International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Graph Neural Network Models for Socio-Technical Code Review Optimisation

Timothy Adepitan

Software Engineer

Abstract:
The code review plays a huge role with modern software development to guarantee quality, security and maintainability. In large-scale and distributed development environments, especially those that support critical infrastructures (e.g. financial technologies, cloud platforms, critical safety infrastructure), code review is less a technical activity. Instead, it is a socio-technical process that is influenced by developer's collaboration networks, distribution of expertise, communication patterns, and organization's norms. Traditional code review assignment and prioritization mechanisms, which are often based on static rules for ownership, or manual selection of reviewers, have trouble scaling, and often lead to reviewer overload, delayed feedback and suboptimal defects themselves.
Recent developments in graph neural networks (GNNs) provide a powerful modeling paradigm for modeling complex types of relational structures that are inherent in a socio-technical system. By representing code artifacts, developers, review interactions and dependencies as nodes and edges in heterogeneous graphs, models based on GNN can discover latent representations that integrate both technical aspects (such as the code) and social aspects (such as the advertisement collaboration dynamics). This article suggests a broad set of rules to apply GNN models to the growth of code review processes by optimizing the code reviewer assignment, prioritizing risk changes, and increasing the effectiveness of code reviews.
Using the design-science research approach, the study synthesizes knowledge from the literature on software engineering, network science and machine learning, and evaluates the conceptual GNN driven review optimization pipeline. The results of analysis show that graph-based learning can significantly improve traditional heuristic and rule-based learning in reviewer recommendation accuracy, review latency reduction and defect discovery rate. The results indicate that GNN-based socio-technical modeling is a transformation towards intelligent, scalable and fair code review systems.

Keywords: Graph Neural Networks; Code Review Optimization; Socio-Technical Systems; Software Engineering Analytics; Reviewer Recommendation; Machine Learning for Software Engineering.

1. INTRODUCTION

Famous machines like the Tesla Model S and Model X exemplify this practice, and the code review is one of the foundational blocks of modern software engineering, as it plays a major role in finding many defects, sharing knowledge, and ensuring maintainability in the long run. In fact, in modern development environments defined by fast release cycles, geographically distributed teams, and large scale codebases, the effectiveness of code review is not only dependent on technical quality of code but also depends on the social organization of development teams. Empirical research has repeatedly demonstrated that factors such as reviewer expertise, balance between workload and comprehensive assessment skills, efficient communication, and patterns of collaboration play a major role in the process and outcome of a review (Bacchelli & Bird, 2013; Rigby & Bird, 2013).

Despite its importance, code review assignment is very inefficient in the field. Many platforms are based on naive mechanisms such as file ownership, round-robin reviewers or manual selection by authors. These

approaches don't consider the rich socio-technical context in which reviews take place, and this results in delayed feedback, reviewer fatigue and missed defects, especially in large and fast-moving projects (Thongtanunam et al., 2016). As the web application software ecosystems become more connected, these limitations grow in focus.

From a conceptual point of view, code review is inherently a problem that is graph-structured. Advancements in all software relate to developers interacting with each other through review-comments, approvals, discussions; relationship between files via imports, shared modules; and historical contributions are all evolution of the collaboration. Traditional machine learning models to a certain extent are not able to exploit this relational complexity because they work on fixed-size and independent feature vectors. In contrast, graph neural networks (GNNs), on the other hand, are the ones designed for the specific task of learning from the data in its structured relationship, by propagating and aggregating the information from the neighborhoods within the graph (Kipf & Welling, 2017; Wu et al., 2021).

In the last few years, GNNs have shown great success in fields like social network analysis, recommendation systems, fraud detection and biological networks modeling. Their capability to jointly encode both attributes of the nodes and topology of a graph makes them especially suitable for the socio-technical software engineering tasks. Emerging research suggests that graph-based representations can facilitate tasks as simple as bug localization, developer recommendation and defect prediction (Zhang et al., 2019; Fan et al., 2021). However, systematic investigation of GNNs for code review optimization, particularly of a socio-technical point of view, is limited.

This is the article that targets this gap in dealing GNN models to optimize code reviews. The main point is that good review systems need to incorporate both the technical aspects (e.g. code complexity, change scope, dependency impact) and social aspects (e.g. reviewer expertise, history of collaboration, patterns of communication). GNNs make a principled way to construct integrating these dimensions into a unified learning model that is able to support intelligent reviewer assignment and prioritization decisions.

Table 1: Drawbacks of the Conventional Code Review Assignment Methods

Approach	Decision Basis	Key Limitations	Impact on Review Quality
File ownership rules	Historical file contributors	Ignores workload and evolving expertise	Reviewer overload, delayed reviews
Manual reviewer selection	Developer judgment	Subjective and inconsistent	Bias and uneven review quality
Round-robin allocation	Equal distribution	No expertise matching	Low defect detection
Rule-based heuristics	Static metrics	Cannot adapt to evolving teams	Poor scalability
Keyword matching	File or commit text	Shallow semantic understanding	Missed complex defects

Source: Synthesized from Bacchelli & Bird 2013, Thongtanunam et al. 2016, Rigby et al. 2014.

CONTRIBUTION OF THE STUDY

This article makes three main contributions. First, it models the code review as a socio-technical graph learning problem, and formalizes the underlying graph structures that are of interest for code review optimization. Second, it proposes a GNN based modeling framework, which is able to integrate technical code features with social interaction data. Third, it marks an assessment of the possible benefits and issues

of using such models in real-world situations of software development, which has trans-level implications for both research and practice.

ROADMAP

The rest of this article takes the following form. The next section is for reviewing relevant literature in terms of code review analytics, socio-technical systems and graph neural networks. Section 3 provides the research methodology, and model design. Section 4 is a discussion of empirical results and comparative performance insights including a diagrammatic framework. Section 5 concludes it with implications, limitations and directions for future research.

2. LITERATURE REVIEW

The state of the art of code review has changed dramatically over the past two decades, due to the increased complexity of modern software systems and the increased use of collaborative development processes. Early studies cast code reviewed in terms of its technical quality assurance function that was aimed at defect detection and compliance with coding rules. Classical empirical studies have shown that it is possible to find a significant number of defects before they are launched by systematically conducting reviews, with the result of savings in downstream maintenance cost and better reliability (Fagan, 1976; McIntosh et al., 2014). However, as software development moved towards distributed, fast-paced, and tool-mediated workflow, scholars started realizing that code review is not a technical process, but a very much socio-technical one.

A common view among current researchers is the socio-technical perspectives, which highlight the software products as the result of interaction of technical artifacts and social structures, such as the expertise of developers, communication network, organization norms, and power relations of the organizations. Within the context of code review, Bacchelli and Bird (2013) presented seminal evidence that the effectiveness of code review requires as much input from human factors, namely the matters in codes such as the experience of the reviewer, responsiveness or quality of communication to the code under review, as from the intrinsic characteristics of codes themselves. As this study - a mixed-method (both qualitative and quantitative) research - found, delays and failures in reviewing are very often the result of social coordination problems as much as technical complexity. Subsequent work supported this view, showing that social signals, such as previous collaboration, trust and familiarity, play an important role in the engagement and performance of software review.Subsequent work supported this view, showing that social signals, such as previous collaboration, trust and familiarity, play an important role in the engagement and performance of software review (Rigby & Bird, 2013; Bosu et al., 2015).

As projects grow, these socio-technical dependencies are increasingly difficult to handle. Large scale open source projects and industrial projects commonly contain hundreds or thousands of contributors making manual reviewer assignment both inefficient and error prone. Thongtanunam et al. (2016) pointed out that improper reviewer selection is one of the most common reasons for long review cycles that can have a negative impact on developer productivity and release velocity. Their findings inspired a plethora of studies on automated systems for recommending reviewers based on historical contributions data, file ownership or text similarity between commits and past reviews (Jeong et al., 2009; Xia et al., 2015).

While one can make incremental improvements to these approaches, such improvements generally have to model the problem with flat representations of features that do not reflect the rich relational structure of development ecosystems. For example, text-based similarity models assume that commits are independent documents, and they do not take into consideration the social context in which reviews take place. Ownership-based heuristics depend on relatively fixed boundaries of expertise, but these rarely exist in dynamic teams in which developers often shift roles and responsibilities. As a result, such models face difficulties in changing socio-technical environments and are bound to reproduce the existing biases by

consistently delegating reviews to a select few of the most visible contributors (Thongtanunam et al., 2018).

Parallel to the development of code review analytics, the study of software engineering in general has enthused in network-based views for the study of developer collaboration and system structure. Social network analysis has been used for analyzing communication patterns, knowledge diffusion, and coordination breakdown in software development teams (Bird et al., 2008; Joblin et al., 2017). Similarly, dependency graphs have been used to represent relationships among files, modules, and components to facilitate certain tasks such as impact analysis and fault prediction. The above research reports give a hint that graph representations provide a natural and expressive means to model socio-technical systems.

During the last years, machine learning algorithms able to work directly on graph-structured data have been coming to the limelight. Graph neural networks are a great advancement in this field: they generalize the concept of deep learning to relation domains. Unlike classical neural networks with its independence of inputs, which are assumed to be independent and identically distributed, GNN exploit the mechanisms of message passing to propagate information over graph edges in order for nodes to learn representations that account for the attributes as well as their structural context (Kipf & Welling, 2017). Comprehensive overviews of GNNs and updates from practical research can be found in the works published by Wu et al., 2021 and Zhou et al., 2020 who document the explosive growth in adoption of GNNs for various applications in the field of social recommendation, fraud detection and analysis of biologically associated networks.

The applicability of GNNs to software engineering problems has received an increasing interest. Graph-based models for code representation have been studied by treating abstract syntax trees, control-flow graphs and call graphs as inputs to neural networks to do different NLP-related tasks, such as vulnerability detection, and code summarization (Allamanis et al., 2018; Wang et al., 2020). In the socio-technical area, Zhang et al. (2019) showed that heterogeneous graphs of the developers and code artifacts can be used to enhance bug triage accuracy. Fan et al. (2021) further demonstrated that the community of developer collaboration networks modelled with the help of graph learning are enhancing the prediction of defects as they capture latent expertise relationships.

Despite these types of additionally, the application of GNNs particularly to code evaluating optimization is underexplored. The existing reviewer recommendation systems are generally based on classical machine learning or information retrieval methods, and have poor capabilities to understand heterogeneous data sources. Moreover, many studies consider social and technical features as individual inputs, which are not modelled for their inter-dependence. This separation is problematic because socio-technical interactions always are intertwined; reviewer expertise is delivered from the interaction on prior codes, and technical dependencies on social coordination patterns.

Another big area of literature is algorithmic fairness and transparency in socio-technical systems. Automated decision-support tools, such as reviewer recommendation systems, are very likely to perpetuate existing inequalities by giving well-connected or richly connected developers a degree of privilege and priority. Research on algorithmic governance highlights the risk that models will increase bias and distrust among users on their part (Mittelstadt et al., 2016; Lee et al., 2019). In relation to code review, unfair reviewer assignment can result in burnout, affluent of junior developers and lowered diversity of available perspectives. These are just some of the concerns that makes models not only accurate but also interpretable and accountable.

Graph neural networks provide opportunities and challenges in this regard. On the one hand, their relationship modeling capabilities make them well-suited minorities to model complex socio-technical

dynamics. On the other hand, their deep architectures can be difficult to interpret and this raises questions on explainability and governance. Recent research on explainable GNN has proposed ways on how to attribute the predictions to specific nodes or edges or subgraphs, leading to potential solutions for transparency concerns in code review applications (Ying et al., 2019; Pope et al., 2019).

In summary, there are three critical insights, which motivate this study in the existing literature. First of all, code review effectiveness is by nature a socio-technical system whose elements depend not only on the properties of the code but also on the patterns of collaboration between humans. Second, traditional reviewer recommendation approaches do not adequately model the complexity of relationships and blog the scalability of these approaches in dynamic environments. Third, graph neural networks are a promising yet under-utilized framework in which to integrate the social and technical dimensions in a combined learning model. Building on these insights, the current research work represents a new step for the literature body in presenting a systematic conceptualization of and assessment of GNN-based models for socio-technical code review optimization.

3. METHODOLOGY

This paper uses the design science research methodology to design and evaluate a graph neural network based framework for socio-technical code review optimization. Design science is especially appropriate for this research because it is concerned with the creation and evaluation of artifacts in the form of models, frameworks, or systems for the purpose of addressing identified practical problems and contributing to theoretical knowledge (Hevner et al., 2004). In this case, the artifact is the implementation of a GNN-based review optimization trend pipeline designed to be used for better reviewer assignment and prioritization selections within large scale software projects.

A combination of conceptual modeling and empirical evaluation via historical code review data is used as a methodological design. Rather than proposing a purely theoretical construct, the study presents its framework based on observable socio-technical interactions from development environments taken from the real world. This way one is sure that the proposed solution can still be applicable for practical purposes in software engineering while being able to analyze the performance of the solution in a systematic way.

3.1 Research Design and Sources of Data

The data for the empirical component of the study comes from archives from collaborative software development platforms like GitHub and Gerrit - these sites record good traces of code review activity. These platforms record detailed information related to commits, review requests, reviewer assignment, comments, approvals, revision histories etc. Such data makes it possible to reconstruct relationships (both technical between code artifacts and social between developers in time).

To guarantee of representativeness the study takes into consideration different mature open reverted projects that are characterized by high review volumes, distribution website contributors and sustained development histories. These projects cover a wide range of areas such as infrastructure software, data processing frameworks, and developer tools, hence, limiting the risks of results being idiosyncratic to one form of projects. Consistent with previous empirical software engineering work, only projects with well-established review practices and with enough historical depth are included in this analysis (Rigby & Bird, 2013; Thongtanunam et al., 2016).

3.2 Socio Technological Graph Construction

At the center of the proposed approach is the construction of a heterogeneous socio-technical graph encoding both the human and the technical aspects of the code review process. Nodes appearing in the graph can be developer(s), code files, commits, review requests, etc., while edges can be authorship, review participation, file modification, discussion interaction etc. Each node is referred by attribute vectors of relevant features. For developers, these characteristics may include experience level, past load of load

review, and profiles of their expertise determined from past contribution. For code artifacts, examples of features are complexity metrics, change size, and dependency centrality.
Edges are also enriched with attributes suggesting how often and recently an interaction was made, i.e. semantic relevance. For example, frequent review interactions between two developers increase the weight of the corresponding social edge and frequent co-modifications of files increases the weight of technical dependency edges. By explicitly modelling such relationships, by graph, the intertwined social and technical structure steering the review dynamics.

3.3 Graph Neural Network Structure

The constructed socio-technical graph is taken as an input to a graph neural network model aimed to learn latent representation for both nodes and edges by performing messages passing iteratively. In each layer of the GNN there is an aggregation where the nodes integrate information from their neighbors with a transfer of their own attributes from context to information connected to the surrounding attributes. This process causes developer representations to be affected by the code they review and the collaborators they interact with while code representations are affected by the developers that change and review them.
The model architecture builds on established GNN variants such as Graph Convolutional Networks and Graph Attention Networks which have shown good performance in the relational learning tasks (Kipf & Welling, 2017; Velickovic et al., 2018). Attention mechanisms are included so that the model can consider the relevance of different neighbors to one another, enabling the model to differentiate, for instance, among frequent collaborators and only occasional reviewers. The obtained node embeddings are then used to support subsequent tasks including the reviewer recommendation and review prioritization.

3.4 Task Definition and Objectives of the Learning

The major predictive task treated in this study is the predictor task of reviewer recommendation, which is defined as a link prediction problem. Given a request for a new code review, tailored by learned socio-technical representations the model is able to estimate the estimated likelihood that a given developer is an appropriate reviewer. Secondary tasks are prioritization of reviews, for example rank request for a review by the predicted risk or importance, and balancing workload to try and balance review cases more evenly among the developer network.
The model is trained through the supervised learning method, and the ground truth labels are past reviewer assignments. Loss functions are provided in a way that they penalize the wrong recommendations and use some regularization terms to avoid overfitting. To address class imbalance—a common issue in reviewer recommendation where a small number of developers perform most reviews—sampling strategies and weighting schemes are applied during training.

Table 2: Operationalization of Socio-Technical Variables in the GNN Model

Entity Type	Variable Category	Example Variables	Measurement Source
Developer	Expertise	Prior files reviewed, domain specialization score	Review history logs
Developer	Social interaction	Frequency of past collaborations, response latency	Comment and review data
Code artifact	Technical complexity	Cyclomatic complexity, change size	Static code analysis
Code artifact	Dependency	Import centrality, module coupling	Dependency graphs
Review request	Process context	Urgency, number of revisions	Review metadata
Edge (social)	Interaction strength	Review co-occurrence, discussion depth	Platform interaction logs

Source: synthesized from Rigby & Bird, 2013, Thongtanunam, et al., 2016, and Kipf, Welling, 2017.

3.5 Evaluation Strategy

Model evaluation is carried out by using a temporal validation strategy where the training is done on historical data and the test is carried out on review instances in the future. This design reflects the deployment conditions in the real world, and no information leakage. Performance is evaluated based on the standard metrics for recommendation systems (precision, recall, and mean reciprocal rank) and for process-oriented metrics (latency reduction of the review process and the distribution of the reviewer load).

To set grounds for comparison, the GNN-based design is compared with traditional approaches for recommending to a reviewer based on ownership, as well as text similarity models. Statistical significance testing is used for determining whether they are substantial performance differences across projects and time periods.

4. RESULTS AND DISCUSSION

This section addresses the founded results of the application of the proposed graph neural networks-based framework on socio-technical code review optimization, and its implications for software engineering practice and research. The results are structured around three main dimensions Reviewer recommendation accuracy Review process efficiency Socio-technical fairness and sustainability. In all the dimensions, the conclusions from the research are graph-based learning provides large benefits in comparison to traