Automatically Deriving Developers' Technical Expertise from the GitHub Social Network

YANCHUN SUN*, Key Laboratory of High Confidence Software Technologies, Ministry of Education, China and School of Computer Science, Peking University, China
JIAWEI WU, School of Computer Science, Peking University, China
XIAOHAN ZHAO, School of Computer Science, Peking University, China
HAIZHOU XU, School of Computer Science, Peking University, China
YE ZHU, Centre for Cyber Resilience and Trust, Deakin University, Australia
ZHENPENG CHEN, Nanyang Technological University, Singapore
SIHAN WANG, School of Computer Science, Peking University, China
HUIZHEN JIANG, School of Computer Science, Peking University, China
GANG HUANG, Key Laboratory of High Confidence Software Technologies, Ministry of Education, China and School of Computer Science, Peking University, China

Developers' technical expertise is crucial for numerous tasks within open-source communities, such as identifying suitable developers and maintainers. Despite its significance, GitHub, the world's largest open-source code hosting platform, does not explicitly display developers' technical expertise. Existing methods fall short in capturing the multifaceted and dynamic nature of developers' skills and knowledge. To address this gap, we propose a novel approach that leverages graph neural networks (GNNs) to express developers' technical expertise. Our method constructs a comprehensive GitHub social network that integrates various social and development activities. We then employ a GNN model to learn a low-dimensional representation vector for each developer, encapsulating their technical expertise across different dimensions. We assess the effectiveness of our model by comparing it against five baselines on three GitHub social relationship recommendation tasks, including SimDeveloper, ContributionRepo, and RepoMaintainer. Our proposed method outperforms these baselines, achieving improvements of 5.6%-9.5% on Hit Ratio@10 and 3.4%-11.1% on F1 score. These results demonstrate promising performance in predicting technical preferences for both repositories and developers. This research contributes to a more nuanced understanding of developer expertise in open-source communities and has potential implications for improving collaboration and project management on platforms like GitHub.

*Corresponding author.

This work was supported by the National Key R&D Program of China (Grant No. 2023YFF0616900).

Authors' Contact Information: Yanchun Sun, sunyc@pku.edu.cn, Key Laboratory of High Confidence Software Technologies, Ministry of Education, China and School of Computer Science, Peking University, China; Jiawei Wu, jw_wu@pku.edu.cn, School of Computer Science, Peking University, China; Xiaohan Zhao, zxh21@stu.pku.edu.cn, School of Computer Science, Peking University, China; Haizhou Xu, xuhaizhou@stu.pku.edu.cn, School of Computer Science, Peking University, China; Ye Zhu, ye.zhu@ieee.org, Centre for Cyber Resilience and Trust, Deakin University, Australia; Zhenpeng Chen, zhenpeng.chen@ntu.edu.sg, Nanyang Technological University, Singapore; Sihan Wang, sihanwang@stu.pku.edu.cn, School of Computer Science, Peking University, China; Huizhen Jiang, jiang2000@stu.pku.edu.cn, School of Computer Science, Peking University, China; Gang Huang, hg@pku.edu.cn, Key Laboratory of High Confidence Software Technologies, Ministry of Education, China and School of Computer Science, Peking University, China.

ACM 1557-7392/2025/6-ART
https://doi.org/10.1145/3746451

CCS Concepts: • **Software and its engineering → Search-based software engineering**.

Additional Key Words and Phrases: GNN, deep learning, technical expertise, recommendation system, GitHub

1 INTRODUCTION

Open-source software and communities attract more developers and enterprises to host their projects on open-source platforms. As the world's largest open-source community and social coding platform, GitHub boasts over 100 million registered users and 370 million code repositories as of January 2023, with at least 28 million open-source repositories[1]. Open-source software development encompasses not only open-source code, but also collaborative development within a community. GitHub provides tools such as Pull Request (PR) and Issue to facilitate contributions to open-source software. However, getting developers' contributions accepted is challenging. Research [13, 14, 24, 27, 48, 60] has consistently shown that building trust between developers and open-source repository maintainers is crucial for successful contributions to open-source software. While this trust can be fostered through communication, it predominantly relies on developers' technical expertise [32, 41]. Mining the technical features of developers is therefore of great significance for the growth and development of open-source communities.

Currently, GitHub does not explicitly display technical features for developers, and there is a problem of data imbalance. For repositories, data sources for obtaining technical features include code, description, READMEs, topics, and statistics automatically generated by GitHub for projects (such as programming languages). These data present different modalities. For developers, the situation is quite different. GitHub provides minimal valuable descriptions except for homepage introductions, which depend on developers to write themselves. Therefore, mining a developer's technical features often requires reliance on social and development activities related to that developer.

Existing research on mining developer technical expertise has several limitations. Most studies focus on specific technical fields, such as keywords, programming languages and frameworks [9, 25]. Some research even limits technical expertise to specific development frameworks, such as React [32, 40]. From an application perspective, these studies are limited. Methodologically, most studies mine developers' technical expertise based solely on their development relationships, such as contributed code and repository README files [12, 22, 50]. However, recent studies [5, 6, 37, 48] highlights that social features are also an important factor in reflecting developers' technical expertise, but there is still a lack of study on integrating social features and technical features to model the technical expertise of developers. The potential of incorporating social relationships and a broader range of development activities, such as pull requests (PR) and issues, are not fully leveraged.

This paper aims to capture developers' technical features by leveraging the extensive collaboration data available on GitHub. We construct a social activity graph that models developers' development and communication activities, encompassing four entity types: repository, developer, PR and issue. From the graph, we derive semantic representations of multiple participants on GitHub, including their technical expertise related to repositories, issues and PRs.

Traditional methods of deriving technical expertise representation merely integrate the technical features of developers, while neglecting social activity features in open-source communities. In this paper, we introduce GitHub social activity relationships into the process of deriving developers' technical expertise representation, generating embeddings that accurately capture the nuances of developers' expertise. These refined embeddings serve as a robust foundation for downstream recommendation tasks. Notably, in the realm of open-source community-based software development, social activity relationships have emerged as a new paradigm shaped by these communities, offering valuable insights into specific areas of expertise. For instance, a developer who frequently commits to a particular type of repository is likely to exhibit a high level of proficiency in that

[1]https://en.wikipedia.org/wiki/GitHub

technology, while one who actively submits pull requests to resolve complex issues often demonstrates a deep understanding of the problem domain and the ability to devise effective solutions.

We utilize multiple data sources and integrate them to obtain comprehensive initial technical expertise representations for all entities except developers. We then apply a Graph Neural Network (GNN) model to the social activity graph, merging relevant entities' embeddings to get developers' representation for technical expertise, as well as enhancing the initial technical expertise representation of other entities. This process results in a more comprehensive and generic technical expertise representation for entities, which can be easily adapted for various downstream recommendation tasks without additional complex feature engineering. We demonstrate the effectiveness of our approach through three selected recommendation tasks.

The main contributions of this paper are as follows:

- To the best of our knowledge, this paper is the first to propose an approach that integrates GitHub social features and technical features to represent the technical expertise of developers. By employing a GNN, we model diverse social relationships and enhance the initial representation of technical expertise.
- We conduct empirical comparisons and analysis of the effect of social relationship features on deriving developers' technical expertise, involving five advanced baselines, and three representative downstream tasks. It is demonstrated that social relationship plays a significant role in driving developers' technical expertise.
- We have made our code open-source at https://anonymous.4open.science/r/DTEM-6830/, enabling other researchers can reproduce and build upon our work. Additionally, the open source community can leverage our method for developers' technical expertise, thereby enhancing open source collaboration.

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 presents the overview of the approach. Section 4 introduces the experiments and evaluation. Section 5 introduces the user evaluation experiment. Section 6 concludes the paper.

A small proportion of the results presented in this paper were previously reported in a two-page poster at ASE 2024 [47]. However, this extended version significantly expands on the prior work by supplementing two existing sections and adding five new sections. In the Introduction (Section 1), we have enhanced the discussion of research motivation and novelty. In the Methodology (Section 3), we have expanded from three to four modules to improve the interpretability of our approach and the traceability of developers' technical decisions. Additionally, we have introduced five new sections: Related Work (Section 2), Evaluation (Section 4), User Evaluation (Section 5), Discussion (Section 6), and Conclusion and Future Work (Section 7). These additions significantly deepen the analysis and contextualization of our results, offering a more comprehensive and rigorous contribution.

2 RELATED WORK

2.1 Developer Technical Expertise Mining

GitHub serves as a collaborative platform where developers share code repositories and engage in activities such as PRs and issues. Many research studies have utilized this data to extract features and perform various automated recommendations [7, 19, 33, 36, 41]. One of the most crucial features in these studies is the technical expertise of developers or repositories. Some works [9, 25, 59] represent developers' technical expertise using keywords derived from sources such as Wikipedia entries and GitHub topics. Others [32, 38, 40] represent it through libraries, which are frameworks commonly used by developers, such as .NET, Angular and React. Additionally, some studies [12] aims to capture developers' technical expertise at a more fine-grained level, such as the code level, often expressing these technical skills as embedding vectors.

Several studies, such as [2, 55], represent developers' technical expertise through their GitHub activities-events. These studies often incorporate temporal sequences, assigning greater weight to recent events and grouping them to address challenges associated with sparse data. Moreover, methods like the Topic method [59] and work [16] focus on GitHub topics. A repository is represented by its topic distribution vector and a developer is represented by the union topic distribution from all repositories he has contributed. This approach is used to recommend repository trends based on developer preferences or to assist developers in selecting topics that best describe their created repositories.

The technical skills mentioned above are often expressed as embedding vectors. Some studies combine embeddings from different aspects; for example, the Dev2Vec method [11] concatenates repository, issue, and API embeddings, while [2] integrates scores derived from different embeddings through linear superposition. Other studies, such as [56], consider historical sequence behaviors to generate embeddings. Liu et al. [34] proposed "GAT method", which leverages user social graph data to enhance recommendations. This method involves transforming social connections into edge embeddings within user-item interaction graph, and feeds them to a modified GAT model. However, none of these studies address the integration of technical expertise across different entities by incorporating a social network. Such an approach can leverage both activity events and various types of embeddings, providing a more comprehensive representation of technical expertise.

Inspired by these works, our study not only utilize data in various modalities—including code data, natural language data, and discrete data to extract developers' technical expertise, but also focus on modeling social relationships. We achieve this by constructing a social network to integrate embeddings from different modalities.

2.2 GitHub Community Study

Developers on GitHub interact and influence each other's development activities, leading researchers to study the social properties and the impact these relationships. Mockus et al. [39] used the Louvain clustering algorithm to group repositories based on common code submissions, identifying similar clusters of repositories. Gote et al. [23] constructed a graph of developer collaboration patterns by extracting co-editing relationships from Git history. Bana et al. [3] analyzed various types of social relationships, such as "developer-developer", "repository-developer" and "repository-repository", to identify influential technologies and repositories. Recently, researchers have started exploring the attributes that define a developer's impact. For example, the D-Index [5] is proposed to meaningfully equate several indicators for the virtues of a developer, such as contributed code, its quality, mentoring in online learning communities, and community engagement. Mezouar et al. [20] demonstrated that stronger social ties between developers and project maintainers lead to faster PR responses. Tsay et al. [48] highlighted the effect of social distance, such as the relationship between developers and repository maintainers, on the acceptance of contributions like PRs. Further research [51, 58] analyzed the correlation between developers' commits and repositories issues. Xiao et al. [54] found that newcomers' expertise preference, open-source software (OSS) experience, activeness, and sentiment are critical to their successful contribution and onboarding.

Building on these insights, our study focuses on the most common development activities on GitHub, such as Commit, PR and Issue, to construct the social relationship between developers. We identify three main collaboration scenarios on GitHub. For each scenario, we design a recommendation task to assess the effectiveness of our method.

3 METHODOLOGY

We formulate our research problem as follows. Given the publicly available GitHub data, our goal is to design an approach that can learn developers' technical expertise in a manner general enough to be applicable to various recommendation tasks.

Our approach to learning GitHub developers' technical expertise can be divided into four main components, as illustrated in Figure 1.

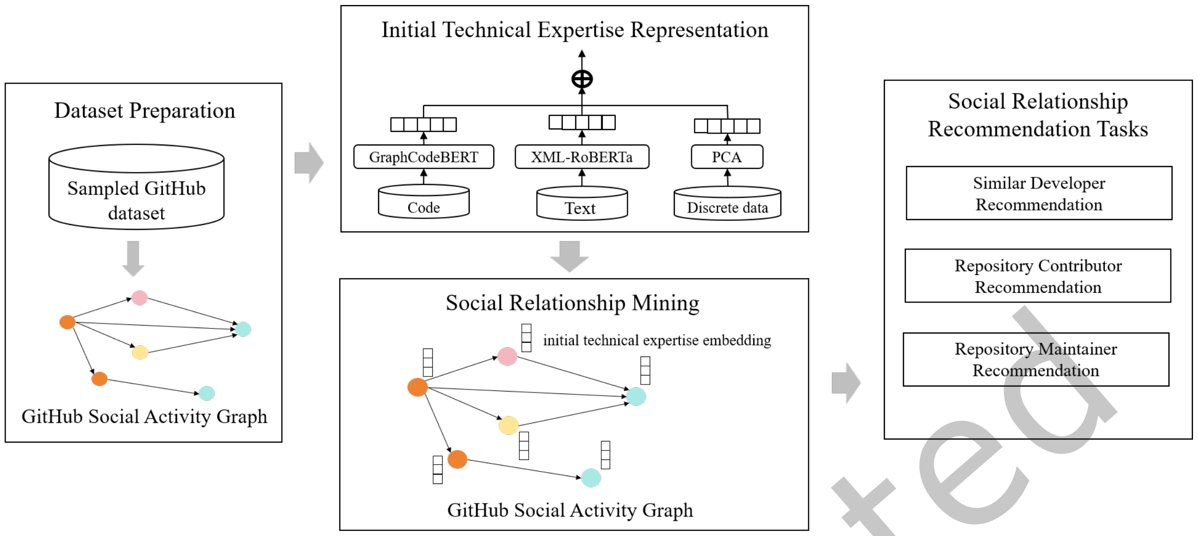

Dataset Preparation
Sampled GitHub dataset
GitHub Social Activity Graph
Initial Technical Expertise Representation
GraphCodeBERT
XML-RoBERTa
PCA
Code
Text
Discrete data
Social Relationship Mining
initial technical expertise embedding
GitHub Social Activity Graph
Social Relationship Recommendation Tasks
Similar Developer Recommendation
Repository Contributor Recommendation
Repository Maintainer Recommendation

Fig. 1. Architecture of proposed method to represent GitHub developers' technical expertise and evaluation

- **Data Preparation module**: Initially, we gather open data from GitHub and construct a social activity graph. This graph encompasses four primary entities on GitHub—repositories, developers, PRs and Issues—as well as their social relationships.
- **Initial Technical Expertise Representation module**: Next, utilizing a deep-learning-based embedding method, we extract the initial technical expertise of GitHub entities.
- **Social Relationship Mining module**: Subsequently, we employ a GNN model and use initial technical expertise embeddings to initialize all node embeddings in the GNN model. The implicit social features are then captured by the GNN through a process that essentially refines these initial technical embeddings, to ultimately generate comprehensive technical representations for all entities.
- **Social Relationship Recommendation module**: Finally, we leverage the technical expertise representations of developers to facilitate various processes within GitHub learning and collaboration. We implement a series of social relationship recommendation tasks to showcase developers' technical expertise and enhance collaborative efforts within the GitHub ecosystem.

3.1 Data Preparation

Social activities on GitHub extend beyond direct developer-to-developer interactions. Developers who contribute to or raise issues for the same repositories often share similar skill preferences, indicating a latent social activities relationship. This perspective allows us to capture more nuanced interactions within the GitHub community, enhancing our understanding of developer skills and potential collaborations.

To construct a GitHub social activity graph, we adopt a sampling method to obtain a representative subset of open-source repositories from GitHub. We refer to the GitHub community dataset provided by [10] and randomly sample 50,000 repositories that had changes in 2022 as our research objects. To focus on repositories with code data for initial technical expertise mining, we select repositories primarily developed in Python, JavaScript, GoLang, Java, PHP, or Ruby, as, these programming languages are commonly used in code-related research (e.g., [21] and [26]). It is important to note that repositories may also use other programming languages besides the primary ones. Table 1 shows the dataset information.

Table 1. Dataset Statistics

Programming Language	Python	JavaScript	GoLang	Java	PHP	Ruby	Total
Number of Repository	18,282	12,839	6,539	5,966	4,264	2,110	50,000

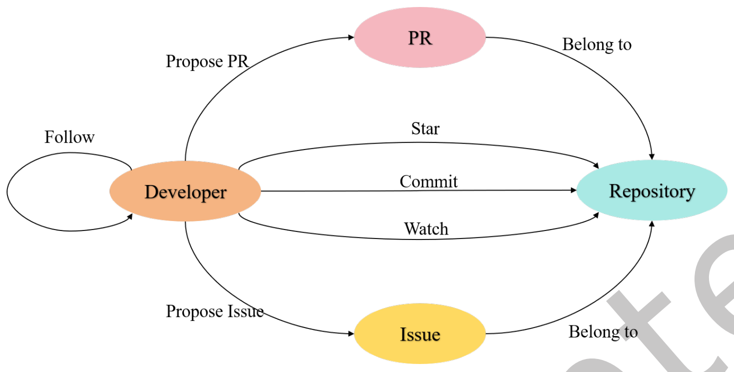


Fig. 2. Scheme of the social activity graph

We utilize GitHub Restful API[2] to obtain the social activity data of the sampled repositories. In this paper, we collected the following social relationships to learn the technical expertise of GitHub entities: Watch, Star, Commit, Follow, and Propose PR/Issue. The scheme of the social activity graph is shown in Figure 2.

Finally, we preprocess the obtained social activity data as follows.

- **Remove bot developers**. According to prior research [15], GitHub bots are automated programs used to perform tasks such as making code commits, and opening, managing or closing issues. Identifying these bots is essential to distinguish automated actions from those actions performed by real developers. This study introduced two bot detection methods: BIN and BIMAN. The BIN method relies on regular expressions applied to user names to identify bots, while BIMAN builds upon BIN by incorporating more features including message features such as commit messages, the number of files or repositories in commits, and employs a random forest classification model to predict bots. In order to assess the performance of these two methods on our dataset, we randomly sampled 10,000 developers and manually labeled them as either GitHub bots or human users based on the bot characterization guidelines outlined in [15]. The results indicate that BIN and BIMAN perform same precision rates of 94.1% and recall rates of 76.2%, successfully identifying 32 bots out of 42 labeled bots, with 2 false positives and 10 false negatives. Given the comparable performance of both methods, we choose to employ BIN method, removing developers with names containing "[special separator] bot [special separator]", where "[special separator]" includes beginning and ending symbols such as "-", "_""[" and "]".
- **Remove noise data**. We remove Issue and PR nodes that lack descriptive information, as these entities are often created by bots or used for experiments and do not represent real development activities. In addition, less representative commits are also filtered out. According to prior work [4, 44], if the frequency of a developer's contributions to a repository falls below a certain threshold, then such contributions are considered negligible. Specifically, the frequencies of contributions are divided into high (the top 1%), medium (the subsequent 10%, i.e. top 1% 11%) and low (the remaining) groups and the low group

[2]https://docs.github.com/en/rest?apiVersion=2022-11-28

represent less representative commit relationships. We apply this criterion to our dataset and find that the top 11.74% 12.57% range of contribution frequencies are precisely 10. Therefore, we remove commit relationships that have a frequency of less than 10.

- **Simplify follow relationship**. Within the scope of this study, only the Follow relationships between developer nodes were considered. This requires both the follower and the followee to be contained in the developer node set defined in this study. All other Follow relationships were removed.

After the data preprocessing, the detailed statistics of the GitHub social activity graph are shown in Table 2 and Table 3. This heterogeneous graph consists of eight types of directed edges and four types of nodes. Among the more than 5 million edges in the graph, the Follow edges between developer nodes account for more than 2 million, while the remaining types of edges are distributed relatively evenly. By conducting the data preprocessing steps including removing bot developers and removing noise data on the 50,000 sampled repositories, we reduced the number of developers from 511,447 to 394,474, issues from 1,336,584 to 692,554, PRs from 1,031,082 to 379,496.

Table 2. Statistics of the edges in GitHub social activity graph

Edge Type	Src Node Type	Dst Node Type	Edge Count
Commit	Developer	Repository	161,241
Star	Developer	Repository	947,423
Watch	Developer	Repository	150,292
Follow	Developer	Developer	2,286,407
Propose Issue	Developer	Issue	692,554
Propose PR	Developer	PR	379,498
Issue belongs to	Issue	Repository	692,554
PR belongs to	PR	Repository	379,498
Total	–	–	5,689,467

Table 3. Statistics of the nodes in GitHub social activity graph

Node Type	Node Count Before Preprocessing	Node Count After Preprocessing
Developer	511,447	394,474
Repository	50,000	50,000
Issue	1,336,584	692,554
PR	1,031,082	379,496
Total	2,929,113	1,516,524

3.2 Initial Technical Expertise Mining

To obtain the initial technical expertise representation of GitHub entities, we consider multiple attributes of the entity nodes across three modalities: code data, natural language text data, and discrete data.

For code data, we utilize tree-sitter[3] to segment each code file into individual code blocks, each only containing a function definition snippet. Other parts of the code files are ignored. It is noteworthy that in all programming

[3]https://tree-sitter.github.io/tree-sitter/

languages of our dataset as shown in Table 1, function definitions are fundamental elements of the program and contain the majority of code semantics. Therefore, to capture the technical expertise embedded in the code, utilizing all function definitions is sufficient.

To obtain the embedding of code data, we considered three models, i.e. GraphCodeBERT [26], Code2vec [1] and GraphCode2Vec [35]. Based on the evaluation presented in [35], we find that both GraphCodeBERT and GraphCode2Vec are designed with a broader range of code-related tasks. Therefore, they are more suitable for generic code semantic modeling. Among three models, GraphCodeBERT achieves superior performance overall, despite that it has higher complexity and is trained on a larger dataset. Therefore, we employ GraphCodeBERT to generate code embeddings for each code block. The code embedding for a repository or pull request (PR) is obtained by averaging the embeddings of all the code files it contains or modifies. Similarly, the embedding of a code file is obtained by averaging the embeddings of all its code blocks.

For natural language text data, we use xml-roberta-base [8], a variant of the BERT model, to generate text embedding. The natural language embedding for a repository, PR, or issue is generated by inputting all its text data into the xml-roberta-base model. We preprocess the text data by removing non-textual elements (e.g., images, hyperlinks, and code snippets) and special HTML or Markdown characters (e.g., the "##" symbol used for second-level headings in Markdown).

For discrete data, we apply one-hot encoding to obtain the embedding for discrete attributes of repositories, including GitHub topics and programming languages. To address the sparsity of these data, we employ Principal Component Analysis (PCA) to reduce the dimensionality to 256. The discrete data embedding for a repository is created by concatenating the embeddings of its topics and programming languages.

Finally, we assign each entity an initial technical expertise embedding by concatenating all the embeddings from its different parts. For example, the initial embedding of a repository is the concatenation of its natural language embedding, code embedding, and discrete data embedding. The embedding dimensions of natural language, code embedding and discrete data are 768, 768 and 512, respectively. These dimensions are determined by referencing the dimensions set in original embedding models, including xml-roberta-base and GraphCodeBERT. For discrete data, standard dimensional settings are applied following common practice. The initial technical expertise embedding is used as part of the input for the subsequent GNN model to acquire the complete technical expertise representations.

3.3 GitHub Social Relationship Modeling

While we have obtained embeddings of entities such as repositories, PRs, and issues, the developers' technical expertise embeddings remain undefined. Existing methods obtain developer embeddings by simply averaging the vectors of entities related to the developer [11]. However, Graph Neural Networks (GNNs) offer a more sophisticated approach, implicitly capturing collaborative effects through message-passing mechanisms. GNNs have proven effective in recommender systems, outperforming traditional methodologies in modeling social relationships [53, 57]. Accordingly, we employ a GNN model on our constructed heterogeneous graph to achieve superior results.

Figure 3 illustrates the architecture of our GNN model. To effectively integrate the social and technical features of GitHub users, we use the initial technical expertise representation as input to train the GNN on the GitHub social activity graph. By simultaneously optimizing for graph structure and attribute learning tasks, we capture the social characteristics of each entity and enable developer nodes without initial representations to acquire technical expertise information from adjacent nodes. That is, the initial technical expertise representation is refined through this GNN-based continual training process. This integration ultimately generates the final representation for each entity. Moreover, to ensure robust connectivity within our GNN model and facilitate the flow of information

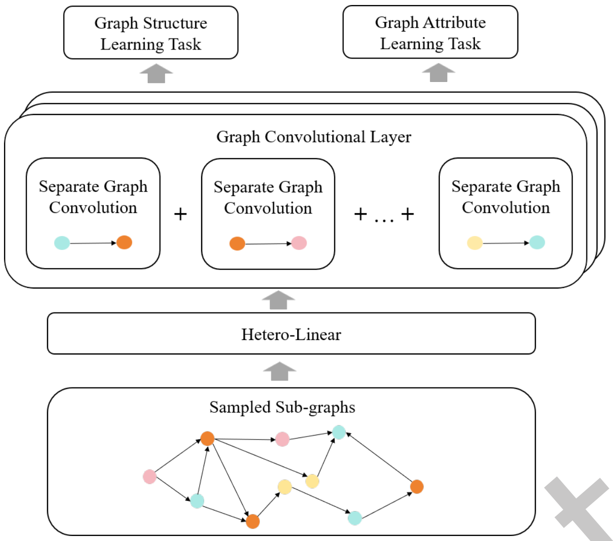

Graph Structure Learning Task
Graph Attribute Learning Task
Graph Convolutional Layer
Separate Graph Convolution
+
Separate Graph Convolution
+ … +
Separate Graph Convolution
Hetero-Linear
Sampled Sub-graphs

Fig. 3. Architecture of the GNN model

across different entity types, we reverse the direction of the commitment edges depicted in Figure 2, transforming GitHub social activities into a cohesive, interconnected graph.

3.3.1 GNN Model. To model the social features in the GitHub social activity graph, we use a GNN model. The input of the GNN model contains the obtained initial technical embedding, and the GitHub social activity graph. By iteratively refining the initial technical embeddings, the GNN model aims to learn the information of graph structure and attributes. Therefore, the output of the GNN model is the final representation of technical expertise of all entities, which contains information about both technical features in initial technical embedding, and the social features learned in the GNN training process.

Specifically, we use GraphSAGE [28] as the graph convolutional layer. During each iteration, a node aggregates information from all of its neighboring nodes and updates its own representation, which is subsequently used for downstream task prediction. Then, the back-propagation algorithm is leveraged to refine the representation of all related nodes. This calculation process in GraphSAGE can be divided into three stages as follows:

- **Message Passing**. In GNN, information is propagated from a node's neighbors to update its own hidden representation, typically through a message function. In GraphSAGE, the messages propagated are just the hidden representations of the neighbors themselves.
- **Aggregation**. In GNN, the messages from neighbors are aggregated using an aggregation function. In GraphSAGE, the aggregation function is defined as follows:

$$H_{\mathcal{N}(i)} = \text{AGGREGATE}(\{H_j,\ \forall j \in \mathcal{N}(i)\}) \tag{1}$$

Where $\mathcal{N}(i)$ denotes the neighbor set of node i and $H_{\mathcal{N}(i)}$ denotes the aggregated messages for the node i. AGGREGATE is the aggregation function. In the original GraphSAGE paper, the authors provided several alternatives for AGGREGATE, including averaging, LSTM networks [29], and pooling. In this paper, we choose the pooling method.

$$\text{AGGREGATE}_{\text{pool}} = \max(\{\text{ReLU}(W_{\text{pool}} h_j + b),\ \forall j \in \mathcal{N}(i)\}) \tag{2}$$

W_{pool} and b are trainable parameters. The ReLU function is used to introduce non-linearity.

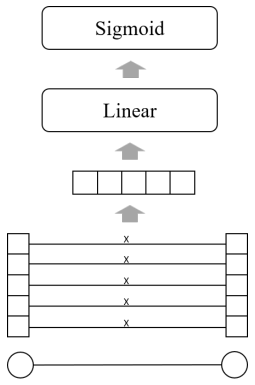


Fig. 4. Architecture of the GNN model

- **Feature Updating**. The hidden representation of a node is updated based on the aggregated messages, which is achieved through a node update function. GraphSAGE computes the update as follows.

$$H_i' = W(H_i \parallel H_{\mathcal{N}(i)}) \tag{3}$$

W is trainable, $\parallel$ denotes a concatenation operation. H_i' is the updated hidden representation of node i.

A single graph convolutional layer typically captures the information of first-order neighbors only. To capture higher-order neighbor information, GNN stacks multiple graph convolutional layers.

To effectively process the heterogeneous GitHub social activity graph, we partition it into sub-graphs based on edge types. Each sub-graph encompasses a single edge type, and we employ separate GraphSAGE models to derive node embeddings within each sub-graph. The final embeddings for the entire graph are then generated by aggregating the node embeddings from all sub-graphs. Due to the graph's extensive size, training a GNN on the complete structure is impractical. Instead, we implement a mini-batch training approach, sampling a small portion of the graph for each training iteration. For every target node, we randomly select 15 of its neighbors as dependent nodes to construct subgraphs. Moreover, the embedding dimensions in three modalities are not consistent, i.e. 768 for text and code data, 512 for discrete data, and different types of entities consists of data in different modalities. Therefore, we incorporate a Hetero-Linear layer within the GNN model to transform the dimension into 512. The GNN architecture is configured with 3 graph convolutional layers, and the neighbor sampling order is also set to 3.

3.3.2 Training. We train our GNN model using two learning objectives: graph structure learning and graph attribute learning. The graph structure learning objective aims to capture the structural information of the graph, including node and edge types as well as their connections. We learn edge types by using a separate convolutional layer for each edge type. We learn the connections of nodes by predicting the existence of links between nodes. The graph attribute learning objective focuses on learning the contribution weight of the committed relationships between developers and repositories. To handle both objectives effectively, we design a unified model architecture, as shown in Figure 4.

The edge representation is generated through the element-wise multiplication of the hidden representation vectors of the two nodes that are connected by the edge. This operation simulates the interaction between the nodes. The following describes the specific design of each learning objective in detail.

Graph Structure Learning. This learning objective involves learning the connection information of the graph. We treat this as a link prediction task, which aims to predict whether there is an edge between two nodes in our GitHub social activity graph. This task is equivalent to a binary classification problem. To facilitate the link prediction task, we sample a subgraph for training and randomly add an equal number of fake edges in the subgraph. The representation vectors for both real and fake edges are generated through the element-wise multiplication of the hidden representations of the respective node pairs linked by the edges. The real and fake edge representation vectors are then processed through a linear layer followed by a sigmoid layer to predict the existence of an edge, providing a probability score that indicates the likelihood of an edge.

Graph Attribute Learning. This learning objective focuses on the technical attributes of the GitHub social activity graph, specifically the contribution ratio between developers and repositories. In the GitHub social activity graph, developers exhibit varying levels of contribution to the same repository. We assume that developers who contribute more to a repository possess stronger technical expertise related to that repository. To quantify these contributions, we use GitHub Restful API to obtain the number of contributions each developer has made to a repository.

We design a model similar to the one depicted in Figure 4, but without the Sigmoid layer, to predict the number of contributions between developers and repositories. This task is also known as an edge regression task. Specifically, for a contribution relationship e between a developer node u and a repository node v, the edge e is assigned a weight w_e, indicating the number of times the developer has contributed to the repository. Because the value range of w_e is too large, we apply a logarithmic processing on w_e, denoted as $w'_e = \log(w_e)$.

3.3.3 Metapath2Vec Embedding. Training the GNNs in mini-batches fails to capture the global structural information of the graph. To address this issue, we use Metapath2Vec [17] to obtain the initial structural embeddings of the nodes. These embeddings are then combined with the initial technical expertise embeddings and used as inputs to the GNN.

Metapath2Vec is a method designed for learning node embeddings in heterogeneous information networks (HINs). Its core idea is to capture the global structural information of nodes through the use of random walks. A heterogeneous information network is typically defined as $G = (V, E, T)$, where V, E and T represent the sets of nodes, edges and types, respectively. For each element $v \in V$ and $e \in E$, there exist mappings $\phi(v) : V \longrightarrow T_V$ and $\gamma(e) : E \longrightarrow T_E$. Here, T_V and T_E represent the types of nodes and edges, respectively. Any graph with more than one type of node or edge is considered a HIN. In a HIN, a meta-path can be usually represented as:

$$V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots \xrightarrow{R_{l-1}} V_l \tag{4}$$

where l denotes the length of the meta-path, and $R_i,\ i \in \{1, 2, \cdots, l-1\}$ represents the edge type connecting nodes of types V_i and V_{i+1}. The random walk strategy used by Metapath2Vec is defined by the following transition probability function at the i-th step:

$$p(v^{i+1} | \, v_t^i) = \begin{cases} \frac{1}{|\mathcal{N}_{t+1}(v_t^i)|} & (v_t^i, v^{i+1}) \in E,\ \phi(v^{i+1}) = t+1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\mathcal{N}_{t+1}(v_t^i)$ denotes the set of neighbors of v_t^i and with type $t+1$. After obtaining the randomly sampled node sequences, Metapath2Vec employs the Skip-Gram model to learn the embedding vectors of nodes. This model uses the context generated by the meta-path-guided random walks to optimize the node embeddings, capturing the structural information of the HIN.

3.4 Social Relationship Recommendation Tasks

This section outlines the practical implications of the user representations produced by our model and explores how developers can integrate these embeddings into their workflows. We focus primarily on key forms of collaboration and social activities on GitHub. For example, developers often seek peers with similar skill sets for

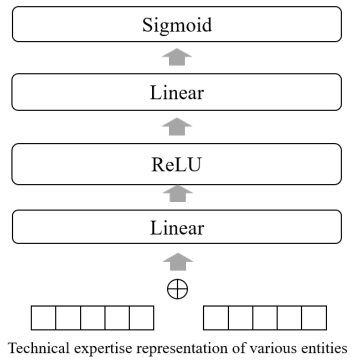


Fig. 5. Scoring model architecture for social relationship recommendations tasks

learning, knowledge exchange, or organizational involvement. Likewise, individuals aspiring to contribute to open-source projects need to identify repositories that align with their expertise, while individual repositories face challenges in attracting suitable maintainers and expanding their communities.

To address these scenarios, we have designed three distinct recommendation tasks and trained dedicated scoring models based on various entity representations. These downstream tasks not only provide practical benefits but also serve as evaluation metrics for assessing the effectiveness of the technical expertise representations generated by our model. Figure 5 illustrates the architecture of our scoring model, offering a clear overview of the integration process and the workflow enhancements enabled by our approach.

For various GitHub social relationship recommendation tasks, we concatenate the technical expertise representation vectors of related entities, and pass them through two fully connected layers to transform their dimensions. A ReLU activation layer is inserted between these two layers to introduce non-linearity. The transformed vectors are then processed through a Sigmoid function to output the predicted label, indicating the likelihood of the recommended relationship. It is important to note that the parameters of the GNN model are not updated during the training of the scoring model, ensuring the consistency and stability of the technical expertise embeddings while the scoring model learns to optimize the recommendations based on these embeddings.

3.4.1 Similar GitHub Developer Recommendation. Recommending developers with similar expertise can help developers find suitable organizations, and promote communication among developers. We abbreviate this task as "SimDeveloper". To train a scoring model for this task, we use GitHub Restful API to collect pairs of developers who either belong to the same organization or do not, serving as positive and negative samples, respectively, in equal amounts. We assign similarity scores of 1 to pairs from the same organization and 0 to those from different organizations. The model is trained using binary cross-entropy loss and outputs a technical similarity score for any two developers. We then build a recommendation system to rank the top 20 most similar developers for a given target developer. Candidates for recommendation include developers who have contributed to the same repository as the target developer. Developers who are in the same organization as the target developer are considered similar developers.

3.4.2 Repository Contribution Recommendation. We address the task of recommending repositories that align with developers' technical expertise for contributions, referred to as "ContributionRepo". We train a scoring model using binary cross-entropy loss to compute the matching degree for each <developer, repository> pair. Positive samples consist of developer-repository pairs with a history of watching and contributing, while negative samples are pairs with no such relationship, both in equal amounts. We assign matching degrees of 1 to positive samples and 0 to negative samples. The model then ranks repositories based on their matching scores, selecting the top 20 repositories for each developer. We only consider repositories that are watched, contributed, starred, or followed by target developer and their followers as candidates for recommendation. Repositories that the target developer has made contributions to are considered valid recommendations.

3.4.3 Repository Maintainer Recommendation. "RepoMaintainer" refers to the task of recommending open-source repository maintainers who can support the healthy and vigorous growth of the repository. We train a scoring model using binary cross-entropy loss to measure the technical expertise matching degree between developers and repositories, as shown in Figure 5. For each repository, we use its owner as a positive sample and other developers as negative samples, with equal amounts. We assign matching degrees of 1 and 0 to these samples respectively. The model ranks developers based on their matching scores and selects the top 20 for each repository. We consider only developers who have contributed to the repository as candidates, with the actual maintainer serving as a valid recommendation.

For fair evaluation, we filter out developers and repositories with minimal social activities, because they provide limited information for recommendation and have much noise information. Specifically, we apply the following task-specific filtering rules to remove these entities, ensuring a more robust evaluation.

- For SimDeveloper, we consider only developers who have at least 5 colleagues in the same organization as target developers for recommendation.
- For ContributionRepo, we consider only repositories with at least 5 contributions from the target developer.
- For RepoMaintainer, we only consider repositories that have at least 10 contributors.

4 EVALUATION

This section evaluates the effectiveness of our model in automatically deriving technical expertise representations. First, we demonstrate that our GNN-based architecture outperforms several state-of-the-art (SOTA) baselines, including non-GNN-based methods such as Dev2Vec [11], Llama-3 [18], Topic Method [59] and Collaborative Filtering [42], as well as GNN-based method such as GAT [34]. Second, to assess the impact of incorporating social features in modeling developers' technical expertise, we conduct an ablation experiment on the social relationship modeling process. Third, after confirming the superiority of our GNN-based approach, we compare our GraphSAGE model with alternative GNN architectures to identify the optimal configuration. Fourth, we perform additional ablation experiments to validate the necessity of each module within our framework. Finally, we employ statistical methods to evaluate how effectively our approach captures differences between entities and demonstrates robust generalizability. This comprehensive evaluation underscores the strength of our model in reflecting nuanced technical expertise across various tasks and configurations.

Therefore, we propose the following five research questions. The first four are progressive, aiming to find the optimal architecture and model, and the fifth RQ provides an assessment from the perspective of statistics.

- RQ1: How does our model perform on recommendation tasks compared to other popular baselines?
- RQ2: What is the effect of social features on deriving technical expertise of developers?
- RQ3: How does our model perform on social relationship modeling compared to other GNN-based models?
- RQ4: How does each module of our model affect its performance?
- RQ5: How effectively does our approach extract developer technical representation?

4.1 Comparisons with Baselines on Recommendation Tasks (RQ1)

To verify the utility of our derived technical expertise representation, we benchmark our approach against established baselines across various recommendation tasks. These downstream applications encompass both scoring models and comprehensive recommendation systems. This comparative analysis serves to demonstrate the practical effectiveness of our method in real-world scenarios.

4.1.1 Experiment Settings. To evaluate our method, we compare its performance on recommendation tasks with five baseline approaches in the field of repository mining and recommendation.

- Dev2Vec Method [11]: This approach generates technical embedding for developers by directly combining the embeddings of relevant GitHub repositories, issues and APIs using Doc2Vec.
- Llama-3 Method [18]: In this method, Llama3-8B is used to embed all text data from GitHub repositories, issues and APIs. The resulting embeddings are then combined directly to represent developers' technical expertise.
- Topic Method [59]: Following this approach, we conduct technical vectors for both repositories and developers based on GitHub topics. Topics with an occurrence frequency of less than 10 are filtered out to remove infrequent and less informative topics.
- GAT method [34]: This baseline also leverages social network information, aiming to identify distinct user behavior patterns by learning heterogeneous relationship features. We adapt this method by dividing our original graph into two subgraphs with one containing the Follow relationship as the social graph, and another one comprising all other relationships as the interaction graph.
- Collaborative Filtering (CF) [42]: A traditional approach for recommendation tasks. We implement a neighbor-based collaborative filtering recommendation system using the data from our GitHub social activity graph.

Since the first two baselines (Dev2Vec and Llama-3) can only generate embeddings for developers, they are limited to the SimDeveloper task. The other baselines can be applied to all recommendation tasks. Additionally, as collaborative filtering is not an embedding-based approach, it does not need trained scoring models.

Moreover, given that social activity relationships and developer activities on GitHub are highly dynamic and embeddings need frequently regenerated, we need to keep a low runtime cost. For methods incorporating Large Language Models (LLMs), although their powerful comprehension ability may yield better results, they also significantly increase the runtime cost. For example, it is observed in the following experiments that when generating code embeddings, Llama3-8B takes about 121̃5 times longer than our method using GraphCodeBERT. Therefore, to meet the performance-cost balance, we choose to compare with Llama3-8B only, instead of more expensive code-aware LLMs.

4.1.2 Metric Selection. As the scoring model in this paper is equivalent to a binary classification model, we use Precision, Recall, and F1 as evaluation metrics. To verify the effectiveness of the recommendation system, we employ Hit Ratio@K (abbreviated as HR@K) and MRR (Mean Reciprocal Rank) as metrics. HR@K measures the proportion of desired items among the top K results produced by the recommendation system, while MRR evaluates the reciprocal rank of the desired items in the recommendation system's output. These metrics provide a comprehensive evaluation of both the classification and recommendation performance of our models. They are defined as follows:

$$\begin{aligned}\text{Hit Ratio@}K &= \frac{1}{N}\sum_{i=1}^{N} I(r_i)\\ \text{MRR} &= \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}(r_i)}\end{aligned} \tag{6}$$

where N is the number of samples, r_i is the target item of sample i, $\text{rank}(r_i)$ is the rank of the target item in the recommendation system output. If r_i is not in the recommendation system's output, then $\text{rank}(r_i)$ is set to ∞. $I(r_i)$ represents whether the target item is in the top K results, with a value of 1 if it is in the recommendation system's output and 0 if it is not. The recorded metrics are computed on the validation and test datasets. Moreover,

Table 4. Performance of the Scoring Models

Task	Configuration	Precision	Recall	F1
SimDeveloper	Topic Method	0.849	0.635	0.727
	GAT Method	0.770	**0.961**	0.855
	Llama-3 Method	0.834	0.873	0.853
	Dev2Vec Method	0.797	0.836	0.816
	Our Method	**0.882**	0.931	**0.906**
ContributionRepo	Topic Method	0.790	0.887	0.835
	GAT Method	0.833	0.924	0.876
	Our Method	**0.885**	**0.938**	**0.910**
RepoMaintainer	Topic Method	0.886	0.597	0.713
	GAT Method	0.890	0.919	0.904
	Our Method	**0.940**	**0.959**	**0.949**

to ensure a robust evaluation across different scoring models, we implement 10-fold cross-validation on each recommendation task.

4.1.3 Experiment Results of Recommendation Tasks. Table 4 presents the performance of scoring models. It is clear that our method achieves the best results across most metrics. For F1 score, compared with the best baseline for each recommendation task, our method improves by 5.1% on SimDeveloper, 3.4% on ContributionRepo, and 4.5% on RepoMaintainer. This demonstrates that our derived representation of developers' technical expertise is highly adaptable to various downstream recommendation tasks. We attribute this improvement to our method of modeling repositories, issues and PRs as individual entities, and incorporating social features into developers' representations, rather than simply treating them as extensions of a developer's technical attributes.

Table 5 shows the recommendation performance across three recommendation tasks. Our method consistently achieves the best performance on the metric HR@10, with improvements of 5.6% on SimDeveloper, 9.5% on RepoMaintainer.

Notably, our method also surpasses GAT in all recommendation tasks. This superiority can be explained by the fact that performing the Metatpath2Vec module without the Follow relationship may cause performance degradation, which cannot be compensated by the attention mechanism in GAT.

It is important to note that the topic method has a limitation: it suffers from data leakage in some tasks. For example, in RepoMaintainer and ContributionRepo, the topic method uses the topics of the repositories that a developer has contributed to as their technical expertise. This approach leads to an artificially high similarity in topic distribution between the developer and their contributed repositories, potentially inflating the performance of the topic method. Despite this unfair advantage, our method still outperforms the topic method in almost all scenarios.

The main drawback of the CF method is its susceptibility to the cold start problem, particularly when little information is available about the recommendation target. This limitation is especially pronounced in the RepoMaintainer task, where the CF method fails to provide any recommendations due to severe cold start issues. Conversely, for tasks like SimDeveloper, where abundant interaction information about the targets is available, the CF method performs considerably better. This improved performance in such scenarios may be attributed to the CF method's tendency to focus on popular items, with popular developers typically attracting more followers.

Table 5. Performance on Recommendation Tasks

Task	Configuration	HR@1	HR@3	HR@10	MRR
SimDeveloper	CF Method	0.340	0.473	0.541	0.414
	Topic Method	0.390	0.682	0.897	0.560
	GAT Method	0.333	0.609	0.839	0.502
	Our Method	**0.439**	**0.741**	**0.953**	**0.615**
ContributionRepo	CF Method	0.191	0.333	0.500	0.288
	Topic Method	**0.310**	0.429	0.714	0.421
	GAT Method	0.167	0.405	**0.810**	0.332
	Our Method	**0.310**	**0.595**	**0.810**	**0.477**
RepoMaintainer	CF Method	–	–	–	–
	Topic Method	**0.400**	0.657	0.857	0.561
	GAT Method	0.029	0.229	0.714	0.200
	Our Method	0.393	**0.710**	**0.952**	**0.587**

By incorporating the social features of different types of entities within the social activity graph, our method derives more expressive and effective representations than non-GNN-based methods, including simple embedding-based methods (Dev2Vec, Topic method), a transformer-based method (Llama-3 method), and CF-based method. Furthermore, our proposed method also outperforms other GNN-based SOTA baseline (GAT method), showing significant improvements and exhibiting superior adaptability, robustness and effectiveness across tasks. Moreover, these findings indicate the generalizability of our method: the learned representations are not limited to a specific task. Rather, our method demonstrates strong generalizability across various recommendation tasks encompassing most open-source collaboration scenarios. In doing so, our method also successfully addresses the limitations of both topic-based and CF-based approaches, offering a more comprehensive and versatile solution for modeling technical expertise within the GitHub ecosystem.

Answer to RQ1: Our GNN-based model outperforms all baseline methods, including both non-GNN-based and GNN-based methods. This demonstrates the effectiveness of our proposed GNN-based architecture in deriving high-quality technical expertise representations.

4.2 Ablation Study on GitHub Social Relationship Modeling (RQ2)

GitHub social relationships capture the connections among developers and between developers and other entities, facilitating more effective propagation of technical representations to relevant GitHub entities, thus establishing appropriate expertise embeddings for developers and other entities. To demonstrate the necessity of incorporating social relationships, we construct an ablation study comparing our approach to methods that reduce or eliminate the effects of social networks.

4.2.1 Experiment Settings. We reduce the effect of the social network by generating entity representations by averaging, an approach inspired by [11]. To get developers' technical expertise embeddings, we utilize GNN constructed based on the GitHub social network, as described in Section 3.3.1. To ablate the impact of the GitHub social network, for each developer, we first find all PRs, repositories, and issues they are related to. Since we use code, natural language, and discrete data to generate initial technical expertise, and the modality type varies among different entities, we average all the related entities' embedding vectors, grouped by each modality for each developer, respectively. Then we concatenate these vectors from different modalities, the resulting vector, represents each developer. The developer representation can be formulated as follows:

$$\begin{aligned} v^i_{\text{dev, code}} &= \sum_{j \in \mathcal{N}_{\text{pr}}(i)} v^j_{\text{pr, code}} + \sum_{j \in \mathcal{N}_{\text{issue}}(i)} v^j_{\text{issue, code}} + \sum_{j \in \mathcal{N}_{\text{repo}}(i)} v^j_{\text{repo, code}} \\ v^i_{\text{nl, code}} &= \sum_{j \in \mathcal{N}_{\text{pr}}(i)} v^j_{\text{pr, nl}} + \sum_{j \in \mathcal{N}_{\text{issue}}(i)} v^j_{\text{issue, nl}} + \sum_{j \in \mathcal{N}_{\text{repo}}(i)} v^j_{\text{repo, nl}} \\ v^i_{\text{dev, dis}} &= \sum_{j \in \mathcal{N}_{\text{pr}}(i)} v^j_{\text{pr, dis}} + \sum_{j \in \mathcal{N}_{\text{issue}}(i)} v^j_{\text{issue, dis}} + \sum_{j \in \mathcal{N}_{\text{repo}}(i)} v^j_{\text{repo, dis}} \\ v^i_{\text{dev}} &= v^i_{\text{dev, code}} \parallel v^i_{\text{dev, nl}} \parallel v^i_{\text{dev, dis}} \end{aligned} \tag{7}$$

where $v^i_{\text{dev, code}}, v^i_{\text{dev, code}}, v^i_{\text{dev, code}}$ denote the code, natural language and discrete modality representation of developer i, respectively. $\mathcal{N}_{\text{pr}}(i)$ denotes the set of all neighbor PRs of developer i, $\parallel$ denotes the concatenate operation.

We demonstrate the effects of the ablation by examining the performance on three downstream tasks, using the constructed developer representations and initial technical expertise representations of other entities. We also constructed an additional embedding dataset by concatenating metapath vectors with the original embeddings. As in RQ1, we employ HR@K and MRR as the evaluation metrics for the recommendation system.

Table 6. Performance comparison of Recommendation Tasks on Our Method and Averaging Method

Task	Configuration	HR@1	HR@3	HR@10	MRR
SimDeveloper	No Social Network	0.340	0.717	0.927	0.583
	No Social Network(w/o mp)	0.355	0.657	0.911	0.541
	Our Method	**0.439**	**0.741**	**0.953**	**0.615**
ContributionRepo	No Social Network	0.286	0.476	0.786	0.438
	No Social Network(w/o mp)	0.286	0.476	0.786	0.438
	Our Method	**0.310**	**0.595**	**0.810**	**0.477**
RepoMaintainer	No Social Network	0.343	0.543	0.943	0.512
	No SOcial Network(w/o mp)	0.314	0.429	0.829	0.436
	Our Method	0.393	**0.710**	**0.952**	**0.587**

4.2.2 Experiment Results. Table 6 shows the performance of 3 scoring models. The absence of the GitHub social network leads to a significant decrease in performance across recommendation tasks. For the MRR score, this absence leads to a decline of at least 3.2% on SimDeveloper, 3.9% on ContributionRepo, and 7.4% on RepoMaintainer. This underperformance suggests that the absence of the social network limits the model's ability to learn the social features of GitHub entities. Compared to directly averaging all other entities related to developers, the better performance of our approach suggests that social network is not just a way to aggregate entity representations, but rather a means to more broadly and deeply diffuse and integrate the embeddings between developers, as well as between developers and other entities. This enables the generated embeddings to more accurately express the implicit technical characteristics of entities, thereby achieving better results in downstream recommendation tasks. These experimental results highlight the importance of social networks in developer recommendation tasks.

Answer to RQ2: The GitHub social features successfully captures the social collaboration information between entities, effectively enhancing the representation of developers' technical expertise, and improving model's performance on recommendation tasks.

4.3 Comparisons between GNN models (RQ3)

This paper presents a GNN model designed to capture the complex social relationships among diverse entities on GitHub. We leverage this GNN model to transform initial technical expertise representation into final, comprehensive representation that seamlessly integrating both social features and technical features. The initial technical expertise is obtained through pre-trained models, allowing us to our evaluation on GNN model's performance in refining and enhancing these representations.

4.3.1 Experiment Settings. We implement the GNN described in Section 3.3.1 naming it HetSAGE. To compare it with other GNN models, we replace the graph convolutional layer with those introduced in GCN [31], GAT [49], RGCN [45], and HGT [30]. These corresponding GNN models are named HetGCN, HetGAT, RGCN, and HGT. All models are implemented using Pytorch [43] and the dgl [52]. We use 3 graph convolutional layers with a neighbor sampling order of 3 and an output dimension of 512. We set the learning rate to 1e-4, the training epoch to 40, and the batch size to 512, using the Adam optimizer. To prevent overfitting, we apply dropout with a probability of 0.2 to all non-last graph convolutional layers and include weight decay with a coefficient of 1e-3 in the loss function, excluding bias parameters.

We partition the GitHub data set into 8:1:1 ratio for training, validation, and testing. For the link prediction task, we use all edges in the GitHub graph. However, for the edge regression task, we used only the Commit edges. The following results are derived from the test set.

4.3.2 Metric Selection. For the link prediction task, which is a binary classification task, we use Precision, Recall and F1 as the evaluation metrics. For the edge regression task, we use the mean absolute error (MAE) and root mean squared error (RMSE) as the evaluation metrics, defined as follows:

$$\begin{aligned} \text{MAE} &= \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| \\ \text{RMSE} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \end{aligned} \tag{8}$$

where N represents the number of edges, y_i represents the actual value of the i-th edge, and $\hat{y}_i$ represents the predicted value of the i-th edge.

4.3.3 Experiment Results. The results are shown in Table 7. In the link prediction task, HetSAGE model outperforms other GNN models selected in this paper. In the edge regression task, the HetSAGE model also performs best. Note that in the data set of the edge regression task, the average value of the label is 3.968, while the MAE is 0.9720, and the RMSE is 1.257. These metrics illustrate that the HetSAGE model in this paper performs well on the edge regression task.

Answer to RQ3: The HetSAGE model based on GraphSAGE achieves the best performance among all GNN models. Therefore, we choose to leverage HetSAGE model in our proposed architecture.

4.4 Effect of Model Modules (RQ4)

4.4.1 Experiment Settings. We conduct ablation experiments to assess the impact of each module on the model's performance. The primary modules are the initial technical expertise module (described in section 3.2) and the

Table 7. GNN performance comparison on GitHub social activity graph

Model	**Link Prediction**			**Edge Regression**	
	Precision ↑	**Recall ↑**	**F1 ↑**	**MAE ↓**	**RMSE ↓**
HetGCN	0.8433	0.8556	0.8494	1.0430	1.3090
HetGAT	0.8246	**0.9245**	0.8717	1.0034	1.2812
HetSAGE	**0.9206**	0.8477	**0.8826**	**0.9720**	**1.2565**
RGCN	0.8719	0.8892	0.8805	0.9843	1.2622
HGT	0.8805	0.8835	0.8820	1.0100	1.2820

Table 8. Ablation study on the HetSAGE model

Model	**Link Prediction**			**Edge Regression**	
	Precision ↑	**Recall ↑**	**F1 ↑**	**MAE ↓**	**RMSE ↓**
HetSAGE-np-mp	0.9086	0.7898	0.8450	1.0364	1.2997
HetSAGE-only-mp	0.9143	0.8432	0.8773	1.0097	1.2868
HetSAGE-without-tech	**0.9277**	0.8256	0.8523	1.0892	1.4246
HetSAGE	0.9206	**0.8477**	**0.8826**	**0.9720**	**1.2565**

Metapath2Vec module which captures global structure (outlined in section 3.3.3). The evaluated models in this section are:

- **HetSAGE**: The final model used in this paper.
- **HetSAGE-no-mp**: We remove Metapath2Vec module from HetSAGE. Because developer entities do not have initial technical expertise representation, we randomly initialize the embedding vector for developer entities.
- **HetSAGE-only-mp**: We remove the initial technical expertise mining module, and use only the embeddings generated by Metapath2Vec as the input for the GNN model.
- **HetSAGE-without-tech**: We train the same model from HetSAGE, but randomly initialize embedding vector for both technical expertise representation and the embeddings generated by Metapath2Vec.

All other settings remain the same as in Section 4.2.1.

We conduct another experiment to evaluate the necessity of the Watch relationship in GitHub's social relationships. In this experiment, we remove the Watch relationship from the social graph and train the Metapath2Vec model without the Watch relationship. We then compare the performance of this modified model with the original model on downstream recommendation tasks.

4.4.2 Experiment Results. The results in Table 8 indicate that both HetSAGE-no-mp and HetSAGE-only-mp models underperform compared to the complete model. This underperformance suggests that the absence of the initial technical expertise mining module constrains the model to learning solely the social features of GitHub entities, failing to extract technical features. This deficiency leads to significantly poorer performance on edge regression tasks. For the HetSAGE-no-mp model specifically, the omission of the Metapath2Vec embedding module results in substantial performance degradation in both link prediction tasks and edge regression tasks. Furthermore, Table 9 illustrates a significant decline in our method's performance on ContributionRepo and SimDeveloper tasks when the Watch relationship is excluded. This decline underscores the critical role of the

Table 9. Ablation study on the GNN watch relationship

Task	Configuration	Precision	Recall	F1
SimDeveloper	with watch	**0.882**	**0.931**	**0.906**
	without watch	0.816	0.920	0.865
ContributionRepo	with watch	**0.885**	**0.938**	**0.910**
	without watch	0.676	0.693	0.683
RepoMaintainer	Our Method	**0.940**	**0.959**	**0.949**
	without watch	0.832	0.909	0.869

Watch relationship in capturing developers' interests and expertise, highlighting its importance within the GitHub social graph.

Answer to RQ4: All of the initial technical expertise embedding, Metapath2Vec embedding, and GitHub social relationships provide indispensable information for the final representation. The removal of any of these modules leads to performance degradation.

4.5 Unsupervised Evaluation of the Technical Expertise Representation (RQ5)

4.5.1 Experiment Settings. To validate the effectiveness of our approach in deriving developer's technical representation, we hypothesize that similar entities should exhibit comparable representations. To test this hypothesis, we follow [12] in employing the t-test, a robust statistical method that assesses significant deviations between two sets of data. Our analysis involves calculating the p-value of similarity for embeddings of similar entities compared to randomly chosen embeddings. This p-value serves as a measure of statistical significance, with smaller values indicating more substantial differences. Specifically, we design two unsupervised tasks, following the methodology outlined in [12]. Note that we make an implicit assumption that similarities between all types of entities follow a normal distribution. We consider this assumption reasonable for all entities that can be deemed generally independent.

Task 1 examines whether GitHub developers tend to join repositories that align with their expertise. For this task, we gather all <developer, repository> pairs where the developer has contributed at least 30 times to the repository as positive samples. We randomly select repositories with the same primary programming language but without any contribution from the developer as negative samples.

Task 2 examines whether developers contribute to repositories that match their technical expertise. For this task, we use all <developer, repository> pairs where the developer has submitted code to the repository as positive samples. We randomly select a repository with no contribution from the developer as a negative sample.

For both tasks, we employ three metrics to calculate the similarity between pairs: cosine similarity, adjusted cosine similarity and Pearson correlation coefficient. Our hypothesis predicts that positive samples will exhibit higher similarity scores across these metrics compared to negative samples.

Table 10. T-test results

distance similarity	cosine	adjusted cosine	Pearson
Task 1 P-value	2.235e-9	3.939e-8	2.464e-13
Task 2 P-value	9.918e-8	1.066e-5	1.515e-9

4.5.2 Experiment Results. The results of the T-test are shown in Table 10. A P-value less than 0.05 typically indicates a significant difference in the mean between two distributions. In our two unsupervised tasks and three distance similarity calculation schemes, the P-value is significantly lower than 0.05. These results provide strong evidence that the technical expertise representation vectors derived by our model effectively capture and preserve the inherent differences between entities. The consistency of these significant results across multiple tests reinforces the robustness and reliability of our approach in generating meaningful technical expertise representations.

Answer to RQ5: Our approach is effective to derive representations that captures and preserves the inherent similarities and differences between entities.

5 USER EVALUATION

While our experimental metrics offer a comprehensive evaluation of various approaches, they are inherently constrained by the specific rules used in dataset construction. To address this limitation and gain a more holistic understanding of our approach's effectiveness, we conducted a user evaluation study. This user-centric assessment complements our quantitative analysis by comparing approaches from the perspective of recommendation system users.

5.1 User Evaluation Settings

Our objective is to evaluate the quality of generated recommendations from the users' perspective. To measure the quality, we mainly consider two aspects of the recommendation results as follows:

- Does the recommendation provide helpful candidates in practical scenarios?
- Does the recommendation recommend entities with similar technical expertise?

Specifically, we leverage the test data from three tasks defined in previous experiments, i.e. SimDeveloper, ContributionRepo and RepoMaintainer. For each task and each model, including our model, CF method, Topic method, GAT method and Dev2Vec method, we randomly select 3 test instances from original test dataset. Since the Dev2Vec method does not work on ContributionRepo and RepoMaintainer task, the total number of the task instances is 39(3*(5+4+4)). We then collect the recommendation predictions of each model. Totally, this sample involves 33 repositories and 63 developers.

We recruit 10 participants for the user evaluation, including 4 Master's students, 3 Ph.D. students, and 3 experienced software engineers. All participants are active GitHub users with 3 to over 10 years of development experience, and nine have contributed to open-source repositories on GitHub.

The 39 task instances described above are organized into an online questionnaire, with hyperlinks to relevant GitHub repositories and users. For each task instance, participants are required to explore the main content of the relevant repositories or the capabilities and experience of developers. This exploration is the key process for participants to gather information and make comprehensive evaluation. Therefore, we required each participant to complete a brief preliminary experiment before the formal experiment. After that, we interviewed each participant to confirm that they fully understood the task and had no remaining questions. The data used in preliminary experiment is guaranteed not to overlap with that used in the formal experiment.

After this process, participants rated each recommendation based on the two previously defined aspects (helpfulness and alignment), which correspond to two questions per task instance, as shown in Table 11. Participants were required to use a 5-point scale to answer these questions, and were informed that 1 indicates the recommendation was completely unsuccessful while 5 indicates the recommendation was completely successful.

Table 11. Questionnaire of User Evaluation

Task	Questions
SimDeveloper	The following consists of two GitHub users with hyperlinks to their homepages. Please look through their information for full understanding, and answer these two questions: 1. To what extent they are similar in working experience. 2. To what extent they are similar in technical stack.
ContributionRepo	The following consists of a GitHub user and repository with hyperlinks to their homepages. Please look through their information for full understanding, and answer these two questions: 1. To what extent you will recommend the user to contribute to the repository. 2. To what extent the user has similar developing experience to the repository.
RepoMaintainer	The following consists of a GitHub user and repository with hyperlinks to their homepages. Please look through their information for full understanding, and answer these two questions: 1. To what extent you will recommend the user to be the maintainer of the repository. 2. To what extent the user has relevant experience and capability as a maintainer.

5.2 Results

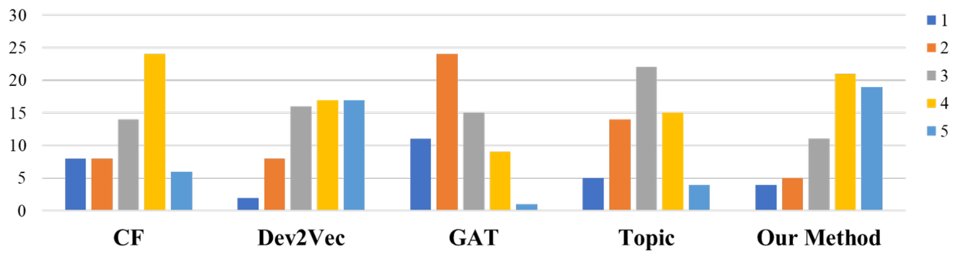
Figure 6 illustrates the distribution of user evaluation scores. The Y-axis represents the number of ratings that each model receives on each task. Since we sampled 3 instances for each model-task pair, with 2 questions per instance and 10 participants, the total count of each model-task pair sums up to 60 (3*2*10). Overall, our method consistently achieves higher ratings across all three tasks, compared with all other methods. These results prove the effectiveness of our proposed method to incorporate GitHub social features into technical expertise representation. In addition, noting that all methods received significantly higher scores in RepoMaintainer task compared to the other two tasks, we attribute this to the fact that all methods are generally capable of capturing basic contribution history, and simply recommending the main contributors of a repository as potential maintainers aligns with participants' expectations when participants lack relevant information.

Furthermore, we also report summary statistics for each model's scores. Since these scores should be regarded as ordinal, we use the median, IQR and mode to reflect the overall performance and dispersion of each model on each task. IQR is defined as follows:

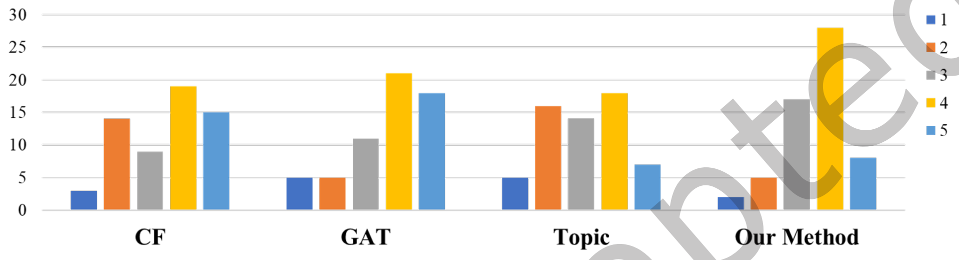
$$\text{IQR} = Q_3 - Q_1 \tag{9}$$

where Q_3 is the third quartile and Q_1 is the first quartile.

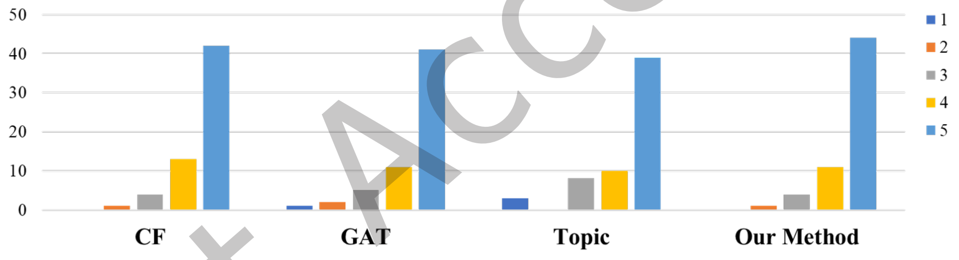
In addition, we conducted both T-tests and ICC tests to ensure the validity of statistical analysis. For the T-tests, we combined the scores of our model across the three tasks and compared them with the corresponding scores of each baseline model to examine whether there are significant differences. One-tailed and paired T-test were



(a) User evaluation results on SimDeveloper task



(b) User evaluation results on ContributionRepo task



(c) User evaluation results on RepoMaintainer task

Fig. 6. Distribution of user evaluation results

conducted with the alternative hypothesis that the mean of our method's scores is significantly higher. Since Dev2Vec is only applicable to the SimDeveloper task, the T-test with Dev2Vec only used scores on one task.

On the other hand, ICC analysis was conducted to assess the reliability of rating scores provided by participants. It evaluates the degree of consistency and agreement among raters by comparing the variability of rating attributed to individual raters with the total variability of ratings. For our ICC analysis, we employed a two-way mixed-effects model with absolute agreement (ICC-3k), which is appropriate for evaluating the reliability of ratings when the same set of raters evaluates all targets [46].

Table 12 shows the statistical results of the user evaluation. Our method achieves the best performance across all three tasks using median, mode and IQR metrics. The only exception is that the GAT method performs a lower IQR in the SimDeveloper task. However, it has a much lower median and mode in this task, indicating less

Table 12. Statistics of User Evaluation Results

Model	**Task**	**Median↑**	**Mode↑**	**IQR↓**	**T-test with our methods**	**ICC coefficient**
CF Method	SimDeveloper	3.5	4.0	2.0	P<0.01 (df=179)	–
	ContributionRepo	4.0	4.0	2.5		–
	RepoMaintainer	5.0	5.0	1.0		–
Topic Method	SimDeveloper	3.0	3.0	2.0	P<0.01 (df=179)	–
	ContributionRepo	3.0	4.0	2.0		–
	RepoMaintainer	5.0	5.0	1.0		–
GAT Method	SimDeveloper	4.0	2.0	**1.0**	P<0.01 (df=179)	–
	ContributionRepo	2.0	4.0	2.0		–
	RepoMaintainer	5.0	5.0	1.0		–
Dev2Vec Method	SimDeveloper	4.0	4.0	2.0	P=0.28 (df=59)	–
Our Method	SimDeveloper	**4.0**	**4.0**	2.0	–	–
	ContributionRepo	**4.0**	**4.0**	**1.0**		–
	RepoMaintainer	**5.0**	**5.0**	**1.0**		–
ICC						0.88(>0.75)

Note: For the T-test results, df is short for degree of freedom, which is the total number of scores minus one in paired T-test. For clarity, when multiple values share the optimal score, only one representative value is highlighted in bolded.

favorable overall performance. Therefore, we conclude that our method outperforms all baselines in the user evaluation.

Moreover, the results of T-test demonstrate that the differences between scores of our method and each baseline model are statistically significant. Despite the higher p-value in the T-test with Dev2Vec method, our method still achieves an equal median and mode, and a better IQR on this task. Also, as Dev2Vec method is only applicable to one task, the smaller sample size (30 instances and 60 scores) may reduce the statistical power of the test.

Finally, the computed ICC value of 0.88 exceeds the conventional threshold of 0.75, indicating high reliability of rating scores. This finding confirms that the user ratings obtained in our study are consistent and reliable, thereby reinforcing the validity of our evaluation results.

In summary, these findings provide strong evidence that the technical expertise representations derived by our method generalize effectively across various recommendation tasks, achieving and in many cases surpassing the performance of state-of-the-art approaches. The robust and consistent performance can be attributed to the comprehensive nature of our approach, which effectively integrates both social and technical features. This versatility and consistency in performance highlight the potential of our method for practical applications within the GitHub ecosystem.

6 DISCUSSION

In this section, we discuss: 1) the key components of the model, including GitHub social relationships and model complexity, their impacts on performance, and the model's limitations and failure cases; 2) the interaction between our model and the open-source community, including the influence of community size on model performance and model's practical benefits for open-source communities.

6.1 Effects of Social Relationships in Deriving Developers' Expertise Representation

From a methodological perspective, the innovation of this paper lies in expanding beyond direct developer relationships to explore interactions between developers and other entities within the open-source community. By leveraging these interactions, we capture hidden relationships between developers, enabling more accurate modeling and representation of developer expertise vectors.

In the task SimDeveloper, we compared our method with other state-of-the-art models like Dev2Vec, and the popular LLM model Llama3-8B. Dev2Vec uses doc2vec as a foundation, which generates embedding vectors to represent developer documents, surpassing other traditional models like term frequency-inverse document frequency (tf-idf) and Bag of Word (BOW) in representation tasks; We feed the same corpus to Llama3. Pretrained on 15T tokens of corpus and a fine-grained tokenizer, Llama3 can capture small semantic differences, which enables it to generate high-quality representation vectors for developer corpus.

Despite their sophistication, these methods underperformed compared to our approach, as shown in Table 4. This underperformance stems from their lack of consideration for social relationships. Commonly, similar techniques have relatively large differences in their corpus. Let's consider "backend" as an example. Imagine two backend developers specializing on Java and Golang, respectively. The corpus related to these developers, such as repository codes, issues, and PRs, can differ semantically in terms of semantics. Conversely, the corpus related to two developers working on Java backend and frontend within a specific repository is highly relevant, despite their technical expertise differs more significantly than in the former scenario. This discrepancy indicates a key limitation: corpus similarities don't always correlate with technical expertise similarities.

Our method addresses this limitation by incorporating social relationships, enabling a more context-aware representation of developer expertise. Given a backend developer is working on both a Java and a Golang repository simultaneously. Our GNN method can capture relationships between developers working across multiple repositories (e.g., Java and Golang). This enhances the representation similarity for developers associated with either repository during training. While initial representations of Java frontend and backend techniques may be similar, they're aggregated into different technical scopes during training. The unsupervised nature of our GNN allows it to leverage latent technical relationships that might otherwise remain unnoticed.

6.2 The Impact of Model's Complexity on Performance

We mainly focus on the number of convolutional layers to discuss the complexity of the GNN model. Intuitively, it might be assumed that a GNN model with more graph convolution layers would produce better results. The rationale behind this assumption is that additional layers enable the model to capture more relationships and aggregate technical expertise from a broader network of contributors.

To test this assumption, we conduct a series of experiments. The experiment setup is almost the same as it is in RQ3. We fit the model to HetSAGE on the link prediction task. The neighbor sampling count is fit to 7, and the layer count varies from 2 to 4. The results are summarized in Table 13.

Table 13. Results on Different GNN layer count

# Layers	Accuracy	Precision	Recall	F1
2	0.8916	0.8940	0.8885	0.8912
3	0.8968	0.9057	0.8857	0.8956
4	0.8945	0.8789	0.9072	0.8929

Our experiments yielded surprising results: while the model with 4 convolutional layers outperformed others in recall, it showed significantly lower precision, with overall accuracy and F1 scores failing to demonstrate significant improvements. This counterintuitive outcome can be attributed to the structure of our heterogeneous graph. Most relevant social relationships can be effectively captured by paths of length 3, such as the connection between two developers where one commits to a repository and another proposes an issue for the same repository (represented by the path "developer propose_issue, issue belong_to, repository commit(reversed) developer"). This relationship is adequately captured by a model with 3 GNN layers. However, a 4-layer model may capture extraneous, less relevant relationships (e.g., two developers who starred different repositories that were both contributed to by a third developer), potentially introducing noise into the learning process and decreasing precision. Given that 4-layer models also require significantly more computational resources, we conclude that 3-layer models strike the optimal balance between performance and efficiency for capturing social relationships in our heterogeneous graph. This finding underscores the importance of tailoring model architecture to the specific characteristics of the underlying data, rather than assuming that increased complexity always yields better results. Therefore, more convolutional layers do not always produce better representation for contributors.

6.3 Case Study of Failures

Despite the fact that our method can efficiently derive developers' technical expertise representations in most common scenarios, it still deteriorates when confronting developers with limited GitHub social activity relationships. When deriving the technical expertise representations, in the initial step we only calculated embeddings for repository, issue and PR nodes. Subsequently, we utilize a GNN to compute embeddings of developers on the GitHub social activity graph. As a result, the effectiveness of the generated embedding is heavily dependent on the degree of the developer node within the graph, that is, the number GitHub social activities of a developer. For developer nodes with a low degree in the graph, there are fewer ways for relevant information to be propagated to these nodes, making it more challenging to accurately model their technical expertise representations. This issue, referred to as the cold start problem, remains to be a significant limitation of our method.

In practical scenarios, developer with limited GitHub social activity relationships may exhibit in the following situations:

- Contributing to only a small number of repositories.
- Utilizing uncommon programming languages.
- Focusing on legacy codebases or highly specialized projects that have minimal interactions with the GitHub community.

In this case study, we present a typical case to illustrate the scenarios mentioned above. For privacy concerns, developers and repositories involved have been anonymized.

In the SimDeveloper task, which aims to recommend developers with similar technical expertise for learning and collaboration, we analyzed User A who contributes to only four repositories, all within the same open-source software project (including its framework, configuration, documentation, and tutorial). This limited interaction diversity complicated the recommendation process. The model recommended User B, the project's lead developer and release manager.

While this recommendation aligned with our SimDeveloper assumption that developers within the same GitHub organization are similar, it oversimplified their relationship. Further analysis revealed that User A specializes in high-performance computing related to the software project, while User B focuses on microarchitecture and compiler optimization. Despite the connection through the same project, these fields require distinctly different skill sets, resulting in below-average user evaluation scores.

However, it's noteworthy that our model, leveraging GNN's message-passing mechanism and social features, still generated recommendations despite User A's limited GitHub presence. In contrast, traditional methods like collaborative filtering, which depend solely on direct user interactions, failed to produce recommendations within the search scope when confronted with such sparse user data.

6.4 Threats to Validity

Threats to Internal Validity. The effectiveness of the model may heavily rely on the quality and comprehensiveness of the GitHub data used, which could potentially be subject to bias or poor quality. To address this concern, our dataset is built based on [10], in which the authors detailed their methodologies for bias elimination. Furthermore, we have implemented additional measures to mitigate the impact of low-quality data, such as removing bot developers to ensure human-generated content, and adopting several pre-trained language models (PLMs) like RoBERTa to encode data of varying qualities. These PLMs are trained on large corpora and can extract the main semantics of the data, thus reducing the impact of low-quality data. Besides, the approach assumes that function definitions encapsulate the primary code semantics for all languages in the dataset, which may not be fully held: critical logic may occur in top-level scripts, decorators, immediately invoked function expressions (IIFEs), static constructors, and inline scripts. Excluding these non-functional code elements might overlook significant aspects of developers' technical contributions, potentially skewing the accuracy of technical expertise representations.

Threats to External Validity. The primary threat to external validity stems from the dataset size: compared to GitHub's vast data volume, our dataset is relatively small, potentially limiting its representation of GitHub's overall data distribution. However, we have implemented strategic measures to mitigate this limitation. Firstly, we focused our research on the active segment of GitHub, specifically sampling repositories with recent changes. This approach ensures better alignment with current development practices and captures the most relevant data for our study. Additionally, we conducted multiple downstream tasks to validate our findings across various contexts. This multi-faceted validation approach enhances the generalizability of our results, providing a more robust assessment of our model's performance across different scenarios within the GitHub ecosystem. While we acknowledge that further expansion of the dataset could strengthen our findings, we believe our targeted sampling strategy and comprehensive evaluation approach significantly reduce the threat to external validity, offering valuable insights into developer expertise representation within active GitHub communities.

6.5 The Robustness and Cost of Model on Different Sizes of Communities

In our research, we define community size as the number of developer nodes in the GitHub social activity graph. One might intuitively expect that larger communities, with their richer network of developer relationships, would yield better GNN model training results and more effective entity embeddings, ultimately leading to superior performance on downstream tasks. This section examines our method's effectiveness across varying community sizes and analyzes the associated computational costs. To validate our approach's scalability, we conducted a series of experiments training the model on communities of different sizes and evaluated their performance.

We evaluated three different community size groups in our experiments: small, medium, and large. The original graph, referred to as the large graph, contained 394,474 developer nodes. For the small graph, we selected one-third of the developers, and for the middle graph, we chose two-thirds of the developers. When selecting developers, we ensured that those participating in downstream tasks were always retained within the community. Robustness was assessed based on the performance across the mentioned three downstream tasks, measured by HR@1, HR@3, HR@10 and MRR. To evaluate computational cost, we recorded training time and GPU memory usage. The experiments were conducted on a Ubuntu 20.04 machine, equipped with two Nvidia A100 40GB GPUs.

The results on the three downstream tasks are shown in Table 14.

It can be observed that for SimDeveloper and RepoMaintainer, the small graph achieved the best results, while for ContributionRepo, the medium graph performed the best. Additionally, all of them outperformed the baseline methods discussed in Section 4.1, demonstrating the robustness of our method.

The cost-related results are shown in Table 15.

Table 14. Downstream Task Results on Different Community Size

Task	Configuration	HR@1	HR@3	HR@10	MRR
SimDeveloper	small	**0.483**	**0.806**	**0.892**	**0.658**
	middle	0.438	0.764	0.869	0.621
	large	0.378	0.709	0.832	0.569
ContributionRepo	small	0.331	0.531	0.655	0.474
	middle	**0.355**	**0.638**	**0.767**	**0.530**
	large	0.333	0.631	0.736	0.512
RepoMaintainer	small	**0.609**	**0.860**	**0.940**	**0.749**
	middle	0.386	0.634	0.800	0.552
	large	0.303	0.574	0.709	0.482

Table 15. Performance Results on Different Community Size

Configuration	Total Train Time	GPU Memory Usage (MB)
small	23:03:51	15737.707
middle	32:56:20	19451:331
large	39:52:45	22893:461

As expected, training time and GPU memory usage increased with graph size. However, the increase was not in a simple 1:2:3 ratio, which may be due to the additional entities in the graph (e.g., PRs, issues, repositories) and the relationships between them.

Our analysis reveals intriguing patterns in model performance across different community sizes. The superior performance of smaller graphs in SimDeveloper and RepoMaintainer tasks suggests that compact communities can effectively capture essential task-specific information. Meanwhile, the medium-sized graph's optimal performance in ContributionRepo indicates that moderate-scale communities may provide more balanced information for certain tasks. The model demonstrates robustness by maintaining strong performance across various community sizes, consistently surpassing our previously established baselines.

The enhanced performance in smaller and medium-sized communities can be attributed to our developer selection strategy. When constructing these communities, we prioritized developers who actively participated in downstream tasks, using contribution levels as a selection criterion. Developers exceeding specific contribution thresholds were included in the downstream task datasets. The superior performance of smaller and medium-sized communities over the complete community suggests that relationship quality outweighs quantity in our GNN model's effectiveness. This indicates that focusing on core contributors can yield better results than expanding to a broader developer base—a finding that warrants further investigation in future research.

Regarding computational efficiency, we observed an expected increase in training time and GPU memory usage with community size. Larger graphs naturally demand more computational resources due to their increased nodes and relationships. However, the non-linear relationship between graph size and computational cost suggests potential optimization opportunities, such as more efficient sampling methods or model compression techniques.

In conclusion, our experiments demonstrate that model robustness isn't directly proportional to community size. Different downstream tasks achieved optimal performance at varying community scales. While computational costs increased with community size, these findings suggest that practical implementations should carefully balance community size against computational constraints to optimize performance for specific use cases.

6.6 Practical Benefits of Technical Expertise Embeddings in Open-Source Ecosystem

The practical implementation of technical expertise embeddings into developer and team workflows is essential for realizing our research's value. These embeddings serve multiple stakeholders in distinct yet complementary ways.

For individual developers. The embeddings act as a strategic tool for both professional growth and collaboration. They highlight in-demand technical skills within relevant open-source projects, guiding developers' skill development. Additionally, the embeddings also facilitate developers to connect with other developers who have similar technical profiles for collaboration and code review. Through GitHub actions, developers can set up automated notifications for skill-relevant discussions and issues, ensuring proactive engagement with emerging project developments.

For repository maintainers. These embeddings offer valuable data-driven insights to enhance project management. By matching technical requirements with developers' expertise, maintainers can assign tasks more effectively. For instance, when addressing complex issues that demand expertise in specific programming languages or frameworks, maintainers can identify the most qualified contributors based on their demonstrated skills. Additionally, this functionality allows maintainers to tracking contributors' skill development, supporting targeted mentorship and support.

For open-source teams. These embeddings offer a systematic approach to team development. Team leaders can analyze skill distribution to identify gaps and organize targeted training programs. Additionally, the embeddings assist in strategic recruitment by helping identify potential team members whose skills complement the existing expertise.

To maximize these benefits, we recommend the following implementation guidelines:

- Developers should systematically review embedding-based recommendations and establish targeted GitHub alerts for skill-related opportunities
- Repository maintainers should implement a structured system for skill-based task allocation and provide regular feedback based on embedding insights
- Open-source teams should develop comprehensive skill development strategies informed by embedding analysis and promote continuous professional growth

7 CONCLUSION AND FUTURE WORK

This paper aims to learn the representations of the technical expertise of developers on GitHub. Unlike existing studies that focus on specific technical areas and data sources, we propose a novel approach that integrates diverse GitHub data sources and social relationships using a GNN-based approach. Our approach generates technical expertise representations for GitHub entities, including developers, projects, and PRs. We leveraged these representations to design and evaluate three social relationship recommendation tasks: recommending maintainers, identifying similar users, suggesting relevant repositories. Experimental results demonstrate the effectiveness and adaptability of our approach across multiple baselines. Future work will focus on taking account for temporal aspects of the data, expanding the dataset to cover a wider range of programming languages, incorporating time-based weighting to prioritize recent activities, and applying the approach to additional recommendation tasks such as predicting issue closure and resolution times.

8 DATA AVAILABILITY

Our code is available at https://anonymous.4open.science/r/DTEM-6830/.

REFERENCES

[1] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2018. code2vec: learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3 (2018), 1 – 29. https://api.semanticscholar.org/CorpusID:4710028

[2] Shuotong Bai, Lei Liu, Huaxiao Liu, Mengxi Zhang, Chenkun Meng, and Peng Zhang. 2022. Find potential partners: A GitHub user recommendation method based on event data. *Information and Software Technology* 150 (2022), 106961. doi:10.1016/j.infsof.2022.106961

[3] Riyu Bana and Anuja Arora. 2018. Influence Indexing of Developers, Repositories, Technologies and Programming Languages on Social Coding Community GitHub. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*. 1–6. doi:10.1109/IC3.2018.8530644

[4] Blerina Bazelli, Abram Hindle, and Eleni Stroulia. 2013. On the Personality Traits of StackOverflow Users. In *2013 IEEE International Conference on Software Maintenance*. 460–463. doi:10.1109/ICSM.2013.72

[5] Ferran Borreguero, Elisabetta Di Nitto, Dmitrii Stebliuk, Damian A. Tamburri, and Chengyu Zheng. 2015. Fathoming Software Evangelists with the D-Index. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. 85–88. doi:10.1109/CHASE.2015.26

[6] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. 2015. Developer onboarding in GitHub: the role of prior social links and language experience. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 817–828. doi:10.1145/2786805.2786854

[7] Aleksandr Chueshev, Julia Lawall, Reda Bendraou, and Tewfik Ziadi. 2020. Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 499–510. doi:10.1109/ICSME46990.2020.00054

[8] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 8440–8451. doi:10.18653/v1/2020.acl-main.747

[9] Eleni Constantinou and Georgia M. Kapitsaki. 2016. Developers Expertise and Roles on Software Technologies. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 365–368. doi:10.1109/APSEC.2016.061

[10] Ozren Dabić, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)* (2021), 560–564. https://api.semanticscholar.org/CorpusID:232147743

[11] Arghavan Moradi Dakhel, Michel C. Desmarais, and Foutse Khomh. 2022. Dev2vec: Representing Domain Expertise of Developers in an Embedding Space. *Inf. Softw. Technol.* 159 (2022), 107218. https://api.semanticscholar.org/CorpusID:250451535

[12] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2020. Representation of Developer Expertise in Open Source Software. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2020), 995–1007. https://api.semanticscholar.org/CorpusID:218718456

[13] Tapajit Dey and Audris Mockus. 2020. Effect of Technical and Social Factors on Pull Request Quality for the NPM Ecosystem. *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (2020). https://api.semanticscholar.org/CorpusID:220424499

[14] Tapajit Dey and Audris Mockus. 2020. Which Pull Requests Get Accepted and Why? A study of popular NPM Packages. *ArXiv* abs/2003.01153 (2020). https://api.semanticscholar.org/CorpusID:211818278

[15] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. 2020. Detecting and Characterizing Bots that Commit Code. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 209–219. doi:10.1145/3379597.3387478

[16] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, and Riccardo Rubei. 2020. TopFilter: An Approach to Recommend Relevant GitHub Topics. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Bari, Italy) *(ESEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 21, 11 pages. doi:10.1145/3382494.3410690

[17] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 135–144. doi:10.1145/3097983.3098036

[18] Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. *ArXiv* abs/2407.21783 (2024). https://api.semanticscholar.org/CorpusID:271571434

[19] Nasir U. Eisty and Jeffrey C. Carver. 2021. Developers perception of peer code review in research software development. *Empirical Software Engineering* 27 (2021). https://api.semanticscholar.org/CorpusID:237605272

[20] M. El Mezouar, F. Zhang, and Y. Zou. 2019. An empirical study on the teams structures in social coding using GitHub projects. *Empirical Software Engineering* 24 (2019), 3790–3823. doi:10.1007/s10664-019-09700-1

[21] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *ArXiv* abs/2002.08155 (2020). https://api.semanticscholar.org/CorpusID:211171605

[22] Haoyu Gao, Christoph Treude, and Mansooreh Zahedi. 2023. Evaluating Transfer Learning for Simplifying GitHub READMEs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1548–1560. doi:10.1145/3611643.3616291

[23] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. 2019. git2net: mining time-stamped co-editing networks from large git repositories. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, 433–444. doi:10.1109/MSR.2019.00070

[24] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355. doi:10.1145/2568225.2568260

[25] Gillian J. Greene and Bernd Fischer. 2016. CVExplorer: identifying candidate developers by mining and exploring their open source contributions. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE '16)*. Association for Computing Machinery, New York, NY, USA, 804–809. doi:10.1145/2970276.2970285

[26] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Jian Yin, Daxin Jiang, and M. Zhou. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. *ArXiv* abs/2009.08366 (2020). https://api.semanticscholar.org/CorpusID:221761146

[27] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 495–504. doi:10.1145/1806799.1806871

[28] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *ArXiv* abs/1706.02216 (2017). https://api.semanticscholar.org/CorpusID:4755450

[29] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. doi:10.1162/neco.1997.9.8.1735

[30] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 2704–2710. doi:10.1145/3366423.3380027

[31] Thomas Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2016). https://api.semanticscholar.org/CorpusID:3144218

[32] Stratos Kourtzanidis, Alexander Chatzigeorgiou, and Apostolos Ampatzoglou. 2021. RepoSkillMiner: identifying software expertise from GitHub repositories using natural language processing. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 1353–1357. doi:10.1145/3324884.3415305

[33] Bo Li, Qiang He, Feifei Chen, Xin Xia, Li Li, John Grundy, and Yun Yang. 2021. Embedding app-library graph for neural third party library recommendation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 466–477. doi:10.1145/3468264.3468552

[34] Xiao Liu, Shunmei Meng, Qianmu Li, Lianyong Qi, Xiaolong Xu, Wanchun Dou, and Xuyun Zhang. 2023. SMEF: Social-aware Multi-dimensional Edge Features-based Graph Representation Learning for Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (Birmingham, United Kingdom) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 1566–1575. doi:10.1145/3583780.3615063

[35] Wei Ma, Mengjie Zhao, Ezekiel Olamide Soremekun, Qiang Hu, Jie Zhang, Mike Papadakis, Maxime Cordy, Xiaofei Xie, and Yves Le Traon. 2021. GraphCode2Vec: Generic Code Embedding via Lexical and Program Dependence Analyses. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)* (2021), 524–536. https://api.semanticscholar.org/CorpusID:244798792

[36] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. 2019. Predicting pull request completion time: a case study on large scale cloud services. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 874–882. doi:10.1145/3338906.3340457

[37] Leena Mathur, Paul Pu Liang, and Louis philippe Morency. 2024. Advancing Social Intelligence in AI Agents: Technical Challenges and Open Questions. *ArXiv* abs/2404.11023 (2024). https://api.semanticscholar.org/CorpusID:269187739

[38] Audris Mockus and James D. Herbsleb. 2002. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 503–512. doi:10.1145/581339.581401

[39] Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing. 2020. A Complete Set of Related Git Repositories Identified via Community Detection Approaches Based on Shared Commits. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 513–517. doi:10.1145/3379597.3387499

[40] João Eduardo Montandon, Luciana Lourdes Silva, and Marco Tulio Valente. 2019. Identifying Experts in Software Libraries and Frameworks Among GitHub Users. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 276–287. doi:10.1109/MSR.2019.00054

[41] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. 2019. FOCUS: a recommender system for mining API function calls and usage patterns. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 1050–1060. doi:10.1109/ICSE.2019.00109

[42] Athanasios N. Nikolakopoulos, Xia Ning, Christian Desrosiers, and George Karypis. 2021. Trust your neighbors: A comprehensive survey of neighborhood-based methods for recommender systems. *ArXiv* abs/2109.04584 (2021). https://api.semanticscholar.org/CorpusID:237485190

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *ArXiv* abs/1912.01703 (2019). https://api.semanticscholar.org/CorpusID:202786778

[44] Ayushi Rastogi and Nachiappan Nagappan. 2016. On the Personality Traits of GitHub Contributors. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 77–86. doi:10.1109/ISSRE.2016.43

[45] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van&nbsp;den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings* (Heraklion, Greece). Springer-Verlag, Berlin, Heidelberg, 593–607. doi:10.1007/978-3-319-93417-4_38

[46] Patrick E. Shrout and Joseph L. Fleiss. 1979. Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin* 86, 2 (1979), 420–428. doi:10.1037/0033-2909.86.2.420

[47] Yanchun Sun, Jiawei Wu, Xiaohan Zhao, Haizhou Xu, Sihan Wang, Jiaqi Zhang, Ye Zhu, and Gang Huang. 2024. Automatically Deriving Developers' Technical Expertise from the GitHub Social Network. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2462–2463. doi:10.1145/3691620.3695329

[48] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 356–366. doi:10.1145/2568225.2568315

[49] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. 2017. Graph Attention Networks. *ArXiv* abs/1710.10903 (2017). https://api.semanticscholar.org/CorpusID:3292002

[50] Yao Wan, Liang Chen, Guandong Xu, Zhou Zhao, Jie Tang, and Jian Wu. 2018. SCSMiner: mining social coding sites for software developer recommendation with relevance propagation. *World Wide Web* 21, 6 (Nov. 2018), 1523–1543. doi:10.1007/s11280-018-0526-9

[51] Liran Wang, Xunzhu Tang, Yichen He, Changyu Ren, Shuhua Shi, Chaoran Yan, and Zhoujun Li. 2023. Delving into Commit-Issue Correlation to Enhance Commit Message Generation Models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 710–722. doi:10.1109/ASE56229.2023.00050

[52] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yujie Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Haotong Zhang, Haibin Lin, Junbo Jake Zhao, Jinyang Li, Alex Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ArXiv* abs/1909.01315 (2019). https://api.semanticscholar.org/CorpusID:202539732

[53] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. 2023. Collaboration-Aware Graph Convolutional Network for Recommender Systems. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 91–101. doi:10.1145/3543507.3583229

[54] Wenxin Xiao, Jingyue Li, Hao He, Ruiqiao Qiu, and Minghui Zhou. 2024. Personalized First Issue Recommender for Newcomers in Open Source Projects. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering* (Echternach, Luxembourg) *(ASE '23)*. IEEE Press, 800–812. doi:10.1109/ASE56229.2023.00158

[55] Yueshen Xu, Yuhong Jiang, Xinkui Zhao, Ying Li, and Rui Li. 2023. Personalized Repository Recommendation Service for Developers with Multi-modal Features Learning. In *2023 IEEE International Conference on Web Services (ICWS)*. 455–464. doi:10.1109/ICWS60048.2023.00064

[56] Bei Yang, Jie Gu, Ke Liu, Xiaoxiao Xu, Renjun Xu, Qinghui Sun, and Hong Liu. 2021. Empowering General-purpose User Representation with Full-life Cycle Behavior Modeling. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2021). https://api.semanticscholar.org/CorpusID:259833436

[57] Liangwei Yang, Zhiwei Liu, Yingtong Dou, Jing Ma, and Philip S. Yu. 2021. ConsisRec: Enhancing GNN for Social Recommendation via Consistent Neighbor Aggregation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 2141–2145. doi:10.1145/3404835.3463028

[58] Chenyuan Zhang, Yanlin Wang, Zhao Wei, Yong Xu, Juhong Wang, Hui Li, and Rongrong Ji. 2023. EALink: An Efficient and Accurate Pre-trained Framework for Issue-Commit Link Recovery. *CoRR* abs/2308.10759 (2023). arXiv:2308.10759 doi:10.48550/ARXIV.2308.10759

[59] Yuqi Zhou, Jiawei Wu, and Yanchun Sun. 2021. GHTRec: A Personalized Service to Recommend GitHub Trending Repositories for Developers. In *2021 IEEE International Conference on Web Services (ICWS)*. 314–323. doi:10.1109/ICWS53863.2021.00049

[60] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: from patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 871–882. doi:10.1145/2950290.2950364