Machine Learning
https://doi.org/10.1007/s10994-023-06449-z

Predicting potential real-time donations in YouTube live streaming services via continuous-time dynamic graphs

Ruidong Jin[1,2] · **Xin Liu**[2] · **Tsuyoshi Murata**[1,2]

Received: 9 February 2023 / Revised: 30 June 2023 / Accepted: 7 October 2023

Abstract
Online live streaming platforms, such as YouTube Live and Twitch, have seen a surge in popularity in recent years. These platforms allow viewers to send real-time gifts to streamers, which can bring significant profits and fame. However, there has been little research on the donation system used on live streaming platforms. This paper aims to fill this gap by building a continuous-time dynamic graph to model the interactions among viewers based on real-time chat messages and predict the real-time donations on live streaming platforms. To achieve this, we propose a novel model called the Temporal Difference Graph Neural Network (TDGNN) that incorporates imbalanced learning strategies to identify potential donors during live streaming. Our model can predict the exact time when donations will appear. We conduct extensive experiments on three live streaming video datasets and demonstrate that our proposed model is more effective and robust than other baseline methods from other fields.

Keywords Online live streaming · Real-time donation · Continuous-time dynamic graph · Dynamic node label prediction

Editors: Dino Ienco, Roberto Interdonato, Pascal Poncelet.

✉ Xin Liu
xin.liu@aist.go.jp

Ruidong Jin
ruidong.jin@net.c.titech.ac.jp

Tsuyoshi Murata
murata@c.titech.ac.jp

[1] Department of Computer Science, School of Computing, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo 152-8550, Japan

[2] Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, 2-4-7 Aomi, Koto, Tokyo 135-0064, Japan

Published online: 10 November 2023

1 Introduction

The growth of the internet and the increasing use of mobile devices have led to a surge in the popularity of live streaming services. These services offer a wide range of content, including news, sports, entertainment, and video games (Yang & Lee, 2018), and provide several advantages over traditional television, such as a large variety of content, low costs, flexibility in viewing, personalized channels, and uninterrupted programming (Yang & Lee, 2018; Lee et al., 2016). The COVID-19 pandemic has significantly impacted the live streaming industry, as many people have turned to online courses and remote work, resulting in an increase in the global market for online video streaming.

YouTube Live[1] is one of the most popular online live streaming platforms, with millions of user-generated content shared among billions of active users. Viewers on YouTube Live can communicate with others by sending real-time chat messages, which streamers can see and interact with in real time. These chat messages bring viewers and streamers closer together, creating a sense of community for viewers and making popular streamers into 'celebrities' for their audiences (Fietkiewicz et al., 2018). Additionally, some viewers are willing to donate money to their favorite streamers through the 'superchat' donation system on YouTube Live. As shown in Fig. 1, superchat is a special chat message associated with an amount of money ranging from $1 to $500 that is highlighted in the live streaming chatbox for some time. This system brings profits, popularity, and motivation for streamers to create high-quality content. This situation raises some interesting questions: Who tends to send superchats? When are superchats sent? Can superchats be predicted? Understanding the answers to these questions will help streamers to predict their expected donation income better.

Research on live streaming services is still in its early stages. Scholars are studying various aspects of video content, such as improving video quality and using image recognition in live streaming. Additionally, live-streaming platforms provide a valuable data source for chat data analysis. Some works focus on highlight detection (Chu & Chou, 2017), sentimental analysis (Kavitha et al., 2018), and fraud detection (Li et al., 2021). However, there is limited research on analyzing virtual donations in live streaming services. Current studies focus on static donation prediction without considering temporal information, using simple machine learning algorithms (Jia et al., 2021; Wang et al., 2019), and are thus not practically applicable.

This paper presents a novel approach to discovering when and who sends superchat messages in YouTube live streaming. We study real-time chat messages and interactions among viewers in live streaming. We find that chat messages increase significantly when a superchat is posted, as streamers appreciate the donation and other viewers respond to the superchat. Additionally, superchat messages are usually longer and better organized to attract attention. Therefore, a superchat message is usually followed by many response chat messages and has unique text content.

Drawing on these insights, we propose an approach that models the dynamic interactions among viewers and the text content of chat messages. Our method utilizes a continuous-time dynamic graph to capture the complex relationships among thousands of viewers and millions of chat messages. By transforming the superchat detection problem into a dynamic node label classification problem in the graph, we are able to predict

[1] https://www.youtube.com/.

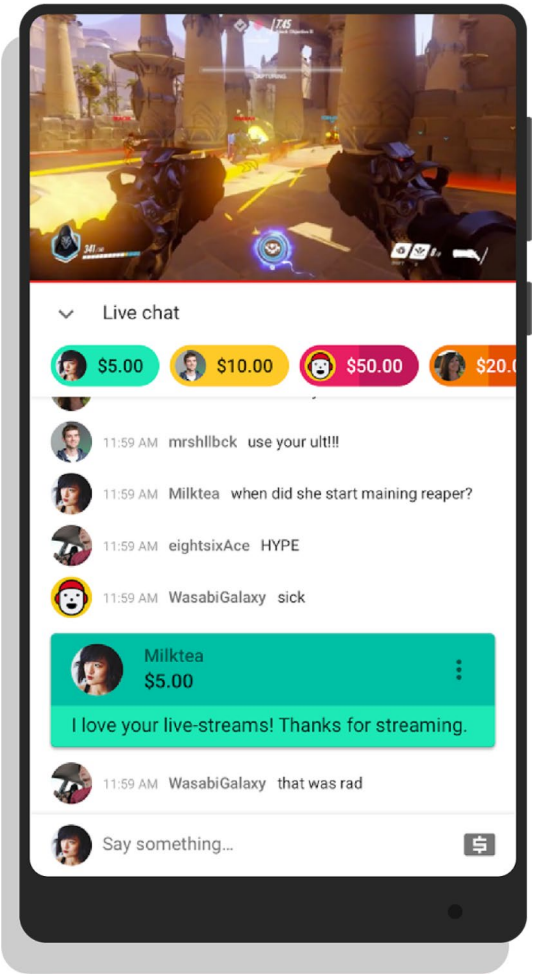

Fig. 1 An example of a superchat on the YouTube Live platform is when a viewer named 'Milktea' donates $5.00 to the streamer and posts a superchat message that reads, 'I love your live-streams! Thanks for streaming', in a live streaming channel

both the superchat messages and the timing of their posting on YouTube Live. To learn temporal node representations in the graph, we introduce a novel Temporal Difference Graph Neural Network (TDGNN) that exploits the information gap between connected nodes.

The experimental results demonstrate that our proposed approach is highly effective in predicting superchat messages with AUC scores up to 0.916, outperforming various baseline methods. These baseline methods include decision tree algorithms(e.g., GBDT, XGBoost), time sequence models (e.g., LSTM-FCN (Karim et al., 2018)), static graph-structured models (e.g., GCN (Kipf & Welling, 2016a), GAT (Veličković et al., 2018)), text classification models (e.g., BERT (Devlin et al., 2018)), and the latest temporal graph neural network (TGNN) models (e.g., TGN (Rossi et al., 2020), APAN (Wang et al., 2021), TGAT (Xu et al., 2020)), MetaDyGNN (Yang et al., 2022).

In this paper, we make several important contributions to the field of donation prediction in live streaming platforms, including:

- We present the first analysis and prediction of real-time virtual donations in live streaming platforms. This is a significant and innovative topic as it has the potential to help live streaming services gain more popularity and increase profits.
- We propose an approach that represents live streaming chat messages and interactions among viewers as a continuous-time dynamic graph. We transform the superchat detection problem into a dynamic node label classification problem in the graph and introduce a TDGNN model that learns temporal node representations and predicts real-time superchat donations.
- Our TDGNN model has a real-time node label updating mechanism that identifies the precise timing of updating node labels. This is in contrast to traditional TGNNs, where node information is not updated until batches are finished. By solving the dynamic node label classification task, our approach can predict the exact time when a superchat appears.
- We conduct experiments to demonstrate the effectiveness of our proposed approach, which outperforms various baseline methods, including traditional machine learning algorithms, time sequence models, static graph-structured models, text classification models, and the latest TGNN models.

We would like to note that an earlier conference version of this paper was presented at the 25th International Conference on Discovery Science (DS'2022) (Jin et al., 2022). In that version, we proposed to represent chat messages and interactions among viewers as a continuous-time dynamic graph and used an existing Temporal Graph Network (TGN) model to solve the dynamic node label classification problem. However, this paper addresses some limitations of TGN, such as indiscriminate neighbor aggregation and inefficient two-step training, by introducing a novel TDGNN model with temporal difference aggregation and an end-to-end training strategy. Our experimental results show that TDGNN significantly outperforms TGN. Additionally, we provide additional experimental results to demonstrate the effectiveness of our proposed model and compare the performance of various methods.

The paper is structured as follows: Sect. 2 provides a literature review of online live streaming services and dynamic graph learning. Section 3 outlines the research goals of this study. Section 4 describes our approach for constructing dynamic graphs from live streaming chat data and predicting potential donations using the TDGNN model. Section 5 presents experimental results that demonstrate the effectiveness of our proposed approach. Finally, Sect. 6 concludes and summarizes our findings.

2 Related work

We represent the related work from two aspects: online live streaming service and dynamic graph learning.

2.1 Online live streaming service

Online live streaming services have become increasingly popular with the advent of high-speed internet and mobile devices. These services have led to the formation of various live streaming communities, which are often centered around different content genres (Hamilton et al., 2014). For instance, YouTube live streamers often share their daily life and experiences and are commonly referred to as 'vloggers' (Ladhari et al., 2020). In contrast, game

streaming channels are more popular on the Twitch[2] platform, where viewers enjoy watching others play video games to relieve stress, pass time, and engage in common topics with friends (Sjöblom & Hamari, 2017).

The live streaming platform enables creators to monetize their live videos and generate revenue from live streaming. Streamers on most live streaming platforms, such as YouTube Live, Twitch, and TikTok, can generate revenue in various ways, such as receiving real-time gifts from fans, paid subscriptions from viewers, and ads. Ads revenue of a live streaming channel is generally stable and positively correlated to the number of viewers and followers, while the amount of real-time gifts and paid subscriptions can vary greatly depending on the popularity of the streamer and the generosity of their viewers. Paid subscriptions and real-time gifts are also referred to as virtual donations. Virtual donation in live streaming is a promising topic and has attracted the attention of many researchers. Current research examines the reasons behind virtual donations, such as how they represent viewers' appreciation and approval of the streamer or recognition and happiness for shared content (Lee et al., 2019). Paid subscriptions enable viewers to make monthly donations to support their favorite channels on a recurring or one-time basis (Yu & Jia, 2022; Wohn et al., 2019; Kim et al., 2019). Subscribers can gain access to some channel benefits, such as customized emotes and badges. Real-time gifts are special gift donations or chat messages associated with a particular amount of money. They are often accompanied by special effects and highlighted in the live streaming chatbox for a while (Zhan & Zhang, 2023; Jin et al., 2022).

To investigate the impact of virtual donation revenue in live streaming, we check the revenue data of streamers from two popular live streaming platforms: Twitch and YouTube Live. According to a Twitch channel ranking site,[3] one of the most popular streamers "JYNXZI" on Twitch had 59, 459 paid subscriptions and 29, 748 gift donations in April 2023, generating more than $294,240 USD revenue. In another hand, according to a YouTube channel ranking site,[4] one of the most popular YouTube Live streamer "Pastor Jerry Eze" had 55, 358 concurrent live streaming viewers in average and 26, 757 superchat donations generating $183,108 USD in April 2023. Assuming that the number of paid subscribers is 10% of the concurrent viewers and the monthly fee of paid subscription (membership) is $5.00 USD, the revenue from paid subscription was $27,679 USD, and the total revenue from virtual donation was $210,787 USD in April 2023. The data above shows that both paid subscriptions and real-time gift donations are significant sources of revenue for streamers. Therefore, it is meaningful to conduct research on the revenue from real-time gift donations in live streaming, and YouTube Live is a suitable platform for this purpose. Some patterns of sending real-time superchat donations during live video streaming have been identified, such as only a small number of viewers sending a majority of the gifts, and viewers being motivated to send gifts after observing other viewers' gift-giving behaviors (Zhu et al., 2017). Additionally, donation information is entirely public on live streaming channels, meaning that when viewers donate to the streamer, others will notice it. Other viewers tend to be affected by such noticeable actions and are likely to follow the group and send more donations (Payne et al., 2017). Therefore, donations in online streaming services signal a group interaction and an event for viewers to interact with others. Some large-scale analyses of real-time virtual donations in on-demand video sites and online live

[2] https://www.twitch.tv/.

