Information diffusion analysis in online social networks based on deep representation learning

Chen, X.

Chapter 5

Extracting Multi-scale Information for Cascades Modeling

5.1 Chapter Overview

Following the successes of deep learning methods in many fields, recently researchers have developed various neural network-based models to extract diverse features from the cascade graph that can be used for cascade prediction. For example, researchers have leveraged RNNs and attention mechanisms to automatically learn the cascade's temporal characteristics in a sequential learning manner by sampling the cascades as random walks or diffusion paths [8, 9, 35, 36]. These approaches, however, fail to capture topological structure features and the dynamic changes of information diffusion. Later studies [10, 69] introduce graph embedding methods to handle the structural diffusion learning problem, and have achieved promising results in cascade prediction. Despite the significant progress made by recent deep learning-based approaches, currently they are still confronted with several limitations:

(L1) Lack of efficient ways to sample cascade graphs. Most of the existing studies try to decompose a cascade graph into a bag of nodes [8] or denote it as a set of diffusion paths [9]. These methods either ignore the structural information or fail to capture the time-evolving structure of the cascade. CasCN [10] breaks down the original cascade graph into a sequence of sub-graphs based on timestamps, which may introduce bias and increase computation cost because there is a large number of timestamps in the diffusion process.

(L2) Incomplete structural feature extraction. Structural features are demonstrated as one of the most powerful features in information cascade prediction [10, 69]. Existing works not only capture nodes' first-order information but also take the edges' directional information into consideration. However, they still fail to capture long-distance message passing between nodes and nodes' position information in the cascade graph.

(L3) Absence of feature-level fusion. After obtaining different features from the cascade graph, e.g., temporal and structural features, most of the current works try to directly concatenate them and then fed them into a fully connected layer to make predictions [8, 9, 10]. However, different features play different roles in information cascades prediction, which necessitates a more fine-grained feature fusion that would facilitate predictions.

To overcome the limitations mentioned above, we first define multi-scale information for cascade graph, including (1) direction-scale, representing the propagating direction of the information between nodes; (2) high-order-scale, which is the higher-order interactions between the nodes; (3) position-scale, which means the sequential/position information of each node (i.e., the emerging time of each node in the diffusion); and (4) dynamic-scale, which is the dynamic information captured from the evolving sub-graph sequence. Then, we propose **MU**lti-scale **Cas**cades model (MUCas) – a novel framework for modeling the information cascades and predicting the increment size of information items. MUCas first employs time interval-aware sub-cascade graph sampling method, which decomposes the observed cascade graph into a sequence of sub-cascade graphs based on *disjoint* time intervals. And then it uses a multi-scale graph capsule network and an influence attention to learn and fuse the multi-scale information to form a unique cascade representation for popularity prediction.

We make the following contributions:

- **Efficient sampling method (L1).** We propose a novel cascade sampling method to sample sub-cascade graphs based on disjoint time-intervals. This method can significantly decrease the number of required sub-cascade graphs and eliminate the bias in processing the dynamic-scale in cascade modeling.
- **Multi-scale information learning (L2).** We design a multi-scale graph network for modeling sub-cascade graphs that can capture direction-scale, high-order-scale, and position-scale features of information diffusion jointly. Simultaneously, we design a neural function to learn the influence-attention between dynamic-scale sub-cascade graphs.
- **Hybrid feature aggregation (L3).** We propose a capsule-based hybrid aggregation layer, which selectively aggregates the learned multi-scale features in a more fine-grained way, i.e., from order-level and node-level to graph-level.
- **Comprehensive evaluations.** We conducted extensive experiments on two benchmark datasets. The results demonstrate that MUCas can significantly improve the prediction accuracy on cascade size prediction compared to the state-of-the-art baselines.

This chapter is based on the following publication:

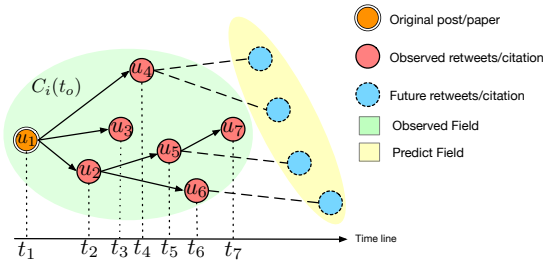


Figure 5.1: A Toy example of cascade graph.

- **Chen, X.**, Zhang, F., Zhou, F., Bonsangue, M.: Multi-scale graph capsule with influence attention for information cascades prediction. International Journal of Intelligent Systems 37 (2022) 2584–2611

5.2 Preliminaries

As illustrated in Figure 5.1, we can find that (1) the cascade graph is dynamic, in other words, cascade graph is evolving over time, i.e., new nodes will join in the diffusion process with time elapsed that leads the cascade size growth. For example, at t_7, u_7 joins in the diffusion process, and make the cascade size increase to 7; (2) the message between two nodes, e.g., u_1 and u_2, can be only passed from u_1 to u_2, i.e., the message passing in cascade graph is directed; (3) nodes are infected in chronological order, and the sequential information can be regarded as the position of each node in the cascade graph; (4) nodes with high influence will indirectly infect the long-distance nodes, and the long-distance dependency is the higher-order information of each node in a cascade graph, e.g., u_1 to u_6.

In this work, we define the aforementioned aspects as the multi-scale information, including (1) **dynamic-scale**, (2) **direction-scale**, (3) **position-scale**, and (4) **high-order-scale** of cascades.

Definition 9 ***Cascade Size Prediction*** – *Given the observed cascade graph $C_i(t_o)$ of post m_i, the goal of cascade size prediction is predicting m_i's incremental size ΔS_i, which is defined as $\Delta S_i = |U^{t_p}| - |U^{t_o}|$, where t_p and t_o are the prediction time and the observation time, respectively; and $|U^t|$ denotes the number of nodes, in terms of the size of cascade graph at time t.*

5.3 MUCas: Multi-Scale Graph Capsule with Influence Attention for Information Cascades Prediction

In this section, we present the proposed MUCas model, as well as its implementation details and computational complexity.

Table 5.1: Main notations used throughout this chapter.

Symbol	Description
$g_i^{T_j}, \mathbf{A}_i^{T_j}$	A sub-cascade graph of $C_i(t_o)$ at time interval T_j and it's adjacency matrix.
$G_i^{T_{iv}}, \mathbf{A}_i^{T_{iv}}$	A sequence of time interval-aware sub-cascade graphs of $C_i(t_o)$ and the corresponding adjacency matrices.
$\mathbf{PE}(u)$	Positional encoding.
$\mathbf{S}$, $\mathbf{H}_l$	A set of node embedding matrix, and a node embedding matrix.
$\mathbf{h}_m$ $\mathbf{n}_m$ $\mathbf{g}_i$	Order-level capsule. Node-level capsule. Graph-level capsule.
$\boldsymbol{\lambda}_j$, α_j	Time decay effect, influence attention.

Overview. Figure 5.2 illustrates the overall framework and the main components of **MUCas**, which consists of four major parts: (1) a time interval-aware sampling layer to generate sub-cascade graphs from observed cascade graph; (2) MUG-Caps to learn the direction-scale, position-scale and high-order-scale information from sub-cascade graphs; (3) influence attention for dynamic-scale learning; and (4) a prediction layer for cascade size prediction.

5.3.1 Time interval-aware Sub-cascade Sampling

Taking the observed cascade graph $C_i(t_o)$ of a given post m_i as input, the existing works try to decompose the observed cascade graph into a bag of nodes [8] or denote it as a set of diffusion paths [9]. Such methods either ignore both local and global structural information or fail to consider the dynamic information. Recently, some works such as CasCN [10] and VaCas [69] use a *Time-aware* sampling method to decompose $C_i(t_o)$ into a sequence of consecutive sub-cascade graphs based on the diffusion timestamp, which has been proved to be an efficient way to treat the observed cascade graph. However, the *Time-aware* sampling method still faces some challenges: 1) the difference between the fine-grained sub-graphs is trivial, which will introduce biases in dynamic modeling; and 2) too many sub-graphs would significantly increase computation cost. Figure 5.3a shows a toy example of *Time-aware* sampling method. Compared with the previous time step, each sub-graph only contains one more node (e.g., t_1 vs. t_2). Finally, it would generate z sub-graphs in total, where z is the number of varying time-stamps in the propagation process, resulting a huge number of sub-graphs within a short time. However, the difference between consecutive graphs are too trivial to be distinguished, which may confuse the model to learn discriminative features of information propagation.

