 matches stars centers but different meaning (none root stars)	 size difference but same meaning (root star)
	 matches chains without direction but different meaning (2 main conversations)	 size difference but same meaning (1 main conversation)

Table 2: Embedding Classification Results

## 6.2 Baseline: Embeddings

Given the dearth of approaches that provide at least parts of our approach (see Section 2.1), we compare our approach with UGraphEmbed [2] for undirected graphs with node ranking, but without detailed structures. None of the more recent work we considered provides runnable code or a better feature set/scalability. We followed the recommendation for train/test split and embedding dimension.

Given the graph sizes, we had to deviate from the suggested (and NP-hard) graph distance measure (GED), replacing with a heuristic that approximates the same idea: Maximum Number of Nodes ( $G_i, G_j$ ) - matching in the main structural pattern (outdegree for the main star or length for the main chain). Clustering these embeddings showed that the clusters were mainly formed by size, while each contained a mix of chains and star-based cascades, as well as stars at different levels (see Table 2).

These results did not change noticeably on a directed version of the heuristic (only allowing rooted chains and stars at the same root distance to match), since the node ranking of UGraphEmbed

cancels out these effects. Although finding suitable embeddings might be feasible, current approaches do not fit our problem.

## 6.3 Static Structure Classification

First, we assess the static approach by generating fine-grained clusters and discuss the observed patterns.

For the number of clusters, we determined them with DSI with 25 and 50 as bounds and 0.5 as a DSI score to separate clusters. This resulted in 39 clusters (DSI score 0.77). Analyzing the cluster features, we observe five dominant patterns (see Table 3):

- **Root-Star Cascades:** Star-shaped diffusion, where the most influential node is the root. Smaller secondary centers are possible, but create only small chains.
- **Non-Root-Star Cascades:** Similar to root-star cascades, but the root is not the center of diffusion.
- **Chain Cascades:** Conversations with message flow.
- **Chain Cascades with Stars:** A combination of small star-shaped structures and larger chains.
- **Mixed-Structure Cascades:** No single dominant pattern, instead multiple stars/chains that are equally important.

Fine-grained clusters generated by hierarchical clustering describe refined subclasses of the given patterns. For example, within the mixed structure cascades, subclasses include **galaxies** (cascades with multiple stars across different levels) and **stars of stars** (where several stars of similar size are influenced by a central star node).

When analyzing the number of fine-grained clusters per pattern, we observe that only a minority of the clusters only contain chain structures (4 out of 39 clusters). Given the dataset, this behavior is expected since only 3.3k cascades are based on a single main chain. Most clusters feature stars, as stars provide more complexity in terms of position, coverage, and structure.

## 6.4 Computation Time Analysis

As expected, operations that involve creating or deleting complex structures (stars) or altering multiple simple structures (star tentacles or chains) are computationally expensive.

**Real-world graphs** typically involve multiple types of alteration, as such changes are more likely in smaller graphs (since adding a new node to a small graph has a higher probability of triggering a costly structural update). The average time to update both the encoding description and shape detection is 51.36  $\mu$ s for tree-structured data and 52.62  $\mu$ s for DAGs.

For the **synthetic test graphs**, the worst-case scenario – designed to trigger the maximum number of recombinations upon insertion of the final node – resulted in an average processing time of 300  $\mu$ s for a graph containing 12.5m nodes and 5,000 recombinations, while the average processing time grows linearly with the root of the number of nodes. For all other synthetic test cases such as changing the star cover of a large star, the size of the graph has no significant influence on the average computation although individual operations can take several seconds (e.g., changing the star cover takes 16 seconds for a star with 10 million nodes), such cases are rare. Moreover, triggering such a costly operation on a large substructure requires a substantial number of basic append operations. Consequently, the average cost of updates remains low.

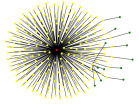
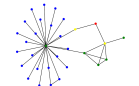
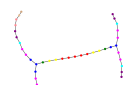
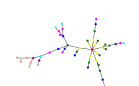
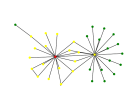
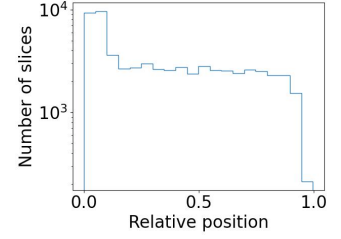
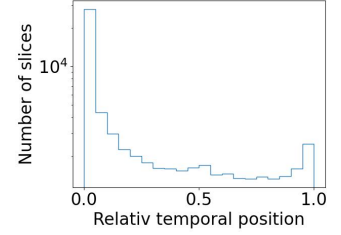
Pattern name	Root influence	Main shapes	Main chains	Main stars	Add. stars	Add. chains	Example	Clusters
Root stars	high	1	0	1	possible but rare	little to none (small)		11
Non-root stars	low	1	0	1	possible but rare	little to none (small to medium)		4
Chains	very low	1	1	0	very rare	Usually 2-4		4
Chains with stars	mostly low	1	1	0	always (size varies)	common		11
Mainshape mixes	depends on cluster	> 1	0-1	1-3	depends on cluster	depends on cluster		9