[3] https://twitchtracker.com/jynxzi/subscribers.

[4] https://playboard.co/en/channel/UCLg4NCAJxhIvD4IRV__LOFg.

streaming platforms have been carried out in several works (Jia et al., 2021; Lu Jia et al., 2020, 2019; Wang et al., 2019). However, those works only cover a short period of virtual donation activities and focus on revealing the static properties of viewers and streamers using naive machine learning algorithms.

2.2 Dynamic graph learning

Graph learning has produced many successful applications (Zhou et al., 2018). The main challenge in graph learning is to find an appropriate method to encode the graph structure, including nodes and edges, into low-dimensional hidden embedding vectors while preserving the topology structure and node information. These embedding vectors can be utilized by machine learning models and deep learning architectures, such as random-walk-based algorithms (Perozzi et al., 2014) and graph neural networks (Wu et al., 2019). Learning embedding vectors on graphs is widely recognized for graph-related downstream tasks, such as node classification (Kipf & Welling, 2016a), link prediction (Zhang & Chen, 2018), community detection (Interdonato et al., 2017), and graph classification (Zhang et al., 2018).

Graph Neural Networks (GNNs) have emerged as a powerful deep learning framework for graph learning. Among various GNNs, Graph Convolutional Network (GCN) (Kipf & Welling, 2016a) is one of the most widely used models that aggregates neighbor node information efficiently using a convolutional layer. To further improve the GNN performance, many researchers have proposed advanced GNN models based on the GCN structure. For example, GraphSAGE (Hamilton et al., 2017) extends GCN into inductive learning that can handle unknown nodes in graphs. Graph Attention Networks (Veličković et al., 2018) employ attention mechanisms that assign importance weights to different neighbor nodes. Graph Autoencoder and Variational Graph Autoencoder (Kipf & Welling, 2016b) are GCN-based encoder-decoder models that can handle unsupervised learning tasks. Moreover, ML-GCN and ML-GAT (Zangari et al., 2021) extend the original GCN and GAT to multilayer networks, which can capture more complex relations among nodes in large-scale graphs.

Learning on dynamic graphs is much more complex than on static graphs. Initially, research on dynamic graphs focused on discrete-time dynamic graphs, which consist of a timed sequence of graph snapshots (Liben-Nowell & Kleinberg, 2007; Sankar et al., 2020; Gao et al., 2022). Existing static graph methods can be directly applied to each graph snapshot. However, most real-life graph-structured data is in a state of constant evolution. A more general style of dynamic graph is the continuous-time dynamic graph, which consists of a timed list of events, including edge creation or deletion, node creation or deletion, and node or edge status evolution. Recently, several studies on continuous-time dynamic graphs have been proposed, including JODIE (Kumar et al., 2019), Continuous-time Dynamic Network Embedding (Nguyen et al., 2018), DyRep (Trivedi et al., 2019), Temporal Graph Networks (TGN) (Rossi et al., 2020), Temporal Graph Attention (Xu et al., 2020), Asynchronous Propagation Attention Network (APAN) (Wang et al., 2021), and Meta-learning framework MetaDyGNN (Yang et al., 2022). However, these continuous-time dynamic graph methods need two-step model training processes that require high data volume and long training time, causing low training efficiency. Additionally, they lack accuracy in predicting the exact timing of superchat messages because they have an update delay in model training and inference, rendering them impractical for our research target. A new approach

Table 1 Superchat purchase details

Purchase amount(USD)	Color	Max. message length	Max. time in the chatbox
$ 1.00–1.99	Blue	0 characters	0 s
$ 2.00–4.99	Light blue	50 characters	0 s
$ 5.00–9.99	Green	150 characters	2 min
$ 10.00–19.99	Yellow	200 characters	5 min
$ 20.00–49.99	Orange	225 characters	10 min
$ 50.00–99.99	Magenta	250 characters	30 min
$ 100.00–199.99	Red	270 characters	1 h
$ 200.00–299.99	Red	290 characters	2 h
$ 300.00–399.99	Red	310 characters	3 h
$ 400.00–499.99	Red	330 characters	4 h
$ 500.0	Red	350 characters	5 h

is needed to address these issues and predict the post time of superchat messages more accurately and efficiently.

3 Problem setup

3.1 Dataset description

We use a publicly available YouTube live streaming dataset in this study: *VTuber 1B: Large-scale Live Chat and Moderation Events Dataset*.[5] VTuber 1B is a massive collection of over a billion live chat messages, superchats, and moderation events (ban and deletion) across hundreds of YouTubers' live streams, especially for English and Japanese Streamers. Our research uses the chat message data from Mar. 2021 to Apr. 2021, including 377 live streaming channels, 5684 streaming videos, and over 58 million live chat messages. Over 54,000 viewers posted 230,025 superchat messages. Each channel has 15.07 live-streaming videos that last for 8.72 h on average. There are 172.76 viewers, 10,245.84 chat messages posted, and 9.66 superchat donations on average in each video. The total purchase amount of superchat exceeds 3.4 million USD.

The dataset contains a considerable amount of superchat donations, which are categorized into multiple levels based on their purchase amount. Each level is represented by a different color, as shown in Table 1. Higher purchase amounts allow for longer message length and longer highlighting time in the chatbox. In our research, we classify all the chat messages into significance levels regarding its donation amount, indicating whether they are superchat messages or not. These labels will be utilized as prediction targets in our proposed framework.

The dataset used in this study is based on a cluster system[6] that was specifically designed to collect data from certain YouTube channels' live streams via YouTube Live

[5] The dataset can be downloaded from https://www.kaggle.com/datasets/uetchy/vtuber-livechat.

[6] https://github.com/sigvt/honeybee.

Table 2 Detailed dataset statistics

Description	Value
Start time	2021–03–15 23:19:38
End time	2021–04–11 15:15:36
# Live streaming channels	377
# Live streaming videos	5684
# Live streaming viewers	981,996
# Chat messages	58,237,423
# Superchat messages	230,025
# Superchat donors	54,918
# Videos per channel	15.07
Duration (hrs.) per video	8.72
# Viewers per video	172.76
# Chat messages per video	10245.85
# Superchat messages per video	9.66
Total amount of super chats (USD)	$3,466,216.61

streaming API. All personal information that could identify individual users, such as usernames and profile images, has been removed from the dataset to protect their privacy. User IDs and channel IDs have also been anonymized using the SHA-1 hashing algorithm and an undisclosed salt, further ensuring user anonymity. The *VTuber 1B* dataset has been used in various research studies, including toxic chat classification, spam detection, demographic visualization, superchat analysis, and training neural language models. With over a billion live chat messages, superchats, and moderation events, this dataset is a valuable resource for researchers interested in studying online behavior and language use. By utilizing this dataset, we can gain a better understanding of how users interact with each other in live streaming chatrooms and develop more effective tools for moderating and managing online conversations.

The detailed statistics are listed in Table 2.

3.2 Reserach problem

In YouTube Live, viewers communicate with others by sending real-time chat messages, which streamers can see and interact with in real-time. In this way, these chat messages bring viewers and streamers closer together. Some of the chat messages are superchat with donations. The chat messages dataset presented above raises interesting questions: Who sends superchat messages? When are superchat messages typically sent? Can we predict superchat messages? These questions can give streamers valuable insights into their core fans and expected donation income. Predicting superchat messages can be helpful for content creators, moderators, and platform administrators to identify and engage with their most valuable viewers, leading to a more engaged and loyal audience. Our research task is to predict when and who sends superchat messages, i.e., we hope to identify the donor as soon as he/she sends a superchat.

We mathematically formalize the research problem shown in Fig. 2 as follows: Consider we have a history sequence of chat messages for the past period [0, T], i.e., $\mathbf{M} = \{\mathbf{msg}_i(t) | 0 \leq t \leq T\}$, where $\mathbf{msg}_i(t)$ include the textual content, the information about the viewer i who sent the chat message, and the corresponding timestamp t. We also

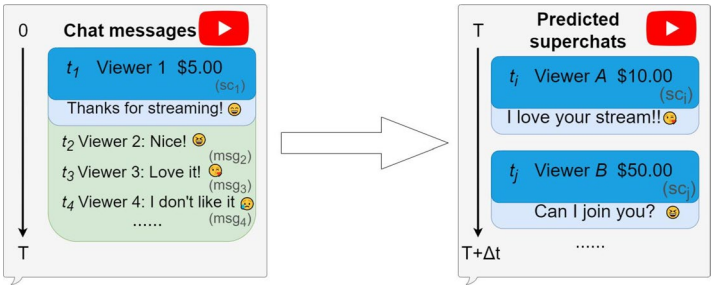

Fig. 2 Problem description: Our objective is to predict the occurrence of superchat messages and their appearance timing in a given chat message stream. Given the timed sequence of chat messages during a certain period $0 \leq t \leq T$, our method aims to predict the viewers i who send superchat message $\mathbf{msg}_i(t)$ after T and the corresponding time t

know which chat messages are superchats in $\mathbf{M}$, i.e., $\mathbf{D} = \{label(\mathbf{msg}_i(t))|0 \leq t \leq T\}$. Here $label(\mathbf{msg}_i(t))$ is a label function where $label(\mathbf{msg}_i(t)) = 1$ means $\mathbf{msg}_i(t)$ is a superchat message and $label(\mathbf{msg}_i(t)) = 0$ otherwise. For the prediction period $[T, T + \Delta T]$, we are given a sequence of chat messages in a stream fashion, i.e., $\mathbf{M}^* = \{\mathbf{msg}_i(t)|T \leq t \leq T + \Delta T\}$. Our aim is to learn the message label $\mathbf{D}^* = \{label(\mathbf{msg}_i(t))|T \leq t \leq T + \Delta T\}$ as soon as the message $\mathbf{msg}_i(t) \in \mathbf{M}^*$ was sent. The message labels indicate the information about viewers who send superchat messages and the timestamp when superchat messages are posted.

4 Methodology

In this section, we describe our proposed method for identifying real-time superchat donations in YouTube live streaming services. We have empirically observed that superchat messages have specific characteristics that distinguish them from regular chat messages. For instance, when a superchat message appears, chat messages tend to increase as the streamer expresses their gratitude for the donation, and other viewers respond. Moreover, superchat messages are often longer and crafted in a way to attract attention. Consequently, superchat messages tend to elicit a high number of response chat messages and have distinctive textual content. To leverage these observations, we use text content information and viewer interactions to identify superchat donations and its donor.

Overall, our approach consists of two main steps. Firstly, we utilize a continuous-time dynamic graph to capture the intricate relationships among a large number of viewers and chat messages, which number in the millions. The graph comprises batches of graph actions over time, such as node creation/deletion, edge creation/deletion, and node/edge status evolution. Nodes represent viewers, and edges represent interactions between them, i.e., whether two viewers have sent similar chat messages. Each node is assigned a dynamic node label, which changes with time. The label represents whether the viewer sends a superchat in a time window, i.e., if a viewer sends a superchat, its label temporarily changes from 0 to 1 until the time window ends. Thus, we transform the superchat message prediction problem into a dynamic node label classification problem in the continuous-time dynamic graph. Secondly, we develop a novel TDGNN model that analyzes the continuous-time dynamic graph. This model continuously learns evolving node representations that are finally used for label classification and superchat message donor prediction.

In the following, we first focus on the construction of the continuous-time dynamic graph. Secondly, we discuss the limitation of traditional TGNN methods and propose our TDGNN model. We explain how TDGNN predicts viewers who send real-time superchat messages and the corresponding posting time. Next, we analyze the computational complexity of TDGNN and then compare its framework with existing TGNN methods. Finally, we elaborate on the strategies used to adapt the model for training on the imbalanced live streaming dataset.

4.1 Construction of a dynamic graph

As previously mentioned, the prediction of superchat messages is based on two aspects: the textual content of chat messages, and the interactions between viewers. To achieve this, we convert the textual content of chat messages into sentence embedding vectors. Also, we measure the interaction between viewers by the similarity in meaning between the latest chat messages they post, i.e., two viewers interact with each other by sending similar chat messages. Specifically, we calculate the cosine similarity of two sentence embedding vectors of their most recently posted chat messages.

