Recent Link Classification on Temporal Graphs Using Graph Profiler

Anonymous authors
Paper under double-blind review

Abstract

The performance of Temporal Graph Learning (TGL) methods are typically evaluated on the *future link prediction* task, i.e., whether two nodes will get connected and *dynamic node classification* task, i.e., whether a node's class will change. Comparatively, *recent link classification*, i.e., to what class an emerging edge belongs to, is investigated much less even though it exists in many industrial settings. In this work, we first formalize recent link classification on temporal graphs as a benchmark downstream task and introduce corresponding benchmark datasets. Secondly, we evaluate the performance of state-of-the-art methods with a statistically meaningful metric *Matthews Correlation Coefficient*, which is more robust to imbalanced datasets, in addition to the commonly used average precision and area under the curve. We propose several design principles for tailoring models to specific requirements of the task and the dataset including modifications on message aggregation schema, readout layer and time encoding strategy which obtain significant improvement on benchmark datasets. Finally, we propose an architecture that we call *Graph Profiler*, which is capable of encoding previous events' class information on source and destination nodes. The experiments show that our proposed model achieves an improved Matthews Correlation Coefficient on most cases under interest. We believe the introduction of recent link classification as a benchmark task for temporal graph learning will be useful for the evaluation of prospective methods within the field.

1 Introduction

Graphs provide convenient structures to represent interactions or relationships between entities by modeling them as edges between vertices. Using this representation allows one to build models that capture the interconnected nature of complex systems such as social networks (El-Kishky et al., 2022; Wu et al., 2022; Gao et al., 2021) or transaction graphs (Liu et al., 2020; Zhang et al., 2022). Graph Representation Learning (GRL) rose in popularity due to the desire to apply deep learning to graph structured problems (Zhou et al., 2020; Wu et al., 2020; Hamilton, 2020). Indeed, GRL has provided significant advances in fraud detection (Liu et al., 2020; Zhang et al., 2022), recommendation systems (Wu et al., 2022; Gao et al., 2021), chemistry and materials science (Pezzicoli et al., 2022; Reiser et al., 2022; Bongini et al., 2021; Han et al., 2021; Xiong et al., 2021), traffic modeling (Rusek et al., 2019; Chen et al., 2022), and weather simulation (Keisler, 2022; Ma et al., 2022), among other possible applications. Many of these graph machine learning tasks can be understood as either link prediction (Chen et al., 2020; Cai et al., 2019; Zeb et al., 2022; Chamberlain et al., 2022) or node classification tasks (Kipf & Welling, 2016; Zhang et al., 2019). Much of the early work focused on the scenario where the graph is static. Acknowledging that many tasks in industrial settings involve graphs that evolve in time, researchers defined a sub-problem of graph machine called Temporal Graph Learning (TGL), with time dependent versions of the original static tasks, yielding *future link prediction (FLP)* and *dynamic node classification (DNC)* respectively (Kumar et al., 2019; Arnoux et al., 2017). The former task, FLP, seeks to predict whether two vertices will be connected at some specified future time; while the latter, DNC, seeks to predict the class of a vertex at a future time. Both of these tasks have a variety of applications in real world, e.g., predicting the probability of two people forming a tie or of a person deactivating their

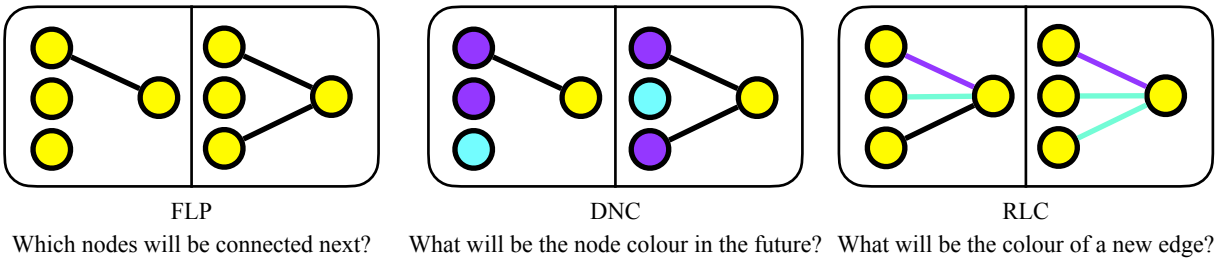

Figure 1: Differences between TGL tasks. **Future Link Prediction (FLP):** Given a temporal graph and a pair of nodes observed at a timestamp, a function is learned which predicts the probability of these two nodes linking at a later timestamp. **Dynamic Node Classification (DNC):** Let blue and purple colours represent the node classes which change over time. Given the current node classes along with the set of edges at a time point, the aim is predicting the node classes at a later timestamp. **Recent Link Classification (RLC):** Let the colours blue and purple represent the edge classes, which are observed some time after the emergence of a link. That is, at the time when a link is first detected, the link class is unknown. Black edge have been recently observed but the class is not yet observed. In this setting, the aim is to predict recent link classes given a history of previous interactions.

account on social media platforms, corresponding to FLP and DNC tasks respectively (Min et al., 2021; Song et al., 2021; Frasca et al., 2020; Zhang et al., 2021a).

This begs the question – "In analogy to the dynamic node classification task, is there a temporal link classification task we can define?" And indeed, there is a third common task that is based on static link classification (Wang et al., 2023), which we term *recent link classification (RLC)*. RLC is present in industrial settings but has yet to be formalized in a research setting. The task of RLC requires that one classify a link that has been observed to exist but does not as yet have a known label. This task is important for settings where we wish to classify the interaction but labels may be delayed. In social media settings, this could involve classifying interactions as abusive, and in a transaction network it could involve classifying a transaction as potentially fraudulent. More formally, we define RLC as a task in which the predictive algorithm is given the source and destination entities of a recent interaction and its features, e.g., textual content of the post; and it must predict the target class value. The task of classifying recent interactions has typically been treated as a tabular data task, arising in applications such as fraud detection (Sarkar, 2022). This approach represents each interaction between two entities as a sample with associated features. The sample can thus be represented as a row of a table, where each column is a feature. There is usually an (implicit) assumption that the samples are independent. This neglects the fact that entities may participate in multiple interactions and thus induce dependencies between the samples. In practice, one option to address this deficiency is feature engineering. For tabular tasks, it has been previously observed that features representing counts, such as how many times one user has previously liked another user's posts, provide significant metric uplift (Wu et al., 2019; Shan et al., 2016). While the incorporation of such features is not explicitly graph machine learning, we take the view that these features are the result of manual feature engineering inspired by graph-based intuition (Zhang et al., 2021b; Martínez et al., 2016; Chamberlain et al., 2022). Therefore, we believe that explicitly formulating RLC as a graph learning task and highlighting suitable datasets for the temporal graph machine learning community will encourage progress towards better solutions for an industrially relevant problem.

With this motivation in hand, we formalize two research questions that we wish to answer in this work: "**Q1: How does recent link classification differ from future link prediction?**" and "**Q2: What are the most critical design principles of recent link classification?**". We answer the first question through a comparative study of baseline methods on both tasks. We answer the second question by exploring a variety of model building blocks, some published previously, and some novel. In answering these research questions, we contribute a new temporal graph learning task, a new figure of merit, a measure of edge-homophily, and a non-exhaustive set of design principles that comprise a design space for this new machine learning task.

2 Related Work

Our work should be understood within the context of the temporal graph learning literature. To that end, we have provided a brief review of the works that we have found influential. TGN (Rossi et al., 2020) is a message passing based encoder which learns graph node embeddings on a continuous-time dynamic multi-graph represented as a sequence of time-stamped events. It involves three main building blocks: (1) message function, (2) memory function and (3) embeddings module. At each event (e.g., a new node, node attribute change or new edge), a message function in the form of event-adopted MLPs, calculates aggregated messages to pass on parties involved (i.e., nodes). A memory function, such as an LSTM or GRU, updates the memory state of each node by the aggregated messages. An embedding module calculates the new state of node embeddings as a function of memory states and events. The authors experiment with simple embedding modules such as identity function which corresponds using memory states directly as well as multi-head attention based procedures. Their datasets include link prediction and node classification tasks. Gao & Ribeiro (2022) develops a framework to analyze the temporal graph learning architectures categorizing the methods literature into two groups 'time-and-graph' and 'time-then-graph'. Time-and-graph based architectures learns evolution of node representations by building a representation for each graph snapshot. DySAT (Sankar et al., 2020) and EvolveGCN (Pareja et al., 2020) fall under this category. Time-then-graph based architectures, on the other hand, constructs a multi-graph by the memory of all past observations and build a static graph to learn the node representations on. TGAT (da Xu et al., 2020) and TGN (Rossi et al., 2020) fall under this category. In addition to the proposed framework that enables analyzing the expressivity of two distinct categories by the employed learnable modules, the authors suggest a simple but efficient algorithm GRU-GCN (Gao & Ribeiro, 2022). GraphMixer (Cong et al., 2023) takes a simpler view by constructing a model that has a fixed time encoding, alongside a node encoder and a link encoder, which are all fed into a link classifier that is trained to predict the existence of a link. Despite this simple infrastructure, GraphMixer is able to achieve state-of-the-art performance on both FLP and DNC tasks. Finally, TGL (Zhou et al., 2022) decomposes the task of training temporal graph neural networks into a general, scalable, framework composed of five core modules and provides primitives for computationally efficient training.

3 Problem Statement

A graph, $\mathcal{G}$, is composed of a set of vertices, $\mathcal{V}$, and edges $\mathcal{E}$, where each edge, $i \to j$, indicates an interaction between a pair of vertices i and j. In most cases, the graph is constructed with entities as the vertices and interactions between those entities as edges. In the case of social networks vertices might be users and their posts, and edges might involve interactions either between users, such as follows or blocks, or interactions between users and posts such as likes or comments. For a general RLC task we are given a set of source entities and destination entities, $\mathcal{V}_{\text{src}}$ and $\mathcal{V}_{\text{dst}}$, and a set of interactions between them $\mathcal{E} = \{(s_i, d_i, t_i, \mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{M}$; such that the interaction from source entity $s_i \in \mathcal{V}_{\text{src}}$ to destination entity $d_i \in \mathcal{V}_{\text{dst}}$ is realized at time t_i and associated with a raw feature vector of $\mathbf{x}_i \in \mathbb{R}^{d_{\text{msg}}}$ where d_{msg} denotes the number of features. We consider that interactions are effectively instantaneous, or that the timestamp marks the completion of the interaction. We consider settings where there are m classes (or types) of interaction. Each interaction is thus associated with a ground-truth target class, represented as a binary vector $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,m})$ such that $y_{i,j} = 1$ if interaction i belongs to the jth class and $y_{i,j} = 0$ otherwise. The aim is to learn a classifier that maps features, timing, source and destination entities of an interaction to a class, given access to a history of preceding interactions. Given a new interaction from source $s \in \mathcal{V}_{\text{src}}$ to destination $d \in \mathcal{V}_{\text{dst}}$ with features $\mathbf{x} \in \mathbb{R}^{d_0}$ which is realized at time t, let $\mathcal{E}_{<t} = \{(s_i, d_i, t_i, \mathbf{x}_i, \mathbf{y}_i) \in \mathcal{E} : t_i < t\}$ denote the preceding observations, and let $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ denote the predicted target class likelihoods by the classifier, i.e., $f(\mathbf{x}, (s, d, t), \mathcal{E}_{<t}) = \hat{\mathbf{y}}$. Traditionally, the quality of estimation is evaluated by the cross entropy loss $\mathbb{L}_{\text{ce}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{m} y_j \log(\hat{y}_j)$ during training. In case that $\mathcal{G}$ is not bipartite, i.e., $\mathcal{V}_{\text{src}} \cap \mathcal{V}_{\text{dst}} \neq \emptyset$, this formulation requires us to learn different representations for a vertex as the sender and the receiver of a message. In the basic problem formulation, we assume that vertices are not attributed, i.e., raw features are not observed for them. At the time of the event occurrence, we observe the identities of the source and destination, as well as the raw edge features. None of these changes over time. The observation of the edge label is delayed.

