CLGT: A Graph Transformer for Student Performance Prediction in Collaborative Learning

Tianhao Peng[1,2], Yu Liang[3]*, Wenjun Wu[1,4], Jian Ren[1,2], Zhao Pengrui[1,2], Yanjun Pu[1,2]

[1]State Key Laboratory of Software Development Environment, Beihang University
[2]School of Computer Science and Engineering, Beihang University
[3]Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology
[4]Institute of Artificial Intelligence, Beihang University
{pengtianhao,wwj09315,renjian,zhaopengrui,buaapyj}@buaa.edu.cn, yuliang@bjut.edu.cn

Abstract

Modeling and predicting the performance of students in collaborative learning paradigms is an important task. Most of the research presented in literature regarding collaborative learning focuses on the discussion forums and social learning networks. There are only a few works that investigate how students interact with each other in team projects and how such interactions affect their academic performance. In order to bridge this gap, we choose a software engineering course as the study subject. The students who participate in a software engineering course are required to team up and complete a software project together. In this work, we construct an interaction graph based on the activities of students grouped in various teams. Based on this student interaction graph, we present an extended graph transformer framework for collaborative learning (CLGT) for evaluating and predicting the performance of students. Moreover, the proposed CLGT contains an interpretation module that explains the prediction results and visualizes the student interaction patterns. The experimental results confirm that the proposed CLGT outperforms the baseline models in terms of performing predictions based on the real-world datasets. Moreover, the proposed CLGT differentiates the students with poor performance in the collaborative learning paradigm and gives teachers early warnings, so that appropriate assistance can be provided.

Introduction

In modern world, the collaborative learning (CL) is a prevalent learning method. In a CL environment, the students of different calibers and intellectual levels work together in teams and engage in a common task (Laal and Laal 2012). The process of modeling and predicting the performance of students is an important task in CL. Software engineering (SE) is an educational program that combines theory and practice. The practical teaching forms the core of SE program. As the practical teaching includes considerable CL, we choose SE as the subject in this work.

The SE courses often incorporate team-based collaborative projects that are designed to mimic the professional software development tasks. These tasks are often more complex as compared to individual coding exercises implemented in other programming classes. In these tasks, each student in the study group needs to participate and work together on a medium scale software project following the classic life cycle of SE, following requirement specifications, architectural design, and coding. The final score of each student depends on their contribution in the project and the quality of the final software artifacts delivered by the team. In order to accurately and efficiently evaluate the academic performance of each student in a team project, the instructor needs an intelligent educational tool for analyzing the process of every project by mining the students' behavior data, especially the interaction behaviors.

However, it remains a challenge for the instructors in the field of SE to score the team members by considering their individual contributions in the projects. This is mainly due to the difficulty in capturing and quantifying the amount of individual effort and work (Parizi, Spoletini, and Singh 2018). A typical solution is to adopt either peer evaluation or assessing the team as a whole. However, both approaches can be inaccurate and prejudiced, resulting in an unfair assessment. Currently, with the introduction of online CL platforms, such as GitHub and GitLab, it is possible to collect the behavioral data of students, such as source code submissions and review postings. The availability of online behavior data enables the researchers to apply the machine-learning algorithms for CL (Olsen, Aleven, and Rummel 2015; Yee-King, Grimalt-Reynes, and d'Inverno 2016; Ekuban et al. 2020; Yee-King and d'Inverno 2016). However, these methods are mostly developed based on the classic machine learning models that choose the activity frequency of students as input features. Such coarse-granular features are unable to capture the rich spatial and temporal information embedded in the interaction of different team members, thus only presenting a limited value to the instructors.

The CL data contains interactive activities with the graph structured features, where the nodes represent the students and the edges between two students represent the interactions between them. There are often different kinds of interactions among the students, such as submitting documents and source codes, as well as posting reviews and issues. Each kind of edge has multiple attributes and has

---

*Corresponding author

a potential impact on the final grade of students. Thus, the graph-structured interactive behaviors are formulated as a weighted heterogeneous graph, which contains not only multiple types of edges, but also the additional information features regarding each type of edge. In order to effectively exploit the rich edge feature information, in this work, we propose a method for processing the CL data and an extended Graph Transformer framework for Collaborative Learning (CLGT). We implement the proposed model and compare its performance with Ada Boost (Freund and Schapire 1997), relational graph convolutional network (R-GCN) (Schlichtkrull et al. 2018), graph transformer network (GTN) (Yun et al. 2019) and Graph Trans (Dwivedi and Bresson 2020) based on the same dataset for predicting the performance of the students. The experimental results show that the proposed model outperforms four other models.