We use a pre-trained Sentence Transformers[7] (Reimers & Gurevych, 2019, 2020) language model to encode all the chat message texts into sentence embedding vectors. Sentence Transformers is a Python framework for generating state-of-the-art sentence, text, and image embeddings. It is commonly used in research tasks such as semantic textual similarity, semantic search, and paraphrase mining. Sentence Transformers has been extensively evaluated for its quality in embedding sentences (Performance Sentence Embeddings) and embedding search queries & paragraphs (Performance Semantic Search). We use a pre-trained multi-lingual model named *paraphrase-xlm-r-multilingual-v1*[8] to encode the chat messages into sentence embedding vectors. This model generates aligned vector spaces, meaning that similar inputs in different languages are mapped close together in the vector space.

Figure 3 illustrates a test of the pre-trained model's performance on sentence similarity. The embedding vectors of sentences with similar meanings have a higher sentence similarity score than those with opposite meanings. Moreover, the pre-trained model can identify the meanings of two sentences written in different languages or even in emojis. This test demonstrated that the model is capable of extracting meaningful information from short chat messages and encoding them into sentence embedding vectors while preserving their meaning.

[7] https://www.sbert.net/.

[8] https://huggingface.co/sentence-transformers/paraphrase-xlm-r-multilingual-v1.

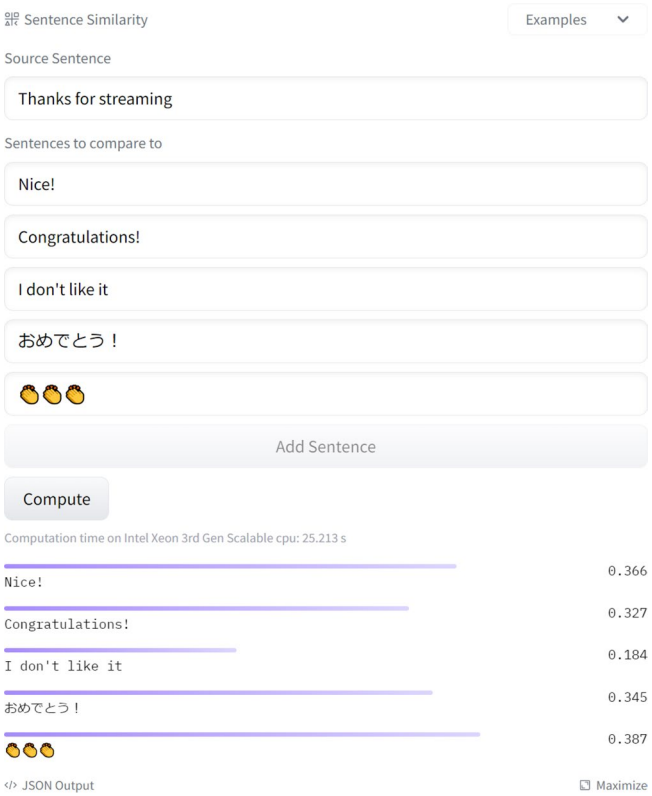

Fig. 3 We conducted a test on the pre-trained Sentence Transformers model to measure sentence similarity. We used the source sentence "Thanks for streaming" and compared it with different target sentences. Sentences with positive and similar meanings, such as "Nice!" and "Congratulations!" received higher similarity scores of 0.366 and 0.327, respectively, whereas the sentence with a negative and opposite meaning, "I don't like it", had the lowest score of 0.184. Additionally, the model could identify the meanings of sentences written in different languages or emojis. For instance, the Japanese sentence "Congratulations!" and the emoji "Clap hands" received almost identical similarity scores (0.345 and 0.387, respectively) as their English equivalents

Data: A timed sequence $\mathbf{M}$ of chat message. The time window Δt for separating batches. The threshold θ_1 for judging duplicated chat messages. The threshold θ_2 for generating edges by cosine similarity.
Result: A continuous-time dynamic graph $\mathbf{G} = (\mathbf{V}(t), \mathbf{E}(t), \mathbf{F}(t) \| t \in \mathbf{T})$, including node set $\mathbf{V}(t)$, edge set $\mathbf{E}(t)$, node feature set $\mathbf{F}(t)$ and the timestamp set of graph actions $\mathbf{T}$

initialization;
Separate $\mathbf{M}$ into several *batches* by time window Δt;
for *batch* $\leftarrow$ *batches* **do**
  for $\mathbf{msg}_i(t_1), \mathbf{msg}_j(t_2) \in \mathbf{M}[batch]$ *and* $t_1 < t_2$ `// t denotes the timestamp`
  do
    if *SequenceMatcher(*$\mathbf{msg}_i(t_1)$, $\mathbf{msg}_j(t_2)$*)* $> \theta_1$ **then**
      delete $\mathbf{msg}_j(t_2)$ `// Drop duplicated chat messages`
    end
  end
  for $\mathbf{msg}_i(t) \leftarrow \mathbf{M}[batch]$ **do**
    if *The viewer* $\mathbf{u}_i(t)$ *who posts* $\mathbf{msg}_i(t)$ *and* $\mathbf{u}_i(t) \notin \mathbf{V}(t)$ **then**
      Create a new node $\mathbf{u}_i(t) \in \mathbf{V}(t)$
    end
    Update the node feature $\mathbf{f}_i(t) \in \mathbf{F}(t)$ by the sentence embedding vector of $\mathbf{msg}_i(t)$;
    for $\mathbf{u}_j(t^-) \leftarrow$ *active_list* **do**
      if *cosine_similarity(*$\mathbf{u}_j(t^-)$, $\mathbf{u}_i(t)$*)*$> \theta_2$ **then**
        Generate an directed edge $\mathbf{e}_{ji}(t) \in \mathbf{E}(t)$ from node $\mathbf{u}_j(t^-)$ to $\mathbf{u}_i(t)$;
        Edge weight $\mathbf{e}_{ji}(t) = cosine\_similarity(\mathbf{u}_j(t^-), \mathbf{u}_i(t))$
      end
    end
    $\mathbf{u}_i(t) \rightarrow$ *active_list* `// Keep` $\mathbf{u}_t$ `active for a time window`
    for $\mathbf{u}_j(t^-)$ *in active_list* **do**
      if $(t - t^-) > \Delta t$ **then**
        delete $\mathbf{u}_j(t^-)$ from *active_list* `// Drop expired nodes`
      end
    end
  end
end

With the sentence embedding vectors, we construct a continuous-time dynamic graph that encodes viewers' chat messages and interactions in live streaming videos. The graph comprises batches of graph actions over time, such as node creation/deletion, edge creation/deletion, and node/edge status evolution. Nodes represent viewers, and edges represent interactions between them, i.e., whether two viewers send similar chat messages. Node creation/deletion occurs when viewers enter or leave the streaming channel, while edge changes occur when viewers send new chat messages and interact with others. Additionally, each node is associated with a node feature vector, which is dynamically updated based on the sentence embedding vector of the latest chat message posted by the corresponding viewer. The edge weight represents the cosine similarity of the node feature vectors of the two nodes at either end, and synchronously changes everytime node feature vectors are updated. The edge direction is always from the formerly updated node to the newly updated nodes. This ensures that new chat messages are influenced by old ones, and information propagates from old chat messages to new ones.

Algorithm 1 outlines how to construct a dynamic graph from a timed sequence of chat messages by utilizing their textual content, sentence embedding vectors, viewer ID,

timestamps, and the dynamic label indicating whether it is a superchat message. The algorithm proceeds as follows:

1. The first step is to preprocess the raw chat messages. All the chat messages $\mathbf{M}$ are separated into several batches by a time window Δt, and any duplicated, nonsensical, or too-short messages are filtered out. The *SequenceMatcher*() method from the Python library *difflib*[9] is used to check for duplicated messages. This preprocessing step helps to ensure that the subsequent graph construction is based on a clean and manageable sequence of chat messages. (lines 1-9)
2. Traverse all the chat messages $\mathbf{msg}_i(t)$ in the batch. And for each $\mathbf{msg}_i(t)$, check if the viewer i who posted it already has a node $\mathbf{u}_i(t)$ in the graph G. If not, create a new node $\mathbf{u}_i(t) \in \mathbf{V}(t)$ for the viewer i. (lines 10-13)
3. Update the node feature vector $\mathbf{f}_i(t) \in \mathbf{F}(t)$ of the node $\mathbf{u}_i(t)$ at time t by the sentence embedding vector of $\mathbf{msg}_i(t)$. (line 14)
4. Compute the cosine similarity between the newly-updated node $\mathbf{u}_i(t)$ and each node $\mathbf{u}_j(t^-)$ in the active list. Node $\mathbf{u}_j(t^-)$ is updated earlier than node $\mathbf{u}_i(t)$ ($t^- < t$). If the cosine similarity of node $\mathbf{u}_i(t)$ and $\mathbf{u}_i(t)$ is greater than the threshold θ_2, a directed temporal edge $\mathbf{e}_{ji}(t) \in \mathbf{E}(t)$ is generated between the two nodes. The edge direction is from the previously updated node $\mathbf{u}_j(t^-)$ to the newly updated node $\mathbf{u}_i(t)$. The edge weight is equal to the value of cosine similarity between the two end nodes. (lines 15-20)
5. Add the newly updated node $\mathbf{u}_i(t)$ into the active list and keep active for a time window Δt. (line 21)
6. Traverse all the nodes $\mathbf{u}_j(t^-)$ in the active list and check their timestamps t^-. The node expiring time is set equal to the time window Δt. If the interval between the current timestamp t and t^- is larger than Δt, it means that the node $\mathbf{u}_j(t^-)$ is expired and will be removed from the active list. (lines 22-26)
7. Repeat step 2 to 6 until all the graph actions are visited. (line 27)

We assign a binary dynamic node label $l_i(t) \in \{0, 1\}$ to each node in the dynamic graph. The dynamic node label, which changes with time, represents whether the viewer $\mathbf{u}_i(t)$ has sent a superchat message $label(\mathbf{msg}_i(t)) = 1$ in the past time window Δt from timestamp t:

$$l_i(t) = max(\{label(\mathbf{msg}_i(t^-))|t - \Delta t < t^- < t\}) \tag{1}$$

Label $l_i(t) = 0$ indicates that the node $\mathbf{u}_i(t)$ did not send any superchat at time t, while label $l_i(t) = 1$ indicates that the node $\mathbf{u}_i(t)$ sent a superchat at time t. More precisely, if a viewer sends a superchat, its label temporarily changes from 0 to 1 until the time window ends. We hope to track the node label changes in a relatively short time window. Thus we could identify the viewer as soon as he/she sends a superchat message.

Figure 4 is an intuitive description of constructing a dynamic graph from chat messages. The input is a batch raw chat messages. Four viewers post chat messages $\mathbf{msg}_1$ to $\mathbf{msg}_4$ at timestamp t_1 to t_4, with the first message being a superchat. Four chat messages appear in the same batch, and the chat messages are encoded to sentence embedding vectors $\mathbf{emb}_1$ to $\mathbf{emb}_4$. Node 1 to 4 are created to represent the corresponding viewers. The node features are initialized by $\mathbf{emb}_1$ to $\mathbf{emb}_4$. A dynamic node label is associated with each node to identify the superchat and normal messages. The blue node represents a viewer who sent a superchat, while the green nodes represent viewers who only sent normal chat messages.

[9] https://docs.python.org/3/library/difflib.html.

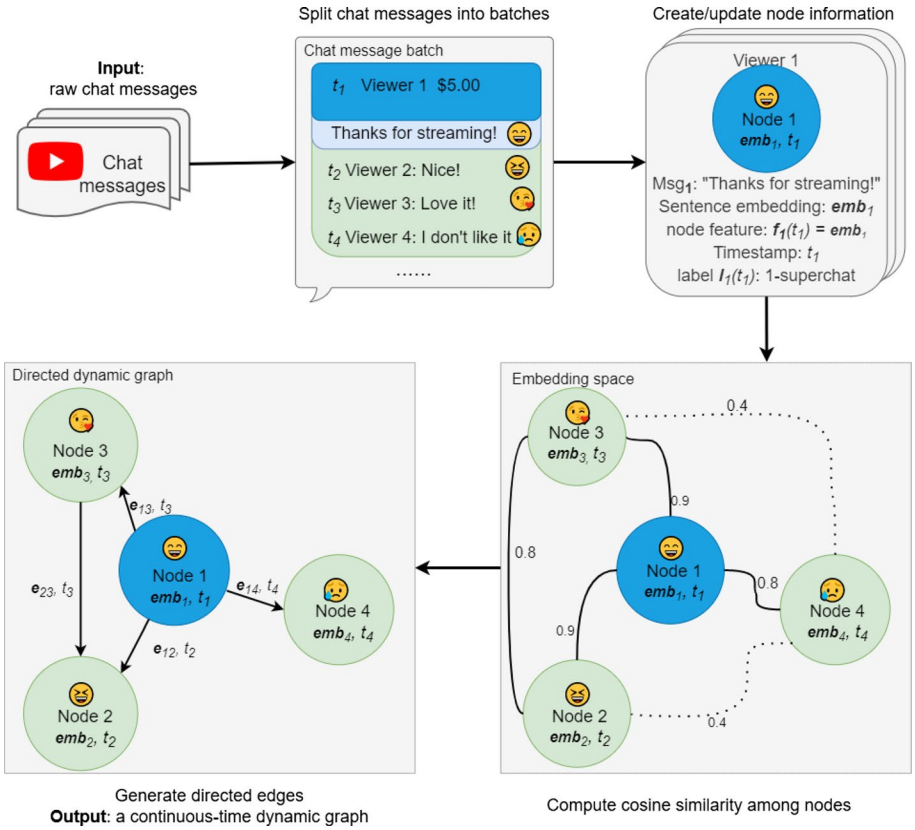

Fig. 4 The diagram illustrates the process of constructing dynamic graphs from live streaming chat messages. In this example, four viewers post chat messages at timestamps t_1 to t_4 in the same batch, with the first message being a superchat. We create nodes 1 to 4 for each viewer and initialize their feature vectors using the sentence embedding vector of the chat message they posted. Next, we compute the cosine similarity between each pair of nodes and generate edges for node pairs with high similarity. The blue node represents a viewer who sent a superchat, while the green nodes represent viewers who only sent regular chat messages

Then we calculate the cosine similarity and generate edges for the node pairs in the active node list. Node 2 and node 3 have similar opinions toward node 1, while node 4 has the opposite. Therefore, edge $\mathbf{e}_{12}$ from node 1 to node 2, edge $\mathbf{e}_{23}$ from node 2 to node 3, and edge $\mathbf{e}_{13}$ from node 1 to node 3 are generated. Node 4 responds to node 1 but does not positively correlate to node 2 and node 3. Thus, only an edge $\mathbf{e}_{14}$ from node 1 to node 4 is generated.

4.2 Limitations of traditional TGNNs

In this section, we discuss the limitations of traditional TGNNs and their potential shortcomings when applied to predicting YouTube live superchats.

TGNNs are designed for learning temporal node embeddings in dynamic graphs. Many existing TGNNs update the temporal node embedding by indiscriminate neighbor aggregation and timestamp information (Rossi et al., 2020; Wang et al., 2021; Xu et al., 2020). Moreover, once the model is successfully trained, it can be flexibly modified for different downstream tasks.

However, there are limitations to existing TGNNs that need to be addressed:

- Existing TGNNs require a two-step training process: First, training the model parameters, and then training the decoders for downstream tasks. This approach requires high data volume and results in low training efficiency.
- The existing TGNNs focus on indiscriminate neighbor aggregation, which updates the central node embedding by collecting information from neighboring nodes. However, predicting node status in the next time window requires knowing in advance the direction and rate of information propagation. This means that more information is needed beyond just collecting information from neighboring nodes.
- In the specific task of predicting superchat messages, it is necessary to predict the exact time when these messages are posted. However, existing TGNNs only update node labels after training batches are finished, resulting in a update time delay in the node label prediction task.

The prediction of superchat messages and their post time is a challenging task, and existing TGNNs need to be more efficient and accurate in this regard. A new approach is needed to tackle the problem of predicting dynamic node labels in continuous-time dynamic graphs more intelligently.

4.3 Temporal difference graph neural network (TDGNN)

To address the gaps mentioned above, we introduce an end-to-end, meticulously designed TDGNN model that predicts dynamic node labels in continuous-time dynamic graphs. The primary objective of TDGNN is to learn temporal node embedding vectors that encapsulate node features and neighborhood node information. The term *temporal difference* here implies that TDGNN's focus is on the difference between the node and its adjacent node embeddings over time. The temporal difference between two adjacent nodes represents the 'gradient', indicating the direction and rate of information propagation. When a graph event occurs, TDGNN calculates the amount of information change on adjacent nodes based on the product of temporal difference and updated node features.

The generated dynamic graph is denoted as $\mathbf{G} = (\mathbf{V}(t), \mathbf{E}(t), \mathbf{F}(t) \mid t \in \mathbf{T})$, where $t \in \mathbf{T}$ represents the timestamp. $\mathbf{u}_i(t) \in \mathbf{V}(t)$ and $\mathbf{u}_j(t) \in \mathbf{V}(t)$ represent the temporal nodes at timestamp t, while $\mathbf{e}_{ij}(t) \in \mathbf{E}(t) \subset \mathbf{V}(t) \times \mathbf{V}(t)$ is the directed temporal edge from node $\mathbf{u}_i(t)$ to node $\mathbf{u}_j(t)$ at timestamp t. The temporal edge weight is computed as the cosine similarity of two node embedding vectors. Each node $\mathbf{u}_i(t) \in \mathbf{V}(t)$ is associated with a node feature $\mathbf{f}_i(t) \in \mathbf{F}(t)$ and a temporal node label $l_i(t) \in \mathbf{L}(t)$. $\mathbf{L}(t)$ is the set of dynamic node labels for all graph actions in $\mathbf{G}$, and graph actions update the node feature $\mathbf{f}_i(t)$ and node labels $l_i(t)$ continuously.

We split timestamps $\mathbf{T}$ into three sets: a training set $\mathbf{T}^{train}$, a validation set $\mathbf{T}^{val}$, and a test set $\mathbf{T}^{test}$. Our research objective can be formalized as follows: Given a period of continuous-time dynamic graph $\mathbf{G} = (\mathbf{V}(t), \mathbf{E}(t), \mathbf{F}(t) \mid t \in \mathbf{T}^{train} \cup \mathbf{T}^{val})$ and a portion of known dynamic node labels $(\mathbf{L}(t) \mid t \in \mathbf{T}^{train} \cup \mathbf{T}^{val})$, we aim to learn a mapping function

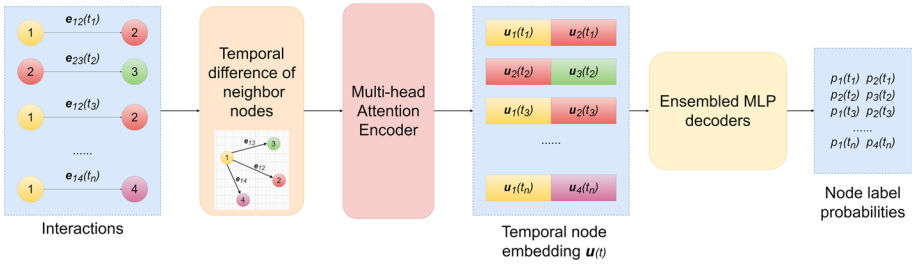

Fig. 5 Overview of the proposed temporal difference graph neural network (TDGNN)

$\mathcal{F} : \mathbf{G} = (\mathbf{V}(t), \mathbf{E}(t), \mathbf{F}(t) \mid t \in \mathbf{T}^{test}) \rightarrow (\mathbf{L}(t) \mid t \in \mathbf{T}^{test})$ to predict the remaining dynamic node labels $(\mathbf{L}(t) \mid t \in \mathbf{T}^{test})$.

The proposed TDGNN framework is illustrated in Fig. 5. It consists of three main components: a temporal difference module, a multi-head attention encoder, and an ensembled Multi-Layer Perceptron (MLP) decoder. The temporal difference module captures the difference between the node and adjacent node embeddings over time. After the graph interactions occur, the temporal difference and the updated node features are passed to the multi-head attention encoder to update the node embeddings. The updated node embeddings are then fed to the ensemble MLP decoder to compute the probabilities of predicted node labels. This work addresses the real-time dynamic node label prediction task, where node labels change over time, in contrast to traditional node label classification, where node labels are static and constant.

The detailed steps are introduced as follows:

4.3.1 Temporal difference aggregation

We define the temporal difference of node $\mathbf{u}_i$ at time t as:

$$\partial \mathbf{u}_i(t) = \sum_{n \in \mathcal{N}_i} \left(\mathbf{u}_n(t) - \mathbf{u}_i(t) \right) \cdot \mathbf{e}_{ni}(t), \tag{2}$$

where $\mathcal{N}_i = \{\mathbf{u}_n(t) \mid \mathbf{e}_{ni}(t) \in \mathbf{E}\}$ denotes the set of neighbor nodes of $\mathbf{u}_i(t)$, and $\mathbf{e}_{ni}(t)$ denotes the edge weight.

The concept of temporal difference is crucial in understanding the direction and rate of information propagation in a directed graph. In the case of our proposed model, the node embedding difference serves as the 'gradient' of information propagation from all neighboring nodes at time t, while the edge weight indicates the strength of the connection. In essence, $\partial \mathbf{u}_i(t)$ represents the total amount of information propagated on node $\mathbf{u}_i$ from all its neighbor nodes at time t. To illustrate this point, consider the example shown in Fig. 6, where node $\mathbf{u}_1$ passes information to its three connected neighbors, namely nodes $\mathbf{u}_2$, $\mathbf{u}_3$, and $\mathbf{u}_4$. In this scenario, $\partial \mathbf{u}_1$ is the amount of information that is passed on to these nodes.

4.3.2 Attention encoder

The TDGNN model updates node embeddings by utilizing both temporal difference aggregation and updated node features. Inspired by the Fundamental theorem of calculus, temporal difference acts as a gradient' in node information propagation, while the updated

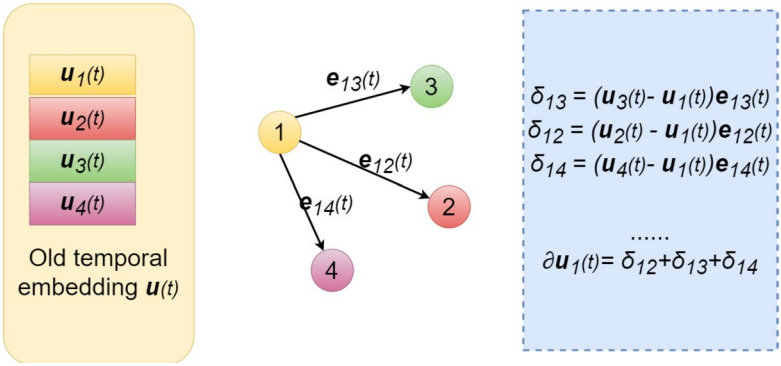

Fig. 6 Compute the temporal difference in a directed graph. Temporal difference $\partial \mathbf{u}$ represents the information difference between node $\mathbf{u}$ and the surrounding neighbor nodes, indicating the amount of information propagation in dynamic graphs

node feature plays the role of step size'. The TDGNN calculates the product of temporal difference and the updated node features, which represents the total amount of information change. It then adds the result to the old node embeddings to compute the new node embeddings. The complete procedure is shown in the following equation:

$$\mathbf{u}_i(t+1) = \mathbf{u}_i(t) + \phi\big(\partial \mathbf{u}_i(t), \mathbf{f}_i(t)\big), \tag{3}$$

where ϕ represents a function that computes the change in the value of updated node feature $\mathbf{f}_i(t)$ regarding the node $\mathbf{u}_i$. The multi-head attention module is an efficient mechanism to combine the temporal difference and node features. The attention layer works by computing the dot product of a query vector with a set of key vectors, resulting in a weight vector that assigns importance scores to the values. The mechanism is defined as follows:

$$Attn(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \tag{4}$$

$$\mathbf{Q} = \big[\mathbf{f}_i(t) \| \delta t_s\big]\mathbf{W}_Q \tag{5}$$

$$\mathbf{K} = \big[\partial \mathbf{u}_i(t) \| \delta t_s\big]\mathbf{W}_K, \mathbf{V} = \big[\partial \mathbf{u}_i(t) \| \delta t_s\big]\mathbf{W}_V \tag{6}$$

$$head_i = Attn\big(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i\big) \tag{7}$$

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [head_1 \| head_2 \| \cdots]\mathbf{W}, \tag{8}$$

where δt_s denotes the time interval since last update, $[\cdot\|\cdot]$ represents the concatenation of matrices, and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_K$ are training parameters. As shown in Fig. 7, the node feature vector is fed into the matrix $\mathbf{Q}$, while the temporal difference vector is fed into matrices $\mathbf{K}$ and $\mathbf{V}$. To incorporate the time order of node updates, a time encoder is used to encode the time information into vectors that are also fed into the multi-head attention layer. This allows the attention mechanism to take into account both the node features and the temporal difference information at each time step, as well as the time order of the updates.