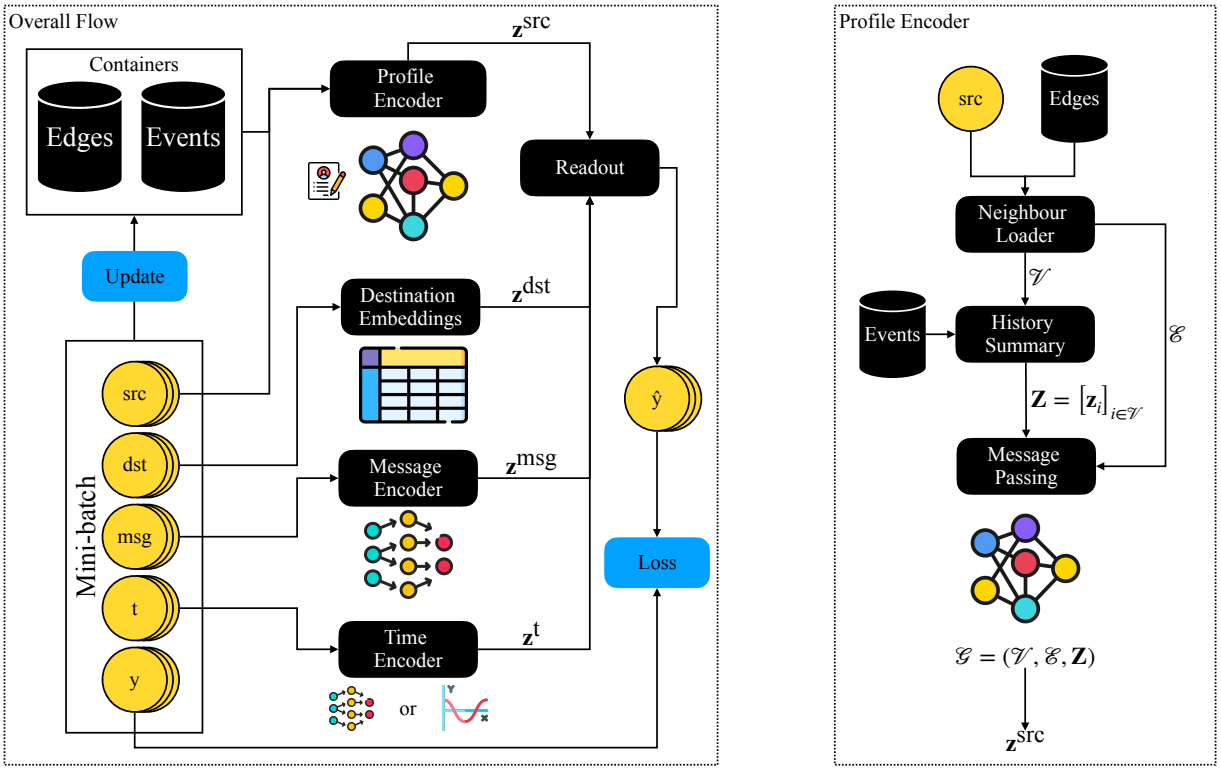

Figure 2: Overview of Graph Builder. **Left:** Overall flow. Given a mini-batch of interactions, where each interaction is from a source node `src` to a destination node `dst`, observed at time `t`, associated with features `msg` and belongs to class `y`, Graph Profiler follows these steps: (1) The profile encoder calculates source node embeddings based on the history of `src` (i.e., previous edges and events that are associated with `src`). (2) The identity of `dst` is used to retrieve the embedding of the destination node from the table of destination embeddings. (3) The message encoder performs a linear transformation on the edge features. (4) The time encoder projects timestamp `t` into a time embedding. (5) The readout layer combines source $\mathbf{z}^{\text{src}}$, destination $\mathbf{z}^{\text{dst}}$, message $\mathbf{z}^{\text{msg}}$ and time $\mathbf{z}^{\text{t}}$ embeddings to predict the class of the interaction. Afterwards, the containers are updated with the information in the mini-batch. **Right:** Profile builder. Given the current set of edges, the neighbourhood of `src` is loaded. The past events are used to summarize the history of all nodes that belong to the neighbourhood of `src`. The node embeddings on the ego graph of `src` are updated using message passing to obtain the final source node embedding $\mathbf{z}^{\text{src}}$.

4 Graph Profiler

In this section, we introduce *Graph Profiler*, a simple architecture that is designed to learn entity profiles, or time-aggregated representations, by processing previously observed interactions. Subsequently it uses the constructed entity profiles, together with observed interaction features, to make classification decisions about recent interactions. Graph Profiler is composed of five main learnable modules; **profile encoder** $f_{\text{profile}}(\cdot)$, **message encoder** $f_{\text{msg}}(\cdot)$, **destination encoder** $f_{\text{dst}}(\cdot)$, **time encoder** $f_{\text{time}}(\cdot)$, and **readout** $f_{\text{rlc}}(\cdot)$, and two containers at time t; previous **events** $\mathcal{H}_t$ and meta-path **edges** $\mathcal{M}_t$. For a given set of interactions (edges), $\mathcal{E} = \{(s_i, d_i, t_i, \mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{M}$, the event and meta-path containers at time t are defined as follows:

$$\mathcal{H}_t = \{i \in \{1, \ldots, M\} : t_i < t\}, \tag{1}$$

$$\mathcal{M}_t = \{(u, v) : \exists i, j \in \mathcal{H}_t; (s_i = u) \wedge (s_j = v) \wedge (d_i = d_j)\}. \tag{2}$$

The event container simply stores the indices of the interactions observed up to time t. The meta-path container stores the pairs of source nodes that have interacted with the same destination node at least once

prior to time t. Graph Profiler proceeds in the following way to perform training. We denote by $\mathcal{E}_{\text{batch}} \subset \mathcal{E}$ the mini-batch of interactions that is being processed. Let $t_{\text{current}} = \min_{j \in \mathcal{E}_{\text{batch}}} t_j$ denote the first interaction time in the batch, and d_{model} denote the dimensionality of the embeddings constructed by the model. Given an interaction $(s_i, d_i, t_i, \mathbf{x}_i) \in \mathcal{E}_{\text{batch}}$, the computation proceeds through each module as follows:

1. The profile encoder calculates the source node profile $\mathbf{z}_i^{\text{src}} \in \mathbb{R}^{d_{\text{model}}}$ based on observations until t_{current}, i.e., $f_{\text{profile}}\left(s_i, \mathcal{H}_{t_{\text{current}}}\right) = \mathbf{z}_i^{\text{src}}$.

2. The message encoder encodes the interaction: $\mathbf{z}_i^{\text{msg}} \in \mathbb{R}^{d_{\text{model}}}$, i.e., $f_{\text{msg}}\left(\mathbf{x}_i\right) = \mathbf{z}_i^{\text{msg}}$.

3. The destination encoder generates destination embeddings: $\mathbf{z}_i^{\text{dst}} \in \mathbb{R}^{d_{\text{model}}}$, i.e., $f_{\text{dst}}\left(d_i\right) = \mathbf{z}_i^{\text{dst}}$.

4. The time encoder converts the interaction timestamp into a time embedding vector $\mathbf{z}_i^{\text{t}} \in \mathbb{R}^{d_{\text{model}}}$, i.e. $f_{\text{time}}\left(t_i\right) = \mathbf{z}_i^{\text{t}}$.

5. The readout layer predicts the interaction class $\hat{\mathbf{y}}_i$, i.e., $f_{\text{rlc}}\left(\mathbf{z}_i^{\text{src}}, \mathbf{z}_i^{\text{msg}}, \mathbf{z}_i^{\text{dst}}, \mathbf{z}_i^{\text{t}},\right) = \hat{\mathbf{y}}_i$.

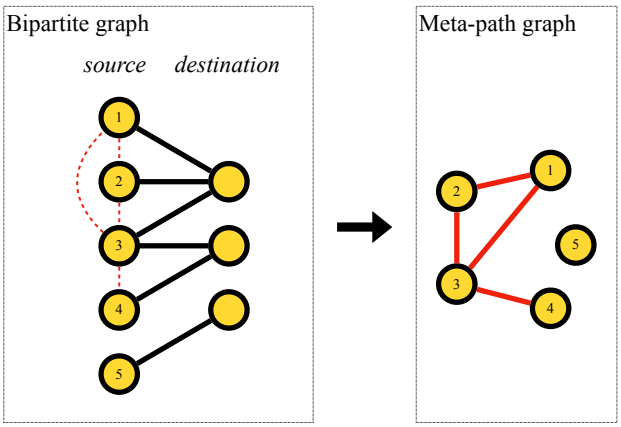

Figure 3: Meta-path construction: On the left, a bipartite graph is provided that consists of edges from a set of source entities to a set of destination entities. The original edges are represented by black lines. Source nodes that have previously interacted with the same destination node are connected by a meta-path edge. These edges are represented by dashed red lines. On the right, the final meta-path graph composed of meta-path edges is shown.

Once the predictions are made on mini-batch $\mathcal{E}_{\text{batch}}$, the containers are updated so that they include the interactions in the mini-batch. The meta-paths are recalculated according to Equation 2. We note that in the procedure outlined above, it is only the construction of the source node profile that uses information from the historical interactions in the calculations. The destination node embedding is computed using only the identity of the node, and the message encoding is derived via a network operating on the message features. The destination embeddings are learnable, as are the weights of the message encoder network, so the historical interactions influence these embeddings via the training procedure. The overall flow is illustrated in Figure 2. Next we explain how individual modules are trained and describe the procedure used to update containers.

Profile Encoder Inspired by our previous experiences working on webscale recommendation systems, we derive graphs that allow us to capture source-source correlations that might be obscured through traditional message passing schemes due to an over-smoothing effect. Similar to previous work, we define a meta-path as an edge that is constructed from a path through the graph (Chen & Lei, 2022; Huai et al., 2023; Huang et al., 2022). In our specific instance, we consider second-order meta-paths that connect a vertex which acts as a source to another which acts as a source through a shared destination vertex. The set of meta-paths is time dependent because the edges are parameterized by time. Given the set of meta-path edges $\mathcal{M}_{t_{\text{current}}}$ observed up until the current time the profile encoder first builds the ego graph $\mathcal{G}_{t_{\text{current}}}(s_i)$ over the set of vertices $\mathcal{V}_{t_{\text{current}}}(s_i) = s_i \cup \{u : (u, s_i) \in \mathcal{M}_{t_{\text{current}}}\}$ with set of edges $\mathcal{M}_{t_{\text{current}}}(s_i) = \{(u, v) : (u, v) \in \mathcal{M}_{t_{\text{current}}}; \forall u, v \in \mathcal{V}_{t_{\text{current}}}(s_i)\}$. Thus, the ego graph $\mathcal{G}_{t_{\text{current}}}(s_i) = (\mathcal{V}_{t_{\text{current}}}(s_i), \mathcal{M}_{t_{\text{current}}}(s_i))$ has a vertex set that consists of s_i and all vertices that are connected via a meta-path to s_i, and an edge set consisting of all meta-paths connecting these vertices. For each node $u \in \mathcal{V}_{t_{\text{current}}}(s_i)$, we define the set of relevant event indices as $\mathcal{H}_{t_{\text{current}}}(u) = \{j : s_j = u, \forall j \in \mathcal{H}_{t_{\text{current}}}\}$.

