Modern code review is a critical and indispensable practice in a pull-request development paradigm that prevails in Open Source Software (OSS) development. Finding a suitable reviewer in projects with massive participants thus becomes an increasingly challenging task. Many reviewer recommendation approaches (recommenders) have been developed to support this task which apply a similar strategy, i.e. modeling the review history first then followed by predicting/recommending a reviewer based on the model. Apparently, the better the model reflects the reality in review history, the higher recommender’s performance we may expect. However, one typical scenario in a pull-request development paradigm, i.e. one Pull-Request (PR) (such as a revision or addition submitted by a contributor) may have multiple reviewers and they may impact each other through publicly posted comments, has not been modeled well in existing recommenders. We adopted the hypergraph technique to model this high-order relationship (i.e. one PR with multiple reviewers herein) and developed a new recommender, namely HGRec, which is evaluated by 12 OSS projects with more than 87K PRs, 680K comments in terms of accuracy and recommendation distribution. The results indicate that HGRec outperforms the state-of-the-art recommenders on recommendation accuracy. Besides, among the top three accurate recommenders, HGRec is more likely to recommend a diversity of reviewers, which can help to relieve the core reviewers’ workload congestion issue. Moreover, since HGRec is based on hypergraph, which is a natural and interpretable representation to model review history, it is easy to accommodate more types of entities and realistic relationships in modern code review scenarios. As the first attempt, this study reveals the potentials of hypergraph on advancing the pragmatic solutions for code reviewer recommendation.

CCS CONCEPTS
• Software and its engineering → Collaboration in software development; • Information systems → Recommender systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Guoping Rong, Yifan Zhang, Lanxin Yang, Fuli Zhang, Hongyu Kuang, He Zhang. 2022. Modeling Review History for Reviewer Recommendation: A Hypergraph Approach. In 44th International Conference on Software Engineering (ICSE ’22), May 21–29, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510213

1 INTRODUCTION
As a popular software practice, code review is believed to be paramount to software quality for both commercial projects and Open Source Software (OSS) projects [9, 41, 43, 49]. Through manually scrutinizing source code, reviewers aim to identify possible issues or improvement opportunities and thereby prevent issue-prone code snippets from being incorporated into project repositories [7]. In addition to secure quality, code review is also helpful in knowledge dissemination, team collaboration [5, 8, 30, 52], etc. However, studies show that code review highly relies on the experience and knowledge of reviewers [24, 32], which implies that the identification of suitable reviewers is crucial to review efficacy.

Nowadays, an informal, asynchronous and tool-based code review practice that is known as Modern Code Review (MCR) is widely adopted in software development [33]. In the OSS community, MCR is an essential step in a so-called pull-request development paradigm [56, 57], where developers make changes to some code snippets and submit a Pull-Request (PR) to the project repository. Then potential reviewers (including project owners) examine the PR and provide feedback through an issue tracking system (e.g., JIRA\textsuperscript{1}, Gerrit\textsuperscript{2}, etc.); if the issues related to the PR were properly addressed, one project owner then merges the PR into the project repository [25, 59]. Studies indicate that MCR quality is subject to many factors, among which the reviewers’ expertise and workload can make a significant difference [6, 24, 40]. In this sense, it is also crucial to find suitable reviewers for a certain PR, especially in the context of the OSS development where the potentially massive participants are usually geographically distributed and not necessarily known to each other. In fact, as Thongtanunam et al. pointed out, inappropriate assignment of code reviewer may take 12 days longer to approve a code change in OSS development, thus a recommendation tool is necessary to speed up a code review process [51].

In the past decade, researchers have worked out a number of reviewer recommendation approaches (recommenders) in order to

\textsuperscript{1}https://www.atlassian.com/software/jira
\textsuperscript{2}https://www.gerritcodereview.com/
Ordinary graph:

Hypergraph:

Recently, a technique namely hypergraph has been utilized to model the complex relationships among multiple entities. Briefly, a hypergraph is a generalization of an ordinary graph in which an edge can associate any number of vertexes. For example, as shown on the right of Figure 1, a hyperedge e2 simultaneously connects three vertexes (n2, n5, n6). In contrast, in an ordinary graph, an edge connects exactly two vertexes (shown on the left of Figure 1). Take Figure 2 as a review example, there are three reviewers, namely Reviewer A, Reviewer B and Reviewer C having reviewed the identical PR. In an ordinary graph, the review history is modeled as reviewer pairs, i.e., Reviewer (A, B), Reviewer (B, C) and Reviewer (A, C) have reviewed one same PR. However, the fact that Reviewer A, B and C actually have reviewed the same PR is not able to be reflected in an ordinary graph. Since certain form of familiarity (e.g., similar review experience/history) forms the basis for most reviewer recommenders, the loss of this information will inevitably impact the recommendation performance. In contrast, since multiple vertexes can be included in one edge, hypergraph offers more natural approaches to model the review history portrayed in Figure 2, which provides more information for recommenders to perform recommendation.

In this study, we applied the hypergraph technique to model the aforementioned complex relationship among various entities in a natural and interpretable way. Based on this model, we also developed a new reviewer recommender, namely HGRec to explore the feasibility and effectiveness of this strategy. An extensive empirical study based on 12 OSS projects with more than 87K PRs and 680K review comments indicates the superiority of HGRec in terms of recommendation accuracy as well as workload balance among reviewers. The contributions of this study can be highlighted as below.

- To the best of our knowledge, this is the first effort that hypergraph is used to model code reviews as well as complex relationships among participants, e.g., one reviewer may be affected by others’ comments.
- We developed a new recommender, i.e., HGRec, based on hypergraph technologies.
- We empirically evaluated HGRec, the results indicate that HGRec not only outperforms the state-of-the-art recommenders in terms of recommendation accuracy, but to some extent mitigates the workload congestion issue.
2 RELATED WORK

2.1 Code Reviewer Recommendation

2.1.1 Recommenders. Automated reviewer recommendation has attracted a lot of attention in the past decade. A number of recommenders have been proposed, which follow a similar strategy in general, i.e., modeling review history and use the result model to recommend a new reviewer. In general, there are three main types of recommenders according to different modeling approaches, i.e., the heuristics-based, learning-based, and graph-based recommenders, respectively.