In addition to performance prediction, an instructor also needs intelligent tutoring tools to infer the latent impact of student interactions on the process of collaborative software development. For instance, the online questioning-answering and code submissions are recorded on the CL platform, however, the implicit activities, such as getting inspiration from reading other's documents and codes is difficult to measure. Moreover, it is noteworthy that most of the deep learning models are inherently designed as black boxes, which are not able to provide good explanations regarding their reasoning mechanisms. Both factors, including the unobserved activities and the nature of the model makes it difficult to present the explainable prediction results to the instructors of SE courses. Therefore, in this work, we build an interpretation model inspired by PGM-Explainer (Vu and Thai 2020) to explain the prediction results of the proposed CLGT model. Based on the results of the interpretation model, we are able to analyze whether the students influence their peers during the learning process and to what extent.

The major contributions of this work are summarized below.

1. In this work, we extend the graph transformer model to develop the CLGT model for accurately predicting and assessing the performance of students in a project-based CL environment.

2. We build an interpretation module for explaining that which part of the graph structure data is responsible for the prediction results of the proposed CLGT framework.

3. To facilitate further research, we publish our dataset of students CL activities generated from an online project-based SE course. The relevant code and dataset are publicly available in the code repository [1].

Related Works

There are some works presented in literature for measuring the contribution of each individual in the projects related to software engineering courses. The log data and activity records available in version control systems are used in collaborative software development (Arcelli Fontana and Raibulet 2017; Loeliger and McCullough 2012; Clifton,

Kaczmarczyk, and Mrozek 2007; Reid and Wilson 2005), thus enabling the researchers to gather data regarding the computing contribution metrics. Most of the methods presented in literature (Parizi, Spoletini, and Singh 2018; Buffardi 2020) utilize classic statistics and regression algorithms for assessing the contribution of a member based on the activity ratio among the metrics of team members, including the number of commits, pull requests, and the lines of code changed by this member. However, these methods fail to consider and reflect the interactive activities that occur during the software development process. Fu et al. (Fu et al. 2021) constructed a developer interaction graph based on the data obtained from GitHub (a git repository manager) for identifying the key developers bridging messages across the GitHub communities. This method only studies the structural properties of large-scale interaction networks in the professional software development communities without any attention to the topics of small- or medium-sized student interaction networks in collaborative learning. To the best of our knowledge, the proposed work is a first attempt that focuses on predicting the performance of students based on their interactive graph in software engineering courses.

It is noteworthy that most of the research efforts regarding the learning outcome prediction have focused on modeling the performance of students based on individual learning process (Corbett and Anderson 1994; Pavlik Jr, Cen, and Koedinger 2009; Piech et al. 2015; Pu et al. 2019; Liang et al. 2022). However, the individual analytic models cannot be applied directly in collaborative learning environments since they do not consider the influence of the students within a team or between different teams. There are few works that make an effort to extend the models designed specifically for individual learning to team-based learning scenarios. Olsen et al. (Olsen, Aleven, and Rummel 2015) extended the additive factors model (Pavlik Jr, Cen, and Koedinger 2009) by using cooperative features to predict the performance of students in a collaborative learning environment. Yee-King et al. (Yee-King, Grimalt-Reynes, and d'Inverno 2016) employed K-nearest neighbour (KNN) model to make predictions on the grades of students in a collaborative learning environment. Ekuban et al. (Ekuban et al. 2020) evaluated multiple machine learning models, including decision tree, random forest, extra trees, Ada Boost, and gradient boosting to predict the performance of students working in a team based on student interactions. However, the models discussed above are shallow and have a limited performance in terms of handling large scale datasets and modeling complex graph-structured data based on spatio-temporal features.

The graph neural network (GNN) is a suitable framework for modeling complex interactions. The applications of GNN are widespread, ranging from image classification and video processing to speech recognition and natural language processing (Wu et al. 2020). However, there are few research works that use GNN for modeling student interactions in a collaborative learning environment. The R-GCN (Schlichtkrull et al. 2018) proposes relation-specific transformation in the message passing steps to deal with various relations in edges. GTN (Yun et al. 2019) pro-

---

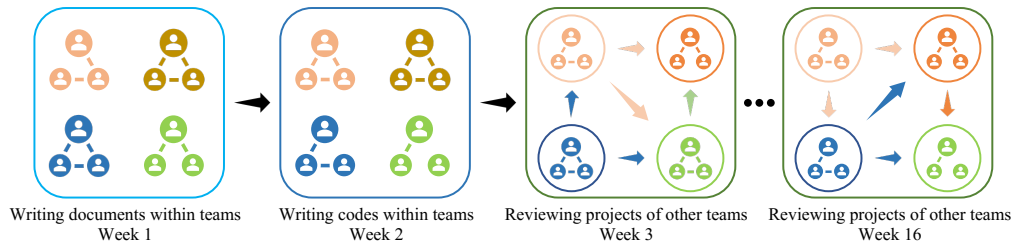[1]https://github.com/Tianhao-Peng/CLGT/

Figure 1: The flowchart of the three cycle phases of the SE course. During the documentation and coding phase, the students interact with other members of their teams, and during the reviewing phase, the integration occurs between the teams as well. As shown in the figure, the interaction graphs within and between the teams may change on weekly basis.