The node embeddings are initialized by aggregating the embeddings of previous events associated with the corresponding node, i.e., $\mathbf{h}^{(0)}(u) = f_{\text{aggregate}}(\mathcal{H}_{t_{\text{current}}}(u))$. For example, using a mean aggregation schema

with single layer of linear transformation, the node embeddings $\mathbf{h}^{(0)}(u) \in \mathbb{R}^{d_{\text{model}}}$ are initialized as follows:

$$\mathbf{h}^{(0)}(u) = \frac{\sum_{i \in \mathcal{H}_{t_{\text{current}}}(u)} \left[\mathbf{z}_i^{\text{msg}} || \mathbf{z}_i^{\text{dst}} || \mathbf{z}_i^{\text{t}} || \mathbf{y}_i\right] \mathbf{W}_{\text{event}}^{\text{T}}}{|\mathcal{H}_{t_{\text{current}}}(u)|}, \tag{3}$$

where $[\cdot||\cdot]$ denotes the concatenation operator, and $\mathbf{W}_{\text{event}} \in \mathbb{R}^{d_{\text{model}} \times d_1}$ are learnable weights for $d_1 = 3 \times d_{\text{model}} + m$. Recall that $\mathbf{z}_i^{\text{msg}}, \mathbf{z}_i^{\text{dst}}, \mathbf{z}_i^{\text{t}} \in \mathbb{R}^{d_{\text{model}}}$ and $\mathbf{y}_i \in \mathbb{R}^m$. Then, using the GCN (Kipf & Welling, 2016) framework, at the kth layer the node embeddings are updated by passing messages between neighbour nodes, i.e., $\mathbf{h}^{(k)}(u) = f_{\text{gcn}}(\mathbf{h}^{(k-1)}(u), \mathcal{M}_{t_{\text{current}}}(s_i))$. Introducing normalization coefficients $c_{u,v} = \frac{1}{\sqrt{\deg(u)} \cdot \sqrt{\deg(v)}}$, with $\deg(\cdot)$ denoting node degree on $\mathcal{G}_{t_{\text{current}}}$, we can write the node embedding update at layer k for a normalized sum aggregation schema as:

$$\mathbf{h}^{(k)}(u) = \sum_{(u,v) \in \mathcal{M}_{t_{\text{current}}}(s_i)} c_{u,v} \left(\mathbf{h}^{(k-1)}(v) \mathbf{W}_k^{\text{T}}\right), \tag{4}$$

where $\mathbf{W}_k \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ are learnable weights, and $\mathbf{h}^{(k)}(u) \in \mathbb{R}^{d_{\text{model}}}$. The profile embedding of source node s_i is set to the final layer node embedding, i.e., $z_i^{\text{src}} = \mathbf{h}^{(K)}(s_i)$ where K denotes the total number of message passing layers.

Time Encoder For the time encoder, we employ either the fixed time encoding function proposed by Cong et al. (2023) or the learnable time projection introduced by Kumar et al. (2019). Given a weight vector $\omega \in \mathbb{R}^{d_{\text{model}}}$, in general the time encoding function follows:

$$f_{\text{time}}(t) = \cos(\omega t) = \mathbf{z}^t, \tag{5}$$

where $\mathbf{z}^t \in \mathbb{R}^{d_{\text{model}}}$ denotes the vector representation of timestamp t. The two variants of time encoding we investigate in this work differ in the calculation of the weight vector. In the *learnable version*, $\omega \in \mathbb{R}^{d_{\text{model}}}$ is simply learned during training, so the time encoding layer is a linear projection, without any bias, followed by cosine scaling. In the case of *fixed time encoding* as proposed by Cong et al. (2023), each dimension of the weight vector is set to $\omega_i = \alpha - \frac{(i-1)}{\beta}$, where α and β are scalars, so that ωt is a vector with monotonically exponentially decreasing values. The hyperparameters α and β are selected according to the scale of the minimum and maximum timestamps in the data. In practice $\alpha = \beta = \sqrt{d_{\text{model}}}$ is found to perform well, and we follow this setting in our experiments.

Other Modules The message encoder uses a single linear layer to compute $f_{\text{msg}}(\mathbf{x}_i) = \mathbf{x}_i \mathbf{W}_{\text{msg}}^{\text{T}} + \mathbf{b}_{\text{msg}}$, where $\mathbf{W}_{\text{msg}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{msg}}}$ are learnable weights and $\mathbf{b}_{\text{msg}} \in \mathbb{R}^{d_{\text{model}}}$ is a learnable bias. For destination encoding, we use an embedding look-up table of size equal to the number of destination nodes. This is initialized randomly, $f_{\text{dst}}(d_i) = \mathbb{1}_{d_i} \mathbf{W}_{\text{dst}}^{\text{T}}$, where $\mathbb{1}_{d_i} \in \mathbb{R}^{||\mathcal{V}_{\text{dst}}||}$ denotes a one-hot vector representation of node d, and $\mathbf{W}_{\text{dst}} \in \mathbb{R}^{d_{\text{model}} \times ||\mathcal{V}_{\text{dst}}||}$ are learnable weights. The predictions at the readout layer are computed as $\hat{\mathbf{y}}_i = \left[\mathbf{z}_i^{\text{src}} + \mathbf{z}_i^{\text{dst}} + \mathbf{z}_i^{\text{msg}} + \mathbf{z}_i^{\text{t}}\right] \mathbf{W}_{\text{rlc}}^{\text{T}}$ where $\mathbf{W}_{\text{rlc}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ are learnable weights.

Unlike static graph learning methods, Graph Profiler is capable of encoding temporal properties of network. Graph Profiler has two main advantages over existing temporal graph learning methods: (1) it enables the construction of dynamic entity profiles that take into account feature and label information of previous interaction in the neighbourhood; and (2) it is capable of maintaining a long-term view of an entity's profile that can capture long-term preferences. In addition, the modular structure of Graph Profiler is flexible, so the model can easily be adapted to suit the contextual properties of individual datasets.

5 Experiments

In order to understand RLC as a novel task within temporal graph learning, we begin by evaluating a two layer MLP, TGN (Rossi et al., 2020), TGAT (da Xu et al., 2020), and Graph Mixer (Cong et al., 2023) on RLC by making the appropriate modifications to the algorithms. We have chosen these methods because they are state-of-the-art TGL baselines developed for the FLP task. Based on our observations concerning

Table 1: Dataset statistics. $|\mathcal{V}_{\text{src}}|$ and $|\mathcal{V}_{\text{dst}}|$ denote the number of source entities and destination entities, respectively; d_{msg} denotes the number of interaction features; $|\mathcal{E}|$ denotes the number of interactions; ρ denotes the fraction of edge labels that are positive; $\bar{\mathcal{H}}_e$ denotes the average edge homophily; $\bar{\mathcal{H}}_e^+$ and $\bar{\mathcal{H}}_e^-$ denote the edge homophily for positive and negative classes, respectively, and $\tilde{\mathcal{H}}_e^b$ denotes the balanced edge homophily. Edge homophily metrics are calculated over all interactions (edges) in the dataset.

	$\lvert\mathcal{V}_{\text{src}}\rvert$	$\lvert\mathcal{V}_{\text{dst}}\rvert$	$\lvert\mathcal{E}\rvert$	d_{msg}	ρ	$\bar{\mathcal{H}}_e$	$\mathcal{H}_e^+$	$\mathcal{H}_e^-$	$\tilde{\mathcal{H}}_e$
Epic Games	542	614	17584	400	0.6601	0.8330	0.9038	0.6955	0.7663
YelpCHI	38,063	201	67,395	25	0.1323	0.7781	0.1589	0.8725	0.2533
Wikipedia	8,227	1,000	157,474	172	0.0014	0.9975	0.0130	0.9988	0.0144
Mooc	7,047	97	411,749	4	0.0099	0.9809	0.0212	0.9904	0.0308
Reddit	10,000	984	672,447	172	0.0005	0.9989	0.0025	0.9995	0.0030
Open Sea	57,230	1,000	282,743	35	0.4601	0.5865	0.5505	0.6171	0.5812

the performance of the methods, we outline a set of design principles that comprise the design space for RLC. With these design principles in mind, we present Graph Profiler and benchmark it on six different datasets. For each dataset, we locate the optimal portion of our design space and discuss the correspondence between that and the underlying dynamics of the dataset under investigation.

Datasets We evaluated our methods on four benchmark datasets that have previously been used by the TGL community – YelpCHI (Dou et al., 2020), Wikipedia, Mooc, and Reddit (Kumar et al., 2019). These datasets are usually employed as benchmarks for future link prediction; we adopt suitable measures to convert them to the RLC setting. In YelpCHI, the source entities are platform users and the destination entities include hotels and restaurants. An interaction happens when a user reviews one of the hotels or restaurants. The reviews are labeled either as filtered (spam) or recommended (legitimate). For Wikipedia, the set of entities is composed of users and pages, and an interaction happens when a user edits a page. For Reddit, entities are users and subreddits, and an interaction represents a post written by a user on a subreddit. Some page edits on Wikipedia and posts on Reddit may be controversial causing the user to be banned. Thus, on both datasets we base the target class of interaction on whether it is controversial or not, i.e., whether the user was banned as a result of posting it. The interaction features for these three datasets are extracted based on the textual content of edit/post/review. The Mooc dataset consists of actions performed by students on a MOOC online course. The source entities are the students and the destination entities are the course contents that the students interact with, e.g., recordings or lecture notes. The interaction features are the types of activities the student performed during an interaction, e.g., viewing the video or submitting an answer on the forum. Sometimes, the students drop out of the course after an activity. We use this as a label to identify the target class of an interaction. Thus, all four datasets are binary recent link classification datasets for which the class imbalance is salient.

In addition to adapting benchmark TGL datasets to RLC task, we process two tabular datasets that are not conventionally investigated in TGL setting; Epic Games [1] and Open Sea (La Cava et al., 2023a;b; Costa et al., 2023) [2]. The Epic Games Store is a digital video game storefront, operated by Epic Games. The dataset includes the critiques written by different authors about the games released on the platform. The source and destination nodes represent authors and games, respectively, and the critiques form the set of interactions. We construct the interaction features by vectorizing the textual content of critiques using TF-IDF. We include the overall rating the author provided as an additional interaction feature. The label of the interaction is determined based on whether it was selected as top critique or not. Once a critique is released all the information regarding the author, game and features of the critique is available. Its selection as a top-critique is a delayed observation, so the data naturally forms an RLC task. Open Sea is one of the leading trading platforms in the Web3 ecosystem. The dataset is a collection of Non-Fungible Token (NFT) transactions. Sourced from Open Sea, it is provided as a natural language processing dataset and is mainly used for multimodal learning classification tasks. To the best of our knowledge the dataset has

[1] `https://www.kaggle.com/datasets/mexwell/epic-games-store-dataset`.
[2] `https://huggingface.co/datasets/MLNTeam-Unical/NFT-70M_transactions`.

not been previously investigated in a TGL framework. In order to make it amenable to graph learning, we identify disjoint sets of sellers and buyers of unique NFTs (these are identified by collection memberships and token IDs) to serve as source and destination nodes. The transaction features are a binary representation of categorical variable fields associated with the transaction, the cryptocurrency exchange rates at the time of the interaction and the associated monetary values. The label of a transaction is determined based on a future transaction of the unique NFT. It is tagged as 'profitable' if the revenue obtained through the final sale is higher than the price paid at the purchase. The labels are thus delayed, because whether it will be a profitable investment is not known at the time of purchase. The data thus aligns with the RLC setting. The links to processed versions of these datasets and the software used to do the processing are available at the repository associated with this work. In Appendix B, further details on pre-processing of Open Sea dataset is shared. The datasets statistics are provided in Table 1. In our experiments, data is divided into training (70%), validation (10%) and testing (20%) sets chronologically.

