TSINGHUA SCIENCE AND TECHNOLOGY
ISSN:1007-0214
DOI:10.26599/TST.2026.90100013

Enhancing Developer Recommendations with Data Augmentation and Meta-Learning under Imbalanced Task Distributions

Lu Zhang, Shizhan Chen, Guodong Fan*, Hongyue Wu, Hongqi Chen, Zhiyong Feng

*

Abstract: Accurate recommendation of appropriate developers for Pull Request (PR) reviews is crucial for ensuring efficient software maintenance in GitHub. However, existing studies primarily focus on developers' review experience, overlooking the issue of data imbalance, which affects the effectiveness of the recommendations, i.e., imbalanced developer workload and reduced recommendation performance. To address this challenge, we propose DAMRec, a developer recommendation method based on data augmentation and meta-learning frameworks. The key point of our approach is artificially diversifying the imbalanced training data, ensuring the model encounters a broader range of developers. In terms of model design, we provide local and global data augmentation strategies to increase the exposure of inactive developers and enhance their chances of being recommended. In the training process, we introduce a meta-learning based training framework to mitigate the impact of data imbalance on model training. Through extensive experiments on popular open-source GitHub projects, we demonstrate that DAMRec outperforms state-of-the-art methods in both recommendation accuracy and mean reciprocal rank (MRR). Compared to existing approaches, DAMRec provides more accurate developer recommendations and alleviates the problem of overburdening a small subset of developers with review tasks.

Key words: Developer recommendation; Imbalance distributions; Data augmentation; Meta-Learning; Workload

1 Introduction

Pull Request (PR) review is a crucial activity for the development and maintenance of the GitHub community. Developers make updates or fixes to a project in a branch and submit a PR. Subsequently, the PR is assigned to other developers for review. If the code quality is satisfactory, it is merged into the project. PR reviews not only control the quality and defects of the code [1] but also facilitate knowledge dissemination and sharing within the GitHub community [2], promoting its sustainable development.

The key to effective PR review is recommending suitable developers for each PR. However, this process is challenging and can be time-consuming. For instance, in the GitHub community, as depicted in Fig. 1, the Angular project currently has 27,348 closed PRs, of which 14,427 are closed but still not reviewed. This situation negatively impacts the maintenance and evolution of GitHub projects. Thongtanunam et al.'s study [3] indicates that 4% to 30% of reviews encounter problems with reviewer assignments, and code change approvals can take up to 12 days, affecting the timeliness of reviews. Therefore, automating the process of recommending suitable developers for PR review is crucial.

Recent research has focused on using developers' re-

• Lu Zhang, Shizhan Chen, Hongyue Wu, Hongqi Chen and Zhiyong Feng are with the Department of Intelligence and Computing, Tianjin University, Tianjin and 300350, China. E-mail: zlu_4435@tju.edu.cn; shizhan@tju.edu.cn; hongyue.wu@tju.edu.cn; hqchen@ahu.edu.cn; zyfeng@tju.edu.cn.
• Guodong Fan is with the School of Information Science and Engineering, Shandong Agriculture and Engineering University, Zibo and 255300, China. E-mail:guodong.fan@126.com.
* To whom correspondence should be addressed.
Manuscript received: 2025-05-15; revised: 2025-09-14; accepted: 2026-01-28

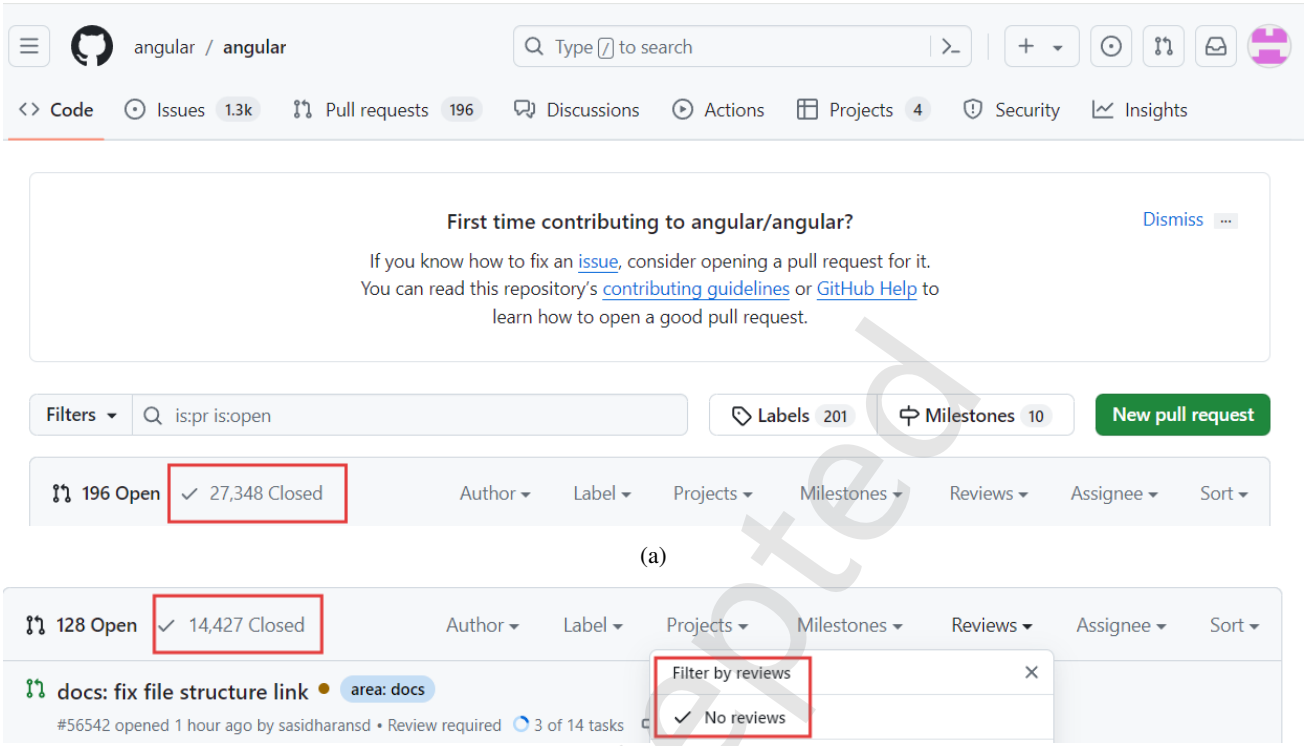
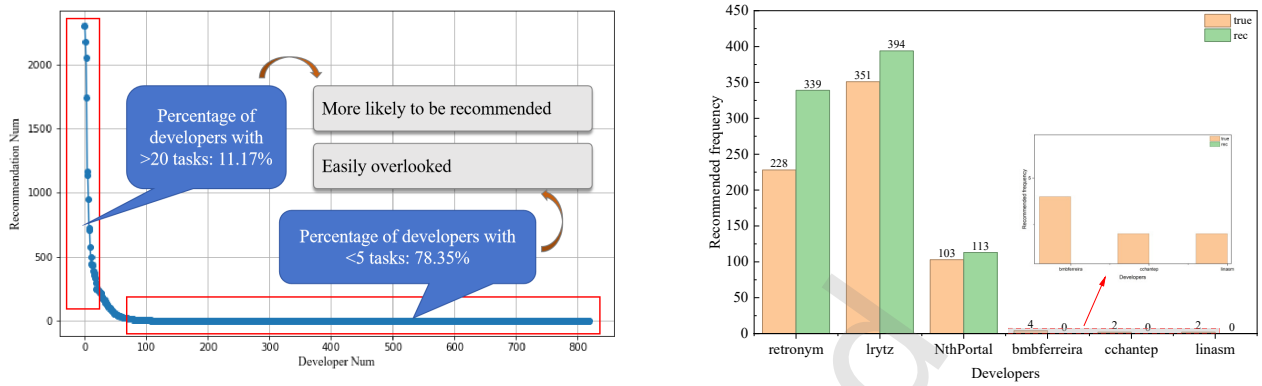


Fig. 1 The Review Status of Pull Requests.

view histories for recommendations, including methods based on historical review documents [4], developers' expertise (textual features) [3,5,6], and developer-task interactions [7,8]. Some of these studies rely on simple rules [9], which can overlook critical information, while others are constrained by extensive feature engineering. The latest work employs graph-based approaches for developer prediction and recommendation. However, there is still room for improvement in the accuracy of these recommendations.

These research efforts overlook the imbalance in the distribution of historical review records. We evaluate the distribution of developers and tasks in the datasets (see Section 5.2 for details) by calculating the average percentage of developers with >20 review tasks and <5 tasks across all datasets. As shown in Fig. 2, 11.17% of developers handled the majority of tasks, while 78.35% were assigned only a small portion, indicating a highly skewed distribution. This leads to popularity bias [10] and a "rich-get-richer" feedback loop [11]. Approaches based on developers' historical review experience and interaction relationships are more susceptible to this imbalance. These methods typically assign more tasks to experienced or active developers while neglecting inactive ones. For example, among six Scala developers: the three with more review tasks received significantly more recommendations than their actual task volume, whereas the three inactive developers (<5 tasks) were overlooked in recommendations (using the advanced developer recommendation method [12]). Consequently, recommendations based on imbalanced task distributions impose excessive review burdens on developers [1,13], ultimately compromising review efficiency.

Compared to the data imbalance issue in traditional recommendation methods, developer recommendation places greater emphasis on contextual and semantic comprehension. Recommending developers for PR review tasks requires a deep understanding of both PR requirements and developers' domain expertise to address potential technical interdependencies between developers and tasks. In traditional recommendations, users incur relatively low and flexible resource costs (time/effort) when consuming items (e.g., clicking/viewing/purchasing), and recommendation methods primarily optimize for preference matching. Conversely, in developer recommendation, reviewing a PR task demands significant time investment and domain expertise, while the effective number of reviews per developer faces strict workload [13] and time constraints [3]. Imbalanced task distribution may cause severe un-



(a) (b)

Fig. 2 Imbalanced Task Distributions and Its Impact .

derutilization of developer resources, elevate project risks, and compromise software quality. Thus, traditional strategies for handling data imbalance are not directly applicable to developer recommendation scenarios.

To mitigate the impact of imbalanced task distributions in PR review data, we propose an enhanced developer **rec**ommendation method integrating **d**ata **a**ugmentation and **m**eta-learning (DAMRec). Specifically, data augmentation mitigates data sparsity and enhances the modeling of low-activity developers, while meta-learning with curriculum scheduling adapts training to enhance generalization on samples from these inactive developers. This dual design aims to optimize both recommendation accuracy and developer workload distribution. Empirically validated through six GitHub projects with 37,000+ pull requests, DAMRec achieves holistic gains—simultaneously elevating recommendation performance and optimizing developer workload distribution. The main contributions of this paper are as follows:

1) To the best of our knowledge, this is the first study that collaboratively mitigate the impact of imbalanced task distribution on developer recommendation from both modeling and training perspectives.

2) We introduce a novel developer recommendation method, DAMRec, which leverages both data augmentation and meta-learning. DAMRec effectively mitigates the effects of imbalanced task distributions and optimizes the recommendation process from both model design and training perspectives.

3) Through extensive experiments on six popular GitHub projects (>37k PRs), DAMRec achieves superior recommendation performance than state-of-the-art baselines while effectively alleviating developer workload imbalance.

The rest of this paper is structured as follows. Section 2 provides the related work about pull request review. Section 3 describes the problem definition of DAMRec. Section 4 presents the details of DAMRec. Section 5 describes the experiment settings and experiment results. Section 6 discusses the threats encountered in this paper. Section 7 concludes the paper.

2 Related Work

2.1 Developer Recommendation

Developer recommendation usually utilizes the historical experiences of developers, collaborative relations and so on to recommend suitable developers to PRs. The related work are researched from four aspects [14, 15]: heuristic-based research, machine-learning-based research, interaction-based research, and mixed-methods-based research.

Heuristic-based research recommend developers based on historical data, e.g., similarity based on files [3, 4, 16, 17] or paths [6], code ownership [5, 18, 19], etc. Thongtanunam et al. [3] proposed a file location-based code reviewer recommendation method that utilizes the similarity of previously reviewed file paths to recommend suitable code reviewers. Sülün et al. [6] introduced an automatic reviewer recommendation method RSTrace+, which can record up-to-date information about traceability links, including support for GitHub utilities. Ye et al. [5] extracted features of PR titles, commit messages and code change information to recommend reviewers for PRs. Machine learning-based research uses Bayesian models [15], RF [20], support vector

machines [21], and latent Dirichlet allocation (LDA) [22] for PR-oriented developer recommendation research. The task of assigning developers to analyze PRs is difficult. Lima et al. [20] proposed Random Forest classifier to mine data and suggest the most appropriate developers. Rahman et al. [21] designed a code reviewer recommendation method based on support vector machine. It considers relevant cross-project work history and the developer's experience with certain PR-related expertise. Interaction-based research [7, 8, 23] recommend the potential developers based on the interaction relationship between developers and PRs. Historical review data has both explicit and implicit relations, modeling only explicit relations often result in sub-optimal recommendation accuracy. To that end, Xia et al. [23] proposed a hybrid recommendation method that combines latent factor models and neighborhood methods to capture implicit relations. Rong et al.[8] proposed a new recommendation method named HGRec, employing hypergraphs to model the one-to-many higher-order relationships between PRs and reviewers, thereby addressing the inadequately modeled problem in existing recommendation methods. Mixed-methods-based research is a hybrid method that may include a combination of above approaches [24]. Of these, a mixture of heuristic and interactive methods is the most commonly used hybrid approach [25–27]. Yu et al. [24] proposed a recommendation method based on comment networks, and extended machine learning, information retrieval, and file location methods. Rebai et al. [27] converted the code reviewer recommendation problem into a multi-objective search question that combines expertise, usability, and collaboration relation.

Most of the researches didn't consider the distribution of historical review data. However, It is an essential factor in the effectiveness of developer recommendations. Hence, we propose DAMRec, which can ensure the performance of developer recommendation, and optimize the issue brought by imbalance distribution.

2.2 Imbalance Task Distribution in Developer Recommendation

Imbalanced distribution is a very common phenomenon in real-world datasets, which seriously affects the performance of different types of tasks, such as classification [28], object detection [29], and recommendation [30]. Developer recommendation are similarly affected by the imbalanced task distribution. The highly skewed imbalance distribution results in a large number of developers being rarely recommended and a few developers being assigned a large number of PR review tasks [14, 31]. The workload of developers not only affects the efficiency of PR review, but also causes a waste of human resources, which is not conducive to the sustainable evolution of the open source community. Currently, common solutions to the imbalanced distribution problem can be categorized into four types: re-sampling, re-weighting, data augmentation and learning strategy.

The main idea of the re-sampling method [32] is to optimize the imbalanced distribution problem at the sample level by adjusting the sampling strategy. Over-sampling [33] and under-sampling [34] are the two classical re-sampling methods. However, these two methods may affect the representation ability of deep learning, resulting in overfitting or loss of important samples [35]. Kang et al. [36] evaluated four sampling strategies, namely random sampling, class-balanced sampling, square-root sampling, and progressively-balanced sampling, and found that all of them, except random sampling, limited to specific data. SMOTE-Tomek method combines SMOTE over-sampling with Tomek under-sampling [37]. It first generates synthetic samples for the minority class using SMOTE and then removes Tomek links from the majority class to enhance classification performance. However, this method may introduce synthetic noise. Re-weighting mitigates bias in the recommendation by adjusting the weights of many-shot sample and few-shot sample through the loss function. The straightforward strategy is to weight the samples according to the reciprocal of the sample frequency [38] [39] or the square root reciprocal [40]. Tan et al. [41] proposed a simple and effective loss of equalization by ignoring the gradients of rare classes to solve the problem of long-tailed rare classes, which protects the learning of less-sample data during network parameter updating. Wang et al [42] proposed a seesaw loss, which dynamically re-balances the gradients of positive and negative samples in each class through two complementary factors, the mitigation factor and the compensation factor. Data augmentation improves the model performance through various data augmentation operations such as feature transformation and data mixup. yao et al. [43] proposed a novel data augmentation method that exploits feature correlations for highly-skewed item recommendations. Zhong et al. [44] found that mixup can remedy over-confidence and have a positive effect on representation learning. Dablain et al. [45] proposed

a novel data augmentation algorithm for imbalanced data called expansive over-sampling (EOS). In the embedding space of the CNN framework, it identified the nearest enemy samples of the minority class samples to form convex combinations, thereby expanding the feature space range of the minority class. Learning strategy guides the training process of the model through learning strategies, such as meta-learning and curriculum learning, thus mitigating the negative effects of imbalanced data. Lee et al. [46] proposed a meta-learning based recommendation system to improve the recommendation effect on users with fewer samples. Wang et al. [47] proposed dynamic curriculum learning (DCL) to adaptively adjust the sampling strategy and loss weight in each batch. Yu et al. [48] proposed a multilevel learning framework for courses with unbalanced recommendations, designing hierarchy-aware gradient descent and course sampling scheduling to mitigate the data imbalance problem.

There is a lack of literature in the developer recommendation domain that considers recommendation performance and the workload of recommendation results from the perspective of the imbalanced task distribution in the training data. Only a few studies address the workload aspect. Mirsaeedi et al. [31] take workload as a consideration in reviewer recommendation, but the recommendation results have little effect on workload. While traditional strategies for handling data imbalance cannot be directly applied to the developer recommendation scenario, they still offer valuable insights for mitigating imbalance issues in developer recommendation. Drawing on data augmentation and learning strategies, we propose a developer recommendation method to address the problems faced in reviewer recommendation, both in model design and training strategies.

3 Preliminaries

Problem Definition. Let $D = \{d_1, d_2, \ldots, d_n\}$ denotes the candidate developer set, and $P = \{p_1, p_2, \ldots, p_m\}$ denotes the PR set. For each developer $d(i) \in D$, its feature is represented by developer's review history. For each $p(j) \in P$, its feature is represented by PR's information, i.e.,descriptions and filepath. In the developer recommendation scenarios, the interactions between developers and PRs are bipartite graph. And, its adjacency matrix can be represented as $R \in R^{N \times M}$, where N and M are the size of developer set and PR set. If there has an interaction between $d_i \in D$ and $p_j \in P$, $R_{i,j} = 1$. Otherwise, $R_{i,j} = 0$.

Table 1 Notations.

Notations	Descriptions
D	Developer set
P	PR set
R	Adjacency matrix
Z_d,Z_p	The CF embedding vectors of d and p
L_d,L_p	The local embedding vectors of d and p
G_d,G_p	The global embedding vectors of d and p
E_d,E_p	The embedding vectors of d and p
$y_{\hat{dp}}$	The prediction score of base model
$\mathcal{L}$	The loss of base model
$\Omega_{support}$, Ω_{query}	Support set and Query set for meta-learning
$g(.)$	Base model
$F(.)$	Meta learning-based model
p	Prediction score of DAMRec

The frequency distribution of developers recommended are imbalanced, that is the PR review tasks assigned to developers are imbalanced, as shown in Fig. 2, which will impact the performance of developers recommendation. So, our goal is to improve developer recommendation performance and optimize the tasks' allocation. The notations are summarized in Tab. 1.

4 Method

DAMRec involves two important stages: base model and meta-learning based model. In base model, we utilize GCN for embedding and propagating the collaborative interactions between PRs and developers, and realize graph contrastive learning with local and global data augmentation. In meta-learning based model, we construct support set and query set by sampling, and learn the evolution of model parameters from easy tasks to hard tasks by meta-learning and curriculum learning strategies.

4.1 Base Model

We detail the framework of the base model, as shown in Fig. 3. First, we implement feature embedding and propagation of local collaborative relationships between developers and PRs through graph learning. Next, we perform local and global augmentation of collaborative relationships based on graph Mask and singular value decomposition (SVD), respectively. The two types of data augmentation are designed to avoid the problem of manually designing comparison views in contrastive learning.

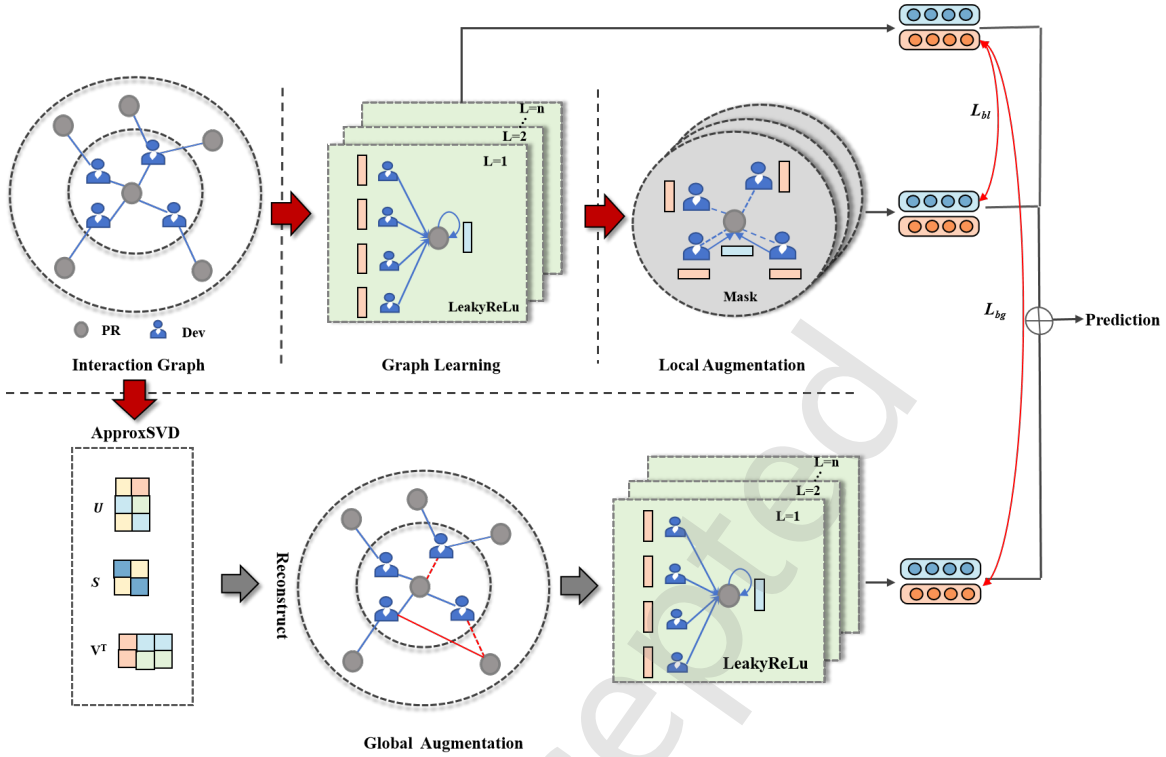


Fig. 3 The Framework of Base Model.

4.1.1 Collaborative Filtering

We first use collaborative filtering method to learn the local collaboration relationship between PRs and developers. We obtain the initial embedding e_p^0, e_d^0 of PR and developer through the pre-trained model Bert [49] and word2vec [50]. Embedding size is set to T. In order to propagate and embed the collaborative relationship between PRs and developers, we use a two-layer GCN [51] to perform neighbour aggregation for each PR and developer node. At layer k, the aggregation process is shown as follows:

$$z_{d,i}^k = \sigma\left(drop(\bar{\mathcal{A}}_{i,:}) \cdot E_d^{k-1}\right), z_{p,j}^k = \sigma\left(drop(\bar{\mathcal{A}}_{:,j}) \cdot E_p^{k-1}\right) \quad (1)$$

where $z_{d,i}^k$ and $z_{p,j}^k$ denote the embedding features of the developer and PR notes in the k-th layer, and $z_{d,i}^k \in Z_d^k, z_{p,j}^k \in Z_p^k$. $\sigma(\cdot)$ is the activation function. $\bar{\mathcal{A}} \in R^{I \times J}$ is the normalized adjacency matrix, and $drop(\cdot)$ is the edge dropout function, which can mitigate the over-fitting issue.

After the neighbour aggregation, we get the updated embeddings as follows:

$$E_d^k = E_d^{k-1} + Z_d^{k-1}, E_p^k = E_p^{k-1} + Z_p^{k-1} \quad (2)$$

4.1.2 Local Data Augmentation

To enhance the learning of the implicit relationship between PRs and developers, inspired by the work of [52–54], we adopt a graph mask-based approach to denoise the developer-PR interaction graphs. Define $\mathcal{M}^k \in R^{I \times J}$ as the mask matrix of the k-th GCN. Each $\mathcal{M}_{i,j}^k$ represents the degree to which the interaction between developer i and PR j is masked. The smaller the value is, the less important the interaction between the developer and the PR is. We utilize cosine similarity [53,55] to calculate the importance of interactions between nodes:

$$cos\left(z_{d,i}^k, z_{p,j}^k\right) = \frac{z_{d,i}^{k\ T} z_{p,j}^k}{\left\|z_{d,i}^k\right\|_2 \left\|z_{p,j}^k\right\|_2} \quad (3)$$

To make $\mathcal{M}_{i,j}^k$ always take values between [0,1], we transform it linearly by the following equation:

$$\mathcal{M}_{i,j}^k = \left(cos\left(z_{d,i}^k, z_{p,j}^k\right) + 1\right)/2 \quad (4)$$

We multiply the above graph mask matrix $\mathcal{M}^k$ with the developer-PR interaction matrix element-by-element to obtain a new matrix $\mathcal{G}^k$ and normalize it:

$$\mathcal{G}^k = \mathcal{M}^k \odot \mathcal{A} \quad (5)$$

$$\overline{\mathcal{G}}_{i,j} = \mathcal{G}_{i,j} / \sum_{j'=0}^{J} \mathcal{G}_{i,j'}, \overline{\mathcal{G}}_{j,i}^T = \mathcal{G}_{j,i}^T / \sum_{i'=0}^{I} \mathcal{G}_{j,i'}^T \quad (6)$$

To obtain local data augmentation in collaborative relationships, we combine the matrix $\mathcal{G}^k$ with a message passing scheme, generating a contrastive learning structure enhanced with adaptive masking. The local data

augmentation can be formalized as follows:

$$L_d^k = \overline{\mathcal{G}} \cdot E_d^k, L_p^k = \overline{\mathcal{G}}^T \cdot E_p^k \tag{7}$$

Therefore, the representations of developer and PR node features updated with local data augmentation are as follows:

$$E_d^k = E_d^{k-1} + Z_d^{k-1} + L_d^{k-1}, E_p^k = E_p^{k-1} + Z_p^{k-1} + L_p^{k-1} \tag{8}$$

We perform graph contrastive learning between the collaborative view and the local augmented data view. Embeddings of the same node across different views are considered positive samples, while embeddings of different nodes are considered negative samples. We use the InfoNCE [56] loss function to compute the loss between the contrastive views, maximizing the consistency between positive samples and the divergence between different nodes.

$$\mathcal{L}_{d,bl} = \sum_{i=0}^{m} \sum_{k=0}^{k} -log \frac{exp\left(cos\left(z_{d,i}^k, l_{d,i}^k\right)/\tau\right)}{\sum_{i'=0}^{m} exp\left(cos\left(z_{d,i}^k, l_{d,i}^k\right)/\tau\right)} \tag{9}$$

where τ is the temperature hyperparameter.

4.1.3 Global Data Augmentation

In contrast to local data augmentation, global data augmentation not only effectively extracts significant collaborative information from the original collaboration graph but also generates new structural information from a global perspective. The primary idea of global data augmentation is reconstructing the normalized adjacency matrix using a randomized singular value decomposition (SVD) method [57] and subsequently performing message propagation and embedding on the reconstructed structure. Given the normalized adjacency matrix $\hat{A}$ and its largest singular value q, we implement the randomized SVD method to carry out the matrix reconstruction:

$$\hat{U}_q, \hat{S}_q, \hat{V}_q^T = ApproxSVD\left(\bar{A}, q\right), \hat{A}_{SVD} = \hat{U}_q \hat{S}_q \hat{V}_q^T \tag{10}$$

where $rank\left(\hat{A}\right) = q$, and $\hat{U}_q \in R^{I \times q}, \hat{S}_q \in R^{q \times q}, \hat{V}_q \in R^{J \times q}$. Next, we perform message propagation based on the reconstructed normalized adjacency matrix $\hat{A}_{SVD}$, which can be formally represented as follows:

$$\begin{aligned} G_d^k &= \sigma\left(\hat{A}_{SVD} E_d^{k-1}\right) = \sigma\left(\hat{U}_q \hat{S}_q \hat{V}_q^T E_d^{k-1}\right), \\ G_p^k &= \sigma\left(\hat{A}_{SVD} E_p^{k-1}\right) = \sigma\left(\hat{U}_q \hat{S}_q \hat{V}_q^T E_p^{k-1}\right) \end{aligned} \tag{11}$$

where G_d^k and G_p^k are the developer and PR embeddings in the new generated graph structure.

After generating the aggregated information from the global collaborative view, we combine the local collaborative filtering relationships with the global collaborative filtering signals to obtain the following representation:

$$\begin{aligned} E_d^k &= E_d^{k-1} + Z_d^{k-1} + L_d^{k-1} + G_d^{k-1}, \\ E_p^k &= E_p^{k-1} + Z_p^{k-1} + L_p^{k-1} + G_p^{k-1} \end{aligned} \tag{12}$$

Similarly, we perform graph contrastive learning between the collaborative view and the globally augmented view. The loss function between the contrastive views is computed using the InfoNCE loss function:

$$\mathcal{L}_{d,bg} = \sum_{i=0}^{m} \sum_{k=0}^{k} -log \frac{exp\left(cos\left(z_{d,i}^k, g_{d,i}^k\right)/\tau\right)}{\sum_{i'=0}^{m} exp\left(cos\left(z_{d,i}^k, g_{d,i}^k\right)/\tau\right)} \tag{13}$$

4.1.4 Model Prediction

After the original collaborative relationship propagation and local and global data enhancements, the final embeddings of developer and PR are calculated as follows:

$$E_d = \sum_{k=0}^{k} E_d^k, E_p = \sum_{k=0}^{k} E_p^k \tag{14}$$

Finally, the predict score is calculated as follows:

$$\hat{y_{dp}} = e_d^T e_p \tag{15}$$

We utilize the classical Bayesian Personalized Ranking (BPR) loss [58] to optimize the parameters of the original collaborative view:

$$\mathcal{L}_{bpr} = \sum_{(d,i,j) \in \mathcal{O}} -ln\sigma\left(\hat{y_{d,i}} - \hat{y_{d,j}}\right) \tag{16}$$

where $\mathcal{O}$ is the set of sampled interactions in training data. $(d,i) \in \mathcal{O}^+$ indicates the positive sample data, and $(d,j) \in \mathcal{O}^-$ indicates the negative sample data.

The contrastive learning loss on the developer-side is the sum of local graph contrastive loss and the global contrastive loss. The graph contrastive learning loss on the PR-side is defined in the same way. The developer-side contrastive learning loss is formalized as follows:

$$\mathcal{L}_{d,cl} = \mathcal{L}_{d,bl} + \mathcal{L}_{d,bg} \tag{17}$$

We integrate the loss of the original collaborative relationship with the contrastive learning loss to obtain the final loss function for base model:

$$\mathcal{L} = \mathcal{L}_{bpr} + \lambda_1 \cdot \left(\mathcal{L}_{d,cl} + \mathcal{L}_{p,cl}\right) + \lambda_2 \cdot \|\Theta\|_2^2 \tag{18}$$

where λ_1 and λ_2 are tunable weights. $\Theta = \{E\}$ indicates the trainable parameters in the base model.

4.2 Meta-Learning Based Model

After high-order propagation and data augmentation in the graph learning process, the inactive developer data is enriched, gradually diminishing the impact on developer recommendations. To further optimize the devel-

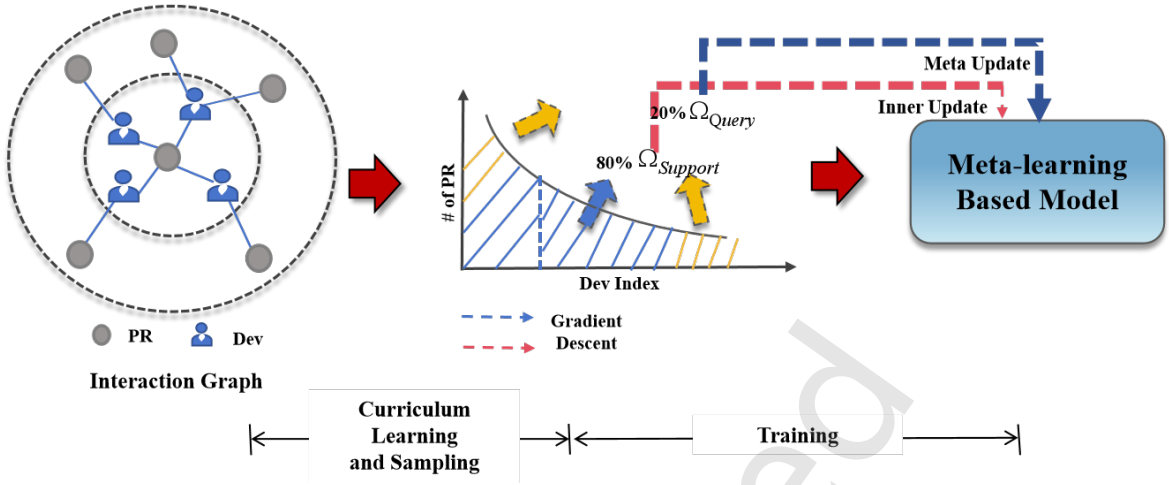


Fig. 4 Meta-Learning Based Training Process.

oper recommendation problem in the context of imbalanced task distribution, we train the model using task sampling and meta-learning to enhance its generalization ability. The training process is illustrated in Fig. 4.

4.2.1 Task Sampling

The construction of samples is crucial for model training based on meta-learning. Inspired by [46] and [59], we divide the training tasks into a support set and a query set. Initially, we adopt a curriculum learning strategy to sort the samples based on their difficulty levels. In this context, data with multiple interactions are considered easier tasks, while data with fewer interactions are regarded as more difficult tasks. Subsequently, we perform random sampling from tasks with varying levels of difficulty, using 80% of the data for the support set and the remaining 20% for the query set. The goal is to facilitate knowledge transfer during the training process through sample selection, thereby enhancing the model's learning ability. The definitions of the Support Set and Query Set are as follows:

$$\Omega_{Support} := \{(p, d, r(p, d))\} \tag{19}$$

$$\Omega_{Query} := \{(p, d, r(p, d))\} \tag{20}$$

If developer d reviewed PR p, $r(p, d) = 1$. Otherwise, $r(p, d) = 0$. $\Omega_{Support}$ and Ω satisfy:
1) $p \in P, d \in D$;
2) $\sum_{(p,d,r(p,d))\in\Omega_{Support}} r(p, d) = \sum_{p\in P} r(p, d), if\ d \in D$;
3) $\sum_{(p,d,r(p,d))\in\Omega_{Query}} r(p, d) = \sum_{p\in P} r(p, d), if\ d \in D$.

4.2.2 Meta-learning

We perform meta-learning training based on the partitioned support and query sets to enhance the model's learning capability. As illustrated in Fig. 4, the training process primarily consists of two phases: inner updates and meta-updates. The parameters involved in the initialization process are defined as meta-parameters θ_{meta}, while the other parameters associated with base model are defined as inner parameters θ_{inner}. During the inner update phase, for each task, we apply multiple gradient updates using the support set to adjust the inner parameters of the model. After completing the inner updates, we use the gradients obtained from the query set to update the meta-parameters, followed by back propagation through the loss function $\mathcal{L}$ of the base model. The detailed training process of meta-learning based model is described in Algorithm 1.

4.3 Final Prediction

The model parameters obtained through the meta-learning training process can enhance the generalization ability of the base model and improve recommendation performance in the face of imbalanced task distributions. Therefore, to better facilitate developer recommendations, we combine the prediction results of the base model with those of the model using parameters obtained through meta-learning. This combination yields the final developer recommendation predictions:

$$p = \lambda_f g\left(x_u, y_i; \theta\right) + (1-\lambda_f) g(x_u, y_i; F(\theta; \omega)) \tag{21}$$

where λ_f is the balance weight between the many-shot model and meta-learning based model. $F(\theta; \omega)$ is the parameters obtained through meta-learning.

Table 2 Statistics of the Datasets.

Projects	Programming Languages	Time Periods	Pull Request#	Reviewers#
Scala	Scala	2017.01.01-2021.01.12	2675	214
Xbmc	C++	2017.01.01-2021.01.12	3367	274
Bitcoin	C++	2017.01.01-2021.01.12	4329	541
Joomla	PHP	2015.01.01-2021.01.12	4741	355
Symfony	PHP	2015.01.01-2021.01.12	10220	1488
Angular	TypeScript	2015.01.01-2021.01.12	12078	812

Algorithm 1: Meta-Learning Process

Input: PR-dev interaction matrix R

Init: step size hyperparameters: α, β

θ_{inner} (inner parameters), θ_{meta} (meta parameters)

while *not converge* **do**

  sample tasks T ($\Omega_{Support}$, Ω_{Query})

  for *each task t in T* **do**

    set $\theta^t_{inner} = \theta_{inner}$

    evaluate $\nabla_{\theta^t_{inner}} \mathcal{L}_t \left(g_{\theta_{meta}, \theta^t_{inner}} \right)$

    inner update $\theta^t_{inner} \longleftarrow$ $\theta^t_{inner} - \nabla_{\theta^t_{inner}} \mathcal{L}'_t \left(g_{\theta_{meta}, \theta^t_{inner}} \right)$

  end for

  meta update $\theta_{meta} \longleftarrow$ $\theta_{meta} - \beta \sum_{t \in T} \nabla_{\theta_{meta}} \mathcal{L}'_t \left(g_{\theta_{meta}, \theta^t_{inner}} \right)$

  $\theta_{inner} \longleftarrow$ $\theta_{inner} - \beta \sum_{t \in T} \nabla_{\theta_{inner}} \mathcal{L}'_t \left(g_{\theta_{meta}, \theta^t_{inner}} \right)$

end while

5 Experiment

We propose four research questions (RQs) for the experimental evaluation, which are:

RQ1: To what extent can the proposed DAMRec accurately recommend code reviewers when oriented to imbalanced task distributions?

RQ2: To what extent can the proposed DAMRec alleviate workload congestion issue? RQ3: How do the major compoments contribute to the performace of developer recommendation?

RQ4: How do the key hyperparameters impact the performance of DAMRec?

RQ5: How is the applicability and usability of the method in practice?

RQ1 evaluates the accuracy and mrr of DAMRec compared with the state-of-art approaches when facing imbalanced task distributions. RQ2 investigates DAMRec's performance with respect to workload. RQ3 assesses the impact of major components of DAMRec. RQ4 evaluates the performance of DAMRec with different settings. RQ5 investigates DAMRec's applicability and usability in practice by means of a manual evaluation.

5.1 Experiment Settings

5.1.1 Datasets

Our experiments are performed on datasets collected from popular projects on GitHub provided in the paper [8]. These GitHub projects are often selected for data crawling by developer recommendation literatures. The datasets include basic information about the PRs, such as description and file path, review records of the PRs, reviewer's comments, and so on. The time span of Scala, Xbmc, and Bitcoin is from 2017.1-2021.1, and the time span of Joomla, Symfony, and Angular is from 2015.1-2021.1. Different sizes of datasets make our experimental validation results more adequate. The specific statistical information of the datasets is shown in Tab. 2.

5.1.2 Evaluation Metrics

We use two widely adopted metrics, Top-k Accuracy (Acc@k) and Mean Reciprocal Rank @k (MRR@k), to evaluate the performance of the method, where k represents the number of top-k recommended developers. Top-k Accuracy measures the percentage of developers in the top-k list of recommended developers who are correctly identified as suitable for reviewing the PR. MRR@k reflects the average reciprocal rank of the first correctly recommended developer in the sorted top-k list. In addition, we propose a weighted average based on entropy to measure the workload of developers and evaluate the performance of the method in the context of imbalanced task distributions. The equations are shown as follows:

$$Accuracy = \frac{\sum_{p \in P} isCorrect(p, k)}{|P|} \tag{22}$$

P denotes the set of PRs. $isCorrect(p, k)$ measures if the correct developers exist in the Top-k recommendation list. If $isCorrect(p, k)$ true, return 1. Otherwise, return 0.

$$MRR = \frac{1}{|P|}\sum_{p\in P}\frac{1}{rank\left(rec\left(p\right),act\left(p\right)\right)} \quad (23)$$

$rec(p)$ returns a sorted list of recommended developers for p, while $act(p)$ provides the actual ranking of developers who reviewed p. The function $rank\left(rec\left(p\right),act\left(p\right)\right)$ returns the rank of the developer in the recommended list who first actually reviewed p.

$$workload = \frac{\alpha * wl\_1 + \beta * wl\_3 + \gamma * wl\_5}{\alpha + \beta + \gamma} \quad (24)$$

$$wl = -\frac{1}{log_2 n}\sum_{i=1}^{n} P(i) log_2 P(i) \quad (25)$$

where wl_k represents the value of workload at k. α, β and γ are the weight of importance of wl. The higher the recommendation ranking, the higher the level of importance. Therefore, as the k-value becomes larger, the weight value decreases in order. n represents the total number of reviewers, and $P(i)$ is the percentage indicating the workload of the i-th developer. The larger the value of $workload$, the more balanced the developer workload is in the recommended results.

5.1.3 Compared Baselines

To fully evaluate TFRec, we compare it with the state-of-the-art developer recommendation approaches and imbalance-oriented approaches:

1) Developer recommendation approaches

RevFinder [3]: RevFinder is a file location-based code-reviewer recommendation approach. It recommends suitable developers to review code by computing the similarity of file paths in the review history. The intuition is that files located in similar file paths will be managed and reviewed by developers with similar experience.

ACRec [25]: ACRec uses temporal locality to measure developer activity to recommend suitable developers for PR reviews.

RFRec [20]: RFRec uses Random Forest (RF) to identify the developers that actually reviewed pull requests.

HGRec [8]: HGRec is a recommender based on hypergraph technologies. It models the high-order relationship between PRs and reviewers (e.g., one PR with multiple reviewers).

TFRec [12]: TFRec is a time-aware developer recommendation approach based on multi-feature fusion.

LightGCN [60]: LightGCN is a simple and linear Graph Convolution Network, including only the neighborhood aggregation component.

2) Imbalance-oriented approaches

LightGCL [61]: LightGCL has an effective and efficient contrastive learning paradigm for graph augmentation.

Meta-learning based method (MLM) [46]: Meta-learning based method uses meta-learning to optimize the training process of the model.

To comprehensively evaluate DAMREc, we carefully choose representative traditional tools, e.g., RevFinder and ACRec, frequently used as comparison benchmarks in numerous prior studies, along with state-of-the-art tools like HGRec and TFRec as our baseline comparators. We also incorporate imbalance-oriented approaches as baseline methods to realize a more comprehensive comparison. To ensure fairness and reliability in the comparative experiments, the initial embeddings of LightGCN, LightGCL and MLM are set to be the same as those of DAMRec. In short, our baseline methods cover two key aspects of PR reviewer recommendations and imbalance-oriented processing.

5.1.4 Hyper-parameter Settings

We implemented DAMRec in a PyTorch environment and initialized the model using the Xavier method [62], with an embedding size of 300. Additionally, we optimized the model using the Adam optimizer [63], setting the learning rate to $1e^{-2}$. The propagation layer of DAMRec is 2, and the batch size is set as 512. The temperature coefficient τ is tuned in $\{0.2, 0.3, 0.5, 1, 3\}$. The regularization weights $\lambda 1$ and $\lambda 2$ are tune from $\{0.1, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}\}$ and $\{1e^{-5}, 1e^{-6}, 1e^{-7}\}$. The number of gradient steps in inner loop are varied from one to five, and the number of gradient steps at test is set as 1. To ensure a fair comparison, we conduct a grid search to optimize the parameters of the baseline methods individually and determine their best solutions. For RF, the number of decision trees n_estimators is tuned in $\{100, 200, 300\}$. For ACRec, the time-decaying coefficient is optimized in $\{0.5, 1, 1.5, 2, 2.5, 3\}$, and the length of the temporal window is searched in $\{7\ days, 15\ days, 30\ days, 60\ days, no\ time\ limit\}$. For HGRec, the regularization parameter α and influence parameter are selected from the range of $\{0.5, 0.6, 0.7, 0.8, 0.9\}$, and the maximum connections m is searched in $\{5, 7, 10, 13, 15\}$. For TFRec, the L2 regularization coefficients λ is tuned in $\{1e^{-6}, 1e^{-5}, \ldots, 1e^{-2}\}$, and the temperature coefficient τ is slected from the range of

Table 3 Performance Comparison of Top-k Accuracy.

Method	K	Top-k Accuracy					
		Scala	**Xbmc**	**Bitcoin**	**Joomla**	**Symfony**	**Angular**
Developer Recommendation Methods							
RevFinder	1	0.3135	0.0807	0.2948	0.4361	0.5017	0.2673
	3	0.6545	0.3627	0.6631	0.6714	0.7454	0.4679
	5	0.7494	0.5157	0.7311	0.7310	0.8544	0.5876
AcRec	1	0.3234	0.1224	0.2976	0.4626	0.5129	0.2017
	3	0.6713	0.3712	0.7190	0.6990	0.7697	0.4079
	5	0.7523	0.4929	0.7514	0.7652	0.8629	0.5066
RFRec	1	0.3346	0.1215	0.3208	0.4594	0.5542	0.2689
	3	0.6905	0.3698	0.7199	0.6925	0.7704	0.4719
	5	0.7485	0.5032	0.7551	0.7491	0.8530	0.5696
HGRec	1	0.3679	0.2107	0.3766	0.4932	0.5664	0.3204
	3	0.7435	0.4839	0.7258	0.7327	0.7815	0.6279
	5	0.7924	0.6234	0.8135	0.7821	0.8692	0.7156
TFRec	1	0.3714	0.2134	0.3802	0.4936	0.5624	0.3219
	3	0.7467	0.4921	0.7323	0.7449	0.7898	0.6396
	5	0.8011	0.6279	0.8196	0.7926	0.8743	0.7215
LightGCN	1	0.3089	0.1365	0.3079	0.4134	0.4947	0.2402
	3	0.6505	0.4011	0.665	0.6716	0.7117	0.5497
	5	0.7196	0.5463	0.7324	0.7204	0.8045	0.6169
Imbalance-oriented Methods							
LightGCL	1	0.3042	0.1584	0.3511	0.4468	0.4916	0.2562
	3	0.6434	0.4185	0.6523	0.7037	0.7229	0.5694
	5	0.7573	0.5302	0.7641	0.7528	0.8390	0.6436
MLM	1	0.2894	0.1414	0.3270	0.4277	0.4804	0.2541
	3	0.6756	0.4202	0.6435	0.6785	0.7172	0.5662
	5	0.7754	0.5357	0.7832	0.7532	0.8386	0.6437
DAMRec	1	**0.3756**	**0.2278**	**0.4168**	**0.5114**	**0.5717**	**0.3321**
	3	**0.7669**	**0.5247**	**0.7578**	**0.7529**	**0.8079**	**0.6639**
	5	**0.8329**	**0.6537**	**0.9005**	**0.8196**	**0.9146**	**0.7446**
imbalance_degree		3.8141	9.2853	4.6177	4.4094	3.1576	5.8270

$\{0.01, 0.05, 0.1, 0.15, 0.2\}$. For LightGCN, We import the textual characteristics into their embedding process, and tune the hyperparameters within the ranges sugessted in TFRec and DAMRec. Like DAMRec, LightGCL and MLM also use LightGCN as the original model.

5.2 Performance Comparisons (RQ1)

To answer RQ1, we compare the performance of DAMRec with both developer recommendation methods and imbalance-oriented methods on datasets with varying degrees of distribution imbalance. The degree of imbalance in the datasets was calculated based on the developers' activity levels, using the following formula. We calculate the accuracy and MRR of DAMRec and the baseline methods at $k = 1$, 3, and 5, with the results shown in Tab. 3 and 4.

$$imbalance\_degree = \frac{std\_dev}{mean\_recommendations} \tag{26}$$

$$std\_dev = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{27}$$

$$mean\_recommendations = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{28}$$

The std_dev represents the dispersion of the number of review tasks among developers. The $mean\_recommendations$ represents the average number of review tasks across developers. n is the number of developers, x_i is the number of review tasks of developer i, and $\bar{x}$ equals $mean\_recommendations$.

Table 4 Performance Comparison of MRR@k.

Method	K	MRR@k					
		Scala	**Xbmc**	**Bitcoin**	**Joomla**	**Symfony**	**Angular**
Developer Recommendation Methods							
RevFinder	1	0.3135	0.0807	0.2948	0.4361	0.5017	0.2673
	3	0.4592	0.2046	0.4516	0.538	0.6169	0.3749
	5	0.4817	0.2409	0.4744	0.5519	0.6318	0.4055
AcRec	1	0.3234	0.1224	0.2976	0.4626	0.5129	0.2017
	3	0.4615	0.2118	0.4758	0.5532	0.6207	0.3315
	5	0.4903	0.2331	0.4892	0.5738	0.6414	0.3476
RFRec	1	0.3346	0.1215	0.3208	0.4594	0.5542	0.2689
	3	0.4854	0.2119	0.4896	0.5504	0.6431	0.3897
	5	0.5003	0.2334	0.5081	0.5625	0.6789	0.4138
HGRec	1	0.3679	0.2107	0.3766	0.4932	0.5664	0.3204
	3	0.5075	0.3295	0.5013	0.5819	0.6486	0.4584
	5	0.5462	0.3584	0.5249	0.5989	0.6757	0.4776
TFRec	1	0.3714	0.2134	0.3802	0.4936	0.5624	0.3219
	3	0.5104	0.3342	0.5099	0.5945	0.6479	0.4602
	5	0.5493	0.3637	0.5374	0.6102	0.6794	0.4835
LightGCN	1	0.3089	0.1365	0.3079	0.4134	0.4947	0.2402
	3	0.4527	0.2534	0.4556	0.5294	0.5789	0.4484
	5	0.4776	0.2818	0.4769	0.5474	0.6001	0.4721
Imbalance-oriented Methods							
LightGCL	1	0.3042	0.1584	0.3511	0.4468	0.4916	0.2562
	3	0.4226	0.2976	0.4486	0.5434	0.5659	0.4501
	5	0.4942	0.3088	0.5049	0.5603	0.5996	0.4769
MLM	1	0.2894	0.1414	0.3270	0.4277	0.4804	0.2541
	3	0.4153	0.2875	0.4397	0.5355	0.5610	0.4495
	5	0.4743	0.2992	0.5053	0.5571	0.5874	0.4754
DAMRec	1	**0.3756**	**0.2278**	**0.4168**	**0.5114**	**0.5717**	**0.3321**
	3	**0.5042**	**0.3725**	**0.5226**	**0.6039**	**0.6599**	**0.4693**
	5	**0.5589**	**0.3912**	**0.5891**	**0.6213**	**0.6826**	**0.4894**
imbalance_degree		3.8141	9.2853	4.6177	4.4094	3.1576	5.8270

A higher $imbalance\_degree$ indicates a greater imbalance in the dataset.

As shown in Tab. 3, DAMRec consistently outperforms the developer recommendation methods and imbalance-oriented methods across six datasets with different $imbalance\_degree$, followed by TFRec and HGRec. For instance, on the Bitcoin dataset, compared to the best-performing baseline method, DAMRec achieves an accuracy improvement of 9.63%, 3.48%, and 9.87% at $k = 1$, 3, and 5, respectively. Moreover, DAMRec shows greater performance improvements on datasets with higher $imbalance\_degree$, such as Xbmc, Angular, and Bitcoin. As shown in Tab. 4, the performance trends of DAMRec and the baseline methods in terms of MRR@k and $imbalance\_degree$ mirror those in Tab. 3. DAMRec outperforms the developer recommendation methods and imbalance-oriented methods in terms of MRR at $k = 1$, 3, and 5. For Bitcoin, compared to the baseline methods, DAMRec shows the highest improvement of 41.38%, 16.50%, and 24.18%, and the lowest improvement of 9.63%, 2.49%, and 9.62% at k = 1, 3, and 5, respectively. The results from Tab. 3 and 4 indicate that: 1) DAMRec outperforms the two types of baseline methods in addressing developer recommendation tasks with imbalanced task distributions; 2) Among the better-performing methods, graph-based approaches excel at extracting feature data, significantly improving the performance of developer recommendations.

Therefore, we can conclude that DAMRec outperforms state-of-the-art methods in both top-k accuracy and MRR@k, demonstrating its superior performance

in developer recommendation tasks on imbalanced task distributions.

5.3 Workload (RQ2)

Workload is an important metric used to measure the task load of developers recommended for PRs. When too many tasks are recommended to the same developer, it can lead to a slower PR review process and waste of other human resources. Therefore, we calculated and compared the workload metrics for developer recommendations using the DAMRec and baseline methods, further evaluating the performance of each method. As shown in Tab. 5, DAMRec performed slightly worse than LightGCL and MLM, demonstrated comparable performance to HGRec, and outperformed the other five methods. On the Xbmc, Bitcoin, Joomla, and Symfony datasets, DAMRec outperforms HGRec in terms of workload. As indicated in Tab. 3 and Tab. 4, these datasets exhibit a relatively high degree of imbalance in distribution. This further demonstrates that the proposed method DAMRec is more suitable for training data with imbalanced task distributions. The results in Tab. 5 indicate that fully leveraging the features of historical review data, increasing the diversity of the data, and optimizing the training process are beneficial for recommending less active or less experienced developers.

A good developer recommendation method should strike an optimal balance between top-k accuracy and workload, i.e., optimizing workload metrics while maintaining high accuracy[64]. To quantify this trade-off, we calculate the mean performance of DAMRec and the baseline methods across six datasets, compare their overall performance in terms of top-5 accuracy and workload metrics, and conduct a Pareto front analysis [65]. Pareto front analysis is commonly used for "trade-off selection". Methods located on the Pareto front curve represent the current optimal trade-off solutions. Compared to any arbitrary method, those on the Pareto frontier point demonstrate superiority in at least one metric, "dominating" the remaining methods. Fig. 5 shows that LightGCL, MLM, HGRec, and DAMRec lie on the Pareto front curve, while the remaining methods represent suboptimal solutions that are dominated. These four methods therefore embody distinct accuracy–workload trade-off strategies. Although LightGCL and MLM achieve high workload scores, their top-5 accuracy lags markedly behind that of HGRec and DAMRec. More importantly, compared to HGRec, DAMRec achieves a notable improvement in average top-5 accuracy (up by 5.87%) while only experiencing a slight decrease in average workload (down by 0.71%). This demonstrates that DAMRec effectively balances accuracy and workload.

In conclusion, DAMRec performs well in terms of workload and alleviates the developer overload issue to a certain extent. Furthermore, DAMRec achieves a favorable trade-off between top-k accuracy and workload compared to baseline methods.

Table 5 Performance Comparison of Workload.

Method	Scala	Xbmc	Bitcoin	Joomla	Symfony	Angular
RevFinder	0.2039	0.1464	0.1534	0.1708	0.1341	0.1791
AcRec	0.1463	0.0982	0.1078	0.0758	0.1125	0.0949
RFRec	0.2119	0.1488	0.1482	0.1532	0.1261	0.1873
HGRec	0.2539	0.2557	0.1992	0.2123	0.1789	0.2493
TFRec	0.2214	0.2012	0.1607	0.1833	0.1332	0.1894
LightGCN	0.2131	0.1823	0.1561	0.1715	0.1283	0.1626
LightGCL	0.2802	0.2961	0.2231	0.2485	0.2375	0.2673
MLM	0.2592	0.2821	0.2087	0.2405	0.2264	0.2423
DAMRec	0.2417	0.2569	0.2005	0.2356	0.1914	0.2136

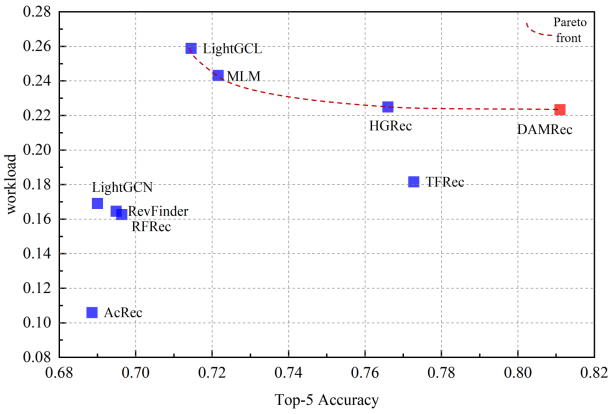


Fig. 5 Comprehensive Performance Evaluation.

5.4 Ablation Studies (RQ3)

To validate the effectiveness of the design of various modules in DAMRec, we conduct an ablation study. From the perspectives of model design and training, we propose four variant models: (w/o) Local, (w/o) Global, (w/o) Meta-learning, and (w/o) Curriculum Learning. We explore the impact of each module on DAMRec by comparing the Top-k Accuracy and MRR@k of the variant models with DAMRec. The details of the variant models and the effects of each module on DAMRec are as follows:

1) (w/o) Local: The variant model (w/o) Local is created by removing DAMRec's local data augmentation

Table 6 Performance Comparison with Ablations.

Method	Bitcoin						Symfony					
	Acc@1	Acc@3	Acc@5	MRR@1	MRR@3	MRR@5	Acc@1	Acc@3	Acc@5	MRR@1	MRR@3	MRR@5
(w/o) Local	0.3822	0.6659	0.8162	0.3822	0.4673	0.5357	0.4934	0.7255	0.8545	0.4934	0.5738	0.6032
(w/o) Global	0.3737	0.7507	0.8708	0.3737	0.5027	0.5649	0.5655	0.795	0.901	0.5655	0.6485	0.673
(w/o) Meta-learning	0.4073	0.7549	0.8536	0.4073	0.5144	0.5679	0.5697	0.8076	0.9088	0.5697	0.6575	0.6816
(w/o) Curriculum learning	0.3955	0.7498	0.8969	0.3955	0.5124	0.5799	0.5721	0.8076	0.9053	0.5721	0.6574	0.6803
DAMRec	0.4168	0.7578	0.9005	0.4168	0.5226	0.5891	0.5717	0.8079	0.9146	0.5717	0.6599	0.6826

module. As observed in Tab. 6, the performance of (w/o) Local is substantially lower than that of DAMRec, highlighting the significant contribution of the local data augmentation module.

2) (w/o) Global: In this variant, we remove the global data augmentation module of DAMRec. As shown in Tab. 6, the removal of this module leads to a significant performance drop in DAMRec. This indicates that the global data augmentation module plays a crucial role in enhancing DAMRec's performance.

3) (w/o) Meta-learning: By removing the meta-learning framework from DAMRec, we obtain the (w/o) Meta-learning variant. It is important to note that, in this variant, the model still uses the curriculum-ordered training data. The performance gap between DAMRec and (w/o) Meta-learning, as shown in Tab. 6, underscores the necessity of the meta-learning module for DAMRec.

4) (w/o) Curriculum Learning: The variant (w/o) Curriculum Learning represents the model trained on data without curriculum learning adjustments. According to Tab. 6, the absence of curriculum learning results in a decrease in DAMRec's performance, demonstrating its positive impact on the model's performance.

In conclusion, the design of each module in DAMRec has a significant impact on its performance. Moreover, the collaboration between these modules enables DAMRec to achieve optimal performance.

5.5 Parameter Analysis (RQ4)

The choice of hyperparameters is crucial to the performance of DAMRec. In this study, we focus on three key parameters: the number of propagation layers, the number of local updates in meta-learning, and the graph sampling ratio, and investigate how variations in these parameters affect DAMRec's performance. Due to space limitations, we present experimental results on two randomly selected datasets, Bitcoin and Symfony.

The Layer parameter refers to the number of propagation layers for node information in the developer-PR interaction graph. To examine the impact of layers on DAMRec's performance, we conducted experiments with Layer $=$ 1, 2, 3, 4, and 5, and evaluated DAMRec using Top-k Accuracy and MRR@k. Fig. 6 illustrates how the performance of DAMRec changes with the number of layers on the Bitcoin and Symfony datasets. As shown in Fig. 6, the performance of DAMRec increases initially with the number of layers and then decreases. The best performance is achieved at Layer = 2. Further, when Layer = 5, performance starts to improve again. This is likely due to over-aggregation of information when the number of layers is 3 or 4, which hampers the model's ability to distinguish node features. At Layer = 5, the model starts to capture richer global information from the graph. Considering the computational cost, setting the number of layers to 2 appears to be the most reasonable choice.

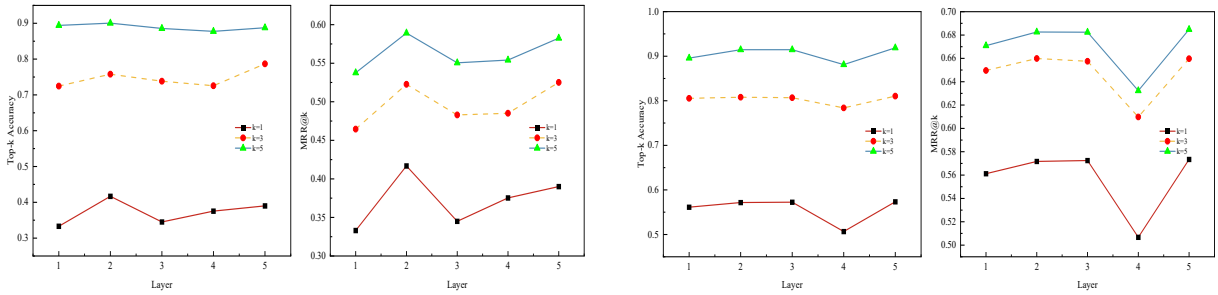
The number of local updates is a critical parameter in meta-learning. A suitable number of local updates helps the model better adapt to the current task. We compared DAMRec's performance with different values of the number of local updates, i.e., $Num\_grad\_steps\_inner$ $=$ 1, 3, 5, and 7, based on Top-k Accuracy and MRR@k, as shown in Fig. 7. As $Num\_grad\_steps\_inner$ increases from 1 to 5, DAMRec's performance gradually improves. However, when $Num\_grad\_steps\_inner$ is set to 7, performance starts to decline. Therefore, we set $Num\_grad\_steps\_inner$ to 5.

The sampling ratio during meta-learning also affects DAMRec's performance. To determine the optimal sampling ratio, we conducted experiments with sampling ratios of 60%, 70%, 80%, and 90%, and compared the results, as shown in Fig. 8. For each value of k and each dataset, we observe that the best performance is achieved when the sampling ratio is 80%. Therefore, we set the sampling ratio to 80%.

Through these experiments, we have identified how the key parameters in DAMRec affect its performance and have selected the optimal values for these parameters based on the experimental comparisons.

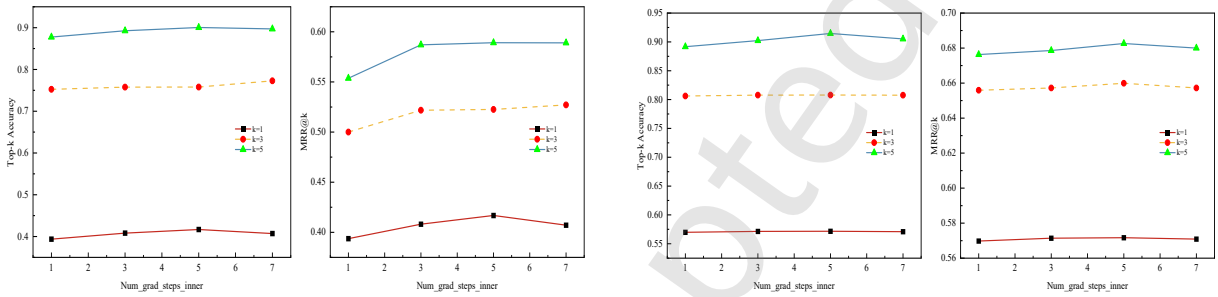
5.6 Applicability and Usability Analysis (RQ5)

To validate the applicability and usability of DAM-



(a) Bitcoin (b) Symfony

Fig. 6 Comparison of the Effect of Layer

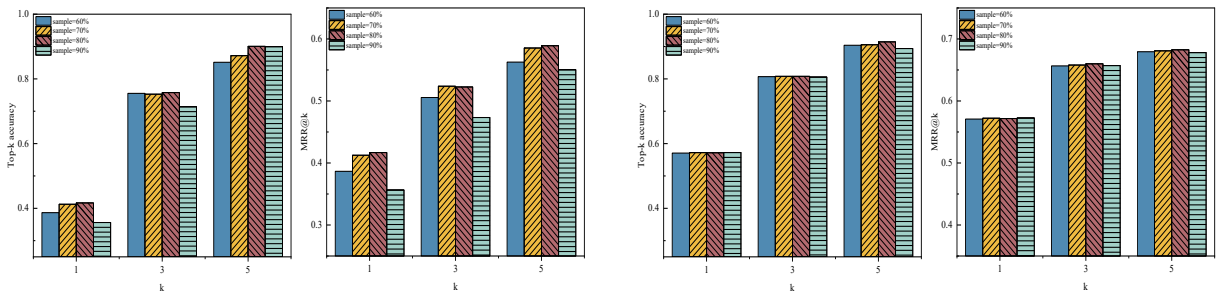


(a) Bitcoin (b) Symfony

Fig. 7 Comparison of the Effect of Inner Grad Steps.

Rec in real-world applications, we employ a manual evaluation approach. We invite ten graduate students majoring in computer science, representing diverse academic years and research specializations, to participate in a questionnaire-based evaluation. The participants included three doctoral candidates and seven master's students. All participants possess at least three years of programming experience and research experience and specialized in programming languages covering the languages involved in the six datasets. Each participant evaluated the applicability and usability of DAMRec, HGRec, and TFRec by reviewing the research methodologies described in these papers, along with their respective 50 selected recommendation instances. These instances included input data, recommendation lists, and key information about both the developers and the PRs. To minimize human evaluation bias, 50 recommendation instances from DAMRec were randomly selected. For the other two methods, corresponding recommendation results from identical PR tasks were chosen. Additionally, questionnaire participants were unaware of which method corresponded to each set of recommendation results.

To ensure the rationality of the questionnaire design, we adopt a combination of qualitative and quantitative evaluation methods. The qualitative evaluation investigated DAMRec's advantages, disadvantages, and potential areas for improvement. The quantitative evaluation assessed three methods' practicality using a five-point Likert scale. The quantitative evaluation items included: data availability(DA), design rationality(DR), comprehensive performance(CP), recommendation ra-



(a) Bitcoin (b) Symfony

Fig. 8 Comparison of the Effect of Sample.

◆ **Overall Assessment**

1. In your opinion, what are the potential positive impacts or advantages of DAMRec in practical application?

2. In your opinion, what are the potential drawbacks or shortcomings of DAMRec at present?

3. What improvements or optimizations do you think are needed to make DAMRec more feasible in practical applications?

◆ **Practicality Assessment**

✓ **Method 1**

Items	**Very Dissatisfied 1**	**Dissatisfied 2**	**Middle 3**	**Satisfied 4**	**Very Satisfied 5**
Data Availability	○	○	○	○	○
Design Rationality	○	○	○	○	○
Comprehensive Performance	○	○	○	○	○
Recommendation Results Rationality	○	○	○	○	○
Platform-Independent Generalizability	○	○	○	○	○

✓ **Method 2**

……

✓ **Method 3**

……

Fig. 9 Sample Questionnaire.

tionality(RR), and platform-independent generalizability(PG). Data availability measures the universality of methods' requirements for input data. Design rationality assesses the reasonableness of methods' architectural design. Comprehensive performance evaluates methods' overall performance balancing top-k accuracy and workload. Recommendation rationality judges whether the professional expertise of developers in the recommendation lists meets the requirements of the PRs. Platform-independent generalizability measures whether methods' application depends on a specific platform. An example questionnaire is shown in Fig.9.

We collect qualitative feedback from all participants. Statistical analysis revealed that the majority of participants believed DAMRec's design effectively mitigates the impact of imbalanced task distribution on developer recommendations and exhibits platform independence, making it applicable to developer recommendation scenarios on other platforms. However, DAMRec requires improvement in explainability and cross-domain generalization. Furthermore, we calculate the average scores given by all participants for the quantitative item of each method, with results shown in Tab.7. DAMRec achieved average scores of 3.7, 3.7, 4.2, 3.8, and 3.8, demonstrating favorable performance across all five quantitative items. Compared to HGRec, DAMRec holds advantages in CP, RR, and PG. Compared to TFRec, DAMRec demonstrates superiority in DR, CP, and RR. These findings align with prior experimental validation and our understanding of these three methods. Based on participant evaluations, DAMRec exhibits applicability and usability in real-world applications.

We utilize Cronbach's alpha coefficient and Fleiss' kappa coefficient to assess the internal consistency of the questionnaire items and the inter-rater agreement among participants, respectively. A Cronbach's alpha value ≥ 0.7 indicates high internal consistency among the questionnaire items. A Fleiss' kappa coefficient > 0.6 indicates a high degree of consistency among the participants. The calculated Cronbach's alpha coefficient for the questionnaire is 0.7237, and the Fleiss' kappa coefficient is 0.6826. This signifies that the questionnaire items consistently focused on method's prac-

Table 7 Practicality Assesement.

Method	DA	DR	CP	RR	PG
HGRec	3.7	3.7	4.1	3.6	3.7
TFRec	3.7	3.6	4	3.7	3.8
DAMRec	3.7	3.7	4.2	3.8	3.8

ticality and that the participants' opinions exhibited a high degree of consistency. Therefore, the questionnaire design and the results of the manual evaluation are reliable.

6 Discussion

External Validity. A major threat to external validity is the generalizability of DAMRec. Due to data accessibility in GitHub, we conducted experiments on GitHub datasets and demonstrated the effectiveness of DAMRec. However, whether DAMRec maintains its superiority when deployed in real-world scenarios remains an important open question. To mitigate the generalizability threat, we validated DAMRec across multiple projects from diverse domains. Simultaneously, we validated the applicability and practicality of DAMRec in real-world applications through manual evaluation. The core component design of DAMRec is platform-agnostic. Thus, we maintain its validity provided that the target platform/environment meets DAMRec's data requirements. We acknowledge, however, that applying DAMRec to proprietary or enterprise platforms may require adaptation due to factors like unique tools for tracking and managing development processes. In future work, we plan to extend our research to real-world application platforms for further validation. Additionally, the participant pool employed in our manual evaluation remains limited. We will continue to attempt to recruit active GitHub developers to participate in the questionnaire. Nevertheless, in this paper we have tried our best to enrich the diversity of the graduate-student respondents by enrolling master's and doctoral candidates with distinct research interests and varying levels of development experience, thereby enhancing the reliability of the questionnaire results.

Internal Validity. Internal validity mainly concerns experimental design and hyperparameter settings. To mitigate potential biases or errors in the experiments, we conducted multiple repeated trials to ensure the reliability of the results. Additionally, to avoid potential biases in reproducing the baseline method, we carefully examined the implementation details of the algorithm to ensure that the code implementation fully aligns with the algorithm description in the original paper. The hyperparameter settings for DAMRec were determined after extensive experimentation, effectively reducing potential threats during hyperparameter tuning.

Construct Validity. DAMRec is a developer recommendation algorithm based on data augmentation and meta-learning. We compared the performance of DAMRec with state-of-the-art baseline methods in terms of Top-k Accuracy and MRR@k, validating its effectiveness. Moreover, we demonstrated, through workload metric, that when performing developer recommendations, DAMRec not only maintains high accuracy but also accounts for the distribution of the developer's workload, improving the diversity of the recommendations. Of course, we also face a common threat in developer recommendation, namely the choice of ground truth in the evaluation. We cannot guarantee that the actual PR reviewers are the most reasonable choice. However, due to the same ground truth selection method between DAMRec and the baseline methods, this does not affect the performance comparison between them.

7 Conclusions

Imbalanced task distributions is an important challenge in developer recommendation. To mitigate the impact of imbalanced task distributions on developer recommendation, we propose DAMRec, a developer recommendation method based on data augmentation and meta-learning frameworks. Our approach improves DAMRec's ability to handle imbalanced task distributions through model design and the training process. We introduce local and global data augmentation strategies that alleviate the effects of task imbalance by increasing recommendations for inactive developers. And then, we adopt a meta-learning-based framework that enhances the model's adaptability and generalization ability, thus reducing the bias caused by imbalanced task distributions. Through experiments on six popular GitHub projects, we demonstrate that DAMRec not only offers accurate developer recommendations but also considers the workload of developers, providing more opportuni-

ties for inactive developers. In the future, we consider integrating DAMRec into real-world applications to further explore the usability and applicability of DAMRec in the real world.

Acknowledgment

This work is supported by the National Natural Science Key Foundation of China grant No.62032016, and National Natural Science Foundation of China grant No. 62102281.

References

[1] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka, Code reviewing in the trenches: Challenges and best practices, *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.

[2] A. Bacchelli and C. Bird, Expectations, outcomes, and challenges of modern code review, in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[3] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, Who should review my code? A file location-based code-reviewer recommendation approach for modern code review, in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 141–150.

[4] A. Chueshev, J. Lawall, R. Bendraou, and T. Ziadi, Expanding the number of reviewers in open-source projects by recommending appropriate developers, in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 499–510.

[5] X. Ye, Y. Zheng, W. Aljedaani, and M. W. Mkaouer, Recommending pull request reviewers based on code changes, *Soft Computing*, vol. 25, pp. 5619–5632, 2021.

[6] E. Sülün, E. Tüzün, and U. Doğrusöz, Rstrace+: Reviewer suggestion using software artifact traceability graphs, *Information and Software Technology*, vol. 130, p. 106455, 2021.

[7] X. Xie, X. Yang, B. Wang, and Q. He, DevRec: Multi-relationship embedded software developer recommendation, *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4357–4379, 2021.

[8] G. Rong, Y. Zhang, L. Yang, F. Zhang, H. Kuang, and H. Zhang, Modeling review history for reviewer recommendation: A hypergraph approach, in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1381–1392.

[9] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, Modern code review: A case study at google, in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 181–190.

[10] H. Abdollahpouri, R. Burke, and B. Mobasher, Controlling popularity bias in learning-to-rank recommendation, in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017, pp. 42–46.

[11] J. Ma, Z. Zhao, X. Yi, J. Yang, M. Chen, J. Tang, L. Hong, and E. H. Chi, Off-policy learning in two-stage recommender systems, in *Proceedings of The Web Conference 2020*, 2020, pp. 463–473.

[12] L. Zhang, S. Chen, G. Fan, H. Wu, H. Chen, and Z. Feng, A time-aware developer recommendation approach based on multi-feature fusion, *Applied Soft Computing*, vol. 169, p. 112609, 2025.

[13] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft, *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, 2016.

[14] H. A. Çetin, E. Doğan, and E. Tüzün, A review of code reviewer recommendation studies: Challenges and future directions, *Science of Computer Programming*, vol. 208, p. 102652, 2021.

[15] J. Lipcak and B. Rossi, A large-scale study on source code reviewer recommendation, in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 378–387.

[16] M. Fejzer, P. Przymus, and K. Stencel, Profile based recommendation of code reviewers, *Journal of Intelligent Information Systems*, vol. 50, pp. 597–619, 2018.

[17] X. Xia, D. Lo, X. Wang, and X. Yang, Who should review this change?: Putting text and file location analyses together for more accurate recommendations, in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 261–270.

[18] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 931–940.

[19] S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal, C. Maddila, S. Mehta, and B. Ashok, WhoDo: Automating reviewer suggestions at scale, in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 937–945.

[20] M. L. Júnior, D. M. Soares, A. Plastino, and L. Murta, Developers assignment for analyzing pull requests, in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1567–1572.

[21] M. M. Rahman, C. K. Roy, and J. A. Collins, Correct: Code reviewer recommendation in github based on cross-project and technology experience, in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 222–231.

[22] J. Kim and E. Lee, Understanding review expertise of developers: A reviewer recommendation approach based on latent dirichlet allocation, *Symmetry*, vol. 10, no. 4, p. 114, 2018.

[23] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, A hybrid approach to code reviewer recommendation with collaborative filtering, in *2017 6th International Workshop on Software Mining (SoftwareMining)*, 2017, pp. 24–31.

[24] Y. Yu, H. Wang, G. Yin, and T. Wang, Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?, *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[25] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development, *Information and Software Technology*, vol. 84, pp. 48–62, 2017.

[26] Z. Liao, Z. Wu, Y. Li, Y. Zhang, X. Fan, and J. Wu, Core-reviewer recommendation based on pull request topic model and collaborator social network, *Soft Computing*, vol. 24, pp. 5683–5693, 2020.

[27] S. Rebai, A. Amich, S. Molaei, M. Kessentini, and R. Kazman, Multi-objective code reviewer recommendations: Balancing expertise, availability and collaborations, *Automated Software Engineering*, vol. 27, pp. 301–328, 2020.

[28] T. Wu, Q. Huang, Z. Liu, Y. Wang, and D. Lin, Distribution-balanced loss for multi-label classification in long-tailed datasets, in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, 2020, pp. 162–178.

[29] S. Ren, K. He, R. Girshick, and J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[30] S. Liu and Y. Zheng, Long-tail session-based recommendation, in *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020, pp. 509–514.

[31] E. Mirsaeedi and P. C. Rigby, Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution, in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1183–1195.

[32] J. Brownlee, *Imbalanced Classification with Python: Better Metrics, Balance Skewed Classes, Cost-Sensitive Learning*, Machine Learning Mastery, 2020.

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[34] J. Laurikkala, Improving Identification of Difficult Small Classes by Balancing Class Distribution, in *Artificial Intelligence in Medicine: 8th Conference on Artificial Intelligence in Medicine (AIME)*, 2001, pp. 63–66.

[35] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9719–9728.

[36] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, Decoupling representation and classifier for long-tailed recognition, *arXiv preprint arXiv:1910.09217*, 2019.

[37] I. Tomek, Two Modifications of CNN, *IEEE Transactions on Systems Man and Cybernetics*, vol. 6, no. 11, pp. 769–772, 1976.

[38] Y. Wang, D. Ramanan and M. Hebert, Learning to model the tail, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 7032—7042.

[39] C. Huang, Y. Li, C. C. Loy and X. Tang, Learning Deep Representation for Imbalanced Classification, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5375–5384.

[40] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, Distributed representations of words and phrases and their compositionality, in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2013, pp. 3111-–3119.

[41] J. Tan, C. Wang, B. Li, Q. Li, W. Ouyang, C. Yin, and J. Yan, Equalization loss for long-tailed object recognition, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11662–11671.

[42] J. Wang, W. Zhang, Y. Zang, Y. Cao, J. Pang, T. Gong, K. Chen, Z. Liu, C. C. Loy, and D. Lin, Seesaw loss for long-tailed instance segmentation, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9695–9704.

[43] T. Yao, X. Yi, D. Z. Cheng, F. Yu, T. Chen, A. Menon, L. Hong, E. H. Chi, S. Tjoa, J. Kang et al., Self-supervised learning for large-scale item recommendations, in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4321–4330.

[44] Z. Zhong, J. Cui, S. Liu, and J. Jia, Improving calibration for long-tailed recognition, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16489–16498.

[45] D. A. Dablain, C. Bellinger, B. Krawczyk, and N. V. Chawla, Efficient augmentation for imbalanced deep learning. in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 1433–1446.

[46] H. Lee, J. Im, S. Jang, H. Cho, and S. Chung, Melu: Meta-learned user preference estimator for cold-start recommendation, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1073–1082.

[47] Y. Wang, W. Gan, J. Yang, W. Wu, and J. Yan, Dynamic curriculum learning for imbalanced data classification, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5017–5026.

[48] S. Yu, J. **, L. Ma, X. Gao, X. Wu, H. Xu, and J. Xu, Curriculum multi-level learning for imbalanced livestream recommendation, in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023, pp. 2406–2414.

[49] J. Devlin, M. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[50] T. Mikolov, Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781*, 2013.

[51] L. Xia, C. Huang, Y. Xu, J. Zhao, D. Yin, and J. Huang, Hypergraph contrastive collaborative filtering, in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 70–79.

[52] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, Learning to drop: Robust graph neural network via topological denoising, in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 779–787.

[53] C. Tian, Y. Xie, Y. Li, N. Yang, W. X. Zhao, Learning to denoise unreliable interactions for graph collaborative filtering, in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 122–132.

[54] X. Ren, L. Xia, J. Zhao, D. Yin, and C. Huang, Disentangled contrastive collaborative filtering, in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023, pp. 1137–1146.

[55] Y. Chen, L. Wu, and M. Zaki, Iterative deep graph learning for graph neural networks: Better and robust node embeddings, *Advances in Neural Information Processing Systems*, vol. 33, pp. 19314–19326, 2020.

[56] A. van den Oord, Y. Li, and O. Vinyals, Representation learning with contrastive predictive coding, *arXiv preprint arXiv:1807.03748*, 2018.

[57] N. Halko, P.-G. Martinsson, and J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[58] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, BPR: Bayesian personalized ranking from implicit feedback, *arXiv preprint arXiv:1205.2618*, 2012.

[59] Y. Zhang, D. Z. Cheng, T. Yao, X. Yi, L. Hong, and E. H. Chi, A model of two tales: Dual transfer learning framework for improved long-tail item recommendation, in *Proceedings of the Web Conference 2021*, 2021, pp. 2220–2231.

[60] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, Lightgcn: Simplifying and powering graph convolution network for recommendation, in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 639–648.

[61] X. Cai, C. Huang, L. Xia, and X. Ren, LightGCL: Simple yet effective graph contrastive learning for recommendation, in *International Conference on Learning Representations (ICLR)*, 2023.

[62] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[63] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*, 2014.

[64] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, The impact of human factors on the participation decision of reviewers in modern code review, *Empirical Software Engineering*, vol. 24, no. 2, pp. 973–1016, 2019.

[65] X. Lin, H. Zhen, Z. Li, Q. Zhang, and S. Kwong, Pareto multi-task learning, in *Advances in neural information processing systems*, 2019.

Lu Zhang is currently pursuing a Ph.D. degree in Computer Science and Technology with the College of Intelligence and Computing, Tianjin University. She received her bachelor's degree and master's degrees from Hebei University of Economics and Business, Shijiazhuang, China. She is working on services computing. Her main research interests are graph representation learning and developer recommendation.

Shizhan Chen (Member, IEEE) was born in 1975. He received the bachelor's degree in engineering and the Ph.D. degree in computer applications from Tianjin University, Tianjin, China, in 1998 and 2010, respectively. He is currently a Professor with the College of Intelligence and Computing, Tianjin University. He has authored or coauthored 50 academic articles in Chinese and international academic journals, magazines, and conferences, and applied for 40 national invention patents and 11 software copyrights. His research interests include service computing and service-oriented architecture.

Guodong Fan received the Ph.D. degree from Tianjin University, Tianjin, China, in 2024. He is currently working at the School of Information Science and Engineering, Shandong Agriculture and Engineering University. He received his bachelor's and master's degrees from Shandong University of Technology. He is working on cognitive services. His main research interests are representation learning, service computing, and software repository mining.

Hongyue Wu received the Ph.D. degree in computer science and technology from Zhejiang University, Zhejiang, China, in 2018. He is currently an Associate Professor with the College of Intelligence and Computing, Tianjin University. He has published more than 30 papers, including international journal papers published on IEEE TPDS, TSC, TASE, ect., and top international conference papers published on ICSOC, IEEE ICWS, etc. He has one paper published on ICSOC 17 and won the Sole Best Paper Award. His research interests include service computing, mobile edge computing, and cloud computing.

Hongqi Chen was born in Rizhao, China, in 1994. She received the Ph.D. degree from Tianjin University, Tianjin, China, in 2025. She is currently working at the School of Computer Science and Technology, Anhui University. She received the master's degree in software engineering from China University of Petroleum (EastChina), Qingdao, China, in 2019. Her current research interests include service computing, graphneural networks, online social networks and recom-mendation systems.

Zhiyong Feng (Member, IEEE) was born in 1965. He received the Ph.D. degree from Tianjin University, Tianjin, China, in 1996. He is currently a Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China. He has authored more than 200 articles, one book and 39 patents. His research interests include service computing, knowledge engineering, and software engineering. Dr. Feng is a distinguished member of China Computer Federation (CCF), a member of the Association for Computing Machinery (ACM), and the Chairman of ACM China Tianjin Branch.