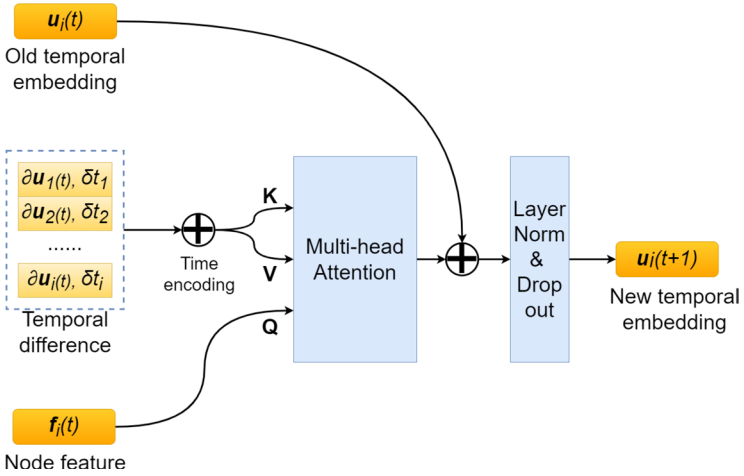

Fig. 7 A multi-head attention module that calculates the new temporal node embedding $\mathbf{u}_i(t+1)$ according to the relativity between the last updated embedding $\mathbf{u}_i(t)$, temporal difference of neighbor nodes, and newly-updated node features

Finally, the output of the multi-head attention encoder is concatenated with the old node embeddings:

$$\mathbf{u}_i(t+1) = \mathbf{u}_i(t) + \phi\big(\partial\mathbf{u}_i(t), \mathbf{f}_i(t)\big) \tag{9}$$

$$= \mathbf{u}_i(t) + MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \tag{10}$$

A normalization layer is set here to limit the mean and variance of the obtained node embeddings.

4.3.3 Ensemble MLP decoder

Node embeddings can be used for various downstream tasks. In this work, an ensemble MLP decoder is used to solve the real-time node label prediction task. Ensemble learning techniques are effective for imbalanced data (Galar et al., 2011). They combine the results from several classifiers to improve the performance of a single classifier by reducing the variance of results. On the other hand, single classifiers are easily affected by the imbalanced dataset and tend to skew towards the majority classes, resulting in poorer performance on the minority classes. The ensemble MLP decoder is composed of multiple individual MLPs. Each MLP maps the learned node embedding $\mathbf{u}_i(t)$ to the probability of node labels. The sum of all MLP outputs is then fed to a softmax function to assign a predicted node label probability $\hat{y}_i(t)$.

$$\hat{y}_i(t) = Softmax\left(\frac{1}{N}\sum_N MLP\big(\mathbf{u}_i(t)\big)\right) \tag{11}$$

Finally, we use a Binary Cross Entropy Loss function to train the TDGNN model.

Table 3 Structure distinction among TGNNs

	Aggregator	Encoder	Classifier	Training
TGAT	Self-attention	Multi-head attention	MLP	Two-step
TGN	Memory & message aggregator	Multi-head attention	MLP	Two-step
APAN	Asynchronous mail propagator	Multi-head attention	MLP	Two-step
MetaDyGNN	Hierarchically adaptive Meta-Learner	Attention mechanism	MLP	End-to-end
TDGNN	Temporal difference aggregation	Multi-head attention	Ensembled MLPs	End-to-end

$$BCELoss(\hat{y}_i, l_i) = -\big(l_i \cdot \log(\hat{y}_i) + (1 - l_i) \cdot \log(1 - \hat{y}_i)\big), \tag{12}$$

where $\hat{y}_i$ is the predicted node label probability and $l_i \in \mathbf{L}$ is the true node label.

4.4 Computational complexity

We analyze the computational complexity of the Temporal Difference Aggregation, Multi-head Attention Encoder, and Ensemble MLP decoders. With regard to Eqs.(2), (4)~(8), (11), the complexity is dominated by matrix multiplication. We assume that the number of edges(graph actions) is N, and the node embedding $\mathbf{V}(t)$ and node features $\mathbf{F}(t)$ have the same dimension D for the hidden units. The complexity of Temporal Difference Aggregation is $\mathcal{O}(ND)$. The complexity of the Multi-head Attention Encoder is $\mathcal{O}(N^2D + ND^2)$. The complexity of Ensemble MLP decoders is $\mathcal{O}(ND)$. Therefore, the total complexity of TDGNN is $\mathcal{O}(ND) + \mathcal{O}(N^2D + ND^2) + \mathcal{O}(ND) = \mathcal{O}(ND(N + D + 2)) = \mathcal{O}(ND(N + D))$.

Considering real-time computation, we must process the input graph actions in batches within a certain time window. We assume that all the batches have the same size K, and then the total number of batches is N/K. In this case, the total complexity of TDGNN is $\mathcal{O}((N/K) \times (KD(K + D))) = \mathcal{O}(ND(K + D))$. Since K is much smaller than N, the complexity in the real-time computation case is smaller than in the normal computation case. The complexity analysis above demonstrates that our framework works more efficiently in the case of real-time computation.

4.5 TDGNN vs. TGNNs

Table 3 compares the structural distinctions between our TDGNN model and three state-of-the-art TGNNs: TGAT (Xu et al., 2020), TGN (Rossi et al., 2020), APAN (Wang et al., 2021), and MetaDyGNN (Yang et al., 2022). TGNNs have similar indiscriminate neighbor aggregators associated with their respective modules: Self-attention, Memory & message aggregator, Asynchronous mail propagator, and hierarchically adaptive Meta-learner. TGAT, TGN, and APAN all use a Multi-head attention layer as the encoder and an MLP as the classifier. Additionally, they all require two-step training: training the model parameters first, and then training the decoder parameters for specific downstream tasks. MetaDyGNN is comparably a more lightweight framework since it develops a novel hierarchically adaptive meta-learner, and does not use the multi-head attention encoder and two-step training. On the other hand, the proposed TDGNN model has a temporal difference aggregator and an ensembled MLP classifier, and it is trained end-to-end. The proposed TDGNN model

addresses the limitations of existing TGNNs mentioned in Sect. 4.2 from the following aspects:

1. TDGNN is an end-to-end model that only needs to be trained once. It is more efficient and convenient than those TGNNs that require two-step training.
2. The temporal difference module computes both the direction and rate of information propagation, which is significant for predicting temporal node embeddings in the future.
3. TDGNN updates temporal node labels in real-time during the training batches, enabling the prediction of dynamic node labels and the time at which they were changed. In contrast, TGNNs update temporal nodes after the training batch is finished, causing a delay in predicting temporal information.

4.6 Strategies for data imbalance

The superchat donations only make up a small fraction of all the chat messages in the live stream, resulting in a highly imbalanced dataset that confuses the model and leads to poor performance. To address this issue, we employ the following strategies to mitigate the negative impact of data imbalance. In Sect. 5.6, we present experimental results that demonstrate how these strategies affect the imbalance ratio and model performance.

4.6.1 Filtering on original data

We have refined the dataset by removing duplicated and meaningless chat messages, as well as filtering out messages that are too short. To do this, we first removed punctuation marks, numbers, and exceptional control characters from the chat messages. We then filtered out messages with fewer than 5 characters. Finally, we used the Python library *difflib* to check for duplicate messages that appear within a certain time window. As a result of this filtering process, 1.6% of the non-superchat samples in the dataset were removed.

4.6.2 Tuning on graph generation

We controlled the number of dynamic edges and further the proportion of positive samples by adjusting the cosine similarity threshold θ_2 in Algorithm 1. A higher θ_2 generates fewer edges, and a lower θ_2 generates more edges. Specifically, we set θ_2 as $\cos(\pi/12)$. Additionally, we tested two other values of θ_2, namely $\cos(\pi/6)$ and $\cos(\pi/3)$, and evaluated their impact on the model performance.

4.6.3 Undersampling on training samples

We employed an under-sampling strategy to address the imbalance between positive and negative samples in our dataset. We kept all the data in the minority class and reduced the size of the majority class during model training. This approach corrected the imbalanced data and reduced the risk of skewing towards the majority class. We tested various undersampling ratios of the two classes in our experiments and ultimately set the ratio as $1:1$.

Table 4 Cost matrix

	Actual negative	Actual positive
Predicted negative	$C(0, 0)$, TN	$C(0, 1)$, FN
Predicted positive	$C(1, 0)$, FP	$C(1, 1)$, TP

4.6.4 Cost-sensitive loss function

We apply a cost-sensitive learning method to self-adjust the penalty factor in the loss function during model training. Most machine learning algorithms assume that all misclassification errors made by a model are equal. This is often not the case for imbalanced classification problems where missing a minority class case is worse than incorrectly classifying an example from the majority class. Cost-sensitive learning is a subfield of machine learning that takes the costs of prediction errors into account when training a machine learning model (Elkan, 2001). In cost-sensitive learning, instead of each sample being either correctly or incorrectly classified, each class is given a misclassification cost. Thus, instead of trying to optimize the accuracy, the problem is then to minimize the total misclassification cost.

Specifically, we set a cost matrix that assigns a cost to each cell in the confusion matrix. Table 4 shows the cost matrix we used, where $C(,)$ denotes the cost of predicting one class when the actual class is another. The acronyms of each cell from the confusion matrix are also listed (e.g., False Positive is FP).

This work defines costs based on the inverse class distribution, assuming a minority-to-majority class ratio of $1 : N$ in the dataset. We invert this ratio to obtain the cost of misclassification errors, where the cost of a False Negative $C(0, 1)$ is N, and the cost of a False Positive $C(1, 0)$ is 1. The cost of True Negative $C(0, 0)$ and True Positive $C(1, 1)$ are set as 0 since they are correctly predicted. Therefore, the total cost of a classifier is defined as the cost-weighted sum of False Negatives and False Positives using this framework:

$$Total\_cost = C(0,1) * FN + C(1,0) * FP + C(0,0) * TN + C(1,1) * TP \tag{13}$$

$$= N * FN + 1 * FP \tag{14}$$

5 Experiments

We conducted experiments on the task of predicting dynamic node labels. First, we explain the setup of experiments and baselines used in our study. Then, we evaluate the experimental results and model performance. Finally, we discuss additional factors such as training time, time delay in real-time prediction, effect of imbalance strategies, parameter sensitivity, and node embedding visualization.

Table 5 Statistics of the live streaming dynamic graphs

Dataset length	Short	Mid	Long
Durations (hrs.)	8.61	47.22	78.49
# Nodes	6,225	28,582	41,156
# Edges	1,660,813	9,498,600	15,097,110
# Positive labels	105,207	258,079	525,964
% Positive labels	6.3%	2.7%	3.4%

5.1 Setup of the experiment

We prepared three continuous-time dynamic graphs from the dataset mentioned in Sec. 3.1. The detailed statistics are listed in Table 5. The three graphs represent videos of different lengths: the *Short* dataset contains chat messages in an 8-hour live streaming video, the *Mid* dataset contains chat messages in a 47-hour video compilation of a week, and the *Long* dataset contains chat messages in a 78-hour video compilation of two weeks. The model training is customized for each streamer to optimize prediction performance. However, it is worth noting that the ratios of positive labels are minimal in all three dynamic graphs as superchat messages only represent a minor portion of all chat messages in real-world scenarios. Therefore, the experiments will be conducted on an imbalanced dataset. We split the chat messages in the dataset into training, validation, and test sets based on the time order. The first 50%/70%/90% of chat messages are used as the training set, and the remaining messages are equally divided into validation and test sets.

Our model processes the chat messages into batches based on the time window. Within each time window, the chat messages (represented as updated node features) and node interactions (represented as edges) are included as a sequence of graph actions. All these sequences are then fed into our model to predict the dynamic node labels (when and who sends superchat). The model updates the involved node status and edges based on the graph action sequence and provides predictions on the changes in dynamic node labels.

We fine-tune the common model hyperparameters, such as learning rate, batch size, and drop-out rate, by manual search. Also we conduct parameter sensitivity experiments for some special hyperparameters in TDGNN by grid search, which will be introduced in Sect. 5.6 and Sect. 5.7. The model is trained using the Adam optimizer with a learning rate of 0.0001, a batch size of 5, 000 for training, validation, and testing, and a dropout rate of 0.2. We set the number of attention heads to 2. For the ensembled MLP decoders, we use a three-layer linear neural network with hidden sizes of 64 and 10. The training process is limited to a maximum of 20 iterations. Furthermore, if the validation loss does not improve for five consecutive iterations, training will be stopped early.

The node embedding dimension and node feature dimension are set to 128. The number of maximum neighbor sampling is 10, and the number of ensembled MLP decoders is 15 for all three datasets in default. We will test the parameter sensitivity in the following experiments and prove that our proposed model results are not sensitive to hyperparameters.

5.2 Baselines

We consider traditional decision tree methods, sequence-based models, static graph representation learning methods, NLP text classification methods, and TGNNs as baselines, as listed below.

Table 6 Input information required by the baselines

		Chat messages texts	Graph structure	Temporal information
Gradient boosting algorithms	GBDT	✓	×	×
	XGBoost	✓	×	×
Time sequence model	LSTM-FCN	✓	×	✓
	ALSTM-FCN	✓	×	✓
Static graph methods	GCN	✓	✓	×
	GAT	✓	✓	×
NLP	BERT	✓	×	×
Dynamic graph methods	TGAT	✓	✓	✓
	APAN	✓	✓	✓
	TGN	✓	✓	✓
	MetaDyGNN	✓	✓	✓
Proposed methods	TDGNN	✓	✓	✓

1. *GBDT* A Gradient Boost Decision Tree (GBDT) classifier from the scikit-learn toolkit.
2. *XGBoost* An ensemble gradient boosting decision tree model from XGBosst library.
3. *LSTM-FCN* (Karim et al., 2018) A time sequence model combining long short-term memory (LSTM) networks and a fully convolutional network (FCN).
4. *ALSTM-FCN* (Karim et al., 2018) An alternative LSTM-FCN with attention layers following the LSTM cells.
5. *GCN* (Kipf & Welling, 2016a) Graph Convolutional Networks (GCN) on static graphs.
6. *GAT* (Veličković et al., 2018) Graph Atention Networks (GAT) on static graphs.
7. *BERT* (Devlin et al., 2018) Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model for NLP pre-training.
8. *TGAT* (Xu et al., 2020) A temporal graph attention structure to aggregate temporal-topological neighborhood features and to learn the time-feature interactions.
9. *APAN* (Wang et al., 2021) An asynchronous continuous time dynamic graph algorithm for real-time temporal graph embedding.
10. *TGN* (Rossi et al., 2020) A generic, efficient neural network framework for deep learning specialized in continuous-time dynamic graphs.
11. *MetaDyGNN* (Yang et al., 2022) A meta-learning framework for few-shot scenarios in dynamic networks.

Table 6 demonstrates the input information required by each baseline method. We keep the default parameter settings in respective methods during training and testing. The detailed experimental settings of baselines are as follows:

1. The gradient boosting algorithms (GBDT and XGBoost) receive every individual sentence embedding vector of chat messages and generate a predicted label indicating

Table 7 AUC scores for predicting the real-time node labels

Model	Training set ratio								
	50%			70%			90%		
	Short	Mid	Long	Short	Mid	Long	Short	Mid	Long
GBDT	0.500	0.472	0.500	0.552	0.469	0.515	0.489	0.458	0.472
XGBoost	0.500	0.509	0.500	0.553	0.495	0.518	0.513	0.462	0.499
LSTM-FCN	0.468	0.505	0.507	0.497	0.499	0.506	0.431	0.499	0.500
ALSTM-FCN	0.485	0.505	0.508	0.499	0.499	0.499	0.500	0.501	0.472
GCN	0.500	0.499	0.499	0.500	0.499	0.499	0.500	0.499	0.499
GAT	0.510	0.510	0.510	0.531	0.531	0.531	0.548	0.548	0.548
BERT	0.630	0.560	0.580	0.570	0.570	0.590	0.620	0.540	0.670
TGAT	0.562	0.655	0.649	0.581	0.739	0.672	0.609	0.785	0.686
APAN	0.493	0.604	0.664	0.484	0.599	0.654	0.553	0.626	0.641
TGN	0.654	0.610	0.784	0.620	0.610	0.795	0.520	0.854	0.902
MetaDyGNN	0.616	0.671	0.673	0.624	0.689	0.684	0.771	0.746	0.695
TDGNN	**0.765**	**0.738**	**0.799**	**0.679**	**0.765**	**0.805**	**0.805**	**0.884**	**0.916**
TDGNN w/o Diff	0.708	0.572	0.688	0.599	0.621	0.653	0.628	0.834	0.830

The best scores are given in bold

whether the corresponding chat message is a superchat. The implementation of these algorithms is based on scikit-learn[10] and XGBoost.[11]

2. The time sequence models (LSTM-FCN and ALSTM-FCN) achieved state-of-the-art performance on the task of time sequence classification. We first separate the sequences of sentence embeddings over time for each viewer and then feed these sequences to the models. The models generate a sequence of hidden state embeddings for each viewer, which are then used to predict the labels of chat messages. The implementation of these models refers to the GitHub repository LSTM-FCN.[12]
3. The static graph methods (GCN and GAT) are commonly used to model graph-structured real-world entities. We built static graphs to represent viewer relations based on the continuously-time dynamic graphs generated in Sect. 4.1. The nodes represent viewers, and the edge weights represent the frequency of edge changes in the dynamic graphs. The implementation is based on the GitHub repositories PyGCN[13] and GAT.[14]
4. BERT is a transformer-based machine learning technique for NLP pre-training. We exploit a Japanese BERT pretrained model to encode the chat messages and integrate them as long paragraphs for each viewer. The BERT for sequence classification model provided by Huggingface[15] is used to process the long paragraphs and predict the appearance of superchats.

[10] https://scikit-learn.org/.
[11] https://xgboost.readthedocs.io/en/stable/.
[12] https://github.com/titu1994/LSTM-FCN.
[13] https://github.com/tkipf/pygcn.
[14] https://github.com/psh150204/GAT.
[15] https://huggingface.co/cl-tohoku/bert-base-japanese

5. The dynamic graph neural networks (TGAT, APAN, TGN, and MetaDyGNN) use the same inputs and hyper-parameters as the proposed TDGNN. The implementation refers to the GitHub repositories TGAT,[16] APAN,[17] TGN,[18] and MetaDyGNN.[19]

5.3 Evaluation

Table 7 presents the experimental results of both the TDGNN model and the baselines. We evaluated the performance in terms of the Area under the ROC Curve (AUC) score, which measures the model's prediction quality, regardless of the classification threshold and how the datasets are imbalanced. Our proposed model shows the best overall performance, achieving the highest AUC scores on all three datasets and all training set ratios, which are marked in bold in the table. We attribute the excellent performance of TDGNN to the deliberately designed continuous-time dynamic graph that considers continuously changing edges and frequently updated node embeddings. We analyzed the performance of the baselines and inferred the reasons for their respective performances as follows:

1. The gradient boosting algorithms (GBDT and XGBoost) only consider the sentence embeddings of chat messages. However, as the textual content of superchat messages is not distinct from normal chat messages, their performance suffers a loss ranging from 0.126 to 0.444 compared to TDGNN.
2. The time sequence models (LSTM-FCN and ALSTM-FCN) receive the sequence of sentence vectors and generate a sequence of hidden state embeddings for each viewer. However, as a majority of viewers never send superchat messages, the sequences of those viewers cannot provide efficient training, resulting in a performance drop ranging from 0.182 to 0.416 compared to TDGNN.
3. The static graph methods (GCN and GAT) are good at exploiting graph structure information and edge interactions among nodes. However, static graph models are not designed to fit temporal data, and each node only stores one sentence embedding, leading to the loss of much temporal and chat message information. The experimental results indicate a performance decline between 0.179 to 0.368 compared to TDGNN.
4. The BERT model is not designed to capture temporal information, and it only stores the all-time sentence embeddings of chat messages, without considering graph structure information. As a result, it exhibits a performance decline between 0.109 to 0.246 compared to TDGNN.
5. Although dynamic graph neural networks (TGAT, APAN, TGN, and MetaDyGNN) are designed for continuous-time dynamic graphs, the performance of these models depends on their underlying architectures and their ability to capture both temporal and graph structure information effectively. TDGNN outperforms TGAT, APAN, TGN, and MetaDyGNN with a performance gain between 0.026 to 0.285, which can be attributed to the distinctive architecture of TDGNN as explained in Sect. 4.5.

[16] https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs.
[17] https://github.com/WangXuhongCN/APAN.
[18] https://github.com/twitter-research/tgn
[19] https://github.com/BUPT-GAMMA/MetaDyGNN.

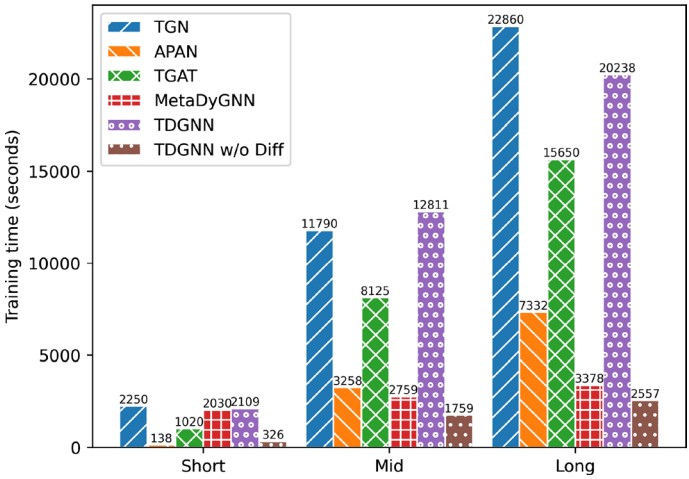

Fig. 8 Training time (seconds) per epoch in three dynamic graph datasets

In addition, we conducted an experiment to verify the contribution of the temporal difference module in the TDGNN model. We tested the TDGNN model by replacing the temporal difference module with the raw aggregated neighbor node embeddings, and the results are presented in the table as TGDNN w/o Diff. The TGDNN w/o Diff model showed a performance drop ranging from 0.040 to 0.177 compared to TDGNN. The results indicate that the temporal difference module plays a crucial role in the TDGNN model and its omission can notably hinder the performance.

5.4 Training Time

In this section, we compare the training time of our proposed model with that of other baselines, particularly the three TGNN baselines: TGAT, APAN, TGN, and MetaDyGNN. Given that our proposed model needs to process real-time live streaming chat messages and predict potential superchat donors as soon as possible, it is crucial to be highly time-efficient. To evaluate the training time, we ran the models on three live streaming dynamic graphs using an Intel Xeon Platinum 8360Y CPU (9 cores, 2.40GHz) and an NVIDIA A100 for NVLink 40GiB HBM2 GPU.

Figure 8 demonstrate the training time (seconds) per each epoch on the three live streaming dynamic graph datasets. We compute the total training time of TGN, TGAT, APAN, and MetaDyGNN, including the model parameter and decoder training. TDGNN has close time efficiency to TGN. Training the TGN model takes longer than the TDGNN model for both short and long datasets, while the TDGNN model takes longer to train for the mid dataset. The TGAT model takes less time to train than both the TDGNN and TGN models, and the APAN and MetaDyGNN models take the least training time out of all the models.