Edge Homophily We introduce a measure of edge homophily to understand the importance of graph information to our edge classification task. Denote by $\mathcal{N}^e(\alpha)$ an edge-wise neighbourhood operator that constructs a set of all edges that are connected to a given edge, $\alpha = (i, j)$, where i and is the source and j the destination. This operator forms the union of two sets, i.e., $\mathcal{N}^e(\alpha) = \mathcal{I}(i) \cup \mathcal{O}(j)$, where $\mathcal{I}(i)$ is the set of incoming edges connected to the source i and $\mathcal{O}(j)$ is the set of outgoing edges connected to the destination. Our edge homophily measure is then defined as:

$$\bar{\mathcal{H}}_e(\mathcal{G}) = \frac{1}{|\mathcal{E}|} \sum_{\alpha \in \mathcal{E}} \sum_{\beta \in \mathcal{N}_\alpha^{(e)}} \frac{\mathbf{1}_{l(\alpha)=l(\beta)}}{|\mathcal{N}_\alpha^{(e)}|}, \tag{6}$$

where $\mathcal{N}^e$ is the edge-wise neighbourhood operator and l is the operator that returns the label of the edge. The edge-wise neighbourhood operator constructs a set of all edges that are connected to a given edge, $\alpha = (i, j)$, where i and j are the source and destination respectively, by constructing the union of two sets $\mathcal{N}^e(\alpha) = \mathcal{I}(i) \cup \mathcal{O}(j)$, where $\mathcal{I}(\cdot)$ and $\mathcal{O}(\cdot)$ construct the set of incoming and outgoing edges respectively. For the simplicity of notation, we have neglected the time dimension but this definition is easy to generalize to temporal graphs through the neighbourhood operators. Edge-homophily measures the fraction of neighbouring edges that have the same class, in analogy to the way node-homophily measures the fraction of neighbouring nodes with the same class. Node-homophily is an important dataset property that can be highly indicative of the value that can be derived by encoding graph structure in node classification tasks (Pei et al., 2020), particularly for embedding procedures that rely on smoothing over a neighbourhood. The edge homophily definition in Equation 6 treats different classes equally, which can be misleading for imbalanced datasets. Therefore, using ρ to denote the fraction of positive edge labels, we also define a balanced edge homophily metric for binary classification as $\tilde{\mathcal{H}}_e^b(\mathcal{G}) = (1-\rho)\mathcal{H}_e^+(\mathcal{G}) + \rho\mathcal{H}_e^-(\mathcal{G})$, where $\bar{\mathcal{H}}_e^+(\mathcal{G}) = \frac{1}{|\mathcal{E}^+|} \sum_{\alpha \in \mathcal{E}^+} \sum_{\beta \in \mathcal{N}_\alpha^{(e)}} \frac{\mathbf{1}_{l(\alpha)=l(\beta)}}{|\mathcal{N}_\alpha^{(e)}|}$ and $\bar{\mathcal{H}}_e^-(\mathcal{G}) = \frac{1}{|\mathcal{E}^-|} \sum_{\alpha \in \mathcal{E}^-} \sum_{\beta \in \mathcal{N}_\alpha^{(e)}} \frac{\mathbf{1}_{l(\alpha)=l(\beta)}}{|\mathcal{N}_\alpha^{(e)}|}$. Table 1 presents the edge homophily values for the datasets we study. In Appendix C, we illustrate the dynamics of edge homophily over time.

Performance Evaluation Two of the most common metrics used for performance evaluation for both FLP and DNC are area under the receiver operating characteristic curve (AUC) and average precision score (APS). These metrics exhibit some undesirable behaviour. AUC is known to saturate, such that it become impossible to differentiate between candidate algorithms. Both metrics can provide skewed measures of quality when applied to imbalanced datasets (Chicco & Jurman, 2020; 2023). As a result, we turn to the Matthews Correlation Coefficient (Yule, 1912; Gorodkin, 2004), which is defined as:

$$\text{MCC} = \frac{cs - \vec{t} \cdot \vec{p}}{\sqrt{s^2 - \vec{p} \cdot \vec{p}}\sqrt{s^2 - \vec{t} \cdot \vec{t}}}, \tag{7}$$

where $\vec{t}$ is a vector, with each element being the number of times a class occurred, $\vec{p}$ is a vector of the number of times each class is predicted, c is the number of samples correctly predicted, and s is the total number of samples. This correlation coefficient has a maximum value of 1, and the minimum value ranges between -1 and 0, depending on the distribution of the underlying data. A score of 0 indicates that the predictor is

Table 2: The performance of TGN variants is presented as the average of 10 random seed runs for each configuration. The best time encoding and aggregator versions are highlighted in bold and the cells representing readout configurations are shaded based on their values, with darker shading indicating higher performance.

		YelpCHI			Wikipedia			Mooc			Reddit		
		MCC	APS	AUC	MCC	APS	AUC	MCC	APS	AUC	MCC	APS	AUC
Time Encoding	fix	0.2624	0.3148	0.7590	**0.2943**	**0.1237**	**0.9086**	**0.1004**	0.0486	0.7634	0.0042	0.0049	**0.6608**
	learn	**0.2866**	**0.3278**	**0.7723**	0.1933	0.0989	0.8728	0.0973	**0.0571**	**0.7730**	**0.0444**	**0.0093**	0.6508
Aggregator	exp	0.2803	0.3262	0.7700	0.1018	0.0712	0.8653	**0.0630**	**0.0415**	**0.7494**	0.0158	0.0036	**0.6608**
	last	**0.2866**	**0.3278**	**0.7723**	**0.2943**	**0.1237**	**0.9086**	0.0477	0.0325	0.7045	**0.0444**	0.0055	**0.6599**
	mean	0.2744	0.3217	0.7666	0.2034	0.0896	0.8955	0.1004	0.0571	0.7730	0.0142	**0.0093**	0.6235
Readout	src	0.2286	0.2391	0.7096	0.1237	0.0828	0.8368	0.0530	0.0416	0.7199	0.0105	0.0045	0.6435
	dst	0.2288	0.2311	0.7015	0.0972	0.0464	0.7298	0.0432	0.0377	0.7195	0.0099	0.0049	0.6188
	src-dst	0.2319	0.2411	0.7094	0.1018	0.0355	0.8908	0.0924	0.0462	0.7485	0.0142	0.0031	0.6608
	src-t	0.2311	0.2426	0.7147	0.1308	0.0844	0.8401	0.0507	0.0386	0.7191	0.0104	0.0021	0.6472
	dst-t	0.2277	0.2381	0.7063	0.1057	0.0469	0.7379	0.0338	0.0411	0.7205	0.0159	0.0124	0.6211
	src-dst-t	0.2290	0.2542	0.7151	0.1442	0.0346	0.9086	0.0903	0.0506	0.7729	0.0444	0.0092	0.6508
	src-msg	0.2732	0.3166	0.7627	0.1530	0.0835	0.8808	0.0996	0.0603	0.7763	0.0074	0.0024	0.6046
	dst-msg	0.2744	0.3209	0.7641	0.1184	0.0564	0.8332	0.0475	0.0289	0.7112	0.0171	0.0144	0.5987
	src-dst-msg	0.2644	0.3147	0.7629	0.2040	0.0714	0.8537	0.0894	0.0497	0.7708	0.0158	0.0033	0.6599
	src-msg-t	0.2866	0.3278	0.7723	0.1764	0.0858	0.8994	0.1004	0.0571	0.7727	0.0040	0.0014	0.6005
	dst-msg-t	0.2803	0.3230	0.7669	0.1300	0.0617	0.7489	0.0462	0.0325	0.7045	0.0113	0.0093	0.5971
	src-dst-msg-t	0.2734	0.3217	0.7666	0.2943	0.1237	0.9020	0.0973	0.0536	0.7730	0.0077	0.0049	0.6089

perfectly random; a score of 1 indicates that the predictor is perfectly accurate; and a score of -1 indicates that the predictor is perfectly inaccurate. As an illustrative example, we present use case A1 from Table 4 in Chicco & Jurman (2020). In this example, we have 100 total data points with 91 in the positive class and 9 in the negative. For a hypothetical classifier that that predicts all but one data point as a member of the positive class; we find $TP = 90, FN = 1, TN = 0, FP = 9$. This yields a respectable APS of 0.90 but a near random MCC of -0.03. While simple, this is just one example where metrics like APS can mask underlying poor performance for imbalanced datasets. Chicco & Jurman (2023) further presents similar failure modes for AUC. Because of this, we choose MCC as our figure of merit for the RLC tasks that we present.

5.1 Key factors to tailor model to specific needs of data

With the introduction of RLC as a benchmark task to evaluate temporal graph learning methods, we explore the performance of a well-known state-of-the art model, TGN (Rossi et al., 2020), together with variants. We create these variants by constructing different message aggregation schema, readout layers, and time encoding strategies to better discover the potential of the overall architecture. More specifically, we experiment with (1) fixed and learnable time encoding as proposed by (Cong et al., 2023) and by (Kumar et al., 2019),

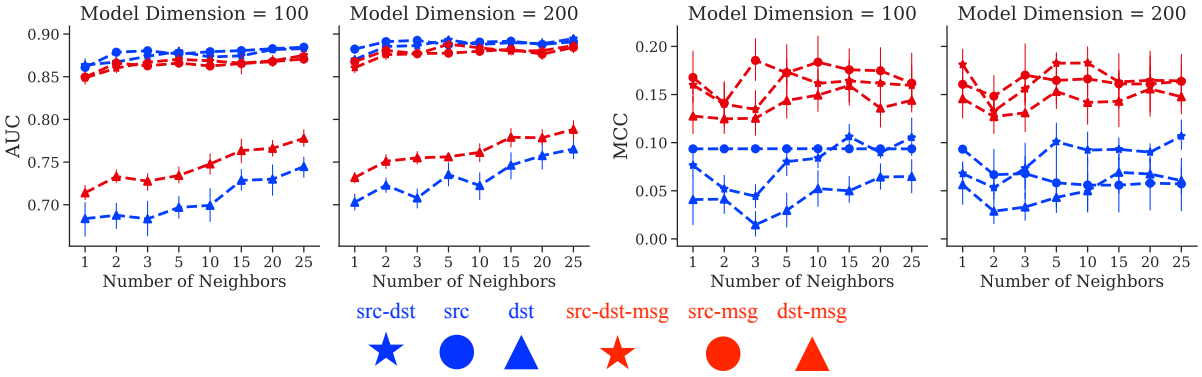

Figure 4: Readout variations on Wikipedia. The blue glyphs correspond to combinations of the vertex features, while the red glyphs correspond to combinations of the vertex *and* message features. The star, circle, and triangle glyphs correspond to the src-dst, src, and dst embeddings respectively.

Table 3: The performance of models is presented as the average of 10 random seed runs. The best and second best results are highlighted in red and blue, respectively.

		Epic Games	YelpCHI	Wikipedia	Mooc	Reddit	Open Sea
MCC	MLP	0.1554	0.2763	0.2354	0.0673	0.0021	0.1106
	TGN	0.8373	0.2372	0.1442	0.0991	0.0174	0.1071
	TGAT	0.5546	0.1890	0.0000	0.0000	0.0000	0.2245
	Graph Mixer	0.2316	0.2830	0.1442	0.1174	0.0000	0.2647
	Modified TGN	0.8713	0.2866	0.2943	0.1004	0.0444	0.2647
	Graph Profiler	0.9355	0.3274	0.2498	0.1739	0.0115	0.2959
APS	MLP	0.6976	0.3254	0.0759	0.0169	0.0012	0.7790
	TGN	0.9850	0.2461	0.0320	0.0526	0.0018	0.7912
	TGAT	0.8844	0.1789	0.0157	0.0190	0.0014	0.6582
	Graph Mixer	0.7522	0.3252	0.0550	0.0711	0.0021	0.8304
	Modified TGN	0.9892	0.3249	0.1148	0.0430	0.0055	0.8225
	Graph Profiler	0.9988	0.4059	0.0955	0.0896	0.0092	0.8459
AUC	MLP	0.6117	0.7694	0.7731	0.6447	0.5486	0.5776
	TGN	0.9734	0.7135	0.7135	0.7672	0.5671	0.5908
	TGAT	0.8470	0.6314	0.8908	0.6383	0.5336	0.6871
	Graph Mixer	0.7132	0.7650	0.7500	0.7515	0.6413	0.6789
	Modified TGN	0.9807	0.7723	0.7723	0.7439	0.6508	0.6544
	Graph Profiler	0.9974	0.8058	0.7821	0.7886	0.6280	0.6740

respectively; (2) mean, last, and exponential decay message aggregators; and (3) six different configurations of the input to the readout layer based on different combinations of source, destination, time and message embeddings calculated for the most recent event. As described by (Rossi et al., 2020), the mean aggregator calculates the state of a node by averaging the interactions held in the memory, whereas the last aggregator uses only the most recent interaction. The exponential decay variant calculates a weighted average over the interactions in the memory by setting weights such that they decrease exponentially with respect to the time that has elapsed. Table 2, summarizes the results. We present violin plots for each variant in Appendix E.