In order to address the aforementioned challenges, we propose a new *Time interval-aware* sampling method, as shown in Figure 5.3b. This sampling method breaks down the observed cascade graph $C_i(t_o)$ into l discrete sub-cascade graphs $G_i^{T_{iv}} =$

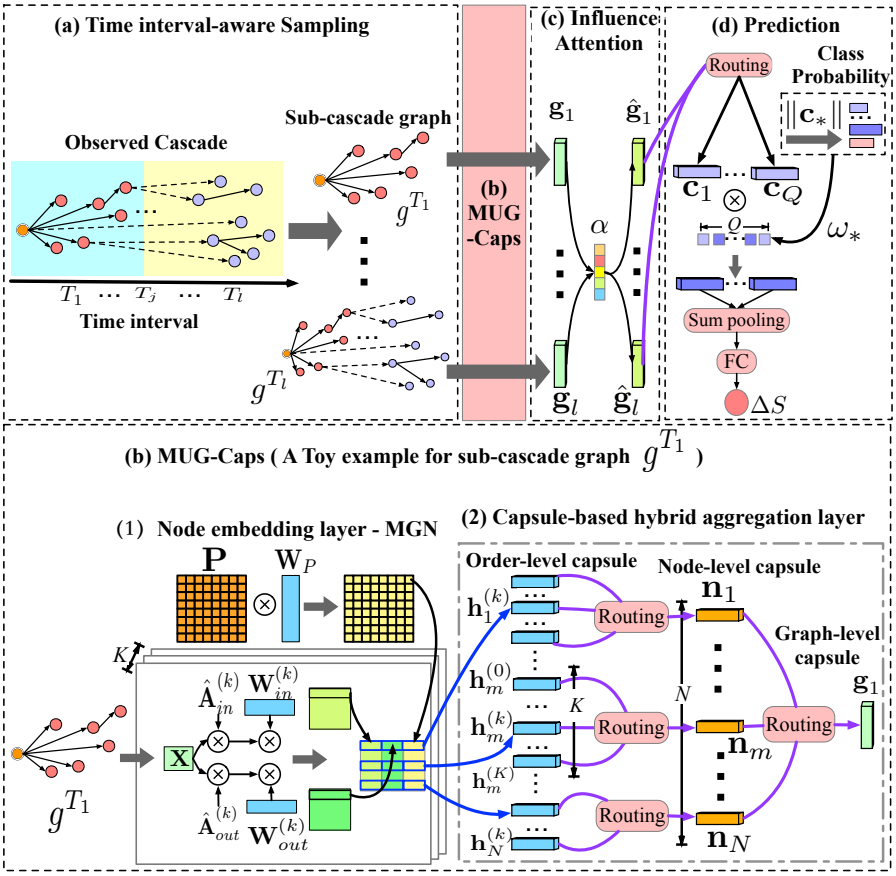


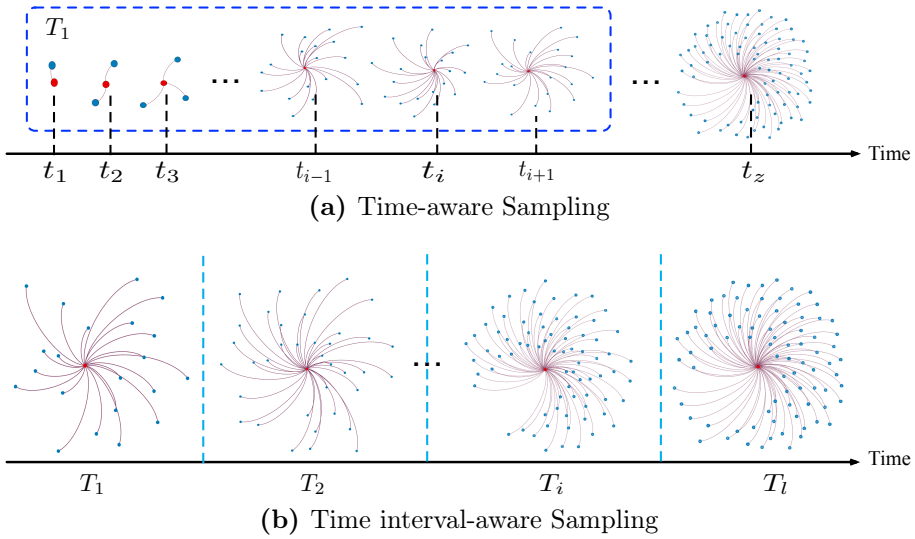
Figure 5.2: Overview of **MUCas**: (a) A time interval-aware sub-cascade graphs sampling layer; (b) MUG-Caps layer; (c) Influence attention layer; and (d) The prediction layer.

$\left\{g_i^{T_1}, \cdots, g_i^{T_j}, \cdots, g_i^{T_l}, |j \in [1, l]\right\}$. Specifically, we first split the observation time window t_o into l disjoint time intervals. Then, we sample sub-cascade graphs based on these intervals. Each sub-cascade graph in $G^{T_{iv}}$ is represented by an adjacency matrix. Thus, $G_i^{T_{iv}}$ is further represented as a sequence of adjacency matrices $\mathbf{A}_i^{T_{iv}} = \left\{\mathbf{A}_i^{T_1}, \mathbf{A}_i^{T_2}, \cdots, \mathbf{A}_i^{T_l}\right\}$.

Since the proposed sampling rule transforms the observed cascade graph into a fixed number of sub-graphs, it is possible that no new retweet/citation occurs in one of the intervals. To address this issue, we use the sub-graph of the interval before the empty one as padding to ensure that the final length of $G^{T_{iv}}$ equals l. Algorithm 3 formalizes the process of the *Time interval-aware* sampling method.

5.3.2 Multi-scale Cascade Representation Learning

After generating l discrete sub-cascade graphs, MUCas turns to learn high-level representation of these sub-cascade graphs, which contains the multi-scale information of cascade graphs. Inspired by the recent success of graph neural networks [41, 61, 126] and capsule network [127] in handling graph structured data,



(a) Time-aware Sampling

(b) Time interval-aware Sampling

Figure 5.3: Illustration of sampling sub-cascade graph sequence: time-aware sampling vs. time interval-aware sampling. z is the number of timestamps; l is the number of time intervals, and l is fixed and set manually.

we propose a MUlti-scale Graph Capsule Network (MUG-Caps) to learn the latent representation for cascade graph from $G^{T_{iv}}$. MUG-Caps is composed of two main parts, i.e., (1) the *node embedding layer* and (2) the *capsule-based hybrid aggregation layer*.

Node Embedding Layer: We propose a Multi-scale Graph Network (MGN) as the node embedding module, which learns node representations at the sub-graph level simultaneously from direction-scale, position-scale, and high-order-scale. The implementation of MGN is based on the graph convolutional networks (GCN) [61]. Original GCN proposes graph convolution approximations in the spectral domain based on graph Fourier transform, which is computationally efficient and achieves competitive performance in many tasks [128, 129]. However, it still faces some limitations in cascade modeling:

(1) GCN focuses on *static* and *undirected* graphs, whereas cascade graphs are *dynamic* and *directed* graphs.

(2) GCN updates a node's representation by aggregating its first-order neighbors and itself, failing to capture each node's infected order, i.e., the node's position information.

(3) GCN aggregates the high-order information for a node through stacking multiply graph convolutional layers. As demonstrated by many later improved works [126, 130], deeper GCN could not improve the performance and even performs worse in graph representation learning.

Our MGN addresses these limitations through revising the convolution kernel of

Algorithm 3: The algorithm for transforming cascade graph into a fixed-length sub-graph sequence: **Time interval-aware sampling**.

Input: Observed cascade graph $C_i(t_o)$, time window t_o, and time interval number l.

Output: A fixed-length sub-graph sequence $G_i^{T_{iv}} = \left\{g_i^{T_1}, \cdots, g_i^{T_l}\right\}$

```
for n = 1,2,...,l do
    for Set of nodes U_i(t_j), set of edges E_i(t_j) and corresponding timestamp t_j in C_i(t_o) do
        Compute the time interval index m = ⌊(t_j − t_1)/⌈t_o/l⌉⌋ + 1.
        if m ≤ n then
            Add U_i(t_j) and E_i(t_j) into g_i^{T_n}.
        end if
    end for
end for
```

GCN, which is defined as:

$$\mathbf{H} = g_\theta * \mathbf{X} = \sigma[\mathop{\Vert}_{k\in\mathcal{O}}, \mathop{\Vert}_{\phi\in\{in,out\}} (\hat{\mathbf{A}}_\phi^{(k)}\mathbf{X}\mathbf{W}_\phi^{(k)}), \mathbf{P}\mathbf{W}_P], \tag{5.1}$$

where $\mathop{\Vert}_{k\in\mathcal{O}}$ and $\mathop{\Vert}_{\phi\in\{in,out\}}$ represent the order-level concatenation and direction-level concatenation, respectively; $[\cdot]$ is a tiling concatenate operation; σ is an element-wise activation such as ReLU; $\hat{\mathbf{A}}_\phi^{(k)}$ denotes the normalized adjacency matrix $\hat{\mathbf{A}}_\phi \in \mathbb{R}^{N\times N}$ multiplied by itself k times; N is the number of nodes in current sub-cascade graph; $\mathbf{X} \in \mathbb{R}^{N\times F}$ is the input graph signal – F is the dimension number; and $\mathcal{O}$ is a set of integer adjacency powers – the value of $\mathcal{O}$ is from 0 to the max-order K of the current sub-cascade graph. $\phi \in \{in, out\}$ represents the in- and out-directions of the adjacency matrix, respectively. $\mathbf{A} = \mathbf{A}_{in} = (\mathbf{A}_{out})^T$. The asymmetric normalized adjacency matrix $\hat{A}_\phi$ of each direction can be calculated as:

$$\begin{aligned} \hat{\mathbf{A}}_\phi &= (\bar{\mathbf{D}}_\phi)^{-1}\bar{\mathbf{A}}_\phi, \\ \bar{\mathbf{A}}_\phi &= \mathbf{A}_\phi + \mathbf{I}_N, \end{aligned} \tag{5.2}$$

where $\mathbf{I}_N$ is the identity matrix, and $(\bar{\mathbf{D}}_\phi)_{ii} = \sum_j (\bar{\mathbf{A}}_\phi)_{ij}$ is the diagonal degree matrix.

In MGN, initially $\mathbf{X} = \mathbf{A}$. Further, $\mathbf{P} \in \mathbb{R}^{N\times F_p}$ is a position embedding matrix for current sub-cascade graph, and F_p is an adjustable dimension. Specifically, we initialize the position embedding matrix $\mathbf{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_u, \cdots, \mathbf{p}_N\}$ through positional encoding $\mathbf{PE}(u)$ [131] as:

$$\begin{aligned} \mathbf{PE}(u)_{2d} &= \sin(u/10000^{2d/d_p}), \\ \mathbf{PE}(u)_{2d+1} &= \cos(u/10000^{2d/d_p}), \end{aligned} \tag{5.3}$$

where $1 \leq d \leq F_p/2$ denotes the dimension index in $\mathbf{p}_u$. The details of this formula are referred to [131].