We believe that the time efficiency of these models is largely dependent on their respective structures. TGAT is a classic TGNN structure that utilizes a neighbor node aggregation module and a multi-head attention encoder. TGN, on the other hand, improves upon TGAT by adding a memory module to update temporal node embeddings. However, this extra component requires more training time. APAN implements an asynchronous propagation

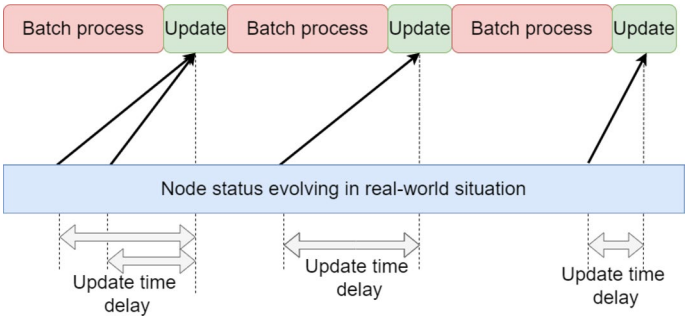

Fig. 9 The update time delay in previous TGNNs. Node status updated during batch process are reflected until the batches are finished, thus resulting in an update time delay

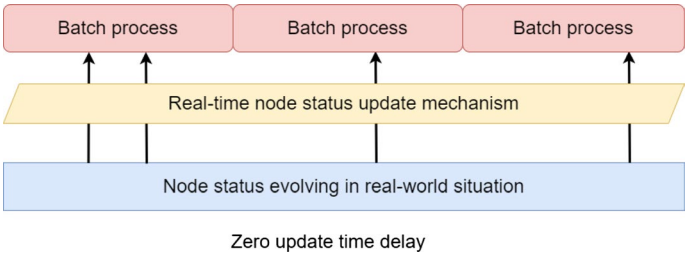

Fig. 10 The zero update time delay in TDGNN. TDGNN have a real-time node status update mechanism that can update node information during batch process

mechanism that greatly reduces training time. MetaDyGNN also performs less training time for its lightweight framework. As a result, our proposed TDGNN model achieves the best AUC score and has training time that is comparable to TGN. Additionally, since TDGNN is an end-to-end structure that only needs to be trained once, it is more convenient than other TGNN baselines. Additionally, it is worth noting that TDGNN without the temporal difference module (TDGNN w/o diff) takes considerably less time than TDGNN. This indicates that the temporal difference module is the primary factor that significantly increases the training time.

5.5 Time delay when updating node status

In the temporal node prediction task, the node status (e.g., node embeddings, node labels) is constantly evolving in continuous-time dynamic graphs. As shown in Fig. 9, TGN, TGAT, and APAN process the graph actions in batches, which may result in some delay in training and inference. This delay can lead to a situation where a node's status has changed in the real world, but the change was not reflected in the models until the batch processing was completed. We refer to this delay as the *update time delay*. The update time delay can significantly affect the timeliness and freshness of the predicted results, especially in constantly evolving situations. If the delay is too long, the predicted results may become irrelevant as the target node status would have changed. Moreover, the outdated node status can negatively impact the model training. Our research target is to identify the exact timing

Table 8 Update time delay (seconds) and AUC scores in TGNNs with different batch sizes

Batch size	Update time delay					AUC score				
	TGN	TGAT	APAN	MetaDyGNN	TDGNN	TGN	TGAT	APAN	MetaDyGNN	TDGNN
100	1.97	2.27	1.21	15.75	0.00	0.392	0.528	0.603	0.655	0.465
200	3.96	4.55	2.42	34.75	0.00	0.425	0.524	0.644	0.771	0.596
500	9.94	9.59	4.35	OOM	0.00	0.410	0.520	0.601	OOM	0.602
1000	20.42	22.79	5.40	OOM	0.00	0.499	0.520	0.656	OOM	0.638
2000	39.76	45.58	12.01	OOM	0.00	0.520	0.591	0.667	OOM	0.741
5000	99.22	117.93	36.72	OOM	0.00	0.609	0.553	0.520	OOM	0.805
10,000	197.76	227.93	48.77	OOM	0.00	0.507	0.528	0.559	OOM	0.608
20,000	392.86	455.87	60.60	OOM	OOM	0.460	0.453	0.519	OOM	OOM
50,000	988.93	941.80	137.42	OOM	OOM	0.513	0.443	0.572	OOM	OOM
100,000	1911.95	1824.74	364.13	OOM	OOM	0.488	0.476	0.502	OOM	OOM

when node labels change and predict the exact timing when superchats are posted. Therefore, it is critical to minimize the update time delay during model training and inference to achieve accurate predictions that are sensitive to time.

To mitigate the impact of update time delay, the proposed TDGNN model is equipped with a real-time node status updating mechanism that allows for the simultaneous updating of node status during training batches, as illustrated in Fig. 10. Unlike TGN, TGAT, and APAN, which are constrained by the batch size, TDGNN can update node status as soon as it changes, regardless of batch size limitations. As a result, TDGNN can provide more timely predictions for predicting the timing of superchat messages.

Table 8 shows the average update time delay (in seconds) and AUC performance of TGNNs with different batch sizes. Unlike TGN, TGAT, APAN. and MetaDyGNN, TDGNN has a real-time updating mechanism, and thus has zero update time delay, regardless of batch size. However, zero update time delay only means that node status in batches is updated as soon as the node changes in the real world, and does not necessarily imply that the predicted results given by TDGNN have zero error. In fact, out-of-memory errors can occur when the batch size is too large, presumably due to the high computational cost associated with the real-time node status updating mechanism for large-scale batches.

The update time delays in TGN and TGAT are almost directly proportional to the batch sizes, while the AUC scores do not fluctuate significantly. In contrast, APAN trains the model asynchronously, and the update time delay is less affected by batch sizes. MetaDyGNN has a hierarchical structure and thus sensitive to the batch size. Small increase in batch size will cause large extra demand in GPU memory and tend to suffer from OOM(Out-of-Memory) problem. However, the AUC scores decrease significantly as batch sizes increase. Notably, TGN and TDGNN perform poorly in terms of AUC scores when batch sizes are too small, likely due to the imbalanced dataset. When the batch size is too small, there may be only a few positive samples or even no positive samples in the batches, causing the model to produce identical prediction results for all samples, which results in low training efficiency. TDGNN performs better than TGN in this scenario, thanks to several strategies employed to alleviate the influence of imbalanced datasets.

In summary, our study highlights the trade-off between model performance and update time delay in the temporal node prediction task. While large batch sizes require significant computational resources, small batch sizes result in poor performance, particularly in imbalanced datasets. Our approach strikes a balance between the two factors by using a batch size of 5,000, enabling us to achieve high performance while maintaining low update time delay. Our findings have implications for other similar tasks, where researchers need to carefully weigh the costs and benefits of different model architectures and batch sizes to optimize both performance and efficiency.

5.6 Effect of the strategies for data imbalance

In this section, we discuss the effects of the strategies for addressing data imbalance that were mentioned in Sect. 4.6. Specifically, we test how the cosine similarity threshold, undersampling ratios, and cost-sensitive loss function impact both the data imbalance and the model performance. The experiment is conducted on the mid dataset, with a training set ratio of 90%.

The cosine similarity thresholds θ_2 have an impact on the number of dynamic edges and further the proportion of positive samples in the dataset. We tested three values of θ_2: $\cos(\pi/3)$, $\cos(\pi/6)$, and $\cos(\pi/12)$. With $\theta_2 = \cos(\pi/3)$, we obtained 840,732

Table 9 AUC scores under the effect of strategies for data imbalance

Cosine similarity threshold θ_2		$\cos(\pi/3)$	$\cos(\pi/6)$	$\cos(\pi/12)$
Imbalance ratio		10.3%	2.4%	2.7%
With cost-sensitive	No undersampling	0.717	0.736	0.716
	Neg:Pos = 1 : 1	0.761	0.780	0.884
	Neg:Pos = 3 : 1	0.742	0.712	0.820
	Neg:Pos = 5 : 1	0.737	0.676	0.700
W/o cost-sensitive	No undersampling	0.498	0.619	0.714
	Neg:Pos = 1 : 1	0.761	0.780	0.884
	Neg:Pos = 3 : 1	0.618	0.683	0.727
	Neg:Pos = 5 : 1	0.653	0.652	0.698

positive samples out of a total of 8,129,340 samples, resulting in an imbalance ratio of 10.3%. With $\theta_2 = \cos(\pi/6)$, we obtained 271,925 positive samples out of 11,176,545 samples, resulting in an imbalance ratio of 2.4%. Finally, with $\theta_2 = \cos(\pi/12)$, we obtained 258,079 positive samples out of 9,498,600 samples, resulting in an imbalance ratio of 2.7%.

The undersampling ratio refers to the ratio between the majority and minority classes in training batches, and it reduces the risk of the model predicting results that skew towards the majority class. We test three different undersampling ratios: 1 : 1, 3 : 1, and 5 : 1. Additionally, we also test the model's performance without using any undersampling strategy, which means that the default imbalance ratio of the original dataset is maintained.

The cost-sensitive loss function adjusts the penalty factor in the loss function during model training. We use the *BCEWithLogitsLoss* as the loss function, and the cost of positive samples is dynamically adjusted based on the ratio of negative samples to positive samples in each training batch. Specifically, when we use undersampling and set the undersampling ratio to 1 : 1, the cost-sensitive loss function is equivalent to the *BCELoss* function.

Table 9 shows the test results of the strategies for addressing data imbalance. Among the three datasets with different cosine similarity thresholds θ_2, the dataset with $\theta_2 = \cos(\pi/12)$ achieved the best overall performance. This may be due to the fact that a high threshold for generating dynamic edges ensures the quality of positive samples in the dataset. The dataset with $\theta_2 = \cos(\pi/6)$ had the lowest imbalance ratio, while the dataset with $\theta_2 = \cos(\pi/3)$ had the highest imbalance ratio but the lowest quality of positive samples, resulting in poorer performance than the dataset with $\theta_2 = \cos(\pi/12)$.

Regarding the undersampling strategy, the best results were obtained with an undersampling ratio of 1 : 1, which is highlighted in bold in the table. As the undersampling ratio increased, the performance gradually decreased, with the worst result being obtained without the undersampling strategy.

In terms of the use of a cost-sensitive loss function, the results were the same as with an undersampling ratio of 1 : 1. This is because, as mentioned earlier, the cost-sensitive loss function is equivalent to the *BCELoss* function when the undersampling ratio is 1 : 1. However, the performance with a cost-sensitive loss function was relatively better than that without.

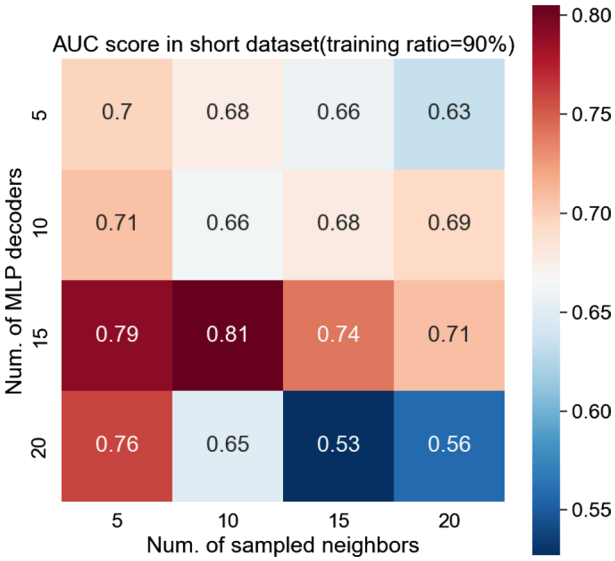

Fig. 11 The AUC score with different numbers of sampled neighbors and MLP decoders

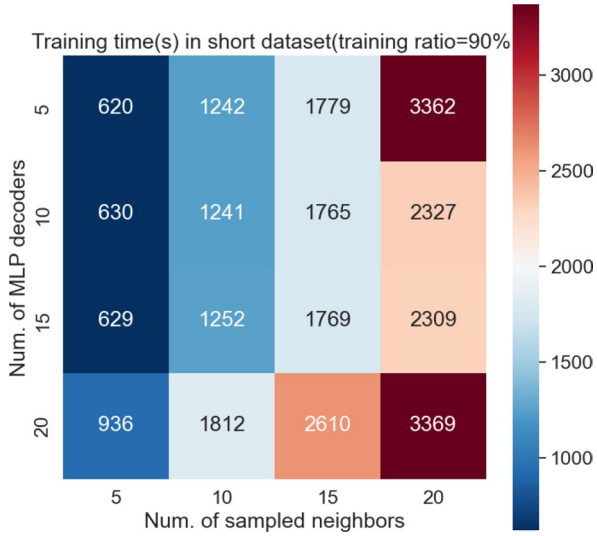

Fig. 12 The training time (seconds) per epoch with different numbers of sampled neighbors and MLP decoders

In summary, we evaluated the model's performance with different strategies for addressing data imbalance, and found that these strategies had a significant effect on performance. Based on the experimental results, the best choices of strategies were a cosine similarity threshold $\theta_2 = \cos(\pi/12)$, an imbalance ratio of $1:1$, and using a cost-sensitive loss function. We used these settings in the test described in Sect. 5.3.