The readout variations appeared to play a significant role in the model's performance, as can be seen in Figure 4. These results on the Wikipedia dataset are demonstrated for different model dimensions, i.e. $d \in \{100, 200\}$ and a varying number of neighbours. We observe that incorporating the edge features as a residual at the final layer of the update helps to improve the performance in terms of MCC, which makes intuitive sense given that the message features for this dataset correspond to the edit's content. Interestingly, while we observe this trend when looking at the MCC curves, which exhibit a dramatic stratification in performance, the AUC curves show the opposite trend. We attribute this to AUC being an unsatisfactory metric for the evaluation of RLC tasks, particularly if there is class imbalance. The APS and Loss curves exhibit trends that are similar to those of MCC (See Appendix F). We conclude that for abuse-related datasets the most recent interaction matters most, and therefore aggregation based on the last event is more useful. In the case of predicting course completion, averaging over multiple previous actions is valuable, which is captured by the outperformance of the mean aggregator. In general, we observe that involving recent interaction features in the readout layer is very useful. The configurations with `msg` included perform significantly better.

5.2 Model Comparison

Using the results from the TGN modifications, we have identified multiple core design principles associated with improved performance on the RLC task, and we have incorporated these principles into Graph Profiler. Specifically, we observed that a learnable time-encoding provided improved results for three of four data sets. The `src-dst-msg-t` readout variant provided strong results across all four data sets. Because of these

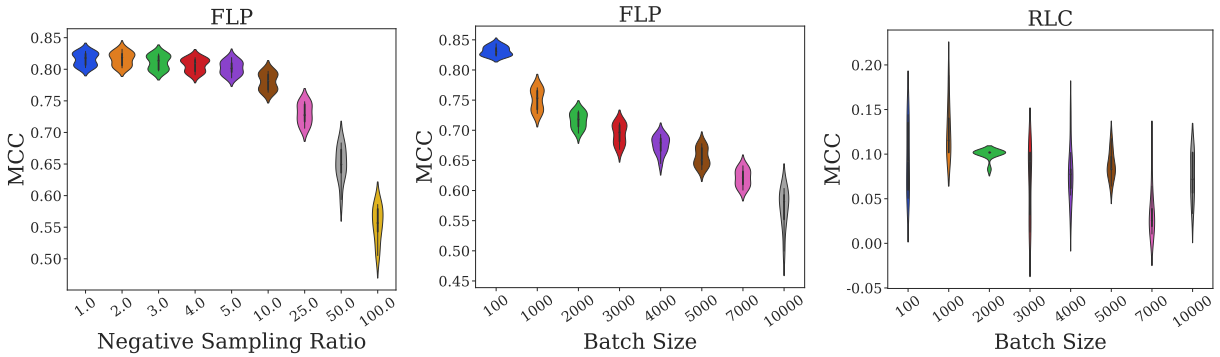

Figure 5: TGN performance on FLP vs RLC on Wikipedia with varying levels of negative sampling ratio and batch size. The negative sampling ratio and batch size are very effective on performance on FLP while RLC is free of negative sampling ratio, and batch size is not an effective factor on the performance.

results, we designed Graph Profiler with a learnable time encoder and a `src-dst-msg-t` readout function. To validate these results, we perform benchmark experiments of baselines and Graph Profiler on RLC. For the existing benchmark datasets, Graph Profiler obtains the best results for YelpCHI and Mooc, but our modified version of the TGN architecture outperforms for Wikipedia and Reddit. The most probable reason is that the ratio of the number of source nodes to the number of destination nodes is much higher for YelpCHI and Mooc compared to Wikipedia and Reddit. Graph Profiler constructs embeddings for the source nodes via calculations over neighbouring interactions, but uses learnable embeddings in a lookup table for the destination nodes. When the number of destination nodes is smaller, this is an easier learning task.

Another observation we draw from these model comparison results is the usefulness of MCC in highlighting the relative capabilities of different models. For example, on Mooc Graph Profiler improves MCC by 73% compared to the Modified TGN, while the observed change for the AUC is only 3% (See Table 3). On the Epic Games and Open Sea datasets, which are less imbalanced, Graph Profiler outperforms other baselines consistently based on MCC and APS, but the performance of the techniques cannot be reliably distinguished using AUC. Revisiting the dataset statistics provided in Table 1, we conclude that encoding node profiles based on their shared history, i.e., employing the Graph Profiler architecture, is more effective on datasets with higher balanced edge homophily and less class imbalance. For such datasets, tracking a graph-wide memory, i.e., employing the TGN framework, is less effective.

5.3 On the importance of hyperparameter sensitivity differences between FLP and RLC

In our reproduction study we explored the effect of negative sampling ratio, batch size, and the number of sampled neighbors on the performance of our TGN baseline for the FLP task. We term these as non-architectural parameters because they influence the training but do not influence the architecture of the model itself. We decided to explore these parameters because they impact the trade-off between model accuracy versus computational performance and utilization. In the example of batch size, this is typically tuned to be as large as possible to maximize GPU utilization, but we see in Figure 5, a steady decline in MCC (other metrics can be found in the appendix) as the batch size is increased. Indeed, we observe variation in model performance due to changes in batch size that are larger than the variations across different model architectures. Intuitively, the decay makes sense, because the gradient updates become less frequent, but this points to a relatively significant but under-discussed trade-off that has major ramifications for production use-cases. In the case of the negative sampling ratio, we observe a slight decline in MCC and a decrease in the consistency between individual training runs as the number of negative samples increases. Thus, it can be concluded that RLC does not have the same dependence on batch-size and does not require the generation of negative samples. These observations lead us to the conclusion that the assumptions made during the development of models for FLP may not hold for RLC, and direct translation of existing TGL methods, which are generally benchmarked on FLP tasks, to perform RLC in industrial settings is not

Table 4: The impact of using destination embeddings

	MCC		APS		AUC	
	with	*without*	*with*	*without*	*with*	*without*
Epic Games	**0.9355**	0.7695	**0.9988**	0.9575	**0.9974**	0.9054
YelpCHI	**0.3274**	0.3071	**0.4059**	0.3892	**0.8058**	0.8000
Wikipedia	**0.2498**	0.1324	**0.0955**	0.0366	**0.7821**	0.6946
Mooc	**0.1739**	0.0000	**0.0896**	0.0011	**0.7886**	0.5906
Reddit	0.0115	**0.0701**	0.0092	**0.0203**	0.6280	**0.6833**

advisable. Because of this, we believe that RLC is an interesting general purpose benchmark task for the TGL community, and should be treated differently from the current common tasks. The full set of results are illustrated in Appendix D.

5.4 Ablation Study

In order to investigate the impact of learnable destination embeddings, we experiment with and without learning a set of embeddings for the destination nodes. The results are presented in Table 4. For most of the datasets (with the exception of Reddit), incorporating learnable destination node representations improves the predictive performance of Graph Profiler. The destination node embeddings are then used for building the source node profiles. It can be inferred that destination encoding enriches the source profile embeddings by temporally smoothing the interactions to build a sense of history. By contrast, using the destination in the readout itself means that the history of destination nodes is ignored. We believe that the difference for the Reddit dataset may arise from the fact that the labels (whether a sub-reddit is controversial) is less dependent on the main post, compared to other Wikipedia. Said another way, there are many Wikipedia pages that are prone to abuse for political reasons, and thus, the page profile (destination node) matters. By contrast, abuse on Reddit is less dependent on the sub-reddit (destination node) than the author (source node).

6 Conclusion

In this work, we introduce recent link classification on temporal graphs as a benchmark downstream task and evaluate the most competitive state-of-the-art method's performance using a statistically meaningful metric, namely *Matthews Correlation Coefficient*. This metric is more robust to imbalanced datasets in comparison to the commonly used average precision and area under the curve. We propose several design principles for tailoring models to specific requirements of the task and the dataset. These design considerations address the choice of message aggregation schema, readout layer and time encoding strategy. We show that appropriate selection can lead to a significant improvement on benchmark datasets. We present a novel architecture that we call *Graph Profiler*. This is a recent link classification algorithm designed for bipartite graphs, which are commonly encountered in industrial settings. We believe the introduction of recent link classification as a benchmark task for temporal graph learning is useful for the evaluation of prospective methods within the field.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Thibaud Arnoux, Lionel Tabourier, and Matthieu Latapy. Combining structural and dynamic information to predict activity in link streams. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pp. 935–942, 2017.

Pietro Bongini, Monica Bianchini, and Franco Scarselli. Molecular generative graph neural networks for drug discovery. *Neurocomputing*, 450:242–252, 2021.

Ling Cai, Bo Yan, Gengchen Mai, Krzysztof Janowicz, and Rui Zhu. Transgcn: Coupling transformation assumptions with graph convolutional networks for link prediction. In *Proceedings of the 10th international conference on knowledge capture*, pp. 131–138, 2019.

Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. *arXiv preprint arXiv:2209.15486*, 2022.

Bo Chen, Di Zhu, Yuwei Wang, and Peng Zhang. An approach to combine the power of deep reinforcement learning with a graph neural network for routing optimization. *Electronics*, 11(3):368, 2022.

Hongxu Chen, Hongzhi Yin, Xiangguo Sun, Tong Chen, Bogdan Gabrys, and Katarzyna Musial. Multi-level graph convolutional networks for cross-platform anchor link prediction. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1503–1511, 2020.

Yali Chen and Xiujuan Lei. Metapath aggregated graph neural network and tripartite heterogeneous networks for microbe-disease prediction. *Frontiers in Microbiology*, 13:919380, 2022.

Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

Davide Chicco and Giuseppe Jurman. The matthews correlation coefficient (mcc) should replace the roc auc as the standard metric for assessing binary classification. *BioData Mining*, 16(1):1–23, 2023.

Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? In *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

Davide Costa, Lucio La Cava, and Andrea Tagarelli. Show me your nft and i tell you how it will perform: Multimodal representation learning for nft selling price prediction. In *Proc. ACM Web Conf.*, pp. 1875–1885, 2023.

da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. Inductive representation learning on temporal graphs. In *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.

Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proc. ACM Int. Conf. Information and Knowledge Management*, 2020.

Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofía Samaniego, Ying Xiao, et al. Twhin: Embedding the twitter heterogeneous information network for personalized recommendation. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 2842–2850, 2022.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. Graph neural networks for recommender systems: Challenges, methods, and directions. *arXiv preprint arXiv:2109.12843*, 2021.

Jianfei Gao and Bruno Ribeiro. On the equivalence between temporal and static equivariant graph representations. In *Proc. Int. Conf. Machine Learning Research (PMLR)*, 2022.

Jan Gorodkin. Comparing two k-category assignments by a k-category correlation coefficient. *Computational biology and chemistry*, 28(5-6):367–374, 2004.

William L. Hamilton. *Graph Representation Learning*, volume 14. Morgan and Claypool, 2020.

Kehang Han, Balaji Lakshminarayanan, and Jeremiah Liu. Reliable graph neural networks for drug discovery under distributional shift. *arXiv preprint arXiv:2111.12951*, 2021.

Zepeng Huai, Yuji Yang, Mengdi Zhang, Zhongyi Zhang, Yichun Li, and Wei Wu. M2gnn: Metapath and multi-interest aggregated graph neural network for tag-based cross-domain recommendation. *arXiv preprint arXiv:2304.07911*, 2023.

Mingyuan Huang, Pengpeng Zhao, Xuefeng Xian, Jianfeng Qu, Guanfeng Liu, Yanchi Liu, and Victor S Sheng. Help from meta-path: Node and meta-path contrastive learning for recommender systems. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 01–08. IEEE, 2022.

Ryan Keisler. Forecasting global weather with graph neural networks. *arXiv preprint arXiv:2202.07575*, 2022.