In Eq.(5.1), $\mathbf{W}_{\phi}^{(k)} \in \mathbb{R}^{F \times F_d}$ is the weight matrix for each direction on different order, and $\mathbf{W}_P \in \mathbb{R}^{d_p \times F_p}$ is another weight matrix used to transform position embeddings. The output node embedding matrix is denoted as $\mathbf{H} \in \mathbb{R}^{N \times K \times (2F_d + F_p)}$ and $\mathbf{H} = \{\mathbf{h}_1, \cdots, \mathbf{h}_m, \cdots, \mathbf{h}_N\}$, respectively. When implementing MGN, there is no need to calculate $\hat{\mathbf{A}}_{\phi}^{k}$ for each order. Instead, we calculate $\hat{\mathbf{A}}_{\phi}^{k}\mathbf{X}$ via right-to-left multiplication. For example, when $k = 2$, $\hat{\mathbf{A}}_{\phi}^{2}\mathbf{X}$ is calculated as $\hat{\mathbf{A}}_{\phi}(\hat{\mathbf{A}}_{\phi}(\mathbf{IX}))$, where $\mathbf{I}$ is the identity matrix. MGN can be regarded as a single layer using multiple times during actual training. The calculation of MGN is outlined in Algorithm 4.

The rationale behind MGN: MGN handles the **direction-scale** of cascade through modeling the incoming and outgoing relations of the cascades, i.e. $\hat{\mathbf{A}}_{\phi}^{(k)}\mathbf{X}\mathbf{W}_{\phi}^{(k)}$ in Eq. (5.1). Moreover, because the directed graph is asymmetric, we use the asymmetric normalization $\hat{\mathbf{A}}_{\phi} = (\hat{\mathbf{D}}_{\phi})^{-1}\bar{\mathbf{A}}_{\phi}$ to replace the symmetric normalization $\hat{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ used in vanilla GCN. MGN utilizes the adjacency matrix's multiple powers to mix the feature representations of higher-order neighbors in one graph convolutional layer via transitive closure, which is used to handle the **high-order-scale**. In our implementation, the sub-cascade graph is an acyclic directed graph, and the value of $\mathbf{A}_{ij}^{(k)}$ can be either 0 or 1. When $\mathbf{A}_{ij}^{(k)} = 1$, there is a path between v_i to v_j. For arbitrary power p and q, $\mathbf{A}_{ij}^{p}\mathbf{A}_{ij}^{q} = 0$ will always be held, which eliminates the problem of layer output – imposing the lower-order information on higher-order relations and increase the feature correlations [132]. As for the **position-scale**, MGN adds position embeddings to the convolution kernel, which enriches each node feature with its corresponding position information.

Capsule-based Hybrid Aggregation Layer From the node embedding layer, we obtain a set of node embedding matrix for each sub-cascade graph, denoted as $\mathbf{S} = \{\mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_l\}$, where $\mathbf{S} \in \mathbb{R}^{l \times N \times K \times (2F_d + F_p)}$. Inspired by the work of capsule networks [127, 133], we design a capsule-based hybrid aggregation layer to aggregate the learned node features from order-level, node-level, and graph-level through dynamic routing, respectively.

The general procedure of dynamic routing is shown in Algorithm 5: (1) Lower-level capsules $\mathbf{U} \in \mathbb{R}^{|\mathbf{U}| \times F_U}$ are linearly transformed through shared matrix $\mathbf{W} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times F_U \times F_S}$, where $|\mathbf{U}|$ and F_U are the number of lower-level capsules and the dimensions, respectively. Here we introduce $\mathbf{W}$ that not only guarantees the feature representation ability of the center vector after clustering, but also being able to identify the order of input features. The result of this step is a set of votes $\hat{\mathbf{V}} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times F_S}$ (cf. line 1 in Algorithm 5), where $|\mathbf{S}|$ and F_S are the number of upper-level capsule and the dimensions, respectively. (2) Upper-level capsules $\mathbf{S} \in \mathbb{R}^{|\mathbf{S}| \times F_S}$ are computed based on the votes via line 3–7 in Algorithm 5, where $c_{ij} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times 1}$

Algorithm 4: Calculation of multi-scale graph network.

Input: Feature matrix $\mathbf{X}$, normalized adjacency matrix $\hat{\mathbf{A}}_\phi$ for both in- and out-directions, a set of order powers $\mathcal{O}$ and its max-value $K = \max(\mathcal{O})$, and positional embedding matrix $\mathbf{P}$.

Parameters: $\{\mathbf{W}_\phi^{(k)}\}_{k\in\mathcal{O}}$ and $\mathbf{W}_P$.

$\hat{\mathbf{P}} := \mathbf{P}\mathbf{W}_P$
$\mathbf{B}_{in}, \mathbf{B}_{out} := \mathbf{X}$
for k in $\mathcal{O}$ **do**
  if $k = 0$ **then**
    $\mathbf{B}_{in} := \mathbf{I}\mathbf{B}_{in}$
    $\mathbf{B}_{out} := \mathbf{I}\mathbf{B}_{out}$
  else
    $\mathbf{B}_{in} := \hat{\mathbf{A}}_{in}\mathbf{B}_{in}$
    $\mathbf{B}_{out} := \hat{\mathbf{A}}_{out}\mathbf{B}_{out}$
  end if
  $\mathbf{H}^{(k)} := \mathrm{CONCAT}(\mathbf{B}_{in}\mathbf{W}_{in}^{(k)}, \mathbf{B}_{out}\mathbf{W}_{out}^{(k)}, \hat{\mathbf{P}})$
end for
$\mathbf{H} := \mathrm{CONCAT}(\mathbf{H}^{(0)}, \cdots, \mathbf{H}^{(k)}, \cdots, \mathbf{H}^{(K)})$
return $\sigma(\mathbf{H})$

Algorithm 5: Dynamic routing mechanism in MUCas.

Input: Lower-level capsules $\mathbf{U}$, iteration number τ.

Output: Upper-level capsules $\mathbf{S}$

for all lower-level capsules i: $\hat{\mathbf{v}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$
$\mathbf{b}_{ij} \leftarrow 0$
for τ iterations **do**
  for all lower-level capsules i: $c_{ij} \leftarrow \mathrm{softmax}(\mathbf{b}_{ij})$
  for all upper-level capsules j: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{v}}_{j|i} + \mathbf{b}_j$
  for all upper-level capsules j: $\widetilde{\mathbf{s}}_j \leftarrow \mathrm{Squash}(\mathbf{s}_j)$
  for all lower-level capsules i to all upper-level capsules j: $\mathbf{b}_{ij} \leftarrow \mathbf{b}_{ij} + \hat{\mathbf{v}}_{j|i} \cdot \widetilde{\mathbf{s}}_j$
end for

is the coupling co-efficiency that helps weight the votes, and the non-linear "squash" function is denoted as $\text{Squash}(x) = \frac{\|x\|^2}{0.5+\|x\|^2}\frac{x}{\|x\|}$.

We add biases to the calculation of $\mathbf{s}_j$ at line 5, which can solve a critical problem in capsule networks – indistinguishableness between the positive inputs and negative inputs [134].

In this layer, we defined three levels of capsules, i.e., order-level capsule, node-level capsule, and graph-level capsule, whose specific definitions are as follows:

Order-level capsule is represented as $\mathbf{h}_m \in \mathbb{R}^{K\times(2F_d+F_p)}$, focusing on specific node embedding in the sub-cascade graph. It aims to aggregate the higher-order information for each node into one node-level capsule $\mathbf{n}_m \in \mathbb{R}^{1\times F_n}$, where F_n is the dimension of node-level capsule.

Node-level capsule: As for a specific sub-cascade graph g^{T_j}, it has a set of node-level capsules $\mathbf{N}_j = \{\mathbf{n}_1, \cdots, \mathbf{n}_m, \cdots, \mathbf{n}_N\}$, $\mathbf{N}_j \in \mathbb{R}^{N\times F_n}$, used $\mathbf{N}_j$ to generate the graph-level capsule $\mathbf{g}_j \in \mathbb{R}^{1\times F_g}$ via dynamic routing, where F_g is the dimension of graph-level capsule.

Graph-level capsule: We have a set of graph-level capsules $\mathbf{G}$, each of which corresponds to a specific sub-cascade graph, i.e., $\mathbf{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_j, \cdots, \mathbf{g}_l\}$, $\mathbf{G} \in \mathbb{R}^{l\times F_g}$. Note that each graph-level capsule contains the properties of the cascade from different time intervals.

How does MUG-Caps work? Above we presented the details of the two components of MUG-Caps, i.e., MGN and the capsule-based hybrid aggregation layer. Now we turn to explain how the two components collaborate. MUG-Caps, as shown in Figure 5.2, takes a subgraph as input, which is first fed into an MGN layer to learn the embedding for each node that contains node direction-scale information, higher-order-scale information, and position-scale information as detailed in Section 5.2. After this layer, each node is represented as K vectors, with each vector matched to a different order level, referred to as the order-level capsules. Next, these order-level capsules are fed into the hybrid aggregation layer. Then dynamic routing is employed to aggregate order-level capsules to form a node-level capsule for each node. Finally, the dynamic routing aggregates these node-level capsules to build a graph-level capsule for the subgraph. The concrete calculation of MUG-Caps is shown in steps 3 to 11 in Algorithm 6.