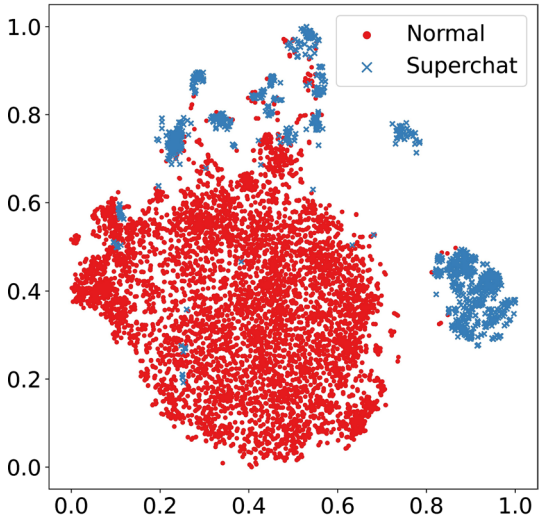

Fig. 13 The t-SNE visualization of the node embeddings learned by TDGNN

5.7 Parameter sensitivity

In Figs. 11 and 12, we present the performance of the TDGNN model with respect to two significant hyperparameters, namely the number of sampled neighbor nodes in temporal difference and the number of ensembled MLP decoders. The results are based on the short dataset with a training set ratio of 90%. The highest AUC score is obtained with 15 MLP decoders and 10 sampled neighbor nodes. It can be concluded that increasing the number of ensembled MLP decoders or sampled neighbor nodes beyond a certain value does not improve the performance. The number of sampled neighbors is a common parameter in GNN models. If too many neighbors are aggregated, the model may fail to identify the most critical information. On the other hand, if too few neighbors are sampled, some important neighbors may be ignored.

Also, the number of sampled neighbor nodes affects training time much more than the number of MLP decoders. The shortest and longest training times fluctuate by 2,742 s with different numbers of sampled neighbors, while the same fluctuation is only 1,060 s with different MLP decoders. This result shows that the number of sampled neighbors is the main factor affecting computational complexity.

5.8 Node embedding visualization

We provide an illustrative visualization of the node embeddings by t-SNE tools (Van der Maaten & Hinton, 2008). Figure 13 shows the distribution of node embeddings learned by our model on the long dataset with the training set ratio equaling 70%. Blue cross markers denote the node embedding of viewers who send superchats, and red circle markers denote the node embedding of those who did not. We observe that node embeddings with the same label are clustering, and a large margin separates node embeddings with different labels. The results prove that our method can learn discriminative node embeddings to distinguish different labels.

6 Conclusion

In summary, our work provides a comprehensive solution for predicting real-time donations in online live streaming services by leveraging dynamic graph neural network models. By constructing a continuous-time dynamic graph representation of viewer interactions and chat messages, our model can capture the temporal dynamics of the data and effectively predict potential donations. The experimental results demonstrate the superiority of our proposed TDGNN model over various baseline methods. Additionally, our study confirms the feasibility of the model's predictions in real-world situations by examining the training time, update delay, and visualization.

Our work also contributes to the rapidly growing and promising area of study on real-time donations in live streaming services. By predicting donations, our model can help content creators better understand their audience and improve engagement. Furthermore, our approach of using dynamic graph neural networks can be applied to other real-time prediction tasks in various domains. Overall, our work provides valuable insights and contributions to both the live streaming and dynamic graph neural network research communities.

Author contributions RJ: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing–original draft, Writing–review and editing, Visualization. XL: Conceptualization, Methodology, Formal analysis, Resources, Writing–original draft, Writing–review and editing, Supervision, Funding acquisition. TM: Conceptualization, Resources, Writing–review and editing, Supervision, Funding acquisition.

Funding This work is partly supported by JST SPRING (grant number JPMJSP2106), JSPS Grant-in-Aid for Scientific Research (grant number 23H03451, 21K12042), and the New Energy and Industrial Technology Development Organization (grant number JPNP20006).

Data availability The YouTube live streaming data that support the findings of this study are available via https://www.kaggle.com/datasets/uetchy/vtuber-livechat.

Code availability The Python code for the TDGNN model can be accessed publicly via https://github.com/Tracy-King/YouTube.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

Chu, W. T., & Chou, Y. C. (2017). On broadcasted game video analysis: Event detection, highlight detection, and highlight forecast. *Multimedia Tools and Applications, 76*(7), 9735–9758.

Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Elkan, C. (2001). The foundations of cost-sensitive learning. *International joint conference on artificial intelligence, Lawrence Erlbaum Associates Ltd, 17*, pp. 973–978.

Fietkiewicz, K.J., Dorsch, I., Scheibe, K., Zimmer, F., & Stock, W.G. (2018). Dreaming of stardom and money: Micro-celebrities and influencers on live streaming services. In *Social computing and social media. User experience and behavior*, pp. 240–253.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2011). A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42*(4), 463–484.

Gao, C., Zhu, J., Zhang, F., Wang, Z., & Li, X. (2022). A novel representation learning for dynamic graphs based on graph convolutional networks. *IEEE Transactions on Cybernetics*.

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems30*.

Hamilton, W.A., Garretson, O., & Kerne, A. (2014). Streaming on twitch: Fostering participatory communities of play within live mixed media. In *SIGCHI2014*, pp. 1315–1324.

Interdonato, R., Tagarelli, A., Ienco, D., Sallaberry, A., & Poncelet, P. (2017). Local community detection in multilayer networks. *Data Mining and Knowledge Discovery, 31*, 1444–1479.

Jia, A.L., Rao, Y., & Shen, S. (2021). Analyzing and predicting user donations in social live video streaming. In *2021 IEEE 24th international conference on computer supported cooperative work in design (CSCWD)*, pp. 1256–1261.

Jin, R., Liu, X., & Murata, T. (2022). Predicting potential real-time donations in youtube live streaming services via continuous-time dynamic graph. In *International conference on discovery science, Springer*, pp. 59–73.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2018). LSTM fully convolutional networks for time series classification. *IEEE Access, 6*, 1662–1669.

Kavitha, G., Saveen, B., & Imtiaz, N. (2018). Discovering public opinions by performing sentimental analysis on real time twitter data. *international conference on management of data, 2018*, pp. 1–4.

Kim, J., Bae, K., Park, E., & del Pobil, A.P. (2019). Who will subscribe to my streaming channel? the case of twitch. In *Conference companion publication of the 2019 on computer supported cooperative work and social computing*, pp. 247–251.

Kipf, T.N., & Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Kipf, T.N., & Welling, M. (2016b). Variational graph auto-encoders. arXiv preprint arXiv:1611.07308.

Kumar, S., Zhang, X., & Leskovec, J. (2019). Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD2019*, pp. 1269–1278.

Ladhari, R., Massa, E., & Skandrani, H. (2020). Youtube vloggers' popularity and influence: The roles of homophily, emotional attachment, and expertise. *Journal of Retailing and Consumer Services, 54*, 102027.

Lee, M., Choi, H., Cho, D., & Lee, H. (2016). Cannibalizing or complementing?. The impact of online streaming services on music record sales. *Procedia Computer Science, 91*, 662–671.

Lee, S. E., Choi, M., & Kim, S. (2019). They pay for a reason! the determinants of fan's instant sponsorship for content creators. *Telematics and Informatics, 45*, 101286.

Li, Z., Wang, H., Zhang, P., Hui, P., Huang, J., Liao, J., Zhang, J., & Bu, J. (2021). Live-streaming fraud detection: A heterogeneous graph neural network approach. In *KDD2021*, pp. 3670–3678.

Liben-Nowell, D., & Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology, 58*(7), 1019–1031.

Lu Jia, A., Shen, X., Shen, S., Fu, Y., & Peng, L. (2019). User donations in a user generated video system. In *Companion proceedings of the 2019 world wide web conference*, Association for Computing Machinery, New York, NY, USA, WWW '19, pp. 1055–1062.

Lu Jia, A., Rao, Y., Li, H., Tian, R., & Shen, S. (2020). Revealing donation dynamics in social live video streaming. In *Companion proceedings of the web conference 2020*, Association for Computing Machinery, New York, NY, USA, WWW '20, pp. 30–31.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research,9*(11).

Nguyen, G. H., Lee, J. B., Rossi, R. A., Ahmed, N. K., Koh, E., & Kim, S. (2018). Continuous-time dynamic network embeddings. In *Companion proceedings of the the web conference, 2018*, pp. 969–976.

Payne, K., Keith, M. J., Schuetzler, R. M., & Giboney, J. S. (2017). Examining the learning effects of live streaming video game instruction over twitch. *Computers in Human Behavior, 77*, 95–109.

Perozzi, B., Al-Rfou, R., & kiena, S. (2014). Deepwalk: Online learning of social representations. In *KDD2014*, pp. 701–710.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

Reimers, N., & Gurevych, I. (2020). Making monolingual sentence embeddings multilingual using knowledge distillation. arXiv preprint arXiv:2004.09813.

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637.

Sankar, A., Wu, Y., Gou, L., Zhang, W., & Yang, H. (2020). Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM2020*, pp. 519–527.

Sjöblom, M., & Hamari, J. (2017). Why do people watch others play video games? An empirical study on the motivations of twitch users. *Computers in Human Behavior, 75*, 985–996.

Trivedi, R., Farajtabar, M., Biswal, P., & Zha, H. (2019). Dyrep: Learning representations over dynamic graphs. In *ICLR2019*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. arXiv preprint arXiv:1710.10903v3.

Wang, X., Tian, Y., Lan, R., Yang, W., & Zhang, X. (2019). Beyond the watching: Understanding viewer interactions in crowdsourced live video broadcasting services. *IEEE Transactions on Circuits and Systems for Video Technology, 29*(11), 3454–3468.

Wang, X., Lyu, D., Li, M., Xia, Y., Yang, Q., Wang, X., Wang, X., Cui, P., Yang, Y., Sun, B., *et al.* (2021). Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *SIGMOD2021*, pp. 2628–2638.

Wohn D.Y., Jough, P., Eskander, P., Siri, J.S., Shimobayashi, M., & Desai, P. (2019). Understanding digital patronage: Why do people subscribe to streamers on twitch?. In *Proceedings of the annual symposium on computer-human interaction in play*, pp. 99–110.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P.S. (2019). A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596.

Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). Inductive representation learning on temporal graphs. arXiv preprint arXiv:2002.07962.

Yang, C., Wang, C., Lu, Y., Gong, X., Shi, C., Wang, W., & Zhang, X. (2022). Few-shot link prediction in dynamic networks. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pp. 1245–1255.

Yang, H., & Lee, H. (2018). Exploring user acceptance of streaming media devices: An extended perspective of flow theory. *Information Systems and e-Business Management, 16*(1), 1–27.

Yu, J., & Jia, A. L. (2022). User donations in online social game streaming: The case of paid subscription in twitch. tv. In *Companion proceedings of the web conference, 2022*, pp. 215–218.

Zangari, L., Interdonato, R., Calió, A., & Tagarelli, A. (2021). Graph convolutional and attention models for entity classification in multilayer networks. *Applied Network Science, 6*(1), 1–36.

Zhan, J., & Zhang, N. (2023). Exploring the impact of virtual anchor features and live content on viewers' willingness to pay for "superchat'' in live entertainment scenarios. *Highlights in Business, Economics and Management, 6*, 189–205.

Zhang, M., & Chen, Y. (2018). Link prediction based on graph neural networks. In *NeurIPS2018*, NIPS'18, pp. 5171–5181.

Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. *AAAI2018*, *32*(1).

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2018). Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434.

Zhu, Z., Yang, Z., & Dai, Y. (2017). Understanding the gift-sending interaction on live-streaming video websites. In G. Meiselwitz (Ed.), *Social computing and social media* (pp. 274–285). Springer International Publishing.