Table 3: Cascade Pattern Descriptions



(a) Position per Cascade



(b) Temporal position per Cascade

Figure 4: Change points

## 6.5 Change Points and Paths

We applied the proof-of-concept change detection based on the dominant structure as described in Section 5.2. The initial ‘setup’ phase of graphs with fewer than five nodes was excluded to avoid false detections. Using a window size of one (only comparing direct neighbor slices), we examined 1166k slices and identified 82k change points ( $\approx 7\%$  of the number of slices). The percentage of change varied considerably, ranging from zero – typically indicative of simple root star structures where the root star forms within the first five nodes and subsequently grows steadily – to around 30 %, which was often observed in Telegram cascades characterized by small substars that are frequently added and deleted. Ignoring the initial peak and the subsequent drop, the change points appeared almost equally likely across different cascade sizes (see Figure 4a). The temporal distribution of changes closely matches the expected activity of the information diffusion cascades, with high activity at the beginning that gradually slowed over time, and a final peak caused by very small cascades (see Figure 4b).

We utilize the encodings obtained after the selected change points to generate clusters for the **change path** analysis, employing the same setup as in the static approach. Due to constraints of time and space, we limit our findings to a brief description of the generated clusters and a simple cross-validation between the identified change points and cluster assignments. This aims to provide preliminary evidence that, in most cases, the identified change points indeed reflect significant structural transformations within the graph when compared to other graphs.

This approach produces clusters for similar shapes as described in 6.3, with the primary differences arising from the exclusion of early-stage changes. Cross-validation between generated change

points and clusters reveals that only 65 % of the change points correspond to an assignment in a different cluster. The reason is minor changes, such as the creation of a small star within a large cascade, that do not necessarily result in a different overall structure.

## 7 CONCLUSION AND FUTURE WORK

In this work, we present a shape-based vector representation for directed DAGs that can be computed efficiently on large-scale graphs. Our analyses show its utility for static and dynamic graph analyses.

There are multiple avenues for future work: Studying a wider range of **graph types** will provide a better understanding of the trade-offs on shape selection and representation. Given the promising results on dynamic graphs, incorporating more **expressive shapes** such as directed temporal cycles and fully overcoming the snapshot-based model seems promising. This opens up means to capture temporal dynamics within information cascades and to study underlying user interaction graphs in conversational settings.

Enhancing the model to support **frequent deletions** would improve its adaptability to real-time data changes. Furthermore, employing diverse activity models for **incremental encoding** may offer more accurate and flexible modeling of dynamic behaviors. Integrating advanced approaches for stream clustering may improve the representation of evolving data streams.

Finally, we plan to conduct a further **analysis of our real-world datasets** to find an optimal change point detection strategy and study graph evolution paths to **predict** future potential structural patterns or anomalies in detected shapes.