5.3.3 Sub-graph Level Influence Attention

Previous works have demonstrated that user influence will decay significantly with time [9, 10]. In our work, we aim to learn such influence changes (dynamic-scale) at the sub-graph level, i.e., we assume that the sub-cascade graph's influence decays as the interval index increases. Inspired by self-attention mechanism [135], we employ a neural function to learn the influence attention. First, we represent the time-interval

as a one-hot vector $\mathbf{t}^j \in \mathbb{R}^l$, and then map $\mathbf{t}^j$ to $\boldsymbol{\lambda}_j$ through a fully-connected layer with sigmoid function. Here the $\boldsymbol{\lambda}_j$ is used to describe the time decay effect.

$$\boldsymbol{\lambda}_j = \text{sigmoid}(\mathbf{W}_t \mathbf{t}^j + \mathbf{b}_t), \tag{5.4}$$

where $\mathbf{W}_t \in \mathbb{R}^{F_g \times l}$ and $\mathbf{b}_t \in \mathbb{R}^{F_g}$. According to the time decay effect vector $\boldsymbol{\lambda}_j$ and graph-level capsules $\mathbf{G}$, we define the influence attention as:

$$\alpha_j = \frac{\exp(\langle \mathbf{w}, \boldsymbol{\lambda}_j \odot \mathbf{g}_j \rangle)}{\sum_{i=1}^{l} \exp(\langle \mathbf{w}, \boldsymbol{\lambda}_i \odot \mathbf{g}_i \rangle)} \tag{5.5}$$

where $\mathbf{w} \in \mathbb{R}^{F_g}$. Given the influence attention α_j, we calculate the graph-level capsule as:

$$\hat{\mathbf{g}}_j = \alpha_j \mathbf{g}_j \tag{5.6}$$

5.3.4 Information Cascade Prediction

Though our work focus on information popularity prediction, unlike existing works, we add an auxiliary task – an extra classification task i.e., whether a cascade would go viral, as a supplementary for the cascade size prediction. That is, we predict whether a cascade can break out a certain threshold value. This step is also implemented using dynamic routing over $\hat{\mathbf{G}} = \{\hat{\mathbf{g}}_j | j \in [1, l]\}$ to generate class capsules $\mathbf{C} \in \mathbb{R}^{Q \times F_c}$, where Q is the number of class, and F_c is the dimension of class capsules. The norm of class capsule $\|\mathbf{c}_q\|$ represents the probability belonging to class q. And, we use a margin loss to calculate the classification loss:

$$\ell_1 = \sum_q \left\{ \mu_q \text{max}(0, m^+ - \|\mathbf{c}_q\|)^2 + \xi(1 - \mu_q)\text{max}(0, \|\mathbf{c}_q\| - m^-)^2 \right\} \tag{5.7}$$

where $m^+ = 0.9$, $m^- = 0.1$, and $\mu_q = 1$ iff the cascade belongs to class q. Here ξ is used to stop initial learning from reducing the length of all class capsules, especially when Q is large.

Subsequently, we use a weighted sum operation on $\mathbf{C}$ to obtain the representation for a cascade $\hat{\mathbf{C}}$. The weight is calculated through $\|\mathbf{c}_*\|$:

$$\begin{aligned} \omega_q &= \frac{\exp(\|\mathbf{c}_q\|)}{\sum_{*=1}^{Q} \exp(\|\mathbf{c}_*\|)}, \\ \hat{\mathbf{C}} &= \sum_q \omega_q \mathbf{c}_q \end{aligned} \tag{5.8}$$

We use a fully-connected layer to predict the increment size $\Delta S = \text{FC}\left(\hat{\mathbf{C}}\right)$. The loss function is:

$$\ell_2 = \left(\log \Delta S - \log \Delta \widetilde{S}\right)^2 \tag{5.9}$$

where $\Delta\widetilde{S}$ is the ground truth. Finally, the overall loss function for a batch is

$$\mathcal{L} = \frac{1}{b}\sum_{i=1}^{b}(\beta\ell_1^i + (1-\beta)\ell_2^i) \tag{5.10}$$

where b is the number of cascades in a batch, and β is used to balance the ℓ_1 and ℓ_2 losses.

Algorithm 6: Learning with *MUCas*.

Input: Time-interval aware sub-cascade graphs for n cascade graphs $\mathbf{A}^{T_{iv}} = \{\mathbf{A}_1^{T_{iv}}, \cdots, \mathbf{A}_i^{T_{iv}}, \cdots, \mathbf{A}_n^{T_{iv}}, \}$; Max order K; Batch size b.
Output: Incremental size $\Delta S = \{\Delta S_1, \cdots\}$ of cascades.

repeat
  $b = 1, 2, ...$
  for adjacency matrix sequence $\mathbf{A}_i^{T_{iv}}$ in batch b **do**
    for $\mathbf{A}_i^{T_j} \in \mathbf{A}_i^{T_{iv}}$ **do**
      Compute the node embeddings $\mathbf{H}_j$ for j^{th} sub-cascade graph according to Eq. (5.1).
      for node embedding $\mathbf{h}_m \in \mathbf{H}_j$ **do**
        Use $\mathbf{h}_m$ to compute the node-level capsule $\mathbf{n}_m$ according to Algorithm 5.
      end for
      $\mathbf{N}_j \leftarrow \{\mathbf{n}_1, \cdots, \mathbf{n}_N\}$
      Use $\mathbf{N}_j$ to generate the graph-level capsule $\mathbf{g}_j$ according to Algorithm 5.
    end for
    $\mathbf{G}_i \leftarrow \{\mathbf{g}_1, \cdots, \mathbf{g}_l\}$
    Compute influence attention α_j according to Eq. (5.4), (5.5).
    Compute newly graph-level capsules $\hat{\mathbf{G}}_i$ according to Eq. (5.6).
    /*Extra classification task*/
    Use $\hat{\mathbf{G}}_i$ to compute class capsules $\mathbf{C}_i$ according to Algorithm 5.
    /*Popularity prediction task*/
    Calculate cascade representation $\mathbf{C}_i^{'}$ according to Eq. (5.8).
    Feed $\mathbf{C}_i^{'}$ into fully-connected layer to compute incremental size ΔS_i of cascade.
    Use Adaptive moment estimation (Adam) to optimize the objective function in Eq.(5.10) and update all the parameters.
  end for
until convergence;

5.3.5 Complexity Analysis

We finish this section with a discussion of the computational complexity of the main components in MUCas, i.e., the cost of (1) node embedding layer, and (2) capsule-based hybrid aggregation layer.

Node embedding layer: We use MGN to learn node embeddings from the sub-cascade graph, whose calculation is shown in Eq. (5.1). As for **direction-scale**, MGN models both the incoming and outgoing relations of the cascades via $\hat{\mathbf{A}}_{\phi}^{(k)}\mathbf{X}\mathbf{W}_{\phi}^{(k)}$. Recall that the dimensions of $\mathbf{X}$, $\mathbf{W}_{\phi}^{(k)}$, and $\mathbf{W}_p$ are F, F_d, and F_p, respectively. Besides, the max-order is K, and the normalized adjacency matrix $\hat{\mathbf{A}}_{\phi}$ in our implementation is a sparse matrix with $|\mathcal{E}|$ nonzero elements – $|\mathcal{E}|$ is the number of edges in current graph. In addition, the number of nodes is denoted as N. According to existing works [61, 126], for a single direction at each order, the calculation is conducted via sparse-dense matrix multiplications, and the computational complexity is $\mathcal{O}(|\mathcal{E}| \times F \times F_d)$. Taking the high-order-scale into consideration, the computational complexity is then $\mathcal{O}(K \times |\mathcal{E}| \times F \times F_d)$. As for **positional-scale**, the calculation of $\mathbf{W}_p\mathbf{P}$ is completed via matrix multiplication, which requires $\mathcal{O}(N \times d_p \times F_p)$ computations. Therefore, the total computational time cost of evaluating Eq. (5.1) is then $\mathcal{O}(2 \times K \times |\mathcal{E}| \times F \times F_d + N \times d_p \times F_p)$.

Capsule-based hybrid aggregation layer: This layer is implemented by executing dynamic routing between different levels capsules. Specifically, we adopt a dynamic routing mechanism for τ iterations over $|\mathbf{U}|$ lower-level capsules and generate $|\mathbf{S}|$ upper-level capsules. This learning process requires $\mathcal{O}(\tau \times |\mathbf{U}| \times |\mathbf{S}|)$ computations [136]. From the MGN, we get the order-level capsule $\mathbf{H} = \left\{\mathbf{h}_1, \cdots, \mathbf{h}_m, \cdots, \mathbf{h}_N | \mathbf{h}_m \in \mathbb{R}^{K \times (2F_d + F_p)}\right\}$. For each node m, it generates the node-level capsule from order-level capsules, whose computational complexity is $\mathcal{O}(\tau \times K \times 1)$. Because there are N nodes, the total time of generating node-level capsules for all nodes is $\mathcal{O}(N \times \tau \times K \times 1)$. Similarly, generating the graph-level capsule from node-level capsules can be done within $\mathcal{O}(\tau \times N \times 1)$ time. The overall computational time is therefore $\mathcal{O}(N \times \tau \times K \times 1 + \tau \times N \times 1)$.

5.4 Evaluation

In this section, we compare the performance of our proposed model *MUCas* with the state-of-the-art approaches, and several variants of *MUCas*, on information popularity prediction. In particular, we provide the quantitative results to answer the following research questions:

- **Q1**: How does MUCas perform on cascade size prediction compared with the state-of-the-art baselines?
- **Q2**: How do different scales of information modeled in MUCas contribute to the overall performance?
- **Q3**: How do the key hyper-parameters affect the performance of MUCas?