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

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2019.

Lucio La Cava, Davide Costa, and Andrea Tagarelli. Sonar: Web-based tool for multimodal exploration of non-fungible token inspiration networks. In *Proc. ACM SIGIR*, 2023a.

Lucio La Cava, Davide Costa, and Andrea Tagarelli. Visually wired nfts: Exploring the role of inspiration in non-fungible tokens. *CoRR abs/2303.17031*, 2023b.

Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pp. 1569–1572, 2020.

Minbo Ma, Peng Xie, Fei Teng, Tianrui Li, Bin Wang, Shenggong Ji, and Junbo Zhang. Histgnn: Hierarchical spatio-temporal graph neural networks for weather forecasting. *arXiv preprint arXiv:2201.09101*, 2022.

Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016.

Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems*, 214:106746, 2021.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proc. AAAI Conf. Artificial Intelligence*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1e2agrFvS`.

Francesco Saverio Pezzicoli, Guillaume Charpiat, and François P Landes. Se (3)-equivariant graph neural networks for learning glassy liquids representations. *arXiv preprint arXiv:2211.03226*, 2022.

Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *Proc. Int. Conf. Learning Representations (ICLR) Graph Representation Learning Workshop*, 2020.

Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 140–151, 2019.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proc. Int. Conf. Web Search and Data Mining*, 2020.

Tushar Sarkar. Xbnet: An extremely boosted neural network. *Intelligent Systems with Applications*, 15: 200097, 2022.

Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 255–262, 2016.

Chenguang Song, Kai Shu, and Bin Wu. Temporally evolving graph neural network for fake news detection. *Information Processing & Management*, 58(6):102712, 2021.

Hewen Wang, Renchi Yang, Keke Huang, and Xiaokui Xiao. Efficient and effective edge-wise graph representation learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2326–2336, 2023.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

Xuyang Wu, Xinyang Gao, Weinan Zhang, Rui Luo, and Jun Wang. Learning over categorical data using counting features: With an application on click-through rate estimation. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, pp. 1–9, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Jiacheng Xiong, Zhaoping Xiong, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. Graph neural networks for automated de novo drug design. *Drug Discovery Today*, 26(6):1382–1393, 2021.

G Udny Yule. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 75(6):579–652, 1912.

Adnan Zeb, Summaya Saif, Junde Chen, Anwar Ul Haq, Zhiguo Gong, and Defu Zhang. Complex graph convolutional network for link prediction in knowledge graphs. *Expert Systems with Applications*, 200: 116796, 2022.

Ge Zhang, Zhao Li, Jiaming Huang, Jia Wu, Chuan Zhou, Jian Yang, and Jianliang Gao. efraudcom: An e-commerce fraud detection system via competitive graph neural networks. *ACM Transactions on Information Systems (TOIS)*, 40(3):1–29, 2022.

Liang Zhang, Jingqun Li, Bin Zhou, and Yan Jia. Rumor detection based on sagnn: Simplified aggregation graph neural networks. *Machine Learning and Knowledge Extraction*, 3(1):84–94, 2021a.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34: 9061–9073, 2021b.

Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. TGL: A general framework for temporal gnn training on billion-scale graphs. *Proc. VLDB Endow.*, 15(8), 2022.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Appendices

A Implementation Details

In an effort to present fair comparisons, we performed 100 steps of hyperparameters optimization to optimize the hyperparameters of all models using the software package OPTUNA (Akiba et al., 2019). Each experiment was run over the same 10 seeds. All tuning was performed on the validation set where we maximize the average accuracy across all 10 seeds, and we report the test-seed averaged results on the test set that are associated with those hyperparameter settings that maximize the validation accuracy. All models were implemented using PYTORCH GEOMETRIC 2.3.1 (Fey & Lenssen, 2019) and PYTORCH 1.13 (Paszke et al., 2019). We implemented TGN using the layers that are publicly available in PYTORCH GEOMETRIC, GraphMixer [3] and TGAT [4] were implemented using the authors opensource implementation provided on their github repository. All computations were run on an `Nvidia DGX A100` machine with 128 `AMD Rome 7742` cores and 8 `Nvidia A100` GPUs.

B Details of Pre-processing on Open Sea Dataset

The OPEN SEA dataset is a collection of Non-Fungible Token (NFT) transactions conducted between 2021 and 2023, provided by La Cava et al. (2023a); Costa et al. (2023); La Cava et al. (2023b). Originally, the data involves 70 million transactions chronologically divided into ten splits. In order to scale our experiments, we use the first split which is composed of 7,097,215 transactions spanned between 2022-12-17 and 2023-02-18. In Table 5 the fields of the raw data is provided. The pre-processing steps are as follows:

1. The unique NFT ids are set grouping the transaction by `token_id` and `collection_name`.
2. The transactions are filtered for each unique NFT, and assigned an `order` based on their position in the sequence and the last ones are flagged. A new field of `next_revenue` is created such that for k^{th} transaction in the sequence is assigned with the `usd_gain` of $(k+1)^{\text{th}}$ one, except the flagged ones.
3. For each transaction without the flag, a new field of `benefit` is calculated by `next_revenue - usd_price` and the flagged transactions are filtered.
4. The number of transactions reduces to 2,979,950 which is still high for the scale of dataset size we experiment with. Thus, we select 1000 different buyers by `winner_account` who conduced most transactions and filter transactions that involve them, which reduces the number of transactions to 282,743.
5. The source node identities are set by `seller_account`, destination node identities by `winner_account`, the transaction time is set by `tx_timestamp`.
6. Feature vectors of transactions are calculated by concatenating normalized quantitative feature fields [`fees_seller`, `fees_opensea`, `fees_seller_usd`, `fees_opensea_usd`, `price`, `gain`, `usd_price`, `usd_gain`, `to_eth`, `to_usd`, `created_date`] and binary representations of categorical feature fields [`token`, `chain`, `token_type`, `asset_contract_type`, `asset_type`].
7. The response variable is set by binarizing `benefit` such that it is set to 1 if `benefit` > 0, and 0 otherwise.

[3] https://github.com/CongWeilin/GraphMixer
[4] https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs

Table 5: Raw data fields of Open Sea dataset

Variable	Type	Description	Processing
token_id	String	The id of the NFT — this value is unique within the same collection	Anonymized
num_sales	Integer	A progressive integer indicating the number of successful transactions involving the NFT up to the current timestamp (cf. *tx_timestamp*)	Original
nft_name	Vector ID	The name of the NFT	Anonymized
nft_description	Vector ID	The description of the NFT as provided by the creator	Anonymized
nft_image	Vector ID	The ID for accessing the NFT image vector	Anonymized
collection_name	Vector ID	The ID for accessing the Collection name vector	Anonymized
collection_description	Vector ID	The ID for accessing the Collection description vector	Anonymized
collection_image	Vector ID	The ID for accessing the Collection image vector	Anonymized
fees_seller	Float	The absolute amount of fees the seller has gained from this transaction expressed in *token*	Original
fees_opensea	Float	The absolute amount of fees OpenSea has gained from this transaction expressed in *token*	Original
fees_seller_usd	Float	The absolute amount of fees the seller has gained from this transaction expressed in US dollars (USD)	Original
fees_opensea_usd	Float	The absolute amount of fees OpenSea has gained from this transaction expressed in US dollars (USD)	Original
payout_collection_address	String	The wallet address where seller fees are deposited	Anonymized
tx_timestamp	String	Timestamp of the transaction expressed in yyyy-mm-ddTHH:MM:SS	Original
price	Float	The price of the transaction expressed in token	Original
gain	Float	The gain after fees (i.e., gain = price - fees_opensea * price - fees_seller * price)	Original
usd_price	Float	The price of the transaction expressed in US dollars (USD)	Original
usd_gain	Float	The difference between the price and the fees expressed in US dollars (USD)	Original
token	Categorical	The token type used to pay the transaction	Original
to_eth	Float	The conversion rate to convert tokens into Ethereum at the current timestamp, such that eth = price * to_eth	Original
to_usd	Float	The conversion rate to convert tokens into US dollars (USD) at the current timestamp, such that usd = price * to_usd	Original
from_account	String	The address that sends the payment (i.e., winner/buyer)	Anonymized
to_account	String	The address that receives the payment (it often corresponds to the contract linked to the asset)	Anonymized
seller_account	String	The address of the NFT seller	Anonymized
winner_account	String	The address of the NFT buyer	Anonymized
contract_address	String	The contract address on the blockchain	Anonymized
created_date	Timestamp	The date of creation of the contract	Original
chain	Categorical	The blockchain where the transaction occurs	Original
token_type	Categorical	The schema of the token, i.e., ERC721 or ERC1155	Original
asset_contract_type	Categorical	The asset typology, i.e., non-fungible or semi-fungible	Original
asset_type	Categorical	Whether the asset was involved in a simple or bundle transaction	Original

C Edge Homophily Trends in Datasets

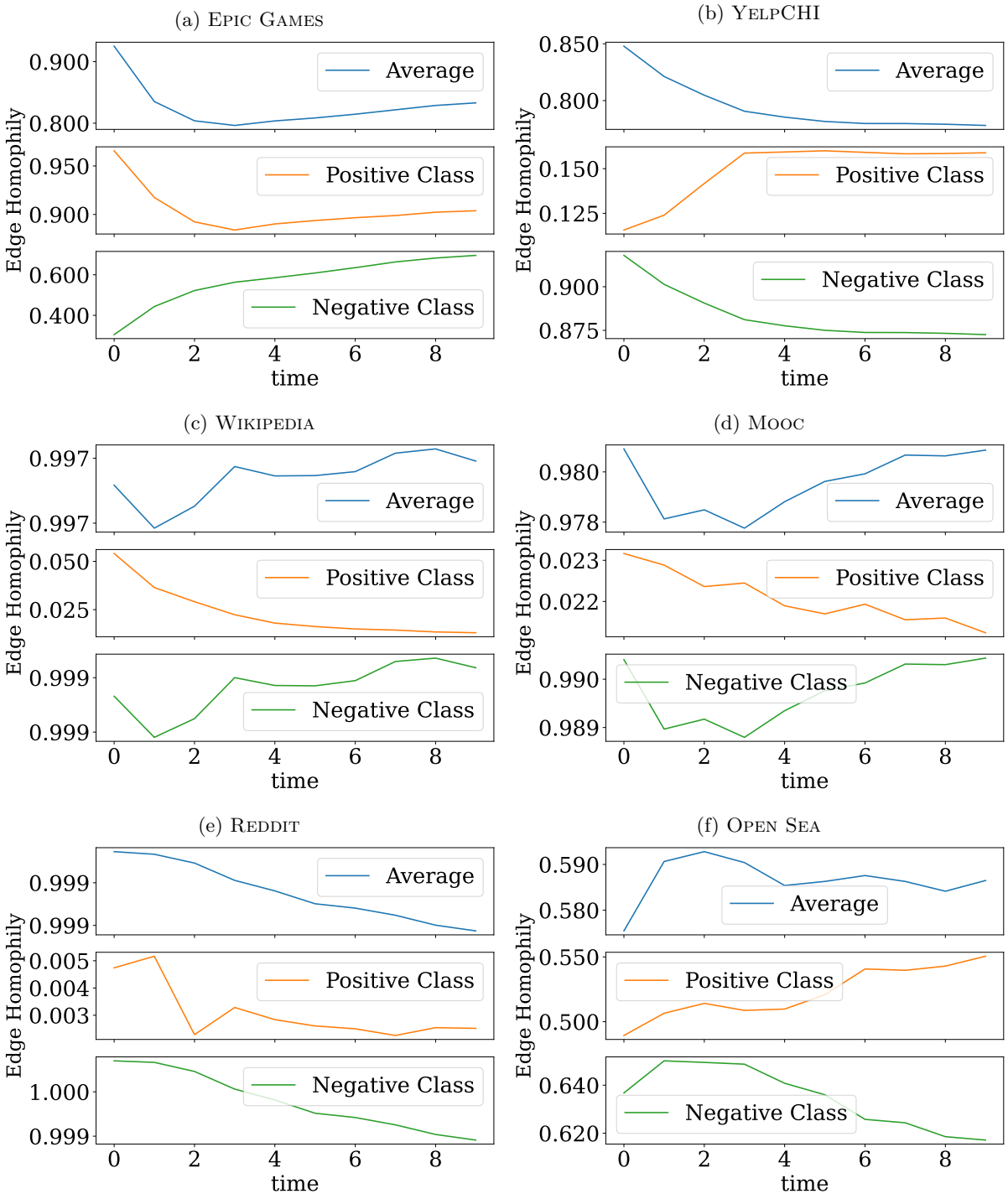

Figure 6: Edge homophily (measured using the metric Equation 6) trends as a function of time (with time measured in batches).

