Holistic Prediction on a Time-Evolving Attributed Graph

Shohei Yamasaki[1,*,†], **Yuya Sasaki**[2,*], **Panagiotis Karras**[3], **Makoto Onizuka**[2]
[1]Nomura Research Institute, Ltd., Japan, [2]Osaka University, Japan
[3]Aarhus University, Denmark
s2-yamasaki@nri.co.jp, {sasaki,onizuka}@osaka-u.ac.jp, piekarras@gmail.com

Abstract

Graph-based prediction is essential in NLP tasks such as temporal knowledge graph completion. A cardinal question in this field is, how to predict the future links, nodes, and attributes of a time-evolving attributed graph? Unfortunately, existing techniques assume that each link, node, and attribute prediction is independent, and fall short of predicting the appearance of new nodes that were not observed in the past. In this paper, we address two interrelated questions; (1) can we exploit task interdependence to improve prediction accuracy? and (2) can we predict new nodes with their attributes? We propose a unified framework that predicts node attributes and topology changes such as the appearance and disappearance of links and the emergence and loss of nodes. This framework comprises components for independent and interactive prediction and for predicting new nodes. Our experimental study using real-world data confirms that our interdependent prediction framework achieves higher accuracy than methods based on independent prediction.

1 Introduction

Real-world language-based data such as blog posts, documents, and user profiles is often interconnected, as in social networks and co-author networks. This interconnection is modeled by graphs where nodes represent objects and edges represent relationships (Bansal et al., 2019; Bai et al., 2021; Zhu et al., 2019; Wu et al., 2021). The structures and node attributes of such graphs often evolve over time by node addition/deletion, link addition/deletion, and node attribute changes. For instance, social networks change over time in terms of participating users, their profiles, and their links. Such temporally malleable graphs are called *time-evolving attributed graphs* (Rossi et al., 2020).

Graph-based methods are essential in various NLP tasks (Zhou et al., 2021; Mondal et al., 2021; Xie et al., 2021). NLP applications often call for *predicting* the future of a time-evolving attributed graph such as completing a temporal knowledge graph (Goel et al., 2020; Mirza and Tonelli, 2016; Xu et al., 2021a), predicting a profile in social networks (Hasanuzzaman et al., 2017), and recommending news articles to readers (Wu et al., 2019); the prediction task involved is composite, including several sub-prediction tasks. Conventionally, this composite prediction task is addressed in a *compartmentalized* manner, separately applying a technique for each component sub-prediction task. This compartmentalized approach treats component prediction tasks independently rather than as an interdependent whole and utilizing one prediction to inform another. Besides, to our knowledge, existing works in time-evolving graph prediction do not predict new nodes and their attributes.

Example 1.1 *Let us consider profile prediction in a social network over a span of time in the future, e.g., job post in two years (Hasanuzzaman et al., 2017). Profile information can be predicted from posts and connections to other users. A social network is time-evolving, user profiles change dynamically, new users register their accounts and some users delete their accounts. Connections also change; newly registered accounts acquire connections to existing accounts with whom they have similar profiles; accounts connected to deleted accounts may connect to other accounts. These predictions are interdependent.*

In this paper, we introduce the problem of holistically predicting the future in a time-evolving attributed graph. Figure 1 illustrates the problem, including multiple sub-predictions, such as those of node loss and new node appearance. To achieve high prediction accuracy, we need to capture the interdependence between sub-prediction tasks. As it is difficult for a single method to capture all in-

[*]These authors contributed equally to this work
[†]This work was done when Shohei Yamasaki was a student at Osaka University

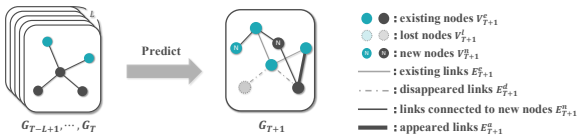


Figure 1: Example of time-evolving attributed graph prediction; node colors represent attributes; T is the current time step; L is the number of time steps.

terdependences, we must effectively combine several methods and interactively reuse the results of one task in other tasks. Further, as existing methods cannot predict new node appearances, we need novel prediction methods for this sub-problem.

We propose AGATE (*A General framework for predicting Attributed Time-Evolving graphs*), a holistic, versatile prediction framework that leverages the interdependence between the tasks of predicting new nodes, lost nodes, appearing links, disappearing links, and node attributes from observed past graphs. AGATE comprises components for predicting the future graph and capturing task interdependence: (i) one to predict the size of an evolving graph, (ii) one to predict existing-node structure and attributes, as well as new nodes, and (iii) a *reuse mechanism* that reuses the results of (ii) to achieve predictions that capture the interdependence between sub-predictions. Our framework can use existing methods as components. We develop a novel prediction method, PROSER for predicting new node appearance with attributes.

We assess AGATE vs. the state of the art on three real-world time-evolving attributed graphs, showing that it benefits from allowing prediction tasks to affect each other, its reuse mechanism improves accuracy, and it effectively predicts new nodes with attributes. Our source code is available.[1]

We summarize our contributions as follows: (1) we study the problem of *holistic* prediction on a time-evolving attributed graph, including new node appearances; (2) we propose the AGATE framework for such predictions, which allows prediction tasks to affect each other; (3) we develop PROSER, a method for the prediction of new nodes with attributes, and (4) we put together best-of-breed methods and show that AGATE improves prediction accuracy and also accurately predicts new nodes with attributes. Further, we validate that tasks are interdependently affected by each other.

[1] https://github.com/yuya-s/AGATE/

2 Problem Statement

An undirected *attributed graph* is a triple $G = (V, E, X)$ where V is a finite set of *nodes*, $E \subseteq V \times V$ is a set of *edges*, and $X \in \mathbb{R}^{|V| \times d}$ is a set of node attributes; $X^{(v)}$ denotes d-dimensional attributes of node v. Each dimension in X may be categorical or numerical. We consider a *time-evolving* graph, where the number of nodes and links and values of node attributes changing over time, with d constant:

Definition 1 *A* time-evolving attributed graph *is a sequence of attributed graphs* $\langle G_1, \cdots, G_T \rangle$ *over discrete time steps, where* T *is the number of observed time steps and* $G_t := (V_t, E_t, X_t)$ *the attributed graph at time step* t*. It is known as discrete-time dynamic graph (DTDG).*

In a time-evolving graph, nodes and links appear and disappear dynamically. We discern three types of nodes at time step t: *existing* nodes V_t^e, *new* nodes V_t^n, and *lost* nodes V_t^l:

$$V_t^e = V_t \cap V_{t-1}, \quad V_t^n = V_t \backslash V_{t-1}, \quad V_t^l = V_{t-1} \backslash V_t.$$

Nodes in V_t^e exist at both time steps t and $t-1$, while those in V_t^n appear and those in V_t^l disappear at time step t. Likewise, we discern existing links E_t^e, links connected to new nodes E_t^n, appearing links E_t^a, and disappearing links E_t^d:

$$E_t^e = E_t \cap E_{t-1}, \qquad E_t^n = E_t \setminus (V_{t-1} \times V_{t-1})$$

$$E_t^a = (E_t \setminus E_{t-1}) \setminus E_t^n, \qquad E_t^d = E_{t-1} \setminus E_t$$

We define the problem as follows:

Problem 1 *Given a time-evolving attributed graph,* $\langle G_{T-L+1}, G_{T-L+2}, \ldots, G_T \rangle$*, including appearing and disappearing nodes and links and changing node attributes, the* **holistic time-evolving attributed graph prediction** *problem asks to predict the graph at time step* $T+1$*,* $G_{T+1} = (V_{T+1}, E_{T+1}, X_{T+1})$.

This problem calls to predict the whole of G_{T+1}, rather than one of its components.

Sub-prediction tasks: We define each sub-prediction task; existing works address some of these sub-prediction tasks independently.

Sub-prediction tasks on existing nodes aim to predict graph structure and attributes on existing nodes. Such tasks have been addressed in prior studies, such as STGCN (Yu et al., 2018) and DynGEM (Goyal et al., 2018). We have five sub-prediction tasks; node loss, link appearance, link

disappearance, and attribute values on existing nodes. As these tasks are intuitive, we describe them in Appendix.

Sub-prediction tasks on new nodes aim to predict new nodes with their links and attributes. Link prediction on new nodes has been addressed recently in (Hao et al., 2020), but there exist no studies for the prediction of new node attributes.

SubProblem 1 (**New node attributes**): *Given a time-evolving attributed graph* $\langle G_{T-L+1}, \ldots, G_T \rangle$ *across* L *time steps, this sub-prediction task is to predict new nodes* $\hat{V}^n_{T+1}$ *and their attributes* $\hat{X}^n_{T+1}$.

As there is no previous work on the prediction of new nodes with attributes, we define an evaluation measure for this task. We evaluate the similarity between the sets of attributes of predicted and real new nodes based on a perfect bipartite matching (Tanimoto et al., 1978):

Definition 2 (**Perfect matching**): *Given a complete bipartite graph* $K = (V, V', E)$ *where* $E = V \times V'$, *a* perfect matching M *in* K *is a set of pairwise non-adjacent edges that cover all vertices in* V.

We construct a complete birpartite graph K from predicted to real new nodes, and compute edge weights by arbitrary similarity functions: $K = (\hat{V}^n_{T+1}, V^n_{T+1}, \hat{V}^n_{T+1} \times V^n_{T+1})$.[2] Similarity functions between attributes of nodes $w(\cdot, \cdot)$ can be cosine similarity and Euclidean distance-based functions, for which we can select the types of node attributes (e.g., word embedding and user profiles).

Our measure is the maximum similarity:

$$\text{M-sim}(\hat{V}^n_{T+1}, V^n_{T+1}, \hat{X}^n_{T+1}, X^n_{T+1}) = \max_{M \in \mathcal{M}} \frac{1}{|M|} \sum_{(\hat{v}, v) \in M} w(\hat{X}^{(\hat{v})}_{T+1}, X^{(v)}_{T+1}) \quad (1)$$

where $\mathcal{M}$ is the set of perfect matchings for the given bipartite graph and $|M|$ is the number of pairs of matching in M. If the similarity is closer to one, attributes on predicted new nodes are similar to those on corresponding real new nodes. By this measure, we enforce that predicted attribute values properly match real ones, rather than merely reiterate values that appear frequently in new nodes of G_{T+1}. In general, the optimal M is hard to compute due to the numerous patterns of perfect matchings, so we use approximate methods such as a greedy method.

[2] If $|\hat{V}^n_{T+1}| \leq \alpha |V^n_{T+1}|$ (α is the minimum integer that satisfies this inequality), we duplicate all nodes in V^n_{T+1} at $\alpha - 1$ times.

Regarding predicting links to new nodes, prior work (Hao et al., 2020) assumes new nodes are *given* along with attributes. Contrariwise, we predict links to *predicted* nodes.

SubProblem 2 (**Links connected to new nodes**): *Given a time-evolving attributed graph* $\langle G_{T-L+1}, \ldots, G_T \rangle$ *across* L *time steps, new nodes* $\hat{V}^n_{T+1}$, *and corresponding attribute values* $\hat{X}^n_{T+1}$ *predicted by subproblem 1, this sub-prediction task is to predict any link between* v *and* $\hat{v}$, $(v, \hat{v}) \in V_T \times \hat{V}^n_{T+1}$.

These sub-prediction tasks on existing and new nodes may be interdependent; for example, attribute prediction affects link prediction and vice versa. The way tasks affect each other is unknown, and the interdependence between predictions regarding new and existing nodes is uncertain.

3 AGATE: Our framework

AGATE holistically predicts the future of time-evolving attributed graphs, exploiting the interdependence of sub-prediction tasks. Figure 2 illustrates its architecture, comprising three main components for graph size, independent, and reuse predictions; the independent prediction component is subdivided in parts for existing and new nodes. AGATE is modular; it can employ any existing method as a component in independent and reuse predictions; as no existing method predicts new nodes with attributes, we develop our own methods for this task. AGATE supports prediction in attributed graphs that are either partially or fully time-evolving, e.g., it can accommodate static attributes or no link disappearance.

Algorithm 1 shows the pseudo code of AGATE. AGATE first predicts the output graph size (lines 1–3). Then it first conducts each sub-prediction task independently to obtain a preliminary predicted graph at time step $T+1$ (lines 4–6), and updates the results of each such task by reusing the results of other tasks (lines 6–10). We explain each component.

3.1 Graph size prediction

We use a component to predict the sizes of V_{T+1} and E_{T+1}; we extract sequences of new nodes $\{V^n_{T-L+1}, \ldots, V^n_T\}$, lost nodes $\{V^l_{T-L+1}, \ldots, V^l_T\}$, appearing links $\{E^a_{T-L+1}, \ldots, E^a_T\}$, disappearing links $\{E^d_{T-L+1}, \ldots, E^d_T\}$, and links to new nodes $\{E^n_{T-L+1}, \ldots, E^n_T\}$ from input graphs $\{G_{T-L+1}, \ldots, G_T\}$, count the elements

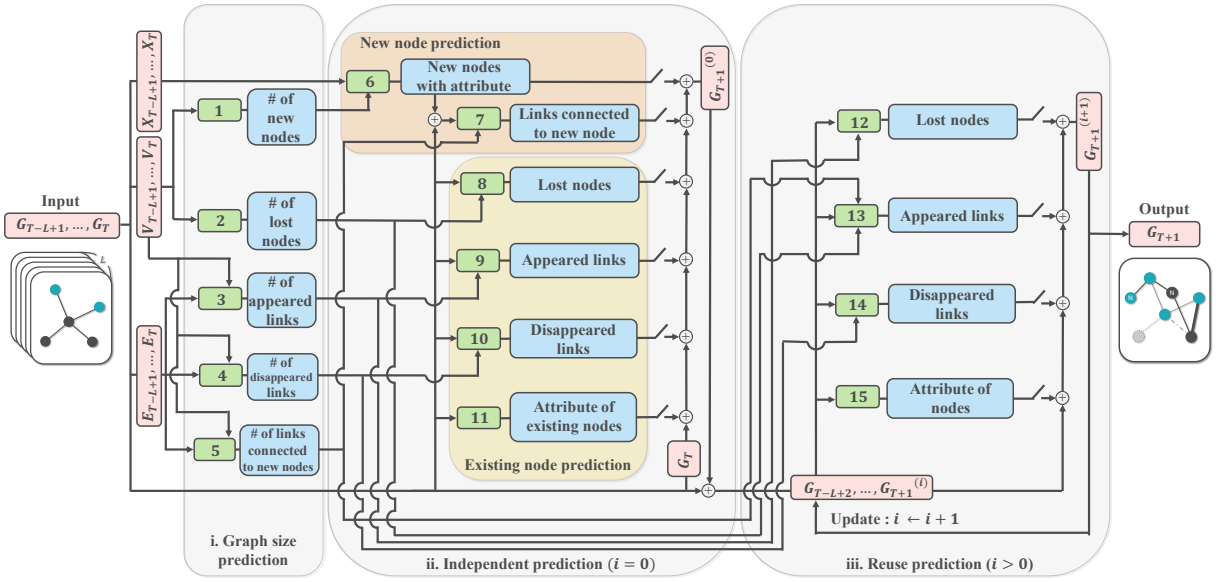


Figure 2: AGATE architecture; input is observed graphs (time $L - T + 1$ to T), and output is predicted graph (time $T + 1$). Pink, green, and blue boxes indicate graph data, learning models, and predicted results, respectively.

Algorithm 1: AGATE

input : $\langle G_{T-L+1}, G_{T-L+2}, \ldots, G_T \rangle$, a natural number I
output: $\hat{G}^{(I)}_{T+1}$
/* Graph size prediction */
Extract sequences of new nodes, lost nodes, appearing links, disappearing links, and links of new nodes from $\langle G_{T-L+1}, G_{T-L+2}, \ldots, G_T \rangle$;
Count and normalize them by dividing its maximum value;
Predicting $|V^n_{T+1}|, |V^l_{T+1}|, |E^n_{T+1}|, |E^a_{T+1}|, |E^d_{T+1}|$ from each sequence of normalized values;
/* Independent prediction */
Predicting $\hat{V}^n_{T+1}, \hat{V}^l_{T+1}, \hat{E}^n_{T+1}, \hat{E}^a_{T+1}, \hat{E}^d_{T+1}, \hat{X}_{T+1}, \hat{X}^n_{T+1}$ from $\langle G_{T-L+1}, G_{T-L+2}, \ldots, G_T \rangle$;
$\hat{V}^{(0)}_{T+1} \leftarrow V_T \cup \hat{V}^n_{T+1} \setminus \hat{V}^l_{T+1}$;
$\hat{E}^{(0)}_{T+1} \leftarrow E_T \cup \hat{E}^n_{T+1} \cup \hat{E}^a_{T+1} \setminus \hat{E}^d_{T+1}$;
/* Reuse prediction */
for $i = 1, \ldots, I$ **do**
  Predicting $\hat{V}^n_{T+1}, \hat{V}^l_{T+1}, \hat{E}^n_{T+1}, \hat{E}^a_{T+1}, \hat{E}^d_{T+1}, \hat{X}_{T+1}, \hat{X}^n_{T+1}$ from $\langle G_{T-L+2}, \ldots, G^{(i-1)}_{T+1} \rangle$;
  $\hat{V}^{(i)}_{T+1} \leftarrow V_T \cup \hat{V}^n_{T+1} \setminus \hat{V}^l_{T+1}$;
  $\hat{E}^{(i)}_{T+1} \leftarrow E_T \cup \hat{E}^n_{T+1} \cup \hat{E}^a_{T+1} \setminus \hat{E}^d_{T+1}$;
return $\hat{G}^{(I)}_{T+1}$;
end procedure

in each sequence, normalize them through division by its maximum value to avoid vanishing gradient, and train the model using the sequence of numbers of elements by time-series data prediction. In the case the methods used in the individual tasks can determine the numbers of nodes and links in the output graph, we do not need a separate prediction on graph size.

3.2 Independent prediction

This component makes initial predictions, which may then be used in reuse prediction; independent prediction is subdivided into parts regarding existing nodes and new nodes. After conducting all prediction tasks, the independent prediction builds ${G_{T+1}}^{(0)}$ to add/remove to/from G_T.

Existing node prediction: This component involves binary classification tasks regarding node loss, link appearance, link disappearance, as well as multi-class classification and regression tasks regarding the attributes of existing nodes. We obtain a probability for each target node/link from each classification model and derive the predicted results as a set of highest probability elements with cardinality given by the respective size prediction module. In node attribute prediction, we can select models to either predict each attribute value individually, or all values simultaneously. We use suitable models (e.g., LSTM (Hochreiter and Schmidhuber, 1997) and DynGEM (Goyal et al., 2018)) to predict categorical and numerical node attributes. To our knowledge, only EvolveGCN (Pareja et al., 2020) supports all tasks on existing nodes. However, EvolveGCN performs poorly compared to other models (see Table 1).

New node prediction. To the best of our knowledge, no previous work addressed new node prediction. Thus, we develop three simple baseline methods for this purpose (Random, FNN, and PointNet), which aim to determine the correspondence between the predicted and real new nodes by maximizing the matching similarity in Equation (1): **Random** outputs randomly sampled nodes

from V_T^n with attributes from X_T^n. **FNN** is a simple feedforward neural network, i.e., a multilayer perceptron, trained by constructing a bipartite graph from predicted new nodes $\hat{V}_{T+1}^n$ to real new nodes V_{T+1}^n, and learning to minimize the loss function: $\mathcal{L}_n = -\log \frac{\text{M-sim}(\hat{V}_{T+1}^n, V_{T+1}^n, \hat{X}_{T+1}^n, X_{T+1}^n)+1}{2}$. FNN's learning depends on the order of input nodes. **PointNet** is a deep learning framework that deals with order invariance (Qi et al., 2017), with input and loss function the same as those of FNN.

We utilize the predicted new node attributes to predict links to new nodes using DEAL (Hao et al., 2020), which handles link prediction for nodes having only attribute information, or any other similar method. We evaluate prediction accuracy by the correspondence between predicted and real nodes.

3.3 Reuse prediction

This component updates prediction results by exploiting task interdependence, aiming to capture the effect of each sub-prediction on others. Reuse prediction repeats the updates of G_{T+1} the given number I of times; it predicts lost nodes, appeared links, disappeared links, and attributes of nodes from $\langle G_{T-L+2}, \cdots, G_{T+1}{}^{(i)} \rangle$ for $0 \leq i \leq I-1$. We can use any model/method in the reuse module.

Reuse prediction follows the specifications of existing node prediction, while now each model reuses already predicted graph characteristics to update the results of all sub-prediction tasks. We may use those independent results that are fit; we study this matter in our experiments.

4 PROSER

We develop a novel method, *Probabilistic Selection Rule* (PROSER), to predict new node attributes.

PROSER aims to maximize the matching similarity in the prediction of new node attributes. In a preliminary analysis, we observed that most new nodes have similar attributes to those of new nodes at the previous time step. Thus, accuracy is relatively high when we randomly sample attributes of new nodes at the previous time step. We refine this process by selecting appropriate nodes to increase the matching similarity. By simple random sampling, matching similarity would be low when the similarity of sampled attributes to any attributes of new nodes at time step $T+1$ is low. PROSER avoids selecting such attributes by estimating the highest similarity between sampled attributes and those of new nodes. Besides, matching similarity would not increase by frequently sampling a few attributes that are highly similar to some real node attributes, as each match in a perfect matching is unique. Instead, we sample attributes X_T^n having a similar *distribution* to that of new node attributes in time $T+1$, X_{T+1}^n.

PROSER employs logistic regression to learn probabilities that similarities between sampled and actual new node attributes are higher than a threshold θ. Before model training, we compute θ to maximize matching similarity, as:

$$\theta = \arg\max_{\theta' \in \mathbb{R}} \sum_{G_i \in \mathcal{T}} \text{M-sim}(V_i^n(\theta'), V_{i+1}^n, X_i^n(\theta'), X_{i+1}^n) \quad (2)$$

where $\mathcal{T}$ denotes the set of training graphs, $V_i^n(\theta')$ is the set of sampled attributes whose similarity to those of the real nodes is higher than θ', and $X_i^n(\theta')$ is the set of their attributes. θ removes nodes that do not increase the matching similarity. To capture the distribution, we use a mean vector of X_T^n as a representative attribute of G_T. Our model computes the probability to maximize the matching similarity for future graphs leveraging this mean and the threshold.

In the inference phase, PROSER randomly samples attributes from the set of new node attributes at time T and calculates probabilities. If these are higher than θ, we add these attributes to prediction results, otherwise discard them. We repeat, until the number of predicted attributes reaches that of new nodes. Since we randomly select candidates of predicted attributes, predictions are diverse.

5 Experiments

We experimentally validate the performance of AGATE and analyze the performance gain due to the exploitation of task interdependence. All experiments are conducted on a Linux server with Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz.

Dataset. We use three datasets with diverse time-evolving features with linguistic node attributes: NBA[3], Reddit[4], and AMiner[5].

- **NBA**: Nodes represent NBA players, with attributes for points, team, age, and position. Two nodes are linked when the respective players are teammates. Nodes, links, and attributes change dynamically; there are 3,781

[3]https://www.basketball-reference.com
[4]http://snap.stanford.edu/data/soc-RedditHyperlinks.html
[5]https://www.aminer.cn/citation

unique nodes, 95,203 unique edges, 35 attributes, and 66 time steps.

- **Reddit**: Nodes stand for subreddits and links represent posts between them; if a subreddit has no posts in a time step, the corresponding node is removed at that time step. Node attributes are embeddings of the subreddits (Kumar et al., 2018). Nodes and links change dynamically, while node attributes are static. Reddit includes 7,756 unique nodes, 23,554 unique edges, 300 attributes, and 14 time steps.

- **AMiner** (Tang et al., 2008): Nodes stand for papers, links for citations, and attributes for research field weights, extracted by computing TF-IDF scores from paper titles. Nodes and links appear over time, but nodes do not disappear, while the links and attributes of existing nodes are static. Aminer includes 10,987 unique nodes, 6,708 unique edges, 9 attributes, and 24 time steps.

We divide time steps into training, validation, and test data with a 7:2:1 ratio; in NBA data, that is time steps 1–48, 49–60, and 61–66, respectively; in Reddit, time steps 1–11, 12–13, and 14; in AMiner, time steps 1–18, 19–22, and 23–24.

Compared methods. We apply different methods for comparison in each sub-prediction task, since not all methods are applicable to all tasks. When predicting existing node features, we use 9 methods grouped in four categories: simple, dynamic graph embeddings, static graph GNNs, and dynamic graph GNNs. Simple methods are: (1) *Baseline*, which outputs G_T as the predicted graph for G_{T+1}, (2) *Random*, which randomly samples the predicted targets, (3) *FNN*, and (4) *LSTM*. We use (5) *DynGEM* (Goyal et al., 2018) as a dynamic graph embedding and (6) *GCN* (Kipf and Welling, 2017) as a static graph GNN. As dynamic graph GNNs, we use (7) *STGCN* (Yu et al., 2018), (8) *EvolveGCN* (Pareja et al., 2020), which has two versions, EvolveGCN-H and EveloveGCN-O, and (9) *TGGNN* that we modified gated CNN to graph tasks (see Appendix in detail).

To predict new nodes with attributes, we use PROSER and the three baseline methods described in Section 3. To predict links connected to new nodes, we use cosine similarity (CS for short), FNN, and DEAL (Hao et al., 2020) based upon the results predicting new nodes with attributes. DEAL is the state-of-the-art method. AGATE repeatedly predicts G_{T+1}; we identify the best model for each sub-prediction task on the validation data, and go on reusing the best model's results.

Hyper-parameters. We run the model with at most 1,000 training iterations, 10–100 early stopping patience, batch size 1 or 2, and learning rate 0.01 with the Adam optimizer. On temporal graphs, we use 3, 3, and 5 as L for NBA, Reddit, and Aminer, respectively. We set the number I of repeated updates in reuse prediction to 3. We use cosine similarity as $w(\cdot, \cdot)$ in Eq. (1). We note that we tune hyper-parameters independently of models close to the input, and in reuse prediction models remain consistent. Thus, hyper-parameter tuning is not hard compared to common graph prediction tasks. Please see the appendix in detail.

The training time totally took about 500, 700, and 200 CPU hours on NBA, Reddit, and Aminer, respectively. Our framework can run in parallel.

5.1 Overall evaluation on Accuracy

We show results for independent prediction and AGATE, which exploits task interdependence.

Prediction on existing nodes: First, we show the results of attribute prediction on existing nodes, which correspond to, for example, future user profile prediction. Figure 3 shows results on the NBA data; MAE and RMSE on the prediction of NBA players' points (where lower is better) and AUC and Average Precision on the binary prediction of whether a player transfers to another team (where higher is better). Note that, Reddit and AMiner data do not change the attributes of nodes. LSTM and TGGNN perform well. However, other dynamic graph GNNs cannot predict player's points and fare worse than the baseline. AGATE enhances accuracy through reuse prediction. In player's point prediction, AGATE reuses the results of LSTM and all sub-prediction tasks; in team transfer prediction, it reuses the results of TGGNN and those regarding links connected to new nodes.

Second, Table 1 presents our AUC and average precision results on node loss, link appearance, and link disappearance predictions on NBA and Reddit; such results correspond to, for example, future connections between users. Note that, in AMiner, existing links and nodes do not change. We observe that, in independent prediction, TGGNN and LSTM often achieve the best performances. This result indicates that node attribute and/or topology

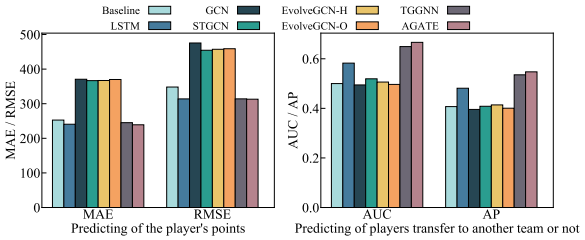


Figure 3: Existing node attribute prediction, NBA data.

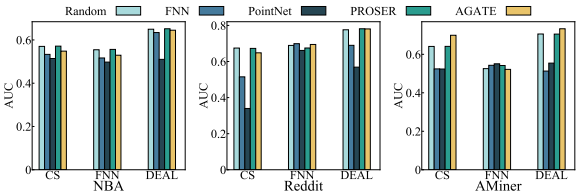


Figure 4: AUC on link prediction for new nodes.

Table 1: Node/link prediction on existing nodes; underlined: best results in independent pred.; bold: best results among both independent and reuse pred. ROC AUC and Average precision are multiplied by 100.

Methods	Node prediction (loss)				Link prediction (appearance)				Link prediction (disappearance)			
	ROC AUC		Average precision		ROC AUC		Average precision		ROC AUC		Average precision	
	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit
Baseline	50.00±0.00	50.00±0.00	18.71±0.00	51.10±0.00	50.00±0.00	50.00±0.00	1.37±0.00	0.05±0.00	50.00±0.00	50.00±0.00	41.36±0.00	39.29±0.00
Random	50.16±1.10	49.75±1.49	18.77±0.34	50.99±0.75	49.86±0.05	50.51±0.75	1.36±0.00	0.05±0.00	49.91±0.40	50.87±0.53	41.32±0.19	39.71±0.26
FNN	85.35±0.06	77.53±0.03	53.51±0.08	74.53±0.04	49.66±0.43	46.31±1.86	1.34±0.02	0.04±0.01	49.65±0.17	48.10±0.57	41.09±0.15	36.77±0.87
LSTM	85.44±0.31	82.55±0.10	53.52±0.88	81.69±0.14	49.83±0.23	**83.07±0.35**	1.34±0.01	**6.05±0.42**	49.53±0.61	**72.13±0.17**	40.52±0.46	56.21±0.37
DynGEM	48.67±1.24	67.53±0.15	18.71±0.41	63.67±0.20	49.66±0.42	44.61±1.56	1.36±0.01	0.04±0.00	50.28±0.68	46.81±1.12	41.47±0.93	35.70±0.61
GCN	56.84±0.97	75.14±0.15	23.40±1.49	71.98±0.40	49.67±0.41	82.10±0.94	1.34±0.02	2.05±0.14	50.79±0.34	63.97±0.90	41.95±1.15	48.77±1.30
STGCN	60.20±0.95	82.22±0.31	26.71±0.15	80.45±0.43	49.71±0.49	73.59±1.52	1.34±0.02	0.12±0.01	49.31±0.57	63.01±2.8	40.45±0.51	48.81±2.05
EvolveGCN-H	56.33±1.18	63.44±1.86	21.97±1.03	59.49±1.37	50.01±1.04	52.21±4.46	**1.38±0.05**	0.08±0.02	49.19±0.58	50.62±0.41	40.47±0.78	39.64±0.15
EvolveGCN-O	56.69±0.58	66.32±2.71	23.01±0.63	62.25±3.08	49.76±0.58	52.43±1.52	1.37±0.03	0.05±0.01	48.83±1.56	51.73±0.68	40.78±1.44	41.07±0.08
TGGNN	85.59±0.58	82.67±0.19	54.57±2.14	81.59±0.42	50.09±0.43	77.93±4.11	1.35±0.02	0.24±0.20	49.81±0.47	66.36±3.91	40.64±0.43	50.80±4.30
AGATE	**98.19±2.60**	**93.49±3.62**	**92.00±11.4**	**93.92±3.51**	**50.36±0.73**	81.27±1.89	**1.38±0.02**	0.45±0.26	**51.34±0.01**	71.99±1.61	**43.38±0.21**	**56.39±2.18**

Table 2: Results on new node attribute prediction; 'mean' indicates matching similarity; underlined fonts indicate the best results in independent prediction; bold fonts indicate best results among both independent and reuse predictions.

Methods	NBA				Reddit			
	mean	median	min	max	mean	median	min	max
Random	0.8143±0.00	0.8176±0.00	0.2055±0.00	**1.0000±0.00**	0.8317±0.00	0.8772±0.00	-0.2507±0.00	0.9947±0.00
FNN	0.7217±0.02	0.7650±0.02	0.2287±0.03	0.9004±0.02	0.7409±0.00	0.7876±0.01	-0.8723±0.01	0.9777±0.00
PointNet	0.6587±0.04	0.7075±0.03	0.2650±0.02	0.8337±0.04	0.6336±0.00	0.6016±0.00	-0.8697±0.00	0.9708±0.00
PROSER	0.8149±0.00	0.8164±0.00	0.2715±0.02	**1.0000±0.00**	0.8329±0.00	0.8780±0.00	**0.0286±0.00**	0.9947±0.00
AGATE	**0.8280±0.00**	**0.8416±0.00**	**0.3775±0.01**	0.9829±0.00	**0.8513±0.00**	**0.8904±0.00**	-0.0238±0.01	**0.9948±0.00**

changes affect the future graphs in these datasets. AGATE improves on node/link prediction accuracy via reuse prediction in most cases. In particular, the accuracy of node loss prediction increases significantly compared to that of independent prediction. Interestingly, the node loss prediction is highly assisted by new node prediction. We here note that link prediction on time-evolving graphs is often difficult, so their AUC and Average precision are very low (e.g., EvolveGCN (Pareja et al., 2020) reported similar values of MAP on different datasets). In the overall link prediction results, AGATE generally achieved the best performance and we reconfirm that prediction accuracy improves when we capture task interdependence.

Prediction on new nodes. Third, Table 2 shows results on attribute prediction on new nodes, which corresponds to, for example, the prediction of user profiles registered in the future. PROSER and Random outperform FNN and PointNet. PROSER and Random output node attributes appearing in the previous time step without modification, whereas FNN and PointNet modify them using neural networks. This result suggests that it is difficult to learn how the attributes of new nodes change, even while they are similar to those of new nodes at the previous time step. Further, PROSER surpasses Random, as it avoids sampling nodes that do not contribute to matching similarity, and thus cosine similarity significantly improves overall. AGATE improves the matching similarity by means of reuse prediction based on supervised learning on the correspondences between predicted and real new node attributes obtained from independent prediction. In reuse prediction, AGATE trains on new nodes as existing nodes, improving on matching similarity.

Finally, Figure 4 shows results on the prediction of links connected to new nodes, which corresponds to future connections among new and existing users. We predict links by the methods on the horizontal axis, each following upon the results on new node attribute prediction using the methods indicated in the bars. The results are generally good when using the previous results of Random, PROSER, and AGATE, which perform well in new node attribute prediction. Among link prediction methods themselves, DEAL outperforms others.

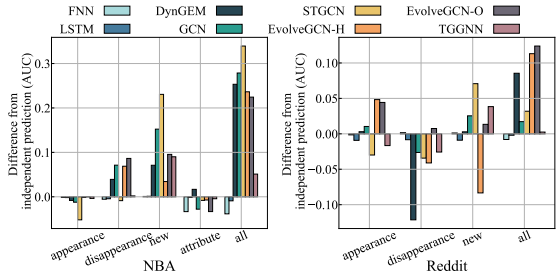


Figure 5: Reuse gain on node loss prediction.

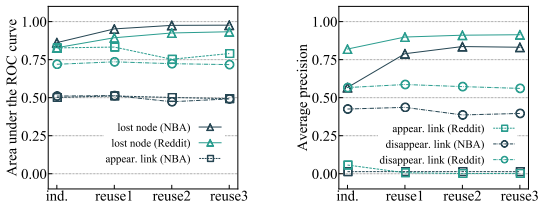


Figure 6: Impact of reuse iterations on node/link prediction, exist. nodes (legend split among two figures).

5.2 Task interdependence analysis

We examine how AGATE improves the accuracy of a sub-prediction task by reusing other tasks' results. Figure 5 shows the difference between the AUC of independent and reuse predictions on node loss. Task names on the horizontal axis indicate the reused task; 'all' denotes the reuse of all tasks. Each method and data reveals a different task interdependence. For example, the accuracy of STGCN improves when it uses results from all sub-prediction tasks with NBA data, but only slightly with Reddit data. The accuracy of TGGNN improves when it uses new node prediction results. Since TGGNN already achieves high accuracy independently, this gain is small. Still, these results confirm our hypothesis that leveraging task interdependence improves performance when it reuses proper sub-tasks. The tasks of new node attribute/link prediction, though not studied previously, exercise a big impact upon other tasks. As AGATE can select what tasks reuse in reuse prediction using validation data, we can remove reused tasks that have a negative impact on performance.

We examine the impact of reuse iterations, varying the number of repetitions i. Figure 6 shows the accuracy of lost node and appeared and disappeared link tasks vs. i; 'ind' indicates the accuracy of independent prediction. We reuse the results of all sub-prediction tasks. Accuracy increases perceptibly in most tasks from 'ind' to 'reuse1'. In node loss prediction, it keeps increasing with reuses; in other tasks, accuracy stays stable or falls after 'reuse1', due to error accumulation.

6 Related Work

We categorize graph prediction methods in four groups: (1) *static embedding* methods (e.g., (Tsitsulin et al., 2018, 2021)), (2) *dynamic embedding* methods (e.g., (Goyal et al., 2018, 2020)), (3) *static graph neural network* methods (e.g., (Kipf and Welling, 2017; Li et al., 2016; Velickovic et al., 2018; Hamilton et al., 2017; Zhang and Chen, 2018; Hao et al., 2020)), and (4) *dynamic graph neural network* methods (e.g., (Li et al., 2019; Xu et al., 2019; Li et al., 2018; Yu et al., 2018; Sankar et al., 2020; Pareja et al., 2020; Xu et al., 2021b; Gao and Ribeiro, 2022; Fu et al., 2022)). Embedding methods first derive an embedding independently of target tasks, and then build a model for each target task using the embedding as input; on the other hand, graph neural network methods directly build a model for each target task.

There are two types of graph prediction tasks: static and dynamic. Most methods assume that the underlying graph is static, and predict or reconstruct links or attributes. Yet real-world systems and data are dynamic. While it may be possible to apply static graph methods (Liben-Nowell and Kleinberg, 2007) to dynamic graphs ignoring temporal evolution, this approach is sub-optimal (Xu et al., 2020). Learning on dynamic graphs has been recently studied, yet limited to the *transductive* case involving *observed* elements (e.g., (Goyal et al., 2018; Yu et al., 2018).) Such approaches are insufficient for real-world settings in which a graph evolves with links and new nodes appearing any time. Some methods (Sankar et al., 2020; Pareja et al., 2020) support the inductive setting, involving nodes *unobserved* in training.

Each method makes its own assumptions about graph properties and inference, which may not apply to all prediction tasks.

Discrete vs Continuous. In this study, we assume discrete-time dynamic network (DTDN) as time-evolving graphs, while continuous-time dynamic network (CTDN) have been actively studied recently (Nguyen et al., 2018; Qu et al., 2020; Dai et al., 2017; Kumar et al., 2019; Trivedi et al., 2019; Liu et al., 2022; Wang et al., 2021). CTDNs are represented by a sequence of temporal graph updates instead of a sequence of temporal graphs. CTDNs are often used for event-based graphs (e.g., e-commerce and message communication) as they

assume each link temporally appears at a given time instead of persistent existences (e.g., friend relationships). They have different semantics and applications. Thus, neural network models for CTDNs do not directly apply to DTDN tasks and vice versa.

Other similar tasks and methods. Graph generation (Leskovec et al., 2005; Wu et al., 2020; You et al., 2018; Bojchevski et al., 2018) and completion (Shi and Weninger, 2018) cannot be applied in our problem; they aim to generate dynamic graph topologies without attributes, or fill out missing information in a static graph.

Summary. There are numerous studies on time-evolving attributed graphs, yet none studies holistic time-evolving attributed graph prediction. In addition, no prior study investigate task interdependence. Our work is the first to holistically predict a future time-evolving graph including new node appearance and analyze task interdependence.

7 Conclusion

We proposed AGATE, a framework for *holistic* prediction on a time-evolving attributed graph that exploits task interdependencem and uses a novel method, PROSER, for predictions about new nodes and their attributes. Our study showed that prediction accuracy largely improves by exploiting task interdependence.

8 Limitations

First, AGATE can handle DTDNs but does not handle CTDNs. In addition, it currently handles neither labeled nor directed edges. In the future, we intend to extend AGATE to cover a more comprehensive collection of graph types. Second, we assumed that most new nodes have similar attributes to those of new nodes at the previous time step, which were observed in our preliminary experiments. We plan to propose new strategies for the new node appearance task. Third, we tuned the hyper-parameters independently of models close to the input, so it may not be the optimal combination of methods. We plan to employ auto-ML techniques to enhance performance and mitigate the learning process.

Acknowledgement

This work was supported by JST PRESTO Grant Number JPMJPR21C5 and JSPS KAKENHI Grant Number JP20H00583, Japan.

References

Xuefeng Bai, Yulong Chen, Linfeng Song, and Yue Zhang. 2021. Semantic representation for dialogue modeling. In *ACL*, pages 4430–4445.

Trapit Bansal, Da-Cheng Juan, Sujith Ravi, and Andrew McCallum. 2019. A2n: Attending to neighbors for knowledge graph inference. In *ACL*, pages 4387–4392.

Nikolaos Bastas, Theodoros Semertzidis, Apostolos Axenopoulos, and Petros Daras. 2019. evolve2vec: Learning network representations using temporal unfolding. In *MultiMedia Modeling (MMM)*, pages 447–458.

Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. Netgan: Generating graphs via random walks. *arXiv*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv*.

Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2017. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv*.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *arXiv*.

Dongqi Fu, Liri Fang, Ross Maciejewski, Vetle I Torvik, and Jingrui He. 2022. Meta-learned metrics over multi-evolution temporal graphs. In *KDD*, pages 367–377.

Jianfei Gao and Bruno Ribeiro. 2022. On the equivalence between temporal and static equivariant graph representations. In *ICML*, pages 7052–7076.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *AAAI*, pages 3988–3995.

Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187(104816).

Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv*.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034.

Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. 2020. Inductive link prediction for nodes having only attribute information. In *IJCAI*, pages 1209–1215.

Mohammed Hasanuzzaman, Sabyasachi Kamila, Mandeep Kaur, Sriparna Saha, and Asif Ekbal. 2017. Temporal orientation of tweets for predicting income of users. In *ACL*, pages 659–665.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with GCNs. In *ICLR*.

Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2018. Community interaction and conflict on the web. In *WWW*, pages 933–943.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, page 1269–1278.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*, pages 177–187.

Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. 2019. Predicting path failure in time-evolving graphs. In *KDD*, pages 1279–1289.

Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence NNs. In *ICLR*.

David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *The JASIST*, 58(7):1019–1031.

Yunyu Liu, Jianzhu Ma, and Pan Li. 2022. Neural predicting higher-order patterns in temporal networks. In *WWW*, pages 1340–1351.

Paramita Mirza and Sara Tonelli. 2016. On the contribution of word embeddings to temporal relation classification. In *ACL*, pages 2818–2828.

Ishani Mondal, Yufang Hou, and Charles Jochim. 2021. End-to-end construction of nlp knowledge graph. In *ACL*, pages 1885–1895.

G. H. Nguyen, J. Boaz Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. 2018. Dynamic network embeddings: From random walks to temporal random walks. *The IEEE International Conference on Big Data*, pages 1085–1092.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolving graph convolutional networks for dynamic graphs. In *AAAI*, pages 5363–5370.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 77–85.

Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. 2020. Continuous-time link prediction via temporal dependent graph neural network. In *WWW*, pages 3026–3032.

E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv*.

Aravind Sankar, Y. Wu, L. Gou, Wei Zhang, and H. Yang. 2020. Dynamic graph representation learning via self-attention networks. In *WSDM*, pages 519–527.

Baoxu Shi and Tim Weninger. 2018. Open-world knowledge graph completion. In *AAAI*, pages 1957–1964.

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: Extraction and mining of academic social networks. In *KDD*, pages 990–998.

Steven L Tanimoto, Alon Itai, and Michael Rodeh. 1978. Some matching problems for bipartite graphs. *Journal of the ACM*, 25(4):517–525.

Phi Vu Tran. 2018. Multi-task graph autoencoders. *arXiv*.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *ICLR*.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: versatile graph embeddings from similarity measures. In *WWW*, pages 539–548.

Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan V. Oseledets, and Emmanuel Müller. 2021. FREDE: anytime graph embeddings. *Proc. VLDB Endow.*, 14(6):1102–1110.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive representation learning in temporal networks via causal anonymous walks. In *ICLR*.

Changmin Wu, Giannis Nikolentzos, and Michalis Vazirgiannis. 2020. Evonet: A neural network for predicting the evolution of dynamic graphs. In *ICANN*, pages 594–606.

Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural news recommendation with multi-head self-attention. In *EMNLP-IJCNLP*, pages 6389–6394.

Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2021. Graph neural networks for natural language processing: A survey. *arXiv*.

Qianqian Xie, Jimin Huang, Pan Du, and Min Peng. 2021. Graph relational topic model with higher-order graph attention auto-encoders. In *ACL*, pages 2604–2613.

Chengjin Xu, Yung-Yu Chen, Mojtaba Nayyeri, and Jens Lehmann. 2021a. Temporal knowledge graph completion using a linear temporal regularizer and multivector embeddings. In *NAACL*, pages 2569–2578.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *arXiv*.

Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. 2019. Spatio-temporal attentive RNN for node classification in temporal attributed graphs. In *IJCAI*, pages 3947–3953.

Dongkuan Xu, Junjie Liang, Wei Cheng, Hua Wei, Haifeng Chen, and Xiang Zhang. 2021b. Transformer-style relational reasoning with dynamic memory updating for temporal network modeling. In *AAAI*, pages 4546–4554.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, pages 5708–5717.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Graph-SAINT: Graph sampling based inductive learning method. In *ICLR*.

Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NeurIPS*, pages 5165–5175.

Yucheng Zhou, Xiubo Geng, Tao Shen, Jian Pei, Wenqiang Zhang, and Daxin Jiang. 2021. Modeling event-pair relations in external knowledge graphs for script reasoning. In *ACL*, pages 4586–4596.

Hao Zhu, Yankai Lin, Zhiyuan Liu, Jie Fu, Tat-Seng Chua, and Maosong Sun. 2019. Graph neural networks with generated parameters for relation extraction. In *ACL*, pages 1331–1339.

A Full Sub Problem Definition

We define each sub-prediction task on existing nodes. Note that existing works address some of these sub-prediction tasks independently.

This group of tasks aims to predict graph structure and attributes on existing nodes. Such tasks are frequently addressed in prior studies, such as STGCN (Yu et al., 2018) and DynGEM (Goyal et al., 2018):

SubProblem 3 (**Lost nodes**): *Given time-evolving attributed graph $G_{T-L+1}, \ldots, G_T$ across L time steps, this sub-prediction task is to predict the nodes V^l_{T+1} that are lost from V_T.*

SubProblem 4 (**Link appearance**): *Given time-evolving attributed graph $G_{T-L+1}, \ldots, G_T$ across L time steps, this sub-prediction task is to predict links appearing between any pair of nodes $u, v \in V_T$, which also exist at time $T+1$ and are not connected at time step T.*

SubProblem 5 (**Link disappearance**): *Given time-evolving attributed graph $G_{T-L+1}, \ldots, G_T$ across L time steps, this sub-prediction task is to predict links disappearing between any pair of nodes $u, v \in V_T$, which exist at time step $T+1$ and are connected at time step T.*

SubProblem 6 (**Attribute values on existing nodes**): *Given time-evolving attributed graph $G_{T-L+1}, \ldots, G_T$ across L time steps, this sub-prediction task is to predict a new attribute value on an existing node at time $T+1$.*

B TGGNN

We develop TGGNN by extending from time-directed convolution (Dauphin et al., 2016). We design TGGNN as a node representation learning model taking into account the nature of sub-prediction tasks: (1) sub-prediction tasks are inductive, since test data graphs may be different from those in the training data; (2) the topologies of graphs $G_{T-L+1}, \ldots, G_T$ may differ from each other, hence we need to handle topological

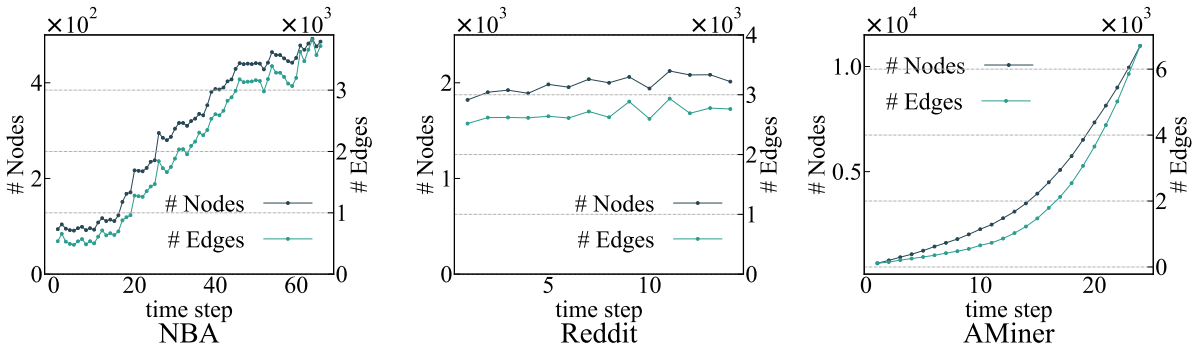


Figure 7: Number of nodes and edges per time step

changes as well as attribute changes. Thus, TGGNN supports inductive tasks on time-evolving attribute graphs whose nodes, links, and node attributes change dynamically.

While graph neural networks perform well on node representation learning, they assume fixed topologies, hence do not support time-evolving graphs. We extend graph neural networks to support induction on graphs with time-dependent topologies and attributes. TGGNN learns node embeddings in G_T that capture topological and attribute changes, informed by the local *dynamic* graph structure, rather than the global static graph structure; in particular, it aggregates embeddings within each node's k-hop neighborhood in all observed time steps $(G_{T-L+1}, \ldots, G_T)$ and captures node attributes by learning temporal attribute changes as graph annotations (Li et al., 2016) independently of graph structure.

TGGNN utilizes a suite of new gating mechanisms, ST-Gate, which improves the ST-Convolution Block (Yu et al., 2018) so as to handle topological changes. ST-Gate first applies time-directed convolution (Dauphin et al., 2016) on L graphs and then aggregates hidden states of nodes and their neighbors at each time step by GRU-like updates (Cho et al., 2014). In parallel, it applies time-directed convolution on graph annotations (Li et al., 2016), which are initially concatenations of node attributes over L time steps; then it aggregates the embeddings of the aggregated hidden states and the graph annotation and applies one more time-directed convolution (Dauphin et al., 2016) on the aggregated embedding.

The inputs to TGGNN are $\mathcal{A} \in \mathbb{R}^{n \times L \times d}$, $\mathcal{H} \in \mathbb{R}^{n \times L \times \hat{d}}$ and $\mathcal{E} \in \mathbb{R}^{n \times L \times n}$ where n is the number of nodes in the observed graphs and $\hat{d}$ the hidden state dimensions; $\mathcal{A}$ denotes a time-series of node attributes, $[X_{T-L+1}; \ldots; X_T]$ as graph annotations, where $[;]$ means concatenation; $\mathcal{H}$ and $\mathcal{E}$ denote the hidden states and a time-series adjacency matrix derived from $\{E_{T-L+1}, \ldots, E_T\}$, respectively. We initialize hidden state $\mathcal{H}^{(0)}$ using $\mathcal{A}^{(0)}$; we may pad with extra 0s to allow hidden states that are larger than the annotation size.

Let $\mathcal{H}^{(N)} \in \mathbb{R}^{n \times (L-2N(K-1)) \times \hat{d}}$ be the hidden state output of the Nth ST-Gate, where K denotes the kernel size of time-directed convolution operators in the ST-Gate. The final node embedding output $\mathcal{H}^{(\text{out})}$ of TGGNN is calculated as:

$$\mathcal{H}^{(\text{out})} = \phi(\{(\mathcal{H}^{(N)} * \Gamma_f + b_f) \otimes \sigma(\mathcal{H}^{(N)} * \Gamma_g + b_g)\} W + b)$$

where Γ_f, $\Gamma_g \in \mathbb{R}^{(L-2N(K-1)) \times \hat{d} \times \hat{d}}$ are 1-dimensional convolution operators for the final embedding whose kernel size is $L - 2N(K-1)$ and $b_f, b_g \in \mathbb{R}^{\hat{d}}$ are correspondence biases; $\otimes$ denotes the element-wise multiplication. $W \in \mathbb{R}^{\hat{d} \times D}, b \in \mathbb{R}^D$ are learnable weights and bias for a linear transformation and ϕ denotes any of the activation function suited for the prediction task; D is the size of predicts (e.g., $D = n$ on link prediction). Since the TGGNN output is computed from temporal graph structures and node attributes, it captures the evolution of a dynamic attributed graph.

C Dataset detail

Figure 7 shows the numbers of nodes and links for each data set in each time step. In NBA and AMiner, the numbers of nodes and edge increase, whereas in Reddit, they are quite stable.

D Additional Experimental Study

D.1 Results for prediction of graph size

Table 3 shows a mean absolute error of baseline and LSTM on the graph size prediction. LSTM generally performs better than baseline; whereas,

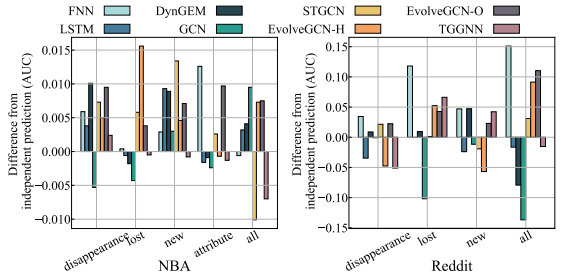


Figure 8: Reuse gain on link appearance prediction (AUC).