poses a novel graph transformer layer to identify the connections between unconnected nodes that are closely related. Graph Trans (Dwivedi and Bresson 2020) proposes an elegant positional encoding strategy based on the eigenvectors of the graph Laplacian to apply attention to neighbouring nodes. However, these models cannot fully and efficiently utilize the extra edge information of weighted heterogeneous graphs in some datasets which contain not only multiple types of edges, but also additional information features of each type of edge.

Moreover, there has been little work devoted to explaining collaborative learning activities with explainable models. Without reasoning the underlying mechanisms behind the predictions, deep models cannot be fully used in collaborative learning. PGM-Explainer (Vu and Thai 2020) is an explanation method that explains the predictions of any GNN in an interpretable manner. It has great performance in interpreting graph-structured data, including node prediction tasks.

Problem Statement

In this work, we aim to model the performance of students during the implementation of SE projects for exploring their interactions with each other team members, and the influence of these interactions on the academic performance of students in a CL environment.

The goal of any software engineering course is to enable the students to master a variety of software engineering skills. The learning activities designed to achieve these learning goals are divided into a three-phase cycle, including writing documents, coding, and reviewing projects of other teams, as presented in Fig. 1. During the documentation phase, the students are required to submit various files. Each change in these files can be tracked by using a version control system. During the coding phase, the students commit code on the course website. Finally, during the reviewing phase, the students review the documents and codes of other teams and point out the problems by raising issues on the course website. Each problem raised by students has a degree rating, which indicates the severity of that problem. When students commit the revised documents or code in the repository, the rest of the team can view these changes and continue their development by considering these changes. Therefore, we assume that the commit behavior affects all the other members of the team. When a student raises a prob-

lem on a project of another team by creating an issue, all the members of that team have to work together to fix the project based on the raised issue. Therefore, we believe that the behavior of issue affects all the members of the team.

As presented in Fig. 1, the collaborative learning activities have peculiar spatial and temporal characteristics. In terms of spatial characteristics, the interaction activities reflect the structural influence relationships among the students in the course. In terms of temporal characteristics, the interaction graph keeps changing as the course progresses. The machine learning models, such as Ada Boost, KNN, and decision trees, currently used in collaborative learning research (Olsen, Aleven, and Rummel 2015; Pavlik Jr, Cen, and Koedinger 2009; Yee-King, Grimalt-Reynes, and d'Inverno 2016; Ekuban et al. 2020) are unable to deal with such complex spatio-temporal data. The GNNs are deep learning models that have the ability to capture the graph dependencies based on message passing between graph nodes. The transformers have the ability to forecast time series data (Min et al. 2022). In this work, we combine GNN and Transformer models to make full use of the spatio-temporal features available in the data. We propose a new model CLGT based on Graph Trans (Dwivedi and Bresson 2020) for modeling the interactions between the students participating in any software engineering course.

Interaction Graph Generation

It is necessary to convert the data collected from the version control system into a format that is usable by the neural networks. The students use a CL platform as a repository for code and document projects. The CL platform records the details regarding each change a student makes in any file. Therefore, the nature and place of the change are known. The data type of the interaction graph can be generated based on the recorded information. In this work, we require a definition that can validate if a construct is an instance of an interaction graph. We identify three necessary and sufficient conditions for interaction graphs.

1. An interaction graph is described by a graph $G = (V, E)$, where V denotes a set of vertices and E denotes a set of edges such that $E \in V \times V$. All the edges are directed from one vertex to another. This indicates that one vertex interacts with other vertices.

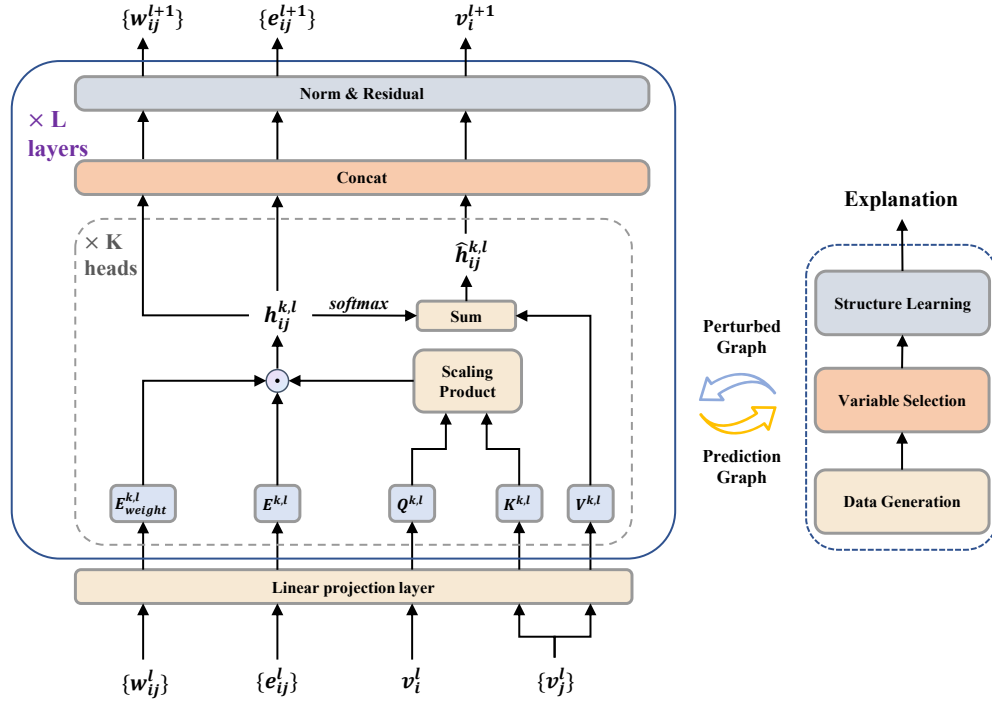2. Each vertex V represents a student, and each student is

Figure 2: The architecture for the CLGT framework, including the prediction module (Left) and the explainer module (Right).

the part of a team.

3. Edges E represents the interactions between different vertexes V. There are two types of edges, including interactions between students within the same team and interactions between different teams. Additionally, each edge type has different weights, depicting the degree of influence of the interaction activity.

Since there are multiple types of edges in an interaction graph and each edge type has multiple types of attributes, an interaction graph is a weighted heterogeneous graph.

Interaction Degree Definitions

In order to accurately depict the interaction activities between the students, we consider the revised lines in the documents and the codes. The issue depict the degree of influence of an interaction activity. The definition of the interaction degree of documents, codes, and issues is as follows.

1. For the interaction activities of document revision a_{doc}, the version control system record the number of lines added or deleted in the document num_{doc}, which indicates the degree of influence of the revision activity. Since the document revision activity is available to the students within the same team, we normalize the number of lines num_{doc} between all commits performed in one week, as follows:

$$I(a_{doc}^i) = \frac{num_{doc}^i}{\sum_{k=0}^n num_{doc}^k}$$

where $I(a_{doc}^i)$ indicates the influence of the document revision activity a_{doc}^i.

2. As the interaction activities regarding code revision a_{code} are available to the students within the same team, we process the data similar to the previous step. We normalize the number of lines num_{code} between all the commits performed in one week, as follows:

$$I(a_{code}^i) = \frac{num_{code}^i}{\sum_{k=0}^n num_{code}^k}$$

where $I(a_{code}^i)$ denotes the influence of the code revision activity a_{code}^i.

3. For the interaction activities regarding the creation of issues a_{issue}, each issue records the severity of the problems raised in the issue. We normalize the severity of the issues num_{issue} between all issues incorporated in one week, as follows:

$$I(a_{issue}^i) = \frac{num_{issue}^i}{\sum_{k=0}^n num_{issue}^k}$$

where $I(a_{issue}^i)$ denotes the influence of the code revision activity a_{issue}^i.

Based on the normalized data of the document revision, code revision, and issue related activities, we generate the interaction matrix, which is used as the input of the model.

Interaction Matrix Generation

Once an interaction graph is generated based on the collected commit records and issue data, it can be converted into an interaction matrix. For a simple graph with vertex set $V = v_1, \cdots, v_n$, the interaction matrix A is a square matrix of dimension $n \times n$. The value of A_{ij} represents whether

vertex v_i interacts with v_j. Zero denotes that there is no interaction, while a positive number indicates the influence of the interaction. Since the interaction graph is directed, the interaction matrix is asymmetric. Especially, the diagonal elements of a matrix are zero, as the edges (cycles) from the vertices pointing towards themselves are not allowed in the interaction graphs.

Prediction Module

The proposed prediction module of CLGT framework is inspired by the Graph Trans (Dwivedi and Bresson 2020).It is designed to effectively utilize the rich edge feature information in collaborative learning. It is notable that original graph trans model only considers heterogeneous graphs that contain multiple types of edges and does not consider the case where the weighted heterogeneous graph contains additional information features regarding the type of each edge. For instance, in our software engineering course dataset, the edges represent different interactions, and each interaction activity has a varying degree of influence. In order to address this problem, we add a pipeline for the weighted edge features based on the original model for making full use of edge information available in the graph. Fig. 2 (Left) illustrates the prediction module of CLGT framework. The prediction module comprises an extra edge feature pipeline and an extra weighted edge feature pipeline to effectively utilize the available edge information.

Given the interaction graph $G = (V, E)$, we pass the input node v_i for each node i, edge features e_{ij} and weighted edge features w_{ij} for each edge between node i and node j via a linear projection layer to embed these to $d-$dimensional hidden features v_i^0, e_{ij}^0 and w_{ij}^0. After the input layer, the pipeline propagates edge attributes from one layer to another, and the layer update equations are as follows.

$$h_{ij}^{k,\ell} = \left(\frac{Q^{k,\ell} v_i^\ell \cdot K^{k,\ell} v_j^\ell}{\sqrt{d_k}} \right) \cdot E^{k,\ell} e_{ij}^\ell \cdot E_{weight}^{k,\ell} w_{ij}^\ell \quad (1)$$

$$\hat{h}_{ij}^{k,\ell} = \sum_{j \in \mathcal{N}_i} softmax(h_{ij}^{k,\ell}) V^{k,\ell} v_j^\ell \quad (2)$$

where $Q^{k,\ell}, K^{k,\ell}, V^{k,\ell} \in \mathbb{R}^{d_k \times d}$ are trainable parameter matrices, k denotes the number of attention heads, $\mathcal{N}_i$ denotes the neighbors of node i. The outputs of $h_{ij}^{k,\ell}$ and $\hat{h}_{ij}^{k,\ell}$ in attention heads are then concatenated and passed via residual connection layers as follows.

$$v_i^{\ell+1} = F_r(F_c(\hat{h}_{ij}^{k,\ell}) + v_i^\ell) \quad (3)$$

$$e_i^{\ell+1} = F_r(F_c(h_{ij}^{k,\ell}) + e_i^\ell) \quad (4)$$

$$w_i^{\ell+1} = F_r(F_c(h_{ij}^{k,\ell}) + w_i^\ell) \quad (5)$$

where $v_i^{\ell+1}, e_i^{\ell+1}, w_i^{\ell+1}$ denotes the output of ℓ layer and the input of $\ell + 1$ layer, F_c denotes the concat operation, F_r denotes the normalization and residual connection operation.

The task of the model is node classification, e_i^ℓ, w_i^ℓ represent the intermediate results, and v_i^ℓ in each layer is passed to a fully connected layer to compute the prediction scores.

Explainer Module

In order to understand the underlying mechanisms behind the predictions, and present the explainable prediction results, we build an explainer module for the prediction module. This explainer module is inspired by the PGM-Explainer (Vu and Thai 2020) and elaborates the process based on which the proposed CLGT makes predictions. Moreover, it also shows the part of the graph structure data that is responsible for the prediction results.

As presented in Fig. 2 (Right), the workflow of the explainer module can be roughly divided into three stages.

1. **Data Generation** The explainer module repeatedly perturbs the nodes in the original graph data and feeds the perturbed graph data to the proposed CLGT for obtaining the sampled data. This sample data contains node information and prediction results.

2. **Variable Selection** Based on the sampled data, a pairwise dependency test is used to form an approximate Markov blanket for the target node to reduce the computational overhead and obtain all the potential statistics of the target node.

3. **Structure Learning** The explainer module uses a hillclimbing algorithm for maximizing the Bayesian information criterion (BIC) score in order to obtain an explanatory Bayesian network. The explainer module generates a weighted graph with the same number of nodes as the original input graph. The edge shows the influence of one node on the prediction result of the other node.

Experiments

Dataset

The object of this work include teams of students from a software engineering graduate course conducted in Spring 2021. In this course, a GitLab-based website [2] is selected to implement the CL platform. We aim to study the interactions of students, when performing software engineering projects assigned in a semester to explore the ways students interact with each other in team projects and the effect of such interactions on the academic performance of students in a CL environment. The software engineering course lasted 16 weeks and comprised three different sessions, including writing documents, writing code, and reviewing projects of other teams. We acquire all the commits and issues available within the GitLab during the 16-week course. We obtain 4,903 commit records and 862 issues. Each commit records the number of lines added or deleted from the documents and code. Each issue records the severity of the problems raised in the issue. Each week, the teacher classified students' grades into three categories (A, B, C) based on the quality and quantity of documentation and code contributed in the software engineering project.

Data Process

Based on the proposed method, the interaction graphs and interaction matrices are easily generated. As the course

---

[2]https://gitlab.com/

(a) Visualization of the output of the explainer module
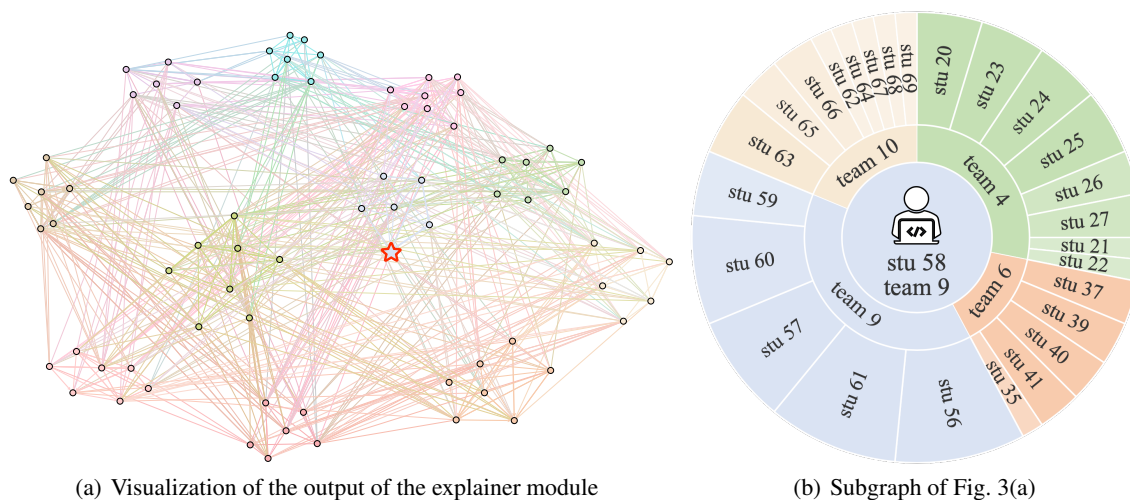
(b) Subgraph of Fig. 3(a)

Figure 3: (a) presents the visualization of the output of the explainer module. The nodes represent the students, and the edges represent the influence of students on each other. Different teams are denoted by different colors. The student 58 of team 9 is specially marked with ☆, which is further analyzed in (b). (b) presents the visualization of part of the Fig. 3(a). Different colors in the inner circle represent different teams, and the outer circular sector represents different students of the corresponding teams. The proportion of the sector indicates the degree of influence of a team and its members on the middle student. The larger the proportion, the greater the influence.

lasted for 16 weeks and the teacher graded each student weekly, we divide the acquired data into 16 sections. In each section, we generate an interaction graph and three corresponding matrices, namely addition, deletion, and issue matrices. The addition matrix (or the deletion matrix) considers the members of the same team only and depict the influence of adding (or deleting) documents and codes on the members of the same team. The issue matrix considers the members of different teams and represents the influence of issues raised by one team on the members of other teams. Since the number of lines of code and documentation in each commit vary considerably, and there is a need for quantifying the severity of raised issues, we divide them into three levels, i.e., minor, moderate, and severe levels. After division, we obtain 48 matrices, i.e., three matrices per week. Now, each matrix consists of three types of elements, representing the influence from one vertex to another.

The purpose of the proposed CLGT is node classification. The labels of each node include the weekly and final grades of the students in the published dataset. The weekly and final grades of students are assigned by the teacher based on the quality and quantity of documentation and code contributed in the software engineering project.

Baselines and Experimental Setup

In the experiments, we compare the proposed CLGT with Ada Boost and three state-of-the-art graph neural networks. The CLGT, Ada Boost, R-GCN, GTN and Graph Trans are constructed using Pytorch (Paszke et al. 2017) and scikit-learn (Pedregosa et al. 2011). For Ada Boost model, we use *sklearn.ensemble.AdaBoostClassifier* with 100 week estimators given the cumulative average number of GitLab pushes per week, and the total number of lines of docu-

mentation and code added, deleted, and modified per week. In the case of R-GCN and GTN models, the Adam optimizer is used and the hyperparameters, including learning rate, weight decay etc. are selected appropriately so that each baseline yields its best performance. For the original graph trans model and the proposed CLGT model, we use 10 graph transformer layers, where each layer comprises 8 attention heads and arbitrary hidden dimensions. Therefore, the total number of trainable parameters is in the range of 588k and 855k. We use the learning rate decay strategy for training the models. The training process is stopped when the learning rate reaches a value of 1×10^{-6}.

Results and Discussion

Prediction Results

The experimental results are presented in Table 1. For the sake of comparison, we implement the proposed CLGT model, R-GCN, GTN, Graph Trans model and Ada Boost on the same dataset we collected. In this work, the accuracy (ACC), the F1-score and the average and standard deviation of the area under the ROC curve (AUC) are used as evaluation metrics. The larger the AUC, F1-score or ACC, the better the model's prediction performance.

Experimental results show that CLGT model performs better as compared to Ada Boost, R-GCN, GTN, and Graph Trans model in terms of ACC, F1-score, and AUC. The outstanding performance of the proposed model shows better utilization of rich edge and weighted edge feature information as compared to the baseline models. The proposed architecture specifically ensures better utilization of datasets that not only contain multiple types of edges, but also additional information features regarding the type of each edge.
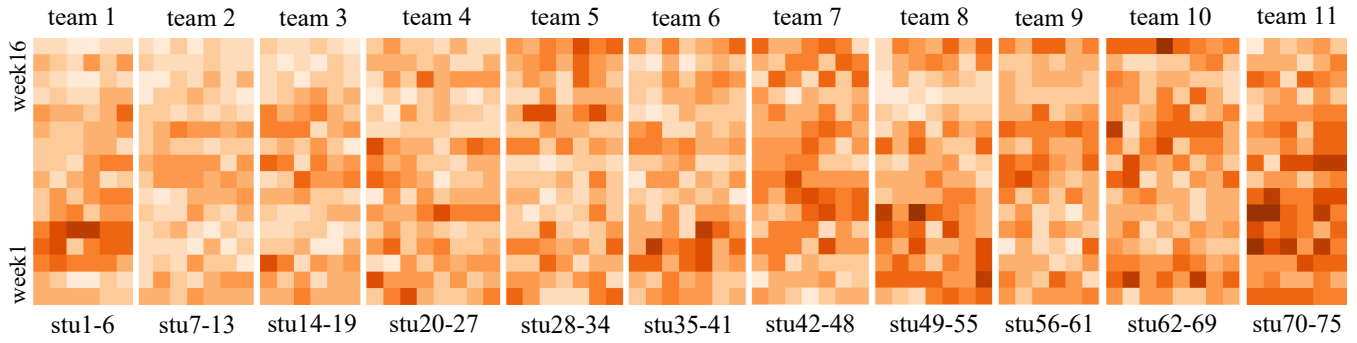
Figure 4: The horizontal axis represents all the 75 students in the course. The vertical axis represents the 16 weeks during which the course is conducted. The shades of color represent the activeness of students in the course, with darker colors representing more active students. Additionally, 75 students are divided into 11 teams, with each block in the figure representing a team.

Method	ACC	F1-score	AUC
Ada Boost	41.53±1.30	29.36±1.82	48.81±1.05
R-GCN	64.81±2.13	52.64±1.83	76.54±2.60
GTN	73.37±2.32	62.79±2.25	78.84±1.12
Graph Trans	72.46±2.08	59.04±1.84	86.21±1.73
CLGT(ours)	**73.92±1.78**	**66.75±2.15**	**90.57±1.88**

Table 1: The prediction performances of the proposed CLGT, Ada Boost, R-GCN, GTN, and Graph Trans model based on the same dataset. The best results are highlighted.

Case Study and Visualization

In order to explore the explainability of the predicted results obtained using the proposed CLGT model, we build an explainer module that generates a weighted graph. The edges of this weighted graph represent the influence of students on each other. The output of the weighted graph is visualized (as presented in Fig. 3(a)) for performing intuitive analysis.

1. After considering a specific student, we explore which students have the greatest influence on him or her. In most cases, the teammates have a greater influence on the students as compared to other team members. However, we can also infer some information from the interpretation results that is not reflected directly in the original interaction graph. We process and visualize the subgraph of Fig. 3(a) in Fig. 3(b). The student 58 (in team 9) does not interact with student 22 (in team 4). However, that student indirectly affects student 58 through the interaction with students in team 10 (there are direct interactions between team 4 and team 10). This shows that we can explore potential relationships among different students to further analyze the role of student behavior.

2. We explore how the influence of each student changes during the course of 16 weeks. We process the data of 16-week course in order to visualize the influence of all students in each week presented in Fig. 4. We find a high correlation between the students' influence and the teams they belong to. Although some teams, e.g., team 10, do not achieve the best grades, their overall influence is significantly bigger. On the contrary, other teams, e.g., team 3, achieve better grades; however, their influence is not as high as their grades. We think this may reflect the underlying characteristics of different teams, such as team 10 being more active, but the overall quality of the project code is poor. On the other hand, team 3 showcases the exactly opposite characteristics. These results enable the course instructors to build better student profiles and perform accurate assessment based on the performance of students working in teams.

Conclusions

In this work, we investigate how students interact with each other in team projects and how such interactions affect the academic performance of these students in the collaborative learning paradigm. We choose one-semester software engineering course to investigate and assess the collaborative learning activities of students. We propose a method for extracting the activity data from an in-house Gitlab platform and constructing the interaction graph. Moreover, we also introduce an extended graph transformer model named CLGT for accurately predicting the performance of students. We also build an interpretation model for explaining the prediction results of the proposed CLGT model and analyze the influence of each student on the teammates during the course project. The experimental results confirm that the proposed CLGT model outperforms the state-of-art models and provides good insights for the instructors to assess the learning progress of students. Moreover, the proposed CLGT differentiates the students with poor performance in collaborative learning and gives the teachers early warnings, so that appropriate assistance can be provided.

Acknowledgments

References

Arcelli Fontana, F.; and Raibulet, C. 2017. Students' feedback in using GitHub in a project development for a software engineering course. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 380–380.

Buffardi, K. 2020. Assessing individual contributions to software engineering projects with git logs and user stories. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 650–656.

Clifton, C.; Kaczmarczyk, L. C.; and Mrozek, M. 2007. Subverting the fundamentals sequence: using version control to enhance course management. *ACM SIGCSE Bulletin*, 39(1): 86–90.

Corbett, A. T.; and Anderson, J. R. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4): 253–278.

Dwivedi, V. P.; and Bresson, X. 2020. A Generalization of Transformer Networks to Graphs. *CoRR*, abs/2012.09699.

Ekuban, A. B.; Mikroyannidis, A.; Third, A.; and Domingue, J. 2020. Using GitLab Interactions to Predict Student Success When Working as Part of a Team. In *International Conference on Interactive Collaborative Learning*, 127–138. Springer.

Freund, Y.; and Schapire, R. E. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.*, 55(1): 119–139.

Fu, E.; Zhuang, Y.; Zhang, J.; Zhang, J.; and Chen, Y. 2021. Understanding the User Interactions on GitHub: A Social Network Perspective. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 1148–1153. IEEE.

Laal, M.; and Laal, M. 2012. Collaborative learning: what is it? *Procedia-Social and Behavioral Sciences*, 31: 491–495.

Liang, Y.; Peng, T.; Pu, Y.; and Wu, W. 2022. HELP-DKT: an interpretable cognitive model of how students learn programming based on deep knowledge tracing. *Scientific Reports*, 12(1): 1–11.

Loeliger, J.; and McCullough, M. 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.".

Min, E.; Chen, R.; Bian, Y.; Xu, T.; Zhao, K.; Huang, W.; Zhao, P.; Huang, J.; Ananiadou, S.; and Rong, Y. 2022. Transformer for Graphs: An Overview from Architecture Perspective. *arXiv preprint arXiv:2202.08455*.

Olsen, J. K.; Aleven, V.; and Rummel, N. 2015. Predicting Student Performance in a Collaborative Learning Environment. *International Educational Data Mining Society*.

Parizi, R. M.; Spoletini, P.; and Singh, A. 2018. Measuring team members' contributions in software engineering projects using git-driven technology. In *2018 IEEE Frontiers in Education Conference (FIE)*, 1–5. IEEE.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. *NeurIPS*.

Pavlik Jr, P. I.; Cen, H.; and Koedinger, K. R. 2009. Performance Factors Analysis–A New Alternative to Knowledge Tracing. *Online Submission*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Piech, C.; Bassen, J.; Huang, J.; Ganguli, S.; Sahami, M.; Guibas, L. J.; and Sohl-Dickstein, J. 2015. Deep knowledge tracing. In *Advances in neural information processing systems*, 505–513.

Pu, Y.; Wu, W.; Jiang, T.; Desmarais, M.; Lynch, C.; Merceron, A.; and Nkambou, R. 2019. ATC Framework: A fully Automatic Cognitive Tracing Model for Student and Educational Contents. In *EDM*.

Reid, K. L.; and Wilson, G. V. 2005. Learning by doing: introducing version control as a way to manage student assignments. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 272–276.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 593–607. Springer.

Vu, M.; and Thai, M. T. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33: 12225–12235.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1): 4–24.

Yee-King, M.; and d'Inverno, M. 2016. Stimulating collaborative activity in online social learning environments with Markov decision processes. In *EDM*, 652–653.

Yee-King, M.; Grimalt-Reynes, A.; and d'Inverno, M. 2016. Predicting student grades from online, collaborative social learning metrics using K-NN. In *EDM*, 654–655.

Yun, S.; Jeong, M.; Kim, R.; Kang, J.; and Kim, H. J. 2019. Graph transformer networks. *Advances in neural information processing systems*, 32.