D Parameter Sensitivity Analysis

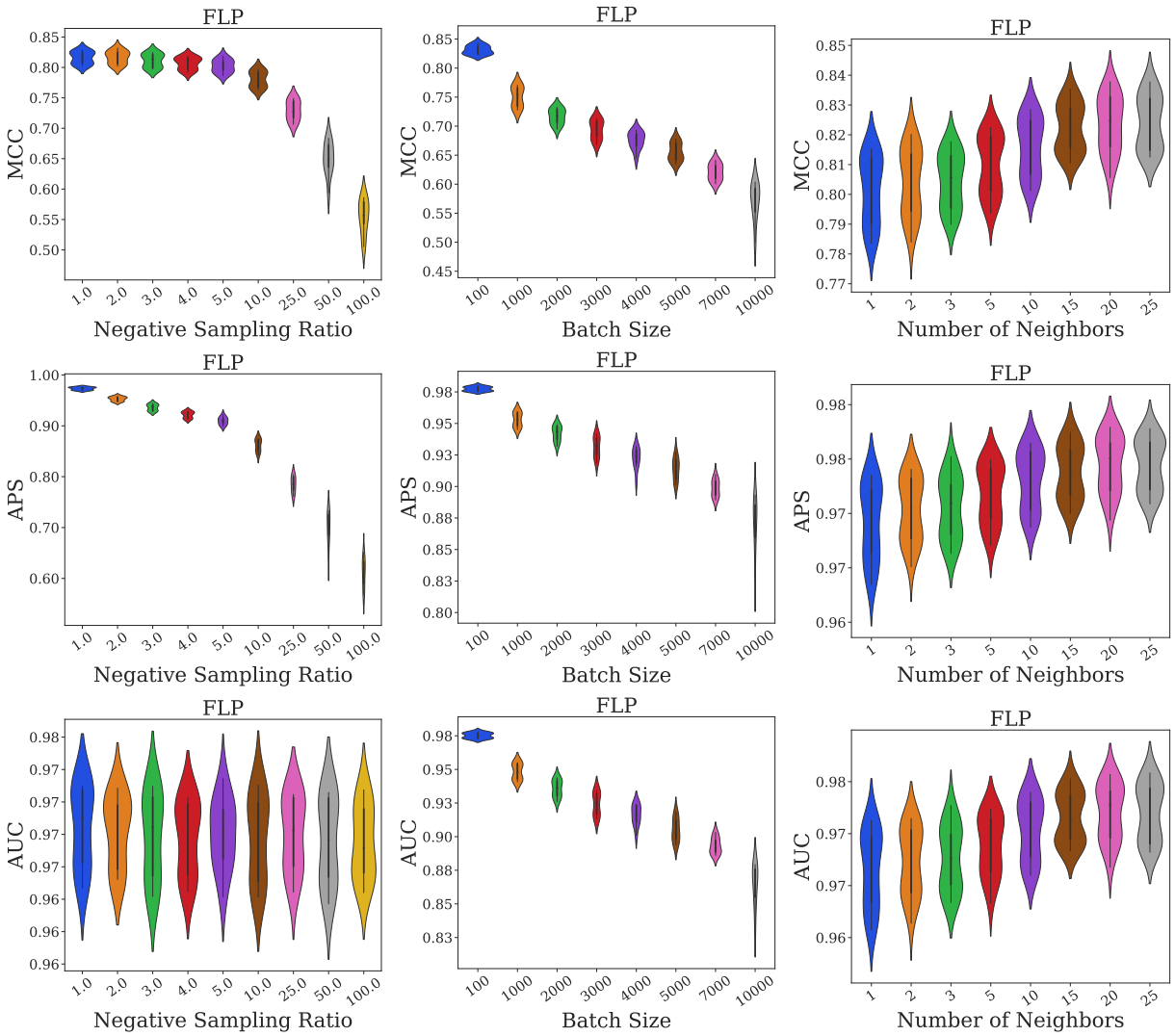

Figure 7: Parameter Sensitivity of FLP - Wikipedia

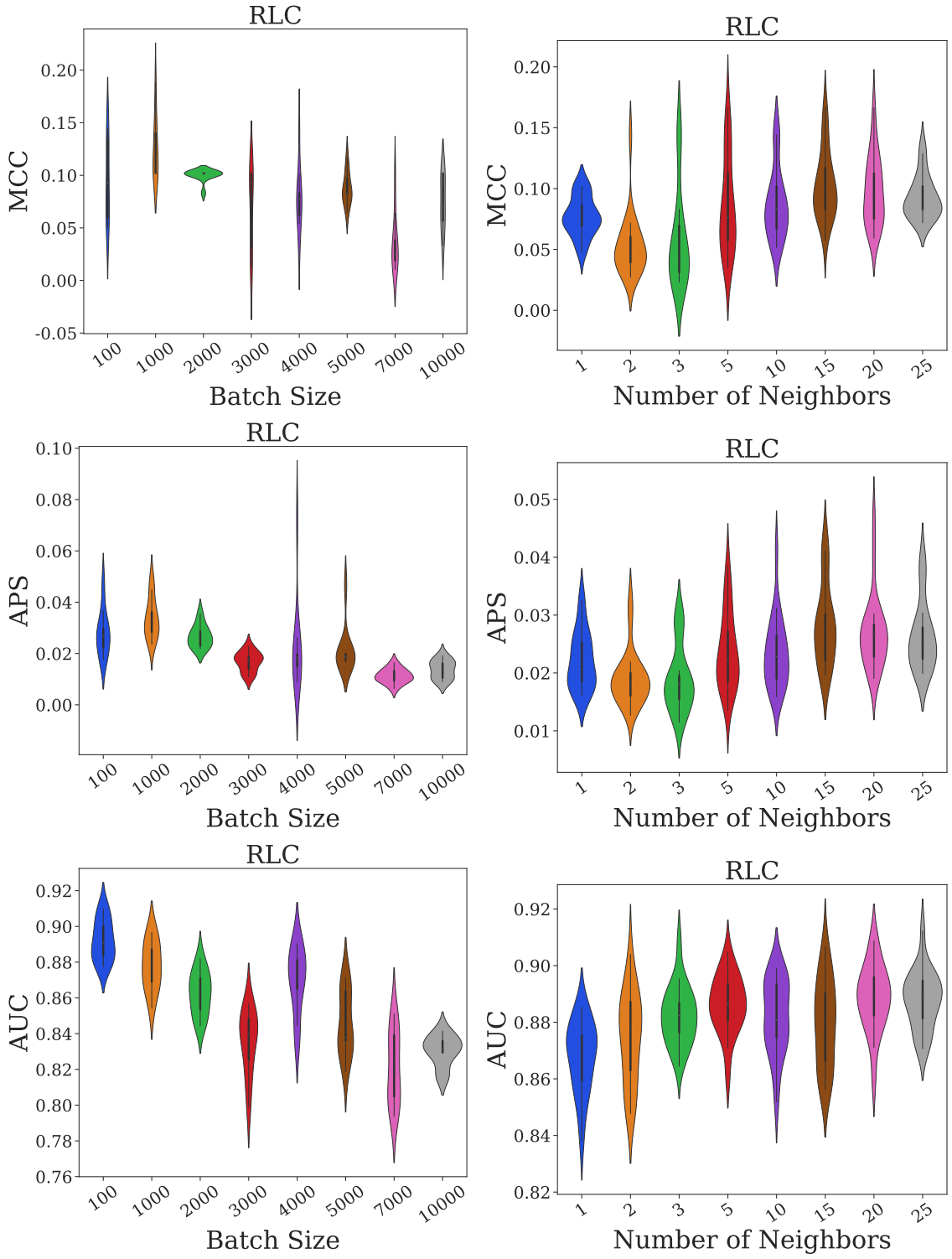

Figure 8: Parameter Sensitivity of RLC - Wikipedia

E TGN Modifications on RLC

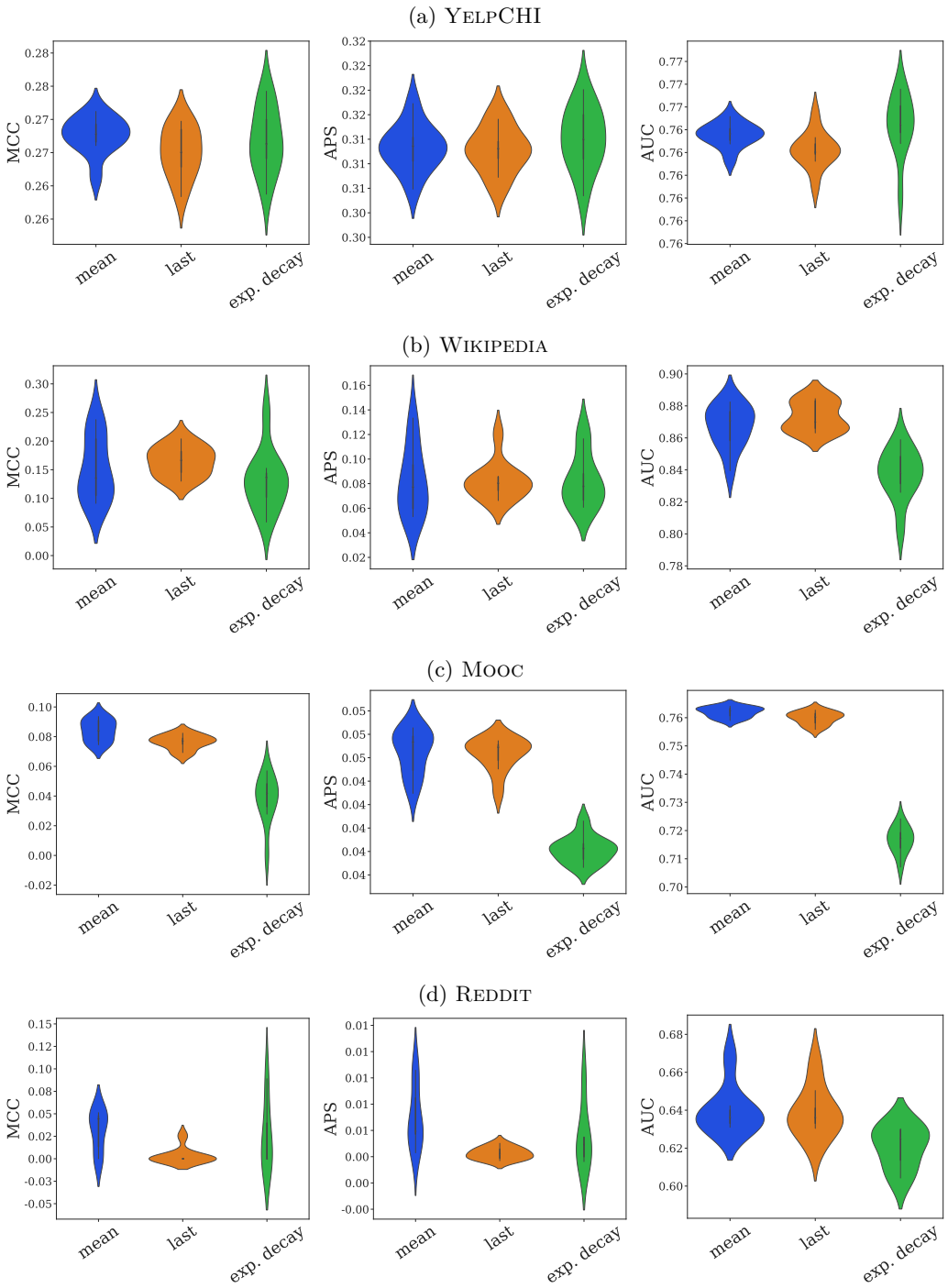

Figure 9: Aggragator Versions