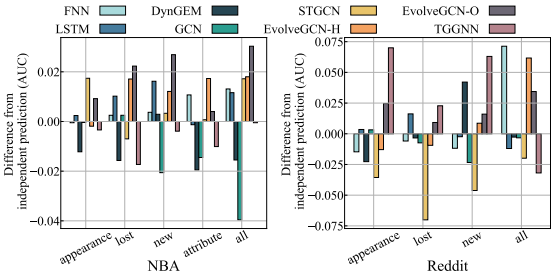


Figure 9: Reuse gain on link disappearance prediction (AUC).

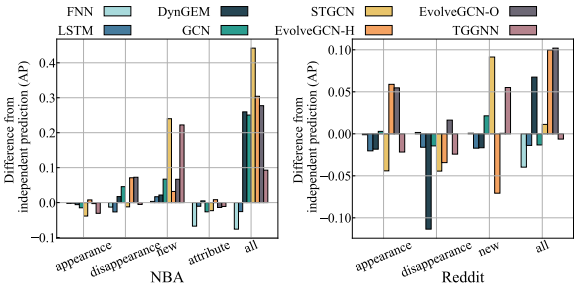


Figure 10: Reuse gain on node loss prediction (Average precision)

baseline performs well when the size of graphs is relatively stable.

Table 3: Mean absolute error of graph statistics prediction. From left to right, the # of new nodes, the # of lost nodes, the # of appeared links, the # of disappeared links and the # of new links, respectively.

	NBA	Reddit	AMiner
Baseline	12/10/122/115/189	43/31/2/10/38	512/-/-/-/5764
LSTM	11/ 8/108/101/180	33/67/3/42/11	618/-/-/-/3266

D.2 New node prediction (cont'd)

Table 4 shows results on the new node prediction in three datasets. We can see that PROSER works well on all the datasets.

D.3 Task interdependence analysis (cont'd)

Figures 8–12 show the difference between AUC/average prediction of independent and reuse predictions on node loss, link appearance, and link disappearance predictions, respectively.

These results are similar tendencies to Figures 8–12, respectively. The results of new node prediction has also large impact in average precision as same as AUC. The difference between the results of average precision and AUC is that the average precision of LSTM in the link appearance prediction on Reddit significantly decreases. This indicates that the reuse prediction in LSTM for the link appearance prediction does not work well on Reddit. While, we can see that the reuse prediction often performs well in sub-prediction tasks in all datasets.

Figure 13 shows the impact of the number of iterations on attribute/link on new nodes and attribute prediction on existing nodes. From these result, these predictions do not improve even if we increase the number of iterations much.

D.4 Scalability

Figure 14 shows inference times on real graphs. All methods take less than 10 seconds; TGGNN needs slightly larger time than others, while PROSER efficiently predicts attributes of new nodes. Figure 15 shows the inference time on synthetic graphs generated by the BA model, varying with the number of nodes. It is worth noting that the number of nodes in the experimental studies of prior works is less than 100,000. The inference time grows linearly with the number of nodes. All methods we use for lost and new node prediction are scalable to large graphs. Contrariwise, predicting link appearances and disappearances is inherently hard to scale for all methods, as it requires prediction on all vertex pairs.

In TGGNN, its architectural hyper parameters have been optimized on the Reddit dataset and are then reused for NBA. The time-directed convolution layers of TGGNN consists of filters whose kernel size $K = 2$ (same as STGCN). The number of aggregate updates by the propagation model is five, which is the same number of times as the pytorch implementation of GGNN for bAbI task 15[6]. ST-Gate is repeated one time. The hidden state's size $\hat{d}$ is the same as attribute dimension.

E Hyper parameters

We describe hyper parameter tuning in each method. Please see our source codes in detail. In

[6]https://github.com/chingyaoc/ggnn.pytorch

Table 4: Results on new node attribute prediction; 'mean' indicates the matching similarity; underlined fonts indicate the best results in independent prediction; bold fonts indicate the best results among both independent and reuse predictions.

Methods	NBA				Reddit				AMiner			
	mean	median	min	max	mean	median	min	max	mean	median	min	max
Random	0.8143±0.00	0.8176±0.00	0.2055±0.00	**1.0000±0.00**	0.8317±0.00	0.8772±0.00	-0.2507±0.00	0.9947±0.00	0.9974±0.00	0.9983±0.00	0.9565±0.00	**0.9998±0.00**
FNN	0.7217±0.02	0.7650±0.02	0.2287±0.03	0.9004±0.02	0.7409±0.00	0.7876±0.01	-0.8723±0.01	0.9777±0.00	0.9904±0.00	0.9912±0.00	0.9779±0.00	0.9993±0.00
PointNet	0.6587±0.04	0.7075±0.03	0.2650±0.02	0.8337±0.04	0.6336±0.00	0.6016±0.00	-0.8697±0.00	0.9708±0.00	0.9915±0.00	0.9924±0.00	**0.9792±0.00**	0.9995±0.00
PROSER	0.8149±0.00	0.8164±0.00	0.2715±0.02	**1.0000±0.00**	0.8329±0.00	0.8780±0.00	**0.0286±0.00**	0.9947±0.00	**0.9975±0.00**	**0.9984±0.00**	0.9553±0.00	**0.9998±0.00**
AGATE	**0.8280±0.00**	**0.8416±0.00**	**0.3775±0.01**	0.9829±0.00	**0.8513±0.00**	**0.8904±0.00**	-0.0238±0.01	**0.9948±0.00**	0.9971±0.00	0.9979±0.00	0.9635±0.00	**0.9998±0.00**

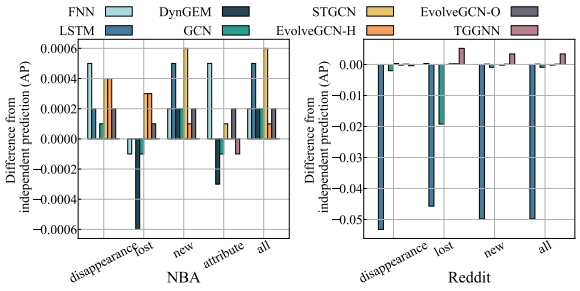


Figure 11: Reuse gain on link appearance prediction (Average precision)

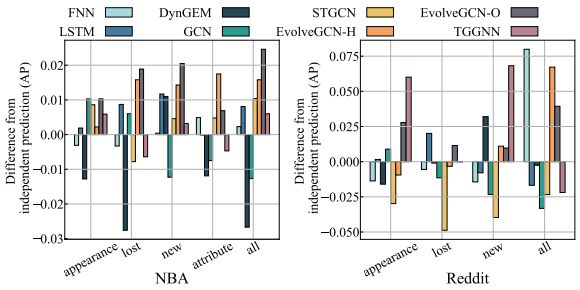
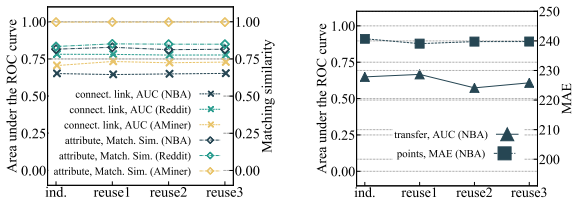


Figure 12: Reuse gain on link disappearance prediction (Average precision)



(a) attr./link prediction, new nodes

(b) attribute prediction, exist. nodes

Figure 13: Impact of reuse iterations.

all experiments, we run the model with 100–1000 training iterations, 10–100 early stopping patience, 1 or 2 batch sizes, and a learning rate of 0.01 with Adam as the optimizer. For methods on temporal graphs, we use 3, 3, and 5 as L for NBA, Reddit, and Aminer, respectively.

For all models, the hidden state's size is the same as the node attribute dimension, only for LSTM is three times the dimension of node attributes. The output layer is a fully-connected layer for all models, and its activation function is a softmax for multi-class classification, a sigmoid for binary classification, and none for regression.

The architecture and hyper parameters in DynGEM[7], DEAL[8], and EvolveGCN[9] are the same to the author's implementation. The embedding size of DynGEM is the same as the node attribute dimension. Hyper parameters in PointNet generally follow the setting on original PointNet but we modify the number of units depending on input and output sizes. FNN, LSTM, and GCN have a single layer.

F Selected models of AGATE

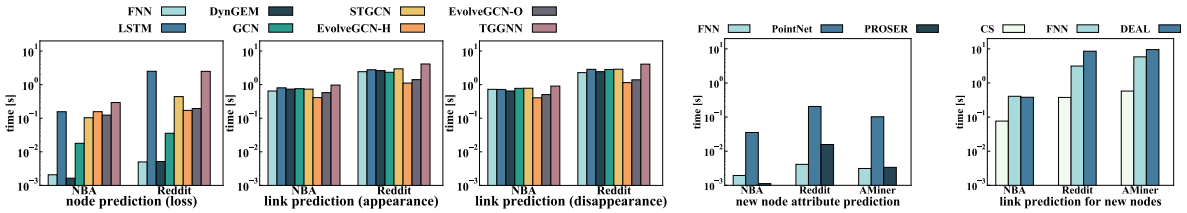
We summarize the models that we use for evaluating AGATE. Tables 5–7 shows models that AGATE uses in NBA, Reddit, and AMiner. We select models that achieve the best performance of validation in each sub-prediction task. Each number is corresponding to the task number in Figure 2. For the task numbers 1–5, we use LSTM in all datasets. The missing numbers (e.g., 11 in Table 6) are regardless tasks to the datasets (e.g., attribute prediction in Reddit). Each model has a small number of parameters because its model architecture is not complicated.

G Related work (cont'd)

Table 8 outlines a summary of existing methods, categorized in four groups: (1) *static embedding* methods (Tsitsulin et al., 2018), (2) *dynamic embedding* methods (Goyal et al., 2018, 2020), (3) *static graph neural network* methods (Kipf and Welling, 2017; Li et al., 2016; Velickovic et al., 2018; Hamilton et al., 2017; Zhang and Chen, 2018; Hao et al., 2020), and (4) *dynamic graph neural network* methods (Li et al., 2019; Xu et al., 2019; Li et al., 2018; Yu et al., 2018; Sankar et al., 2020; Pareja et al., 2020; Xu et al., 2021b).

Among existing methods, only EvolveGCN (Pareja et al., 2020) predicts

[7]https://github.com/palash1992/DynamicGEM
[8]https://github.com/working-yuhao/DEAL
[9]https://github.com/IBM/EvolveGCN



(a) Existing node prediction (node loss, link appearance, link disappearance)

(b) New node (attributes and links of new nodes)

Figure 14: Inference time on real graphs.

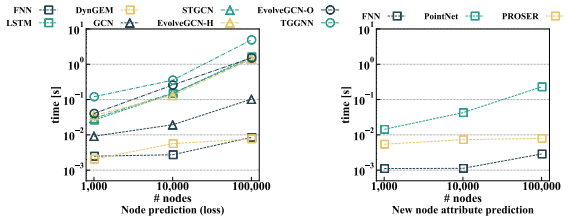


Figure 15: Inference time on synthetic graphs.

changes of nodes, links, and node attributes, hence supports time-evolving attributed graphs, notwithstanding new node appearances. TGGNN supports what EvolveGCN supports, yet differs in two ways. First, EvolveGCN uses graph convolutional networks (GCNs) (Kipf and Welling, 2017); upon a significant change in graph structure, GCNs can neither recognize that a node may have a different aggregation vector, nor distinguish nodes having the same vector. EvolveGCN recursively updates GCN weights using a recurrent neural network (e.g., GRU and LSTM), yet retains the GCN shortcomings. Contrariwise, TGGNN learns node embeddings by aggregating hidden node vectors within k hops at each time step, while computing aggregation weights for hidden vectors. Second, EvolveGCN incorporates node attributes in hidden vectors at each time step, whereas TGGNN captures temporal changes in node attributes via graph annotations handled separately from graph structure, enhancing attribute prediction accuracy. In effect, TGGNN handles graph structure and node attribute changes more effectively than EvolveGCN.

EvoNet (Wu et al., 2020) and open-world knowledge-graph (OWKG) completion (Shi and Weninger, 2018) tackle problems different from ours. EvoNet generates a future graph without correspondences between nodes at different time steps. OWKG completion predicts unseen nodes from the text data of existing ones. Contrariwise, we track evolving nodes without specifically relying on additional text data; to our knowledge, no existing method supports such a task.

Besides, some methods assume *continuous-time* dynamic graphs that are continuously modified (Rossi et al., 2020; Xu et al., 2020; Kumar et al., 2019; Nguyen et al., 2018; Bastas et al., 2019; Trivedi et al., 2019); however, methods for continuous-time graphs do not support discrete-time graphs, and vice versa.

Discrete vs Continuous. In this study, we assume discrete-time dynamic network (DTDN) as time-evolving graphs, while continuous-time dynamic network (CTDN) have been actively studied recently (Nguyen et al., 2018; Qu et al., 2020; Dai et al., 2017; Kumar et al., 2019; Trivedi et al., 2019; Liu et al., 2022; Wang et al., 2021). CTDNs are represented by a sequence of temporal graph updates instead of a sequence of temporal graphs. CTDNs are often used for event-based graphs (e.g., e-commerce and message communication) as they assume each link temporally appears at a given time instead of persistent existences (e.g., friend relationships). They have different semantics and applications. For example, current models for CTDNs do not handle node attribute changes and disappearances of persistent edges, and models for DTDNs do not directly handle continuous time information on edges. Thus, neural network models for CTDNs do not directly apply to DTDN tasks and vice versa. It is possible to use CTDN methods for link appearance tasks on DTDNs after DTDNs change to CTDNs (with ignoring either temporary or persistent edges) unless datasets have node attribute changes and disappearances of links. In the holistic time-evolving attributed graph prediction problem, graphs have node attribute changes and disappeared links, so CTDN methods are inapplicable to the problem.

Other similar tasks and methods. Tasks such as graph generation (Leskovec et al., 2005; Wu et al., 2020; You et al., 2018; Bojchevski et al., 2018) and completion (Shi and Weninger, 2018) cannot be

applied in our study, as those problem definitions are different from our problem; rather than predicting the evolution of a time-evolving graph, they aim to generate learned dynamic graph topologies without considering attributes instead of predicting the future graph, or fill out missing information in a static graph. Multi-task learning (Tran, 2018) can be incorporated into AGATE. It is unsure what time-evolving attributed graph tasks should be addressed by a single model yet.

Summary. There are numerous studies on time-evolving attributed graphs, yet no one studies the holistic time-evolving attributed graph prediction problem. In addition, there are no studies that investigate task interdependence. Our work is the first to holistically predict a future time-evolving graph including new node appearance and analyze task interdependence.

Table 5: Methods used in AGATE of NBA Dataset

Independent prediction										
Task number	6	7	8	9	10	11				
						team (transfer or not)	team (which team)	position	points	age
Method	PROSER	DEAL	TGGNN	GCN	DynGEM	TGGNN	GCN	Baseline	LSTM	TGGNN

Reuse prediction										
Task number	12	13		14	15					
		existing	new		team (transfer or not)	team (which team)	position	points	age	new
Method	TGGNN	FNN	DEAL	DynGEM	TGGNN	GCN	Baseline	LSTM	Baseline	LSTM

Table 6: Methods used in AGATE of Reddit Dataset

	Independent prediction					Reuse prediction				
Task number	6	7	8	9	10	12	13		14	15
							existing	new		
Method	PROSER	DEAL	TGGNN	LSTM	LSTM	STGCN	TGGNN	DEAL	LSTM	FNN

Table 7: Methods used in AGATE of AMiner Dataset

	Independent prediction		Reuse prediction	
Task number	6	7	13	15
Method	PROSER	DEAL	DEAL	FNN

Table 8: A summary of existing works and their characteristics.

	methods	graph property			inference		prediction target				
		attribute	temporal	notation	transductive	inductive	attribute	appearance link	disappearance link	lost node	new node
embedding	VERSE (Tsitsulin et al., 2018)	✘	static	$G = (V, E)$	✔	✘	✘	✔	✔	✔	✘
	G2G (Bojchevski and Günnemann, 2018)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GraphSAINT (Zeng et al., 2020)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	DynGEM (Goyal et al., 2018)	✘	dynamic	$G_t = (V_t, E_t)$	✔	✘	✘	✔	✔	✔	✘
	dyngraph2vec (Goyal et al., 2020)	✘	dynamic	$G_t = (V_t, E_t)$	✔	✘	✘	✔	✔	✔	✘
graph neural network	GCN (Kipf and Welling, 2017)	✔	static	$G = (V, E, X)$	✔	✘	✘	✔	✔	✔	✘
	GGNN (Li et al., 2016)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GAT (Velickovic et al., 2018)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GraphSAGE (Hamilton et al., 2017)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	SEAL (Zhang and Chen, 2018)	✔	static	$G = (V, E, X)$	✔	✘	✘	✔	✘	✘	✘
	DEAL (Hao et al., 2020)	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✘	✘	✘
	SAPE (Li et al., 2019)	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✘	✘	✔	✘	✘
	STAR (Xu et al., 2019)	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✔	✔	✔	✔	✘
	DCRNN (Li et al., 2018)	✔	dynamic	$G_t = (V, E, X_t)$	✔	✘	✔	✔	✔	✔	✘
	STGCN (Yu et al., 2018)	✔	dynamic	$G_t = (V, E, X_t)$	✔	✘	✔	✔	✔	✔	✘
	TRRN (Xu et al., 2021b)	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✔	✔	✔	✔	✘
	DySAT (Sankar et al., 2020)	✘	dynamic	$G_t = (V_t, E_t)$	✔	✔	✘	✔	✔	✔	✘
	DANE (Li et al., 2017)	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✘	✔	✘	✘	✔	✘
	EvolveGCN (Pareja et al., 2020)	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✘
ours	TGGNN	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✘
	PROSER	✔	dynamic	$G_t = (V_t, X_t)$	✔	✔	✔	✘	✘	✘	✔
	AGATE	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✔