Table 5.2: Statistics of datasets

Dataset	Weibo	APS
# Ori. Cascades	119,313	636,294
# Nodes	5,918,473	636,294
# Edges	12,204,245	3,425,508
Avg. popularity	173	13
Train (0.5 Hour/ 3 Years)	19,472	24,636
Val (0.5 Hour/ 3 Years)	4,173	5,279
Test (0.5 Hour/ 3 Years)	4,172	5,278
Train (1 Hour/ 5 Years)	19,124	33,408
Val (1 Hour/ 5 Years)	4,098	7,159
Test (1 Hour/ 5 Years)	4,097	7,158

5.4.1 Datasets

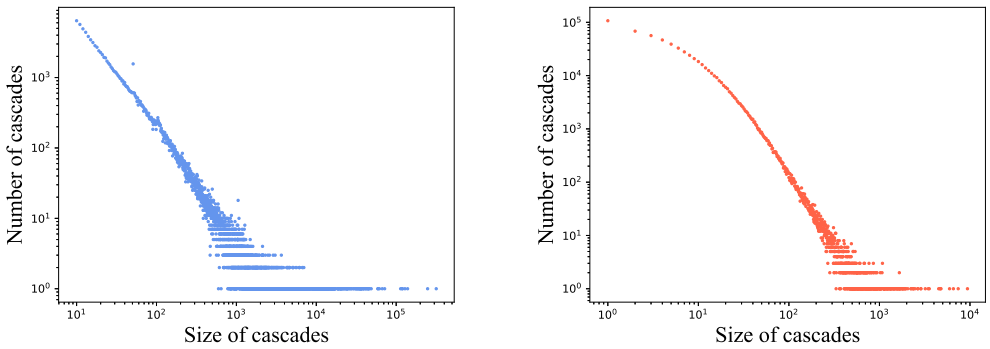
We evaluate the effectiveness and generalizability of *MUCas* on two scenarios. The first one is to predict the size of retweet cascades in Sina Weibo and the second one is to forecast the citation count of papers in citation dataset APS. The statistics of the datasets used in this work has shown in Table 5.2.

Sina Weibo: The description of the Weibo dataset is given in Section 4.4.1. Figure 5.4a plots the distribution of the cascade size (the number of re-tweets of each post), which, obviously, follows an power-law distribution and reflects the Pareto principle (80/20 rule). Figure 5.5a shows the distribution of depth over all cascades, which roughly follows an exponential distribution, indicating that the majority of cascades have a shallow depth, i.e., most of them are less than 5. The depth of a cascade is the length of the longest path, which also equals to the max-order of the cascade. Due to the effect of diurnal rhythm in Weibo [9], in our experiments, the cascades with the publication time before 8 am and after 6 pm were filtered out, leaving each post at least 6 hours to obtain retweets. As shown in Figure 5.6a, on average, a message receives about 70% retweets within 5 hours.

APS: APS [1] is provided by American Physical Society (APS), which consists of pairs of APS articles that cite each other for the corpus of Physical Review Letters, Physical Review, and Reviews of Modern Physics from 1893 to 2018. The papers from 1893 to 1997 are selected as observations so that each of the papers is allowed to develop for at least 20 years. In the citation scenario, the size of a cascade is the citation count. Figure 5.4b shows the distribution of cascade size in the APS dataset, which exhibits a power-law distribution. Figure 5.5b shows the distribution of the depth over all cascades in APS, which has a similar trend with Weibo. As shown in Figure 5.6b, on average, the citation reaches around 50% of the final size within 5 years.

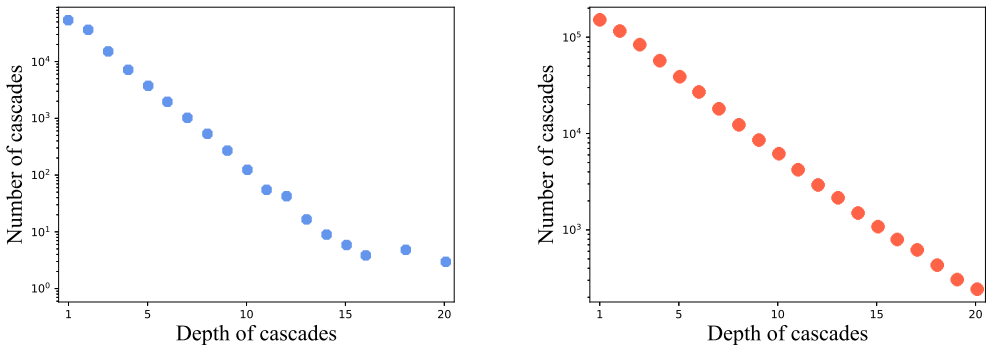
[1] http://journals.aps.org

Settings: In our experiments, we use the same dataset settings as in previous works [9, 69]. The observation time window in Weibo data is set to $t_o = 0.5$ and $t_o = 1$ hour. For the observation window of the APS dataset, we choose $t_o = 3$ and 5 years. For both Weibo and APS datasets, we filter out cascades whose observed size $S_{obs} < 10$. And for those cascades whose $S_{obs} > 100$, we only track the first 100 retweets. In addition, we randomly split each dataset into a training set (70%), a validation set (15%), and a testing set (15%) following existing works [9, 69].



(a) Weibo dataset **(b)** APS dataset

Figure 5.4: Cascade size distributions of Weibo (left), and APS (right) in log-log scales.



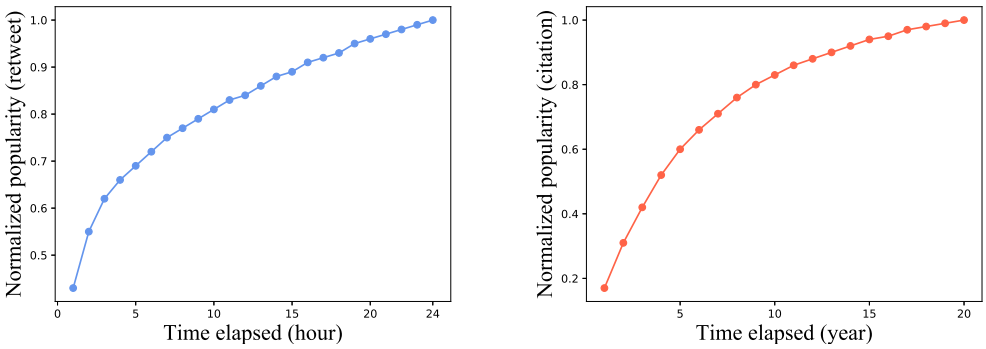
(a) Weibo dataset **(b)** APS dataset

Figure 5.5: Distribution of depth (max-order) of cascades.

5.4.2 Baselines

To validate MUCas's performance in cascade prediction, we select following state-of-the-art baselines for comparison:

- **Feature-Linear**, **Feature-Deep**, **DeepCas** [8] and **DeepHawkes** [9] are already introduced in Section 4.4.2, for brevity, we will not repeat them here.



(a) Weibo dataset **(b)** APS dataset

Figure 5.6: Normalized popularity distribution of cascades: retweets (citations) vs. time.

- **CasCN**: CasCN (in Chapter 4) is the first graph convolution network (GCN)-based framework exploiting both temporal and structural information for cascade size prediction. It decomposes a cascade graph into a sequence of sub-cascade graphs based on propagation time, while learning the local structure and its evolving process of cascade structure through the combination of graph convolutions and LSTM.

- **VaCas** [69]: VaCas is the first Bayesian learning-based approach that uses pre-trained node embeddings of the cascade as input and leverages a hierarchical variational information diffusion model to learn the posterior of cascade distribution with variational inference.

- **Cascade2vec** [62]: Cascade2vec is an improvement of CasCN, which proposes a new graph convolutional kernal – graph perception network (GPN) to replace the original GCN in CasCN. It also introduces the attention mechanism to learn different importance of each sub-graph.

5.4.3 Evaluation Metric

Following previous studies [8, 9, 35], we use mean square logarithmic error (MSLE) and symmetric mean absolute percentage error (SMAPE) for prediction performance evaluation. In addition, we also report the coefficient of determination (R^2) of

different models. The formulation of all evaluation metric is defined as:

$$\begin{aligned} \text{MSLE} &= \frac{1}{n}\sum_{i=1}^{n}\left(\log \Delta S_i - \log \Delta \widetilde{S}_i\right)^2 \\ \text{SMAPE} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\log \Delta S_i - \log \Delta \widetilde{S}_i|}{(|\log \Delta S_i| + |\log \Delta \widetilde{S}_i|)/2} \\ \text{R}^2 &= 1 - \frac{\sum_{i=1}^{n}(\log \Delta S_i - \log \Delta \widetilde{S}_i)^2}{\sum_{i=1}^{n}(\log \Delta \widetilde{S}_i - \frac{1}{n}\sum_{i=1}^{n}\log \Delta \widetilde{S}_i)^2} \end{aligned} \tag{5.11}$$

where n is the total number of posts, ΔS_i is the predicted incremental size for post m_i, and $\Delta \widetilde{S}_i$ is the ground truth. Note that, the value of MSLE and SMAPE the smaller the better, in contrast, the value of R^2 the bigger the better.

5.4.4 Experimental Settings and Parameter Tuning

Parameter settings: Models mentioned above are optimized to the best performance which involves several key hyper-parameters. The L_2 coefficient of Feature-Linear is chosen from $10^{\{0,-1,-2\cdots,-8\}}$. The node embedding size for DeepCas, DeepHawkes, CasCN, and Cascade2vec is set to 50. The hidden layer of each GRU has 32 units, and the hidden dimensions of the two-layer fully-connected layers for all deep learning-based methods are 32 and 16, respectively. The learning rate for node embeddings in DeepCas and DeepHawkes is 5×10^{-4} and the learning rate for other methods are 5×10^{-3}. The batch size is set as 64. All other hyper-parameters are set to the same values as used in the original papers.

As for our MUCas, the basic parameters (e.g., the learning rate is 5×10^{-4} and batch size is 64, etc.) are the same as above deep learning-based approaches, except that the max-order K and iteration number τ are chosen from 1 to 5. The embedding size of positional embedding is chosen from $\{30, 50, 100, 150, 200\}$, the hidden size for MGN is 60, and the hidden size for node-level capsule, graph-level capsule and the class capsule is 30, 8 and 16, respectively. In addition, the number of time intervals is set to 6. For the auxiliary classification task, the number of classes Q is equal to 2. Specifically, we label a cascade as 1 if its increased size is twice or more than its observed size, 0 otherwise. All methods, including ours, are tuned to the best performance with early stopping when validation errors has not declined for 10 consecutive epochs.

Experimental environment: The experiments are conducted on a sever with Intel E5-2680 v4 2.40GHz, one NVIDIA GeForce GTX 3090, and 256GB memory.

5.4.5 Performance Comparison (Q1)

The overall performance of MUCas as well as the state-of-the-art baselines are shown in Table 5.3 and Table 5.4, from which we have the following important observa-

Table 5.3: Performance comparison between baselines and MUCas on Weibo datasets. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.001$ as compared to the best baseline method.

Model	Weibo					
	0.5 Hour			1 Hour		
	MSLE	SMAPE	R^2	MSLE	SMAPE	R^2
Feature-Linear	2.959	0.331	0.351	2.710	0.356	0.461
Feature-Deep	2.815	0.311	0.379	2.646	0.353	0.465
DeepCas	2.914	0.330	0.352	2.747	0.358	0.459
DeepHawkes	2.891	0.321	0.379	2.632	0.352	0.468
CasCN	2.804	0.311	0.381	2.601	0.350	0.468
Cascade2vec	2.752	0.308	0.384	2.589	0.348	0.479
VaCas	2.586	0.291	0.504	2.359	0.333	0.518
MUCas	**2.081**$*$	**0.271**$*$	**0.621**$*$	**1.882**$*$	**0.308**$*$	**0.647**$*$
(Improvements)	19.53%	6.87%	23.21%	20.22%	7.51%	24.90%

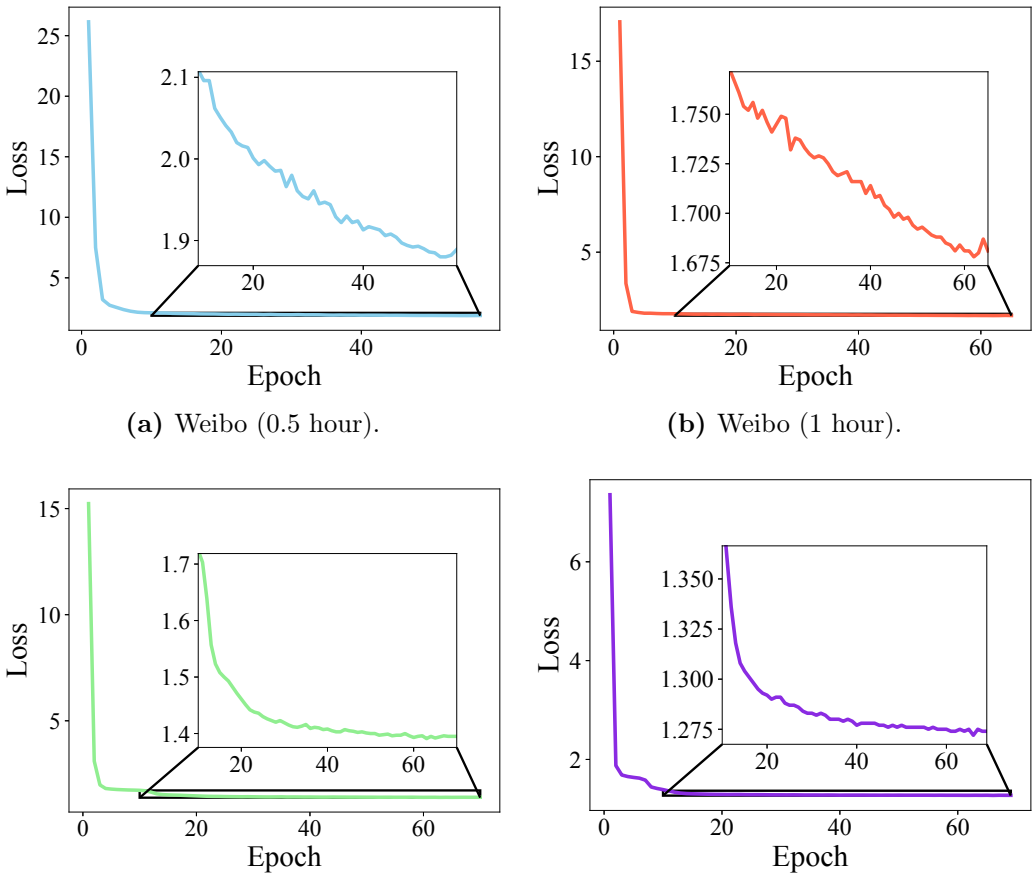
Table 5.4: Performance comparison between baselines and MUCas on APS datasets. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.01$ as compared to the best baseline method.

Model	APS					
	3 Years			5 Years		
	MSLE	SMAPE	R^2	MSLE	SMAPE	R^2
Feature-Linear	2.100	0.289	0.126	2.087	0.358	0.311
Feature-Deep	1.996	0.358	0.221	1.874	0.352	0.322
DeepCas	2.033	0.361	0.213	1.944	0.365	0.318
DeepHawkes	1.831	0.344	0.241	1.588	0.337	0.363
CasCN	1.818	0.274	0.244	1.574	0.337	0.367
Cascade2vec	1.783	0.272	0.258	1.560	0.336	0.373
VaCas	1.723	0.268	0.283	1.507	0.335	0.394
MUCas	**1.557**$*$	**0.263**$*$	**0.355**$*$	**1.439**$*$	**0.333**$*$	**0.426**$*$
(Improvements)	9.63%	1.87%	25.44%	4.51%	0.59%	8.12%

tions.

(**O1**) MUCas outperforms the baselines by a large margin, e.g., as for the MSLE, it reduces the prediction error up to 19.53%, 20.22% on the Weibo dataset and 9.63% and 4.51% on the ASP dataset when compared to the best baseline – VaCas, when t_o is set to 0.5, 1 hour and 3, 5 years on Weibo and APS, respectively. We plot the training process of MUCas on the Weibo and APS dataset and show the results in Figure 5.7. Clearly, the training loss of MUCas consistently decreases and converges to a lower value.

(**O2**) The gap between handcrafted feature-based methods and most deep learning-based baselines are quite small. In some cases the handcrafted feature-based methods even beat some deep learning-based methods. Comparing the Feature-Deep with DeepCas, for example, we can observe that a fully connected layer is enough



(a) Weibo (0.5 hour). **(b)** Weibo (1 hour).

(c) APS (3 years). **(d)** APS (5 years).

Figure 5.7: Convergence of MUCas on Weibo and APS datasets.

to achieve competitive results than complicated neural networks (DeepCas) if we have a set of well-designed hand crafted features. However, obtaining such features requiring extensive domain knowledge, which is hard to be generalized to new domains.

(**O3**) DeepCas – the first deep-learning based approach for cascade size prediction – performs the worst among the deep learning baselines, because it simply learns the cascade representation based on sampled random walks but ignores temporal and topological information. DeepHawkes, while being successful in modeling temporal information for cascades in a generative learning manner, does not perform well due to its weak ability to learn structural information.

(**O4**) The rest of the baselines, i.e., CasCN, Cascade2vec and VaCas, generate competitive results because they explore structural and temporal information at the same time. When comparing CasCN with Cascade2vec, the performance of Cascade2vec is slightly better, due to the modified convolutional kernel in Cascade2vec, which indeed improves the ability of learning structural features. Besides, VaCas employed VAE [70] to solve the uncertainty problem in structural representation

learning and therefore achieves higher accuracy compared to other baselines. Our MUCas not only combines the advantages of CasCN and VaCas, but also takes into account the high-order-scale and position-scale of a cascade graph, leading to a significant improvement in prediction performance.

(**O5**) When examining the methods with different observation window t_o, we can observe a general trend, i.e., the larger the t_o, the accurate the predictions. This is intuitive because longer observation time reveals more temporal and structural knowledge regarding information diffusion that helps cascade size prediction.

5.4.6 Ablation Study (Q2)

To better investigate the contribution of each scale of information modeled in MUCas, we derive the following variants of MUCas:

-Direction: In "*-Direction*", we do not consider the directional relation in cascades, i.e., regarding the cascade graphs as undirected graphs. We replace the MGN to a vanilla GCN [61] and calculate the normalized adjacency matrix according to $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$.

-Order: In "*-Order*", we only focus on 1^{st}-order neighbors in the cascade graphs, i.e., setting the max-order number K to 1.

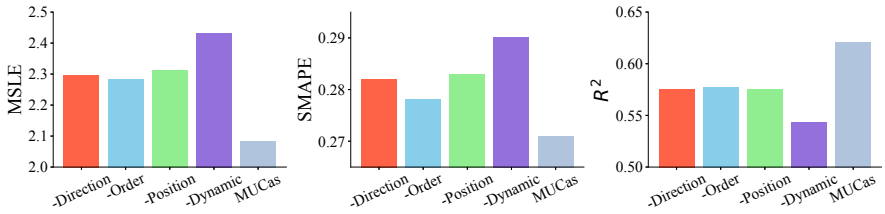
-Position: In "*-Position*", we ignore the node's relative position in cascade graph, i.e., removing the computation of position information $\mathbf{PW}_p$ in MGN.

-Dynamic: In "*-Dynamic*", we remove the sub-graph level influence attention component, and use $\mathbf{G}$ directly.

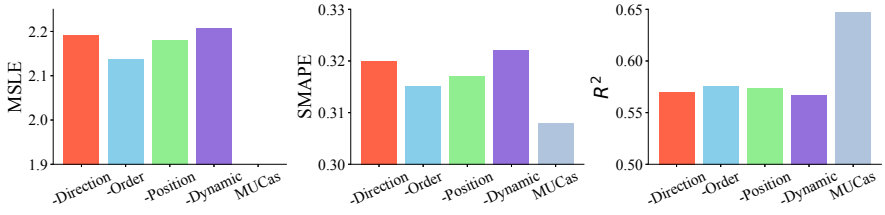
Figure 5.8 shows the performance comparisons among MUCas and its variants, which illustrates that: **(i)** the original MUCas achieves the best performance compared with other variants, demonstrating the motivation of our work, i.e., considering the four different scale information for cascade modeling. **(ii)** From the comparison between "-Direction" and "-Position", we find that effectively modeling the directional relation and node's relative position in the cascade graph will improve the prediction performance. **(iii)** Removing "-Order" and "-Dynamic" bring a remarkable decrease of the prediction performance, which implies that: (a) nodes with different orders play different importance in prediction task, and (b) the influence decreases as the cascade graph evolves.

In order to quantify the effectiveness of different levels of capsules, i.e., order-level, node-level, and graph-level capsule, we test the model performance by designing several single capsule-based variants of MUCas, including:

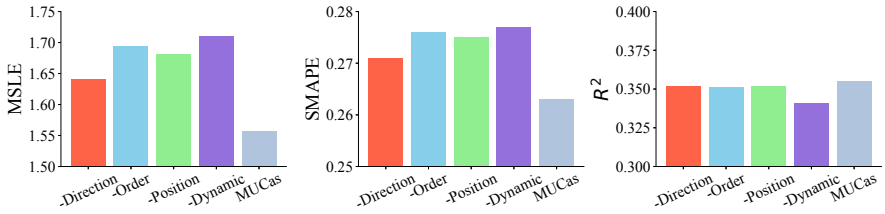
Order-level: In "*Order-level*", we apply sum-pooling to aggregate the order-level capsules $\mathbf{h}_*$ for each node and form node-level capsule $\mathbf{n}_*$. Finally, the sum-pooling



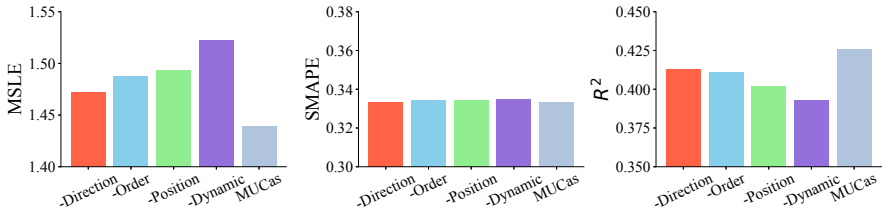
(a) Weibo – 0.5 hour



(b) Weibo – 1 hour



(c) APS – 3 years



(d) APS – 5 years

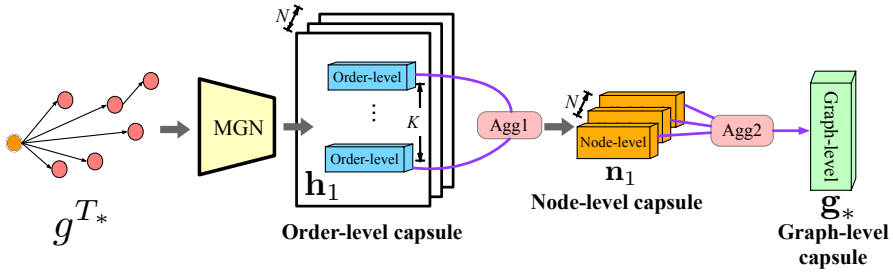
Figure 5.8: Ablation study of MUCas on two datasets.

operation is employed again to aggregate node-level capsules to form the graph-level capsule $\mathbf{g}_*$.

Node-level: In "*Node-level*", we apply dynamic routing to aggregate order-level capsules $\mathbf{h}_*$ to form node-level capsule $\mathbf{n}_*$. Subsequently, sum-pooling is used to aggregate node-level capsules to form the graph-level features $\mathbf{g}_*$.

Graph-level: In "*Graph-level*", we apply sum-pooling to generate node-level capsule $\mathbf{n}_*$ from $\mathbf{h}_*$, and then employ dynamic routing to aggregate node-level capsules to form the graph-level capsule $\mathbf{g}_*$.

Figure 5.9 illustrates the differences between three single capsule-based variants and MUCas. The experimental results are shown in Figure 5.10, where we can find that: (i) Compared to all single capsule-based variants, the original MUCas performs the best; (ii) Even keeping only one single-level capsule, the model performance is



Methods	Agg1	Agg2
MUCas	Dynamic routing	Dynamic routing
Order-level	Sum-pooling	Sum-pooling
Node-level	Dynamic routing	Sum-pooling
Graph-level	Sum-pooling	Dynamic routing

Figure 5.9: Illustration of single capsule-based variants of MUCas.

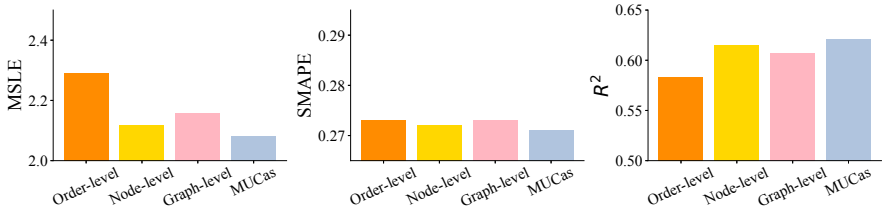
still superior to all the baselines in Table 5.3 and Table 5.4; and (iii) Compared to original MUCas, the "order-level" variant performs the worst, while the performance of the other two variants (i.e., node-level and graph-level) do not drop a lot. This result demonstrates that (1) the three-level capsules are indispensable, and (2) the dynamic routing is efficient in aggregating features.

5.4.7 Hyper-parameter Sensitivity (Q3)

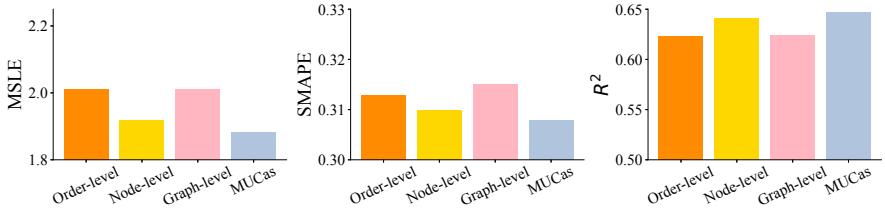
We study several important hyper-parameters that may influence the prediction performance of our model. Here we select the Weibo data set for the experiments and omit APS as the results are expected to be similar. The impact of the choice of hyper-parameters is shown in Figure 5.11.

- *Impact of max-order* K: we try different values of max-order K from 1 to 5, and the prediction results of MUCas are shown in Figure 5.11a and Figure 5.11b. When the observation time is 0.5 hour, our MUCas achieves the best performance if $K = 3$. But for 1 hour observation, the optimal value of K is 4. The reason is that with the increase of observation time, the information are more likely to propagated to more nodes with deeper depth.

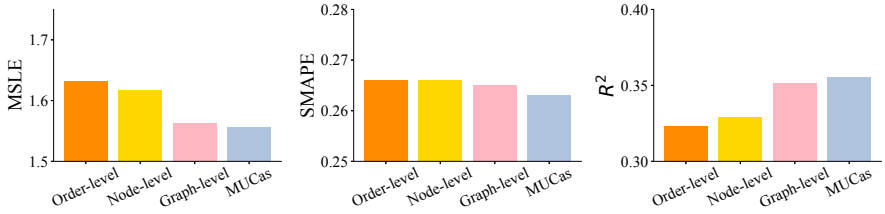
- *Impact of embedding size of positional embedding* F_p: we change the dimensions F_p of positional embedding $\mathbf{P}$ within $\{30, 50, 100, 150, 200\}$. Compare the results on 0.5 hour with the results on 1 hour (Figure 5.11c and Figure 5.11d), we can see that larger embedding size sometimes may degrade the performance, and the proper embedding size should fall into the scope of $[50, 150]$. We hypothesize the reason is that most cascade graphs are small, though a few of them may diffuse to a large number of nodes, i.e., the 80/20 rule as shown in Figure 5.4a and 5.4b. Therefore, a larger embedding dimensions would incur overfitting issue for those smaller cascade graphs.



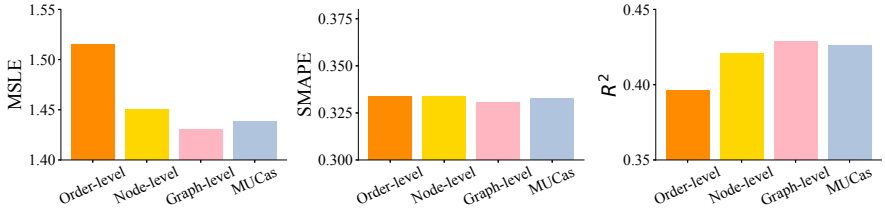
(a) Weibo – 0.5 hour



(b) Weibo – 1 hour



(c) APS – 3 years

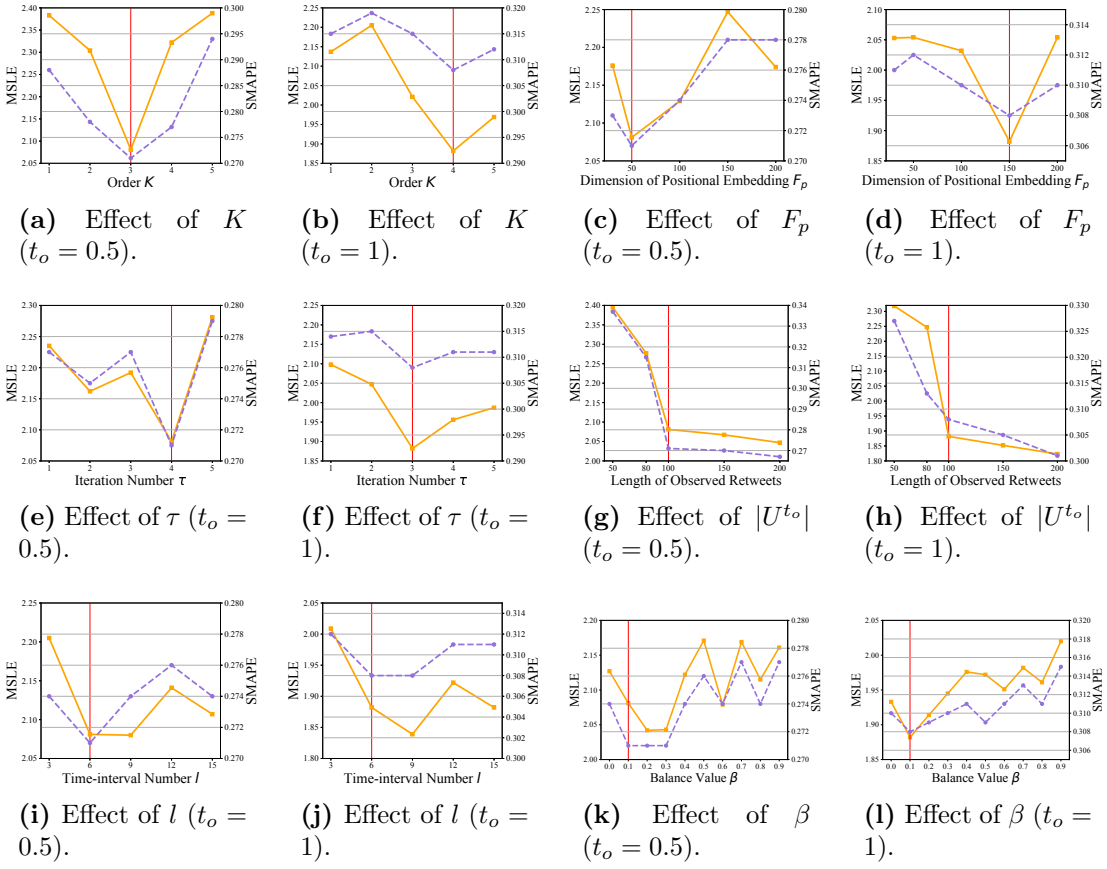


(d) APS – 5 years

Figure 5.10: Capsule study of MUCas on two datasets.

- *Impact of iteration number* τ: we try different values of the iteration number τ in dynamic routing, specifically, by increasing the value from 1 to 5. The results are shown in Figure 5.11e and Figure 5.11f, which suggests that increasing the number of iterations would improve the performance first, but deteriorate the model soon. This result raises an open issue in capsule network learning. That is, the appropriate iterations requires careful tuning that makes the capsule network unstable. This issue should be addressed for robust representation learning, which is beyond the scope of this work and left as future work.

- *Impact of the length of observed retweets* $|U^{t_o}|$: we track the first 50, 80, 100, 150, and 200 retweets and report the performance of MUCas based on these observed retweets. The experimental results show in Figure 5.11g and Figure 5.11h, where we can observe the improved performance with the increase of observation retweets, which is a natural result of including more training



(a) Effect of K ($t_o = 0.5$). **(b)** Effect of K ($t_o = 1$). **(c)** Effect of F_p ($t_o = 0.5$). **(d)** Effect of F_p ($t_o = 1$).

(e) Effect of τ ($t_o = 0.5$). **(f)** Effect of τ ($t_o = 1$). **(g)** Effect of $|U^{t_o}|$ ($t_o = 0.5$). **(h)** Effect of $|U^{t_o}|$ ($t_o = 1$).

(i) Effect of l ($t_o = 0.5$). **(j)** Effect of l ($t_o = 1$). **(k)** Effect of β ($t_o = 0.5$). **(l)** Effect of β ($t_o = 1$).

Figure 5.11: Impact of the important hyper-parameters on MUCas. Vertical lines are settings we used in previous experiments. Orange solid lines represent the results of MSLE, and the purple dashed lines denote the results of SMAPE.

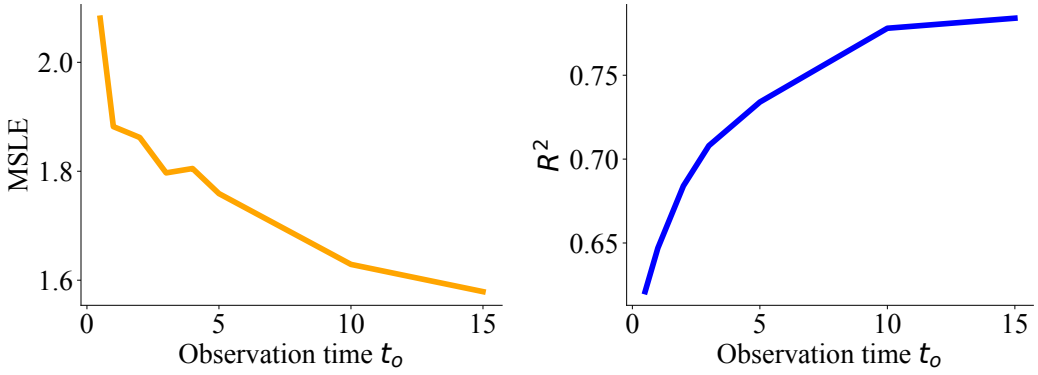
data.

- *Impact of the number of time intervals* l: when sampling sub-cascade graphs from the observed cascade, we set different numbers of time interval, i.e., $l = [3, 6, 9, 12, 15]$. The results are shown in Figure 5.11i and 5.11j, which reveal that: (1) the fine-grained sampling does not always perform better. For example, when the observation time is 0.5 hour, the performance of MUCas at $l = 6$ is much better than the results when $l = 9, 12$ and 15. This finding also proves our hypothesis in Section 5.3.1 that the differences between fine-grained sub-graphs become trivial as l grows, which will introduce biases in dynamic modeling. Besides, increasing the number of sub-graphs would significantly increase the computation cost. (2) The choice of the number of time interval heavily depends on the distribution of dataset. When the observation window is 0.5 hour and 1 hour, the model performance achieves the best at $l = 6$ and $l = 9$, respectively.

- *Impact of the balance value* β: we change the hyper-parameter β in loss function (Eq. (5.10)) from 0 to 0.9, and report the results in Figure 5.11k and

Figure 5.11l. We can see that a smaller value of β always achieves better performance. Furthermore, $\beta = 0.5$ can be regarded as a watershed, with a value less than 0.5 being far more suitable for model selection than a value greater than 0.5. Setting $\beta = 0$, which can be regarded as removing the auxiliary task from the original model. And from the results of $\beta = 0$, we can find that adding an auxiliary task indeed helps us to improve the model performance.

- In order to further support the finding (**O5**) in Section 5.4.5, we conduct an extra experiment to assess the model sensitivity varied with the observation time. Specifically, we consider the propagation in the first 15 hours because most of the messages in the Weibo dataset will decay after this time [11]. The results are shown in Figure 5.12, where we can clearly see that the longer the observations, the better the performance of the model.



(a) MSLE **(b)** R^2

Figure 5.12: The influence of time window in Weibo dataset.

5.4.8 Model Parameters and Computation Cost

Table 5.5: Model parameters and computation time measured by seconds when t_o = 0.5 hour and 3 years for Weibo and APS, respectively. "∼" means "approximatively".

Methods	Parameters	Time cost per epoch (in seconds)	
		Weibo (t_o = 0.5 hour)	APS (t_o = 3 years)
DeepCas	∼250M	∼150s	∼170s
DeepHawkes	∼250M	∼128s	∼158s
CasCN	∼278K	∼320s	∼400s
Cascade2Vec	∼368K	∼50s	∼75s
VaCas	∼2M	∼83s	∼98s
MUCas	∼495K	∼104s	∼105s

We compute the time cost of training MUCas and the baselines, as well as the required parameters. The results are reported in Table 5.5. First, the memory required for DeepCas and DeepHawkes is much higher because they need the embeddings of

all users in the social network, which required $|U| \times F_u$ parameters – $|U|$ represents the total number of users and F_u denotes the embedding size. Besides, the training time for each epoch of MUCas is around $104s$ and $105s$ in Weibo and APS dataset when $t_o = 0.5$ hour and $t_o = 3$ years, respectively. In contrast, Cascade2Vec is the fastest model that only needs 50s and 70s for Weibo and APS datasets, respectively, but the quality of the model is much lower.

5.5 Summary

In this chapter, we proposed a novel cascade prediction model – MUCas, which capture the multi-scale features regarding information diffusion comprehensively and make good predictions. Specifically, MUCas consists of four components: (1) a time interval-aware sampling layer used to generate sub-cascade graphs from the observed cascade graph, (2) MUG-Caps extracts the direction-scale, position-scale, and high-order-scale information from sub-cascade graphs, (3) an attention layer applied to learn dynamic-scale, and (4) a prediction layer to make predictions. We conducted extensive experiments based on two real-world datasets, i.e., Weibo and APS. The experimental results demonstrate that our method achieves state-of-the-art performance on information cascade size prediction of tweet propagation in social networks and scientific papers' impact.