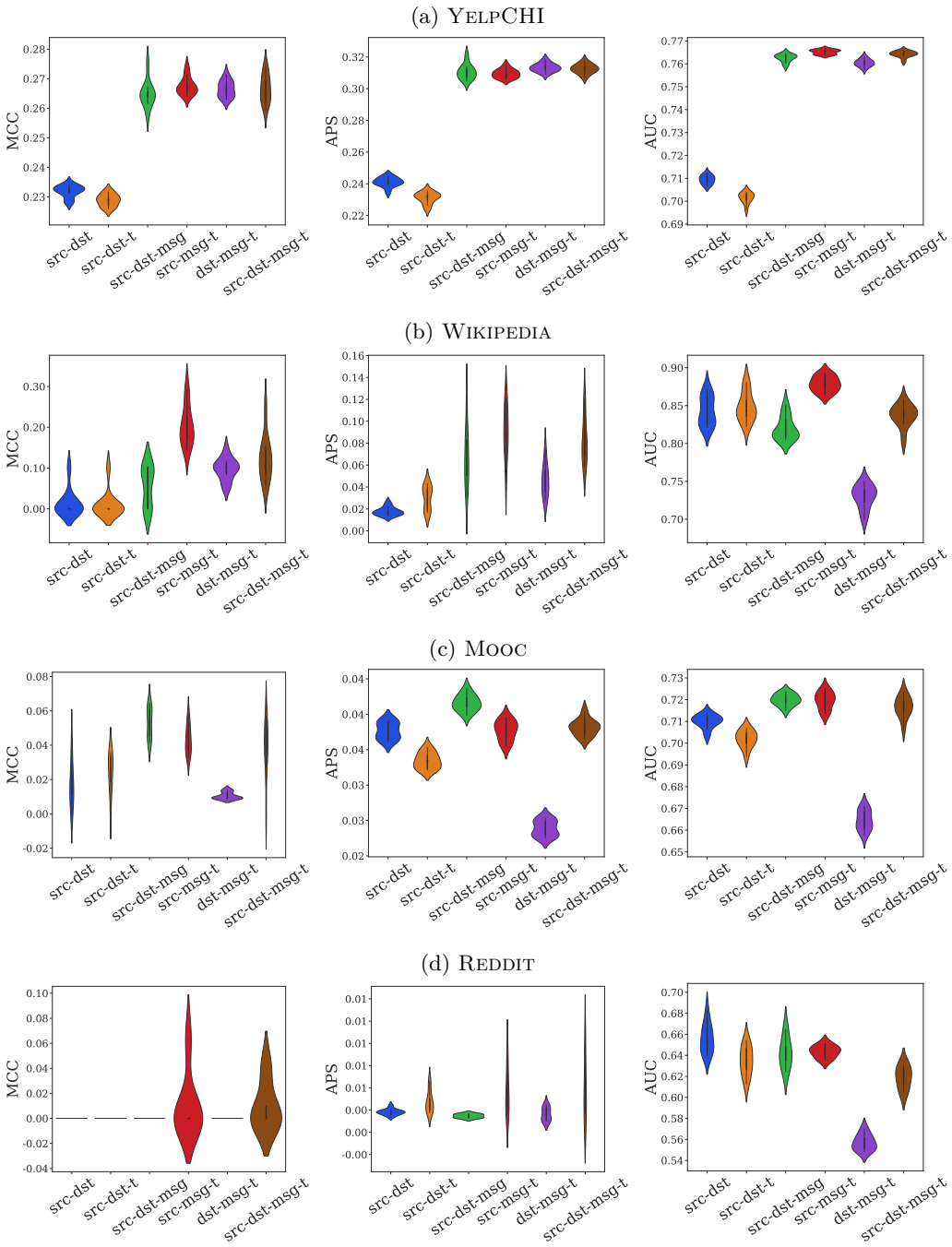

Figure 10: Readout Versions

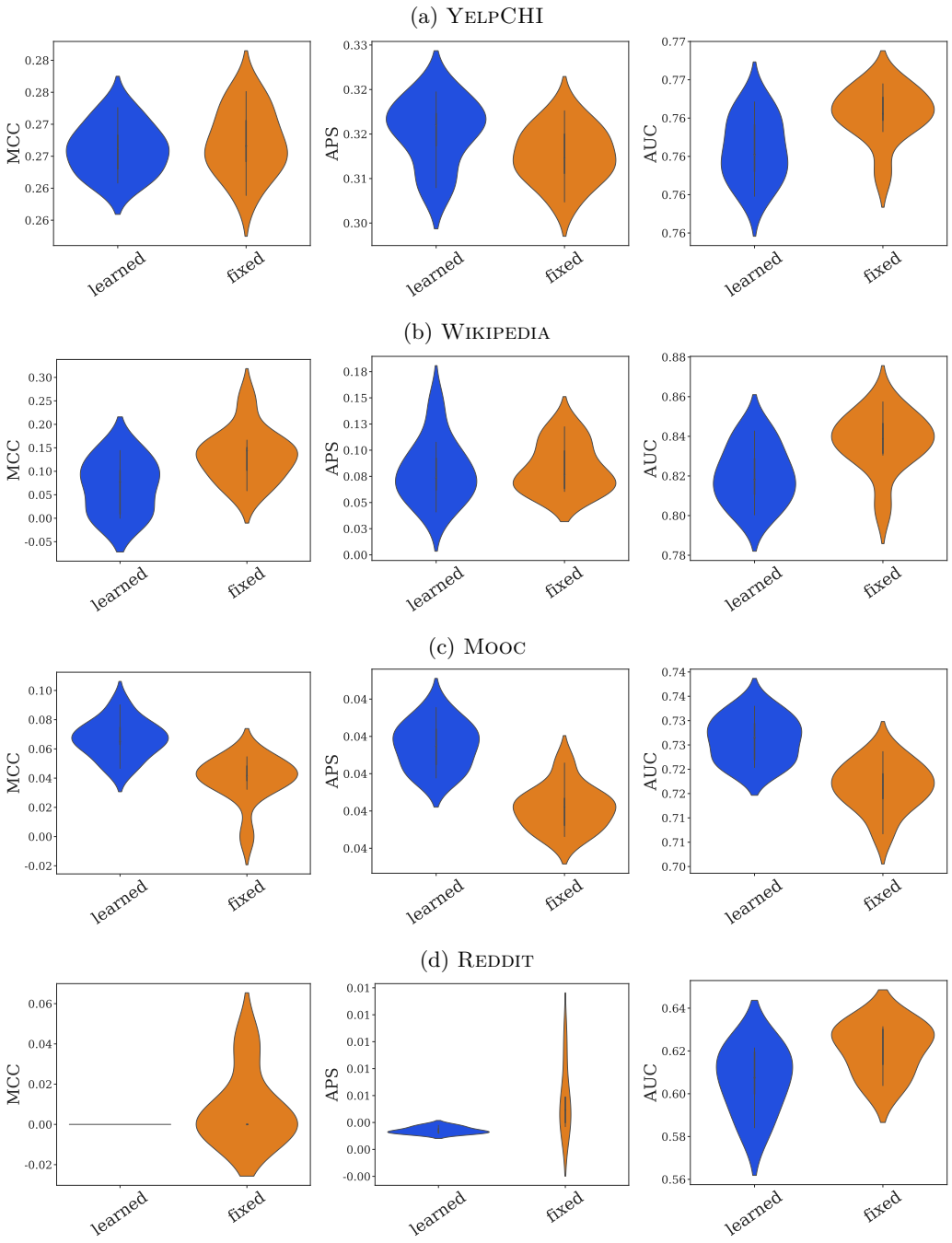

Figure 11: Time Encoding Versions

F Readout Variations on RLC

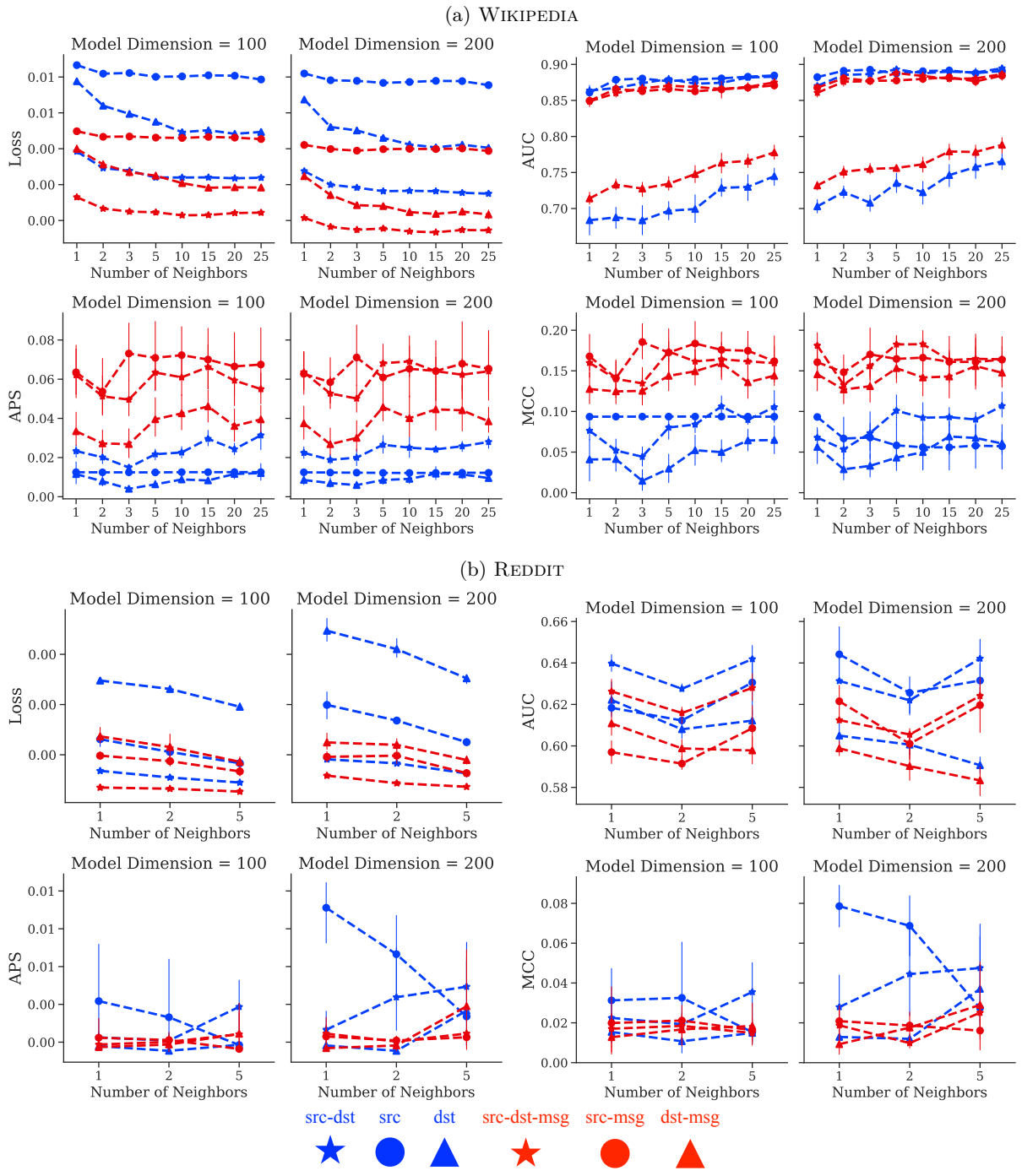

Figure 12: The performance on RLC using different variations of readout layer