Heuristics-based recommenders. This type of recommenders suggests new reviewers with simple heuristic rules. For example, Thongtanunam et al. [50] proposed a recommender based on file path similarity, which subsequently evolved into RevFinder [51]. The RevFinder is based on the similarity between the file paths of a previous PR and a new PR. Zanjani et al. [60] developed a recommender (cHR) that determines candidates on a basic premise that the reviewers who have reviewed target code snippets before are most likely to be recommended. Rahman et al. [38] proposed a recommender (CORRECT) that utilizes external library similarity and technology expertise similarity of reviewers, which provides a possibility for cross-project reviewer recommendation. Jiang et al. [21] analyzed several attributes related to the code review and found that activity-based recommenders (AC) performed the best. Other rules adopted in the heuristics-based recommenders include Line 10 Rule [42], Expertise Recommender [31], Code Ownership [14] and Expertise Cloud [2], etc. Usually, the ‘models’ used by the heuristics-based recommenders are merely simple statistics or comparison results on the original review history. Most heuristic-based recommenders are easy to understand. However, research indicates that most of them suffer from low accuracy. Moreover, it is usually hard to add more elements (information) to enhance the models based on simple heuristic rules, which impacts their evolvability.

Learning-based recommenders. This type of recommenders assumes that the PR profile and reviewers’ personal expertise can be automatically learned from the review history by training. Among them, Support Vector Machine (SVM), Random Forest (RF), and Bayesian Network (BN) are widely applied [16–19]. de Lima Júnior et al. [11] investigated several kinds of learning-based recommenders, including Naïve Bayes (NB), Decision Tree (J48), RF, and Sequential Minimal Optimization (SMO) and found that RF outperforms others in terms of recommendation accuracy. In general, learning-based recommenders usually perform better than simple heuristics-based recommenders, however, the models behind these recommenders need a heavy workload on feature engineering, training, and long-term maintenance. Besides, they are normally not interpretable also, which becomes a barrier for future extension and improvement for the recommenders.

Graph-based recommenders. Recently, graph techniques have been adopted to model the review history [26, 36, 44, 45, 58], through which personal profiles and social relationships or networks between developers and reviewers are thus formalized into graph vertexes and edges. Using graph as the model, both sophisticated heuristics and learning algorithms can be used to design recommenders. For example, Yu et al. [58] found that developers who share common interests with a PR originator are potentially suitable reviewer candidates. Liao et al. [26] combined PR topic model with social networks to build the connections between collaborators and PRs. Sülün et al. [44] used software artifact traceability graphs to recommend reviewers who potentially are familiar with a given artifact.

2.1.2 Recommendation distribution. The rationale behind nearly all the recommenders implies that one reviewer who conducted the most reviews in the history tends to be recommended in a future review. As a matter of fact, it is common that a few core reviewers took over the most workloads on code review [54], which becomes a severe issue of “workload congestion” for some core reviewers, leading to review overload for these core reviewers [35]. Recent studies have proposed some solutions to alleviate the workload congestion. Asthana et al. [4] proposed a recommender (WhoDo) where reviewers’ scores are reduced by his/her incomplete PRs so as to decrease his/her chance to be recommended. Al-Zubaidi et al. [1] presented a workload-aware recommender (WLRRec) by utilizing NSGA-II, a multi-objective search-based approach to address two main objectives – maximizing the chance of participating in a review and, minimizing the skewness of review workload distribution. Rebai et al. [39] balanced the conflicting objectives of expertise, availability, and history of collaborations with multi-objective search techniques. Mirsaedi et al. [35] systematically take expertise, workload, and knowledge distribution for collaborators in recommending new reviewers.

In short, the workload congestion issue has raised wide concern in the research community on reviewer recommenders and should not be neglected in designing and evaluating recommenders.

2.2 Hypergraph Approach for Software Engineering

A hypergraph is an extension of the ordinary graph that consists of multiple vertexes and hyperedges, which can depict the high-order relationships among entities [13, 61]. Therefore, unlike the pairwise relationships depicted in an ordinary graph, hypergraph has the ability to express complex relationships in the real world, which prevents information loss as far as possible [27, 29, 37, 62]. This merit enables hypergraph techniques to be used in some software engineering scenarios. For example, Göde et al. [15] used hypergraph-based models on cloned code fragments and analyzed clone evolution in mature projects. Thomé et al. [48] used hypergraph to implement a search-driven string constraint solving algorithm to detect vulnerabilities in the program. Jiang et al. [20] used hypergraphs to represent code and implemented a framework for interferring program transformations. While the studies that use hypergraph techniques to model the complex relationships among software artifacts are not rare, to the best of our knowledge, this technique has never been used in reviewer recommendation, which usually involves both entities such as humans and artifacts as well as the complex and high-order relationships among different entities. This motivates the hypergraph-based recommender (i.e., HGRec) that is proposed in this study.
3 APPROACH
There are two major steps to design and implement HGRec, i.e., hypergraph construction and reviewer recommendation, respectively. In this section, we elaborate these two steps in detail.

3.1 Approach Overview
Figure 3 depicts the two major steps of HGRec. The top segment shows the process to construct a base hypergraph ($G_{base}$), which is based on the review history retrieved from project repositories; the bottom of Figure 3 presents the process to recommend potentially suitable reviewers for an incoming new PR, say $p_1$. The basic idea is to add $p_1$ and corresponding contributor ($a_1$) to the existing hypergraph $G_{base}$ to form a new hypergraph $G_t$ using the similar strategy to construct $G_{base}$. Then a hypergraph-based search strategy which calculates vertex score using a localized first-order approximation [10] is applied to rank and recommend candidate reviewers. Details of the hypergraph construction and reviewer recommendation will be elaborated in the following subsections.

Table 1: Key notations

| Notations | Descriptions |
|-----------|--------------|
| $PRS$ | the set of Pull-Request/PRs with $n$ PRs at first |
| $p_i$, $i \in [1...n]$ | a PR in $PRS$ |
| $a_i$ | the contributor of PR $p_i$ |
| $R_i$ | the set of reviewers to PR $p_i$ |
| $r_{ij}$, $j \in [1...m]$ | a reviewer in $R_i$ |
| $F_i, k \in [1...l]$ | the set of changed file paths involved in PR $p_i$ |
| $G$ | the hypergraph constructed based on $PRS$ |
| $V$ | the set of vertices in $G$ |
| $E$ | the set of hyperedges in $G$ |

In order to eliminate ambiguity, we first define some key notations in Table 1. To be specific, the review history of an OSS project is represented by a set of PRs ($PRS$), including the contributors ($a_i$), reviewers ($R_i$) and changed file paths ($F_i$) involved in each PR ($p_i$). A hypergraph ($G_{base}$) is used to model the review history, based on which, a new hypergraph ($G_t$) is generated by adding an incoming new PR, say $p_1$, to $G_{base}$.

3.2 Hypergraph Construction
Intuitively, for a target PR, the adjacent PRs in terms of file paths share certain similarities regarding content or function, which may also be able to reflect the similarity regarding experience and familiarity towards the target PR among the contributors and reviewers involved in these PRs. Using hypergraph the relationships among different entities involved in these PRs can be created in a succinct and natural representation. Two major steps are included to construct a hypergraph, i.e., the architecture building and the edge weight, respectively. As shown in Algorithm 1, function Construction depicts the former step, while the latter step is described by the function BuildEdge.

Algorithm 1: Hypergraph Construction

```plaintext
Function Construction(PR$S$, A, R)

1. $V_{base} \leftarrow \emptyset$; $E_{base} \leftarrow \emptyset$;
2. for $p_1 \in PRS$ do
   3. $V_{base} \leftarrow V_{base} \cup \{p_1\};$ // add vertexes
   4. Edge $epc = BuildEdge(p_1, a_1);$ // add PR-Contributor edge
   5. Edge $epr = BuildEdge(p_1, R_1);$ // add PR-Reviewer edge
   6. $E_{base} \leftarrow E_{base} \cup \{epc\} \cup \{epr\};$
3. end
4. for $p_1 \in PRS$ do
   5. $V_{base} \leftarrow V_{base} \cup \{p_1\};$ // add vertexes
   6. Edge $epc = BuildEdge(p_1, a_1);$ // add PR-Contributor edge
   7. $E_{base} \leftarrow E_{base} \cup \{epc\};$
   8. for $p_1 \in PRS$ do
      9. $V_{base} \leftarrow V_{base} \cup \{p_1\};$ // add vertexes
      10. Edge $epc = BuildEdge(p_1, a_1);$ // add PR-Contributor edge
      11. $E_{base} \leftarrow E_{base} \cup \{epc\};$
      12. SortWeightAddFilter($E_{base}$); // select high weight edges in set $E_{base}$
      13. $E_{base} \leftarrow E_{base} \cup \{epc\};$
      14. for $p_1 \in PRS$ do
         15. $V_{base} \leftarrow V_{base} \cup \{p_1\};$ // add vertexes
         16. Edge $epc = BuildEdge(p_1, a_1);$ // add PR-Contributor edge
         17. $E_{base} \leftarrow E_{base} \cup \{epc\};$
      18. $G_{base} \leftarrow \{V_{base} \cup E_{base}\};$
   19. end
20. EndFunction
```

In general, function Construction describes the main logic for hypergraph construction, which takes review history (contributors, reviewers, PRs, etc.) as inputs. Lines 2 initializes the vertex set $V_{base}$ and hyperedge set $E_{base}$. The ‘for loop’ in lines 3-8 updates hypergraph by adding new vertexes of PRs, contributors and reviewers as well as hyperedges of PR-Reviewer and PR-Contributor.
The hyperedges representing PR-PR relationship should be separately processed (in lines 9-17) since global information is needed to calculate edge weight. Finally, line 19 returns the result hypergraph $G_{base}$. Note that function Construction invokes the function BuildEdge to calculate the weights for hyperedges according to different relationships, which is detailed in lines 22-25. Since different types of relationships require different methods to calculate the edge weight, we elaborate them in detail as the following.

**PR-Reviewer:** The relationships between PRs and reviewers are necessary for all kinds of graph-based recommenders. In a pull-request development paradigm, one PR may experience multiple revisions and re-submissions, which would usually engage multiple reviewers and they may impact each other by publicly posted review comments. Therefore, in addition to the regular relationship, i.e. a pair of one reviewer and one PR, reviewers who comment on the same PR are connected with a hyperedge in a hypergraph.

In HGRec, the weight of a PR-Reviewer edge is set by aggregating all reviewers’ contributions, which is formulated in Equation 1.

$$w = \sum_{r_s \in E_{prs}} \frac{\alpha_i}{t_e - t_s}$$  \hspace{1cm} (1)

where reviewers in set $R_1$ participated in the PR $p_1$, reviewer $r_s$ made $\alpha_i$ comments in the PR $p_1$. The creation time of each comment is $t_s$. The hyperparameter $\lambda$ in Equation 1 works for mitigating the influence of comments (cf. subsection 3.4 for details). Moreover, reviewers’ activeness was also considered in HGRec, i.e. the closer reviewers are, the greater influence they carried. $t_s$ and $t_e$ are the start time and the end time of dataset in Equation 1.

**PR-Contributor:** Contributors and reviewers may play different roles in a pull-request development paradigm. Therefore, they are treated differently in HGRec by defining the PR-Contributor relationship and the corresponding weight. As Equation 2, the more recent activity is, the higher weight.

$$w = \frac{t_e - t_s}{t_e - t_s}$$  \hspace{1cm} (2)

$t_1$ is the creation time of PR $p_1$, $t_s$ and $t_e$ take the same meaning as in Equation 1.

**PR-PR:** The profiles (e.g., language, code lines) and content (e.g., the source code) included in PRs to a certain degree can reflect the expertise of contributors. Moreover, it is also common that closely located source files share similar functions, and hence can be used for reviewer recommendation [51]. Therefore, the weight of PR-PR relationship is achieved by considering the distances between PRs in the file path set (as shown in Equation 3).

$$w = \sum_{f_1 \in F_1, f_2 \in F_2} \frac{Similarity(f_1, f_2)}{|F_1| \times |F_2|}$$  \hspace{1cm} (3)

where function Similarity is calculated as Equation 4,

$$Similarity(f_1, f_2) = \frac{LCF(f_1, f_2)}{max(len(f_1), len(f_2))}$$  \hspace{1cm} (4)

where $F_1$, $F_2$ are the file path sets contained in two PRs, say $p_1$ and $p_2$. $f_1$ and $f_2$ are the specific file paths that belong to the file path sets $F_1$ and $F_2$.

We also model developers’ turnover as an exponential function to smooth the distance between two PRs. In the exponential function of Equation 3, $t_1$ and $t_2$ are the creation and end time of dataset, $t_1$ and $t_2$ are the creation time of two PRs respectively. Through this way, within a certain time scope, the latest PRs are preferentially considered. To reduce calculation cost, we restricted the number of neighbors for a certain PR and simplified the hypergraph that only top-$m$ PR-PR connections (cf. subsection 3.4 for details) were included. Moreover, we employed a MIN-MAX strategy to normalize the weight for each type of edge.

### 3.3 Reviewer Recommendation

With a constructed hypergraph in hand, we then can perform a recommendation calculation based on the hypergraph. In general, we formulated reviewer recommendation as a ranking task on a hypergraph. Previous studies (e.g., [55]) used ‘random walk’ strategy to choose neighborhood as the next vertex with a certain probability, which is somehow low-effective. Inspired by [46], we applied an advanced ‘search and ranking’ strategy in HGRec, which is elaborated briefly in this subsection.

Given a hypergraph $G_{base}$ and a newly-submitted $p_1$, we first develop PR-PR relationship by calculating its file path similarities with existing PRs in $G_{base}$ and then we connect $p_1$ with the most similar PRs. By following a similar strategy (Algorithm 1), we can establish the PR-Contributor relationship. In this way, both new PRs and contributors are merged into the original hypergraph $G_{base}$ to form a new $G_t = \{V_t, E_t\}$.

For a hypergraph $G$, the key of this ranking strategy is to find the appropriate ranking vector $f^* \in \mathbb{R}^{|V|}$ which is able to minimize the objective function $Q(f)$ defined as below:

$$Q(f) = f^T (I - A) f + \mu (f - y)^T (f - y)$$  \hspace{1cm} (5)

where $y \in \mathbb{R}^{|V|}$ is a query vector with multiple elements, one for each vertex of the hypergraph $G$ which will be set to 1 for a target PR and its contributor, otherwise, $0$. $H^G \in \mathbb{R}^{|V| \times |E|}$ is a vertex-hyperedge incidence matrix, $W^G \in \mathbb{R}^{|V| \times |E|}$ is a weight matrix, $D^G$ is a vertex degree matrix and $D^G$ is a hyperedge degree matrix, $A = D^{-1}_o H^G W^G D^{-1}_e H^T$, and $\mu$ is the regularization parameter.

Through a series of deductions and transformations, we have the optimal $f^*$ as:

$$f^* = (I - \frac{A}{1 + \mu})^{-1} y = (I - \alpha A)^{-1} y$$  \hspace{1cm} (6)

where $\alpha = \frac{1}{1 + \mu}$.

Having ranked on the hypergraph, we can recommend the top-$k$ reviewers as the candidates. The whole recommendation process is presented in Algorithm 2.

Algorithm 2 takes hypergraph $G_{base}$, PR set $PRS$, and target PR as its inputs. Line 2 initializes candidate list $C_t$. Line 3 is to add vertexes of three types of entities. Lines 4-12 also invoke function BuildEdge (cf. Algorithm 1) to build relationships of PR-Contributor and PR-PR. Line 13 generates the query vector $y_t$ by $p_t$. Line 14 optimizes objective function $Q(f)$ and get the ranking vector. Lines 15-16 rank and return a recommendation list according to ranking strategy.
Algorithm 2: Hypergraph-based Recommendation

\[\textbf{Function}\;
\text{Recommendation}\((\text{Hypergraph } G_{\text{base}}, \text{PR}, PR, a_1)\)
\]

1. \begin{align*}
C_0 & \leftarrow \emptyset; \\
V_1 & \leftarrow \{V_{\text{base}} \cup \{p_1\} \cup \{a_1\}\}; \quad \text{add new vertexes to } G_{\text{base}} \\
E_0 & \leftarrow \emptyset; \\
\text{Edge set } E_t & \leftarrow \emptyset; \\
\text{Edge } e_{pp} & \leftarrow \text{add PR-Contributor edge} \\
\text{for } p_t \in \text{PR}\do \\
& \text{Edge } e_{pp} \leftarrow \text{add PR-PR edge} \\
& \text{end} \\
\text{SortWeightAddFilter}(E_t); \quad \text{select high weights in set } E_t \\
E_t & \leftarrow \{e_t \cup \{e_{pp}\}\}; \\
G_t & \leftarrow \{V_t, E_t\}; \\
y_t & \leftarrow \text{QueryVector}(V_t, e_t); \quad \text{use search and ranking strategy} \\
f_t & \leftarrow \text{Ranking}(G_t, y_t); \quad \text{get candidates’ score} \\
C_t & \leftarrow \text{FilteredAndSort}(f_t); \quad \text{get recommendation list} \\
\end{align*}

4.3 Experiment Settings

4.3.1 Baselines. To evaluate HGRec thoroughly, the following representative traditional recommenders and state-of-the-art recommenders as well are compared as the baselines, which are

- **AC** [21] that recommends reviewers based on recent activities of the candidates. Reviewers who leave comments frequently in recent PRs are determined to be active and prone to be recommended; otherwise inactive.
- **RevFinder** [51] that recommends reviewers by leveraging the file path similarities of PRs, i.e. the files located in close files may share similar functionality and therefore should be reviewed by reviewers with similar experience.
- **cHRev** [60] that recommends reviewers on the premise that who previously reviewed the code files is tended to be candidate reviewers for a target PR. cHRev formulates reviewers’ expertise based on “how many”, “who performed”, and “when reviews were performed”.
- **CN** [58] that recommends reviewers by aggregating developers who share common interests with the contributor of target PR. CN mines historical comment traces to construct a comment network to make recommendations.
- **RF** [11] that recommends reviewers by applying supervised machine learning, i.e. collecting project attributes and PRs to construct classifiers and rank candidates.
- **EARc** [55] that recommends reviewers by constructing a graph architecture to depict the expertise and authority of developers as well as their interactions. The recommendation is performed using graph searching algorithms.

The considerations are three-fold. First, AC and RF have shown impressively good performance in terms of accuracy in many studies [12, 21]. Second, CN and EARc both adopt graph (an ordinary graph) as the underlying model. Last but not least, as two classical recommenders, RevFinder and cHRev have been used as the comparison basis frequently in many existing studies.

4.3.2 Metrics. To address RQ1, we need to evaluate the performance of HGRec in terms of accuracy. We take two common metrics (cf. subsection 4.3.1). As a result, we chose 12 well-known projects in GitHub to evaluate the performance of HGRec as well as other recommenders in order to position our recommender. The time span of the dataset is from 2017-01-01 to 2020-06-30. Detailed demographics of the dataset are presented in Table 2.

| Project | #PRs | #Comments | #Reviewers | #Contributors |
|---------|------|-----------|------------|---------------|
| akka    | 4073 | 45677     | 864        | 921           |
| angular | 12817| 110178    | 2806       | 2233          |
| Baystation | 9471| 41373     | 676        | 576           |
| bitcoin | 7113 | 91092     | 1012       | 896           |
| cakephp | 3319 | 17281     | 864        | 976           |
| django  | 6027 | 31607     | 2952       | 3691          |
| joomla-cms | 10327| 94791    | 2122       | 1184          |
| rails   | 7912 | 37720     | 5651       | 4943          |
| scala   | 3478 | 24091     | 778        | 651           |
| scikit-learn | 6315| 68903     | 2378       | 2627          |
| symfony | 11283| 77548     | 3949       | 3477          |
| xbee    | 5959 | 40451     | 1596       | 1141          |

Total 87194 680622 25644 23316
in recommender evaluation studies, i.e. Accuracy (ACC) (defined as Equation 7) and Mean Reciprocal Rank (MRR) (defined as Equation 8).

- **Accuracy**

\[
ACC = \frac{1}{|PRS|} \sum_{p \in PRS} isTrue(p, k)
\]

where, \(PRS\) is a set of target PRs, indicator function \(isTrue(p, k)\) returns 1 if the recommended reviewer within top-\(k\) candidates finally reviewed the target PR \(p\), otherwise returns 0.

- **Mean Reciprocal Rank**

\[
MRR = \frac{1}{|PRS|} \sum_{p \in PRS} \frac{1}{rank(p, k)}
\]

where function \(rank(p, k)\) returns the location where the true reviewer places in the sorted reviewer list. MRR rewards score 1 if the first choice was correct and rewards 1/2 if the second choice was correct, and so on. While if the recommended reviewer is not contained in the candidate list, then MRR rewards 0. The final MRR is calculated as the average value of all the scores.

To answer RQ2, we defined Recommendation Distribution (RD) as Equation 9 to measure the extent that diverse reviewers can be recommended.

- **Recommendation Distribution (RD)**

\[
RD = \frac{1}{\log_2 n} - \sum_{i=1}^{n} P(i) \log_2 P(i)
\]

where, \(n\) is the total number of reviewers, \(P(i)\) is a percentage that indicates the workload of the \(i^{th}\) reviewer. The larger RD, the more diverse that a recommender recommends reviewers.

As a popular standard in the related studies, we evaluated the top-\(k\) (\(k=1, 3, 5\)) performances of the recommenders. To further test the difference, we established hypotheses and applied the Wilcoxon Signed Rank Test on ACC, MRR and RD. The null and alternative hypotheses can be stated as follows,

\(H_0,M\) : There is no significant difference on the metrics \(M\) between HGRec and R.
\(H_{1a,M}\) : HGRec is significantly better than R on metrics \(M\).
\(H_{1b,M}\) : HGRec is significantly worse than R on metrics \(M\).

where \(M\) can be ACC, MRR and RD and correspondingly, \(R\) represents one recommender introduced in Section 4.3.1.

4.3.3 Data pre-processing. Following the similar method in [12, 28], we applied a time series strategy to evaluate the recommenders’ performance in terms of ACC, MRR and RD. To be specific, all the reviews in 2017 were initiated as the original training set and hereafter each monthly review until Jun, 2020 played the role of the test set. Therefore, we eventually performed 30 rounds of evaluation in total, as shown in Figure 4. Take the first round for example, the first 12-month data is fed into all the recommenders and then the data of the 13th month is used to calculate ACC, MRR and RD using Equation 7, 8 and 9, respectively.

5 RESULTS AND ANALYSIS

Following a common strategy [51, 58, 60], we compare HGRec with other recommenders in terms of ACC, MRR and RD using top-1, top-3 and top-5 criteria, respectively. This section presents the results and the corresponding analysis.

5.1 Accuracy (RQ1)

Table 3 shows the performance of each recommender in terms of ACC. The results in bold mean the best recommender regarding ACC for a certain project. For example, HGRec performed the best in project ‘akka’ with all the top-1, top-3 and top-5 criteria. In general, HGRec, AC and RF performed relatively better than other recommenders in most cases. To be specific, HGRec takes the lead on 8 (7, 8) projects in terms of top-1 (top-3, top-5) ACC. As the close competitors, AC wins on 2 (4, 4) projects, RF leads on 2 (1, 0) projects using the same top-1, (top-3, top-5) criteria. The last row in Table 3 lists the average ACC for all the recommenders, which further indicates HGRec’s superiority. With all the top-1, (top-3, top-5) criteria, HGRec produces the best average ACC. Besides, compared with other two recommenders (i.e. CN and RARec) using graph techniques, HGRec outperforms the others regarding ACC for both solo project and the overall average, indicating the advantage of hypergraph technique to model the review history. Moreover, as the comparison basis, the ACC given by recommender RevFinder and cHRRec is not ideal, which to a fair degree is in line with other studies [21, 35, 58]. To further test the difference, a Wilcoxon Signed Rank Test has been conducted on ACC using the data from all the 12 projects. Note that there are 30 data points in each project according to the experimental setting elaborated in subsection 4.3.3. Due to page limits \(^3\), we present the number of projects in which we are not able to reject a certain hypothesis (i.e.\(H_{0,M}\), \(H_{1a,M}\) and \(H_{1b,M}\), where \(M\) denotes ACC) with \(p\)-value 0.05. The results are listed in Table 6 (the 3 columns under ACC). Take the first row as an example, in 9 out of 12 projects, HGRec produces a significantly better ACC than recommender RevFinder using the top-1 criteria, meanwhile, there are 3 projects in which no significant difference on ACC between RevFinder and HGRec has been observed. The rest is similar, which also confirms our intuitive observation derived from Table 3, i.e. HGRec performed the best on ACC among the recommenders involved in this study.

\(^3\)The dataset, source code and complete results are now public online through https://doi.org/10.6084/m9.figshare.19199981.v1
Table 3: ACC of recommenders

| Recommender     | ACC | MRR | RD |
|-----------------|-----|-----|----|
| RevFinder       | 0.52 | 0.09 | 0.06 |
| CN              | 0.50 | 0.08 | 0.06 |
| eRev            | 0.51 | 0.10 | 0.07 |
| RF              | 0.50 | 0.09 | 0.06 |
| HGRec           | 0.52 | 0.09 | 0.06 |

Table 4: MRR of recommenders

| Recommender     | MRR | RD |
|-----------------|-----|----|
| RevFinder       | 0.36 | 0.05 |
| CN              | 0.35 | 0.05 |
| eRev            | 0.36 | 0.05 |
| RF              | 0.35 | 0.05 |
| HGRec           | 0.36 | 0.05 |

Table 5: RD of recommenders

| Recommender     | RD |
|-----------------|----|
| RevFinder       | 0.06 |
| CN              | 0.06 |
| eRev            | 0.06 |
| RF              | 0.06 |
| HGRec           | 0.06 |

Table 6: Number of projects by Wilcoxon Signed Rank Test on ACC & MRR & RD of recommenders

| Recommender     | ACC | MRR | RD |
|-----------------|-----|-----|----|
| RevFinder       | 0.52 | 0.09 | 0.06 |
| CN              | 0.50 | 0.08 | 0.06 |
| eRev            | 0.51 | 0.10 | 0.07 |
| RF              | 0.50 | 0.09 | 0.06 |
| HGRec           | 0.52 | 0.09 | 0.06 |

5.2 Workload (RQ2)

Recently, researchers raise a new concern other than the accuracy of recommenders. That is, most recommenders tend to suggest a small group of core reviewers (i.e. the reviewers who reviewed the most PRs in a certain project have more chance to be recommended). This phenomenon may cause severe problems in projects with massive PRs within a relatively short period. RD is quotient to show the status of workload congestion.

Similarly, we also evaluated recommenders’ performance regarding RD and further tested the difference using Wilcoxon Signed Rank Test. Table 5 lists the results of RD for all the recommenders on all the 12 projects. In general, eRev performs the best in all the recommenders. Meanwhile, next to eRev, HGRec and CN present comparable performance regarding RD. Nevertheless, eRev and CN showed relatively poor performance regarding ACC. A noteworthy point is that among the best 3 recommenders according to ACC, HGRec outperforms the others, i.e. RF and AC by a discernible margin. The Wilcoxon Signed Rank Test results listed in Table 6 also confirm these observations.
To present an intuitive concept, take project ‘angular’ as an example. Figure 5 shows the workload distribution resulting from diverse recommenders based on the number of PRs in one month for project ‘angular’. Each column represents one reviewer’s workload (i.e. # PRs assigned), the broken line represents accumulated workload on diverse reviewers. For top-3 accurate recommenders, HGRec tends to create a relatively balanced workload for top-10 core reviewers. Take RF for example, the top-2 core reviewers are recommended for reviewing 77 and 64 PRs in just one month, which might be huge burdens for them. As a comparison, the top-2 core reviewers are recommended by HGRec for merely 39 and 31 PRs.

The reason behind this phenomenon is that by properly tuning $\alpha$ and $\lambda$ in HGRec, the importance (score) for some reviewers sharing the review experience on the same PRs has been increased, which increases their chances to be recommended, even they may not be active reviewers in the past.

6 DISCUSSION

6.1 Graph Technology for Recommenders
To represent relationships in a review recommendation paradigm is a basis of a recommender. Although some graph technologies have been adopted to model the relationships when developing recommenders, the primary innovation in HGRec lies in the introduction of hypergraph to model multiple participants involved in one PR, which is very common in OSS projects and easy to understand. Compared with the traditional recommenders that are based on the ordinary graphs, HGRec consists of multiple vertexes and hyperedges that can naturally model the complex high-order relationships among PRs, contributors and reviewers. Besides, HGRec supports a flexible recommendation architecture, that is more entities and relationships, e.g., organization, comments can be involved in HGRec if necessary.

When it comes to recommenders with similar technologies, CN only considers simple relationships such as developer vertexes, their interactive relationships (e.g., review activities) are formulated by directed edges. CN suggests candidate reviewers who share common interest with contributors but neglects the PR information itself. On the contrary, EARec includes both PR and reviewer vertexes, and recommends candidates by matching the characteristics of target PR and expertise of candidates. However, EARec does not consider the information of contributors’ internal relationships, e.g., the social relationships, which may not be able to be directly obtained from reviewer’s profile. HGRec systematically combines multiple roles, including PRs’ content and interactive relationships among the three entities (i.e. PRs, contributors and reviewers). More importantly, HGRec is able to model the complex in an intuitive manner close to the reality.

6.2 Model Interpretability
In recent years, AI (Artificial Intelligence) /ML (Machine Learning) techniques are widely used in software engineering, such as software defect prediction [22], continuous Integration prediction [59], software defect developer recommendation [3], code reviewer recommendation [60], etc. The interpretability of AI/ML model has naturally become the focus of these studies. According to [34], the interpretability of AI/ML model is the degree to which a human can understand the reasons behind a decision. For example, the model interpretability should reflect the relationship of feature on the outcome, the importance of each feature, the decision rule of each
feature, etc [23]. The importance of model interpretability in software engineering is obvious since without proper understanding towards the model, practitioners may not trust and adopt the model in practice [47]. HGRec models the review history using hypergraph, which explicitly includes the interaction among contributors, PRs and reviewers. Besides, the setting of parameters directly reflects the recommendation inclination. These characteristics of HGRec obviously make the rationale of reviewer recommendation much easier to be understood.

6.3 Capability to Support Future Improvements
Hypergraph distinguishes itself not only by its straight adaption to code review but also its architecture’s flexibility and extensibility, which supplies a promotion for improvement in the future. For example, due to its architecture and search strategies, CN and EARec hardly make any adjustments. On the contrary, HGRec at this stage has presented the advantages to model review history using the hypergraph technique, yet it still has the capability to involve more entities and relationships, which is worthy of exploration. For example, potential reviewers belonging to the same organization may share a similar background, thus impacting the weight (w) calculation. In addition, the content of review comments may bring a new feature to characterize a PR.

To conclude, HGRec supports a flexibility to adjust diverse contexts and future improvements.

6.4 Recommender Selection

![Figure 6: Synthetical evaluation regarding ACC and RD](image)

We use the top-3 accuracy, a common way to evaluate recommenders, combined with recommendation distribution to visually illustrate the performance of all the recommenders involved in our study. Figure 6. presents the average ACC and RD for the 12 projects. The advantages of HGRec are thus easy to identify, i.e., it achieves the most accurate recommendation in all recommenders and the best balanced workload in the top three accurate recommenders.

Nevertheless, although recommendation accuracy should be the primary consideration in most cases (otherwise the value of recommendation will be lost), reviewer recommendation should be applied with sufficient considerations in the application context, which involves multiple factors such as the number of potential reviewers, the number of PRs, etc. Take project ‘angular’ for example (as shown in Figure 5), workload balance may not be an ignorable factor since the core reviewers have already undergone heavy review tasks. To present a general concept, we portray the number of PRs per month for the 12 projects involved in our study, as depicted in Figure 7. Obviously, recommendation distribution (aka, workload balance) means more in projects such as ‘angular’, ‘joomla-cms’, etc. where there are normally hundreds of newly-submitted PRs need to be reviewed. On the contrary, in projects such as ‘cakephp’ and ‘scala’ where there are usually only dozens of new PRs per month, the core two or three reviewers to handle all the PRs may seem to be acceptable.

6.5 Threats to Validity

Several threats to validity are elaborated in this subsection.

6.5.1 Construct validity. The threats to the construct validity of this study may be related to one of the common concerns of research on reviewer recommendation, i.e., the ground-truth set of reviewers for evaluation[36]. The actual reviewers recorded in review history may not be able to guarantee “suitable reviewers” and further justify an appropriate recommendation. In this sense, the recommended reviewers are only potentially suitable reviewers for a certain PR.

6.5.2 Internal validity. The threats to the internal validity of this study may result from the data preparation phase. The personal organization of OSS projects is significantly loose, which brings participants’ frequent turnovers and once-in-all reviews. Recommending these gone or accidental reviewers is inappropriate. In this study, we left out reviewers who had already deleted accounts or participated in less than two reviews, so did the robot users. On the other hand, the opening PRs were also removed as they are uncertain. Another related threat is that some noise data (e.g., casual/superficial comments such as “OK”, “fine”, etc.) exists in both the training set and test set, which may not be able to guarantee a qualified reviewer recommendation. However, the evaluation on HGRec and other recommenders is based on the same dataset, which may mitigate these threats to a fair degree. Meanwhile, reviewers who posted these casual/superficial comments may also have subtle relationships (e.g., certain familiarity, mutual influence, etc.). Therefore, we did not refine the dataset to remove noise data at this stage.

Instead, HGRec takes the advantage to use the possible relationships behind the casual/superficial comments and their corresponding reviewers.

6.5.3 External validity. We experimented with the proposed recommender on the 12 OSS projects that are retrieved from GitHub. However, the proposed recommender could suffer risks on external validity, as several studies investigated other contexts, e.g., Gerrit
projects or mixed projects (both OSS and industrial projects). Therefore, the findings and conclusions are only valid in the given context. We have confidence that this study is representative because all the included projects were mentioned in the previous studies and the data is up-to-date. Besides, given the population of OSS projects from GitHub, 12 projects tested in our study may only represent a small portion. Nevertheless, the comparably consistent performance (i.e., recommendation accuracy) in our study and previous studies is able to mitigate this external threat to validity to a fair degree.

6.5.4 Conclusion validity. To avoid threats to conclusion validity, we followed a systematic, rigorous experiment and analysis procedure. The recommender proposed in this study has been experimented on 12 OSS projects with the history in three and a half years, including more than 87K PRs, 680K review comments. All the dataset is clearly elaborated (e.g., the name of projects, the time range, etc.) and publicly accessible online. This ensures a high degree of reliability that the conclusion drawn in the study is directly traceable to the raw data and hence can be replicated by other researchers.

7 CONCLUSIONS
With the proliferation of the pull-request development paradigm nowadays, as a key and perhaps daily practice, Modern Code Review (MCR) may impact massive software projects. While the importance of suitable reviewers has been widely recognized among OSS projects, their identification is indeed a challenge. Although many recommenders have been proposed in the past decade, their adoption is far from satisfactory [60]. Several critical issues such as low accuracy, workload congestion, incapable of extension and improvement have been raised and investigated in several related studies.

This paper proposes HGRec, a hypergraph based recommender to perform automatic reviewer recommendation in OSS projects. By applying hypergraph, we managed to model high-order relationships in MCR, an essential step in OSS development. A relatively extensive evaluation based on 12 OSS projects with more than 87K PRs and 680K comments indicates that the proposed approach (i.e., HGRec) outperforms the state-of-the-art recommenders in terms of accuracy. Moreover, among the top-3 accurate recommenders, HGRec is more likely to recommend new reviewers out of core reviewers, which may help to alleviate the workload congestion issue to some extent. Last but not least, with flexible and natural model architecture, HGRec can support modeling more elements (e.g., entities, attributes and relationships) in a way that more modern learning techniques or sophisticated heuristic algorithms could be incorporated into the recommender. To this end, better performance can be expected with exploration in the future.

ACKNOWLEDGMENTS
This work is supported by the National Natural Science Foundation of China (No.62072227, No.61802173), the National Key Research and Development Program of China (No.2019YFE010550) jointly with the Research Council of Jiangsu (No.309494), the Key Research and Development Program of Jiangsu Province (No.BE2021002-2), as well as the Intergovernmental Bilateral Innovation Project of Jiangsu Province (No.BZ20200017).

REFERENCES
[1] Wisam HAitham Abbood Al-Zubaidi, Patanamon Thongtanum, Hoa Khanh Dam, Chakkrit Tantithamthavorn, and Aditya Ghose. 2020. Workload-aware reviewer recommendation using a multi-objective search-based approach. In Proceedings of the 16th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE ’20). ACM, 21–30.
[2] Omar Alonso, Premkumar T Devanbu, and Michael Gertz. 2008. Expertise identification and visualization from cvs. In Proceedings of the 5th international working conference on Mining software repositories (MSR ’08). IEEE, 125–128.
[3] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug? In Proceedings of the 28th International Conference on Software Engineering (ICSE ’06). ACM, 361–370.
[4] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Maddila, Sonu Mehta, and B Ashok. 2019. WhoDoes: automating reviewer suggestions at scale. In Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ’19). ACM, 937–945.
[5] Alberto Baccelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In Proceedings of the 5th International Conference on Software Engineering (ICSE ’13). IEEE, 712–721.
[6] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2013. The influence of non-technical factors on code review. In Proceedings of the 20th Working Conference on Reverse Engineering (WCER’ 13). IEEE, 122–131.
[7] Moritz Beller, Alberto Baccelli, Andy Zaidman, and Elmar Jaeger. 2014. Modern code reviews in open-source projects: Which problems do they fix? In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR’ 14). ACM, 202–211.
[8] Amani Shou Bou and Jeffrey C Carver. 2013. Impact of peer code review on peer impression formation: a survey. In Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM’ 13). IEEE, 133–142.
[9] Amani Shou Bou, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Checkley. 2016. Process aspects and social dynamics of contemporary code review: insights from open source development and industrial practice at microsoft. IEEE Transactions on Software Engineering 43, 1 (2016), 56–75.
[10] Jujuan Bu, Shulong Tan, Chun Chen, Can Wang, Hao Wu, Lupin Zhang, and Xiaofei He. 2010. Music recommendation by unified hypergraph: combining social media information and music content. In Proceedings of the 18th International Conference on Multimedia (MM’ 10). ACM, 391–400.
[11] Manoel Limeira de Lima Júnior, Darielício Moreira Soares, Alexandre Plastino, and Leonardo Murta. 2013. Developers assignment for analyzing pull requests. In Proceedings of the 30th annual ACM symposium on applied computing (SAC ’15). ACM, 1567–1572.
[12] Manoel Limeira de Lima Júnior, Darielício Moreira Soares, Alexandre Plastino, and Leonardo Murta. 2018. Automatic assignment of integrators to pull requests: the importance of selecting appropriate attributes. Journal of Systems and Software 144 (2018), 181–196.
[13] Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. 2011. A tensor-based algorithm for high-order graph matching. IEEE transactions on pattern analysis and machine intelligence 33, 12 (2011), 2383–2395.
[14] Tudor Gîrba, Adrian Kuhn, Mauricio Seeberger, and Stéphane Ducasse. 2005. How developers drive software evolution. In Proceedings of the 8th international workshop on principles of software evolution (WPSE ’05). IEEE, 113–122.
[15] Nils Göde and Rainer Koschke. 2011. Frequency and risks of changes to clones. In Proceedings of the 33rd International Conference on Software Engineering (ICSE ’11). 311–320.
[16] Christoph Hannebauer, Michael Patalas, Sebastian Stinkel, and Volker Grunn. 2016. Automatically recommending code reviewers based on their expertise: an empirical comparison. In Proceedings of the 31st International Conference on Automated Software Engineering (ASE ’16). IEEE, 149–159.
[17] Gaeul Jeong, Bongjun Kim, Thomas Zimmermann, and Kwangkeun Yi. 2009. Improving code review by predicting reviewers and acceptance of patches. Research on software engineering for error-free computing center Tech-Memo (2009), 1–18.
[18] Jing Jiang, JiHuan He, and XueYuan Chen. 2015. Coredevrec: automatic core member recommendation for contribution evaluation. Journal of Computer Science and Technology 30, 5 (2015), 998–1016.
[19] Jing Jiang, David Lo, Jiaying Zheng, Xin Xia, Yun Yang, and Li Zhang. 2019. Who should make decisions on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction. Journal of Systems and Software 154 (2019), 196–210.
[20] Jiajun Jiang, Luyao Ren, Yingfei Xiong, and Lingming Zhang. 2019. Inferring program transformations from singular examples via big code. In Proceedings of the 34th International Conference on Automated Software Engineering (ASE’ 19). IEEE, 255–266.
[21] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanck, and Li Zhang. 2017. Who should comment on this pull request? analyzing attributes for more accurate commentor