## REFERENCES

- [1] Samaneh Aminikhanghahi and Diane J Cook. 2017. A survey of methods for time series change point detection. *Knowledge and information systems* 51, 2 (2017), 339–367.
- [2] Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. 2019. Unsupervised inductive graph-level representation learning via graph-graph proximity. *arXiv preprint arXiv: 1904.01098* (2019).
- [3] Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. 2021. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–37.
- [4] Jason Baumgartner, Savvas Zannettou, Megan Squire, and Jeremy Blackburn. 2020. The pushshift telegram dataset. In *Proceedings of the international AAAI conference on web and social media*, Vol. 14. 840–847.
- [5] Horst Bunke. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern recognition letters* 18, 8 (1997), 689–694.
- [6] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering* 30, 9 (2018), 1616–1637.
- [7] Dongdong Chen, Yuxing Dai, Lichi Zhang, Zhihong Zhang, and Edwin R Hancock. 2023. Position-aware and structure embedding networks for deep graph matching. *Pattern recognition* 136 (2023), 109242.
- [8] Dongqi Fu and Jingrui He. 2022. DPPIN: A biological repository of dynamic protein-protein interaction network data. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 5269–5277.
- [9] Sara E Garza and Satu Elisa Schaeffer. 2019. Community detection with the label propagation algorithm: a survey. *Physica A: Statistical Mechanics and its Applications* 534 (2019), 122058.
- [10] Shuyue Guan and Murray Loew. 2020. An internal cluster validity index using a distance-based separability measure. In *ICTAI 2020*. IEEE, 827–834.
- [11] Zexi Huang, Arlei Silva, and Ambuj Singh. 2021. A broader picture of random-walk based graph embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 685–695.
- [12] Di Jin, Zhizhi Yu, Pengfei Jiao, Shirui Pan, Dongxiao He, Jia Wu, Philip S Yu, and Weixiong Zhang. 2021. A survey of community detection approaches: From statistical modeling to deep learning. *IEEE Transactions on knowledge and data engineering* 35, 2 (2021), 1149–1170.
- [13] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. 2015. Summarizing and understanding large graphs. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8, 3 (2015), 183–202.
- [14] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. 2020. A survey on graph kernels. *Applied Network Science* 5 (2020), 1–42.
- [15] Xiaodong Li, Reynold Cheng, Kevin Chen-Chuan Chang, Caihua Shan, Chenhao Ma, and Hongtai Cao. 2021. On analyzing graphs with motif-paths. *Proceedings of the VLDB Endowment* 14, 6 (2021).
- [16] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *ACM computing surveys (CSUR)* 51, 3 (2018), 1–34.
- [17] Lorenzo Livi and Antonello Rizzi. 2013. The graph matching problem. *Pattern Analysis and Applications* 16 (2013), 253–283.
- [18] Fragkiskos D Malliaros and Michalis Vazirgiannis. 2013. Clustering and community detection in directed networks: A survey. *Physics reports* 533, 4 (2013), 95–142.
- [19] Fanhui Meng, Jiarong Xie, Jiachen Sun, Cong Xu, Yutian Zeng, Xiangrong Wang, Tao Jia, Shuhong Huang, Youjin Deng, and Yanqing Hu. 2025. Spreading dynamics of information on online social networks. *Proceedings of the National Academy of Sciences* 122, 4 (2025), e2410227122.
- [20] Abdolreza Mosaddegh, Amir Albadvi, Mohammad Mehdi Sepehri, and Babak Teimourpour. 2021. Dynamics of customer segments: A predictor of customer lifetime value. *Expert Systems with Applications* 172 (2021), 114606.
- [21] Jennifer Neumann and Peter M Fischer. 2024. Multi-Means Analysis of Information Diffusion in Telegram. In *Companion Publication of the 16th ACM Web Science Conference*. 8–9.
- [22] Marco Piccininni, Stefan Konigorski, Jessica L Rohmann, and Tobias Kurth. 2020. Directed acyclic graphs and causal thinking in clinical risk prediction modeling. *BMC medical research methodology* 20 (2020), 1–9.
- [23] Satrio Adi Priyambada, Mahendrawathi Er, Bernardo Nugroho Yahya, and Tsuyoshi Usagawa. 2021. Profile-based cluster evolution analysis: Identification of migration patterns for understanding student learning behavior. *IEEE Access* 9 (2021), 101718–101728.
- [24] Roni Ramon-Gonen and Roy Gelbard. 2017. Cluster evolution analysis: Identification and detection of similar clusters and migration patterns. *Expert Systems with Applications* 83 (2017), 363–378.
- [25] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. 2020. Self-supervised graph transformer on large-scale molecular data. *Advances in neural information processing systems* 33 (2020), 12559–12571.
- [26] 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 (2008), 1118–1123.
- [27] Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. *Advances in neural information processing systems* 32 (2019).
- [28] Io Taxisidou and Peter M Fischer. 2014. Online analysis of information diffusion in twitter. In *23rd WWW*. 1313–1318.
- [29] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.
- [30] Tatiana Von Landesberger, Felix Brodkorb, Philipp Roskosch, Natalia Andrienko, Gennady Andrienko, and Andreas Kerren. 2015. MobilityGraphs: Visual analysis of mass mobility dynamics via spatio-temporal graphs and clustering. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 11–20.
- [31] Zhihao Wu, Zhao Zhang, and Jicong Fan. 2023. Graph convolutional kernel machine versus graph convolutional networks. *Advances in neural information processing systems* 36 (2023), 19650–19672.
- [32] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *International conference on machine learning*. PMLR, 7134–7143.
- [33] Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. 2021. Magnet: A neural network for directed graphs. *Advances in neural information processing systems* 34 (2021), 27003–27015.
- [34] Fan Zhou, Xovee Xu, Goce Trajcevski, and Kunpeng Zhang. 2021. A survey of information cascade analysis: Models, predictions, and recent advances. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.
- [35] Yuxuan Zhou, Shang Gao, Dandan Guo, Xiaohui Wei, Jon Rokne, and Hui Wang. 2025. A Survey of Change Point Detection in Dynamic Graphs. *IEEE Trans. on Knowl. and Data Eng.* 37, 3 (March 2025), 1030–1048. <https://doi.org/10.1109/TKDE.2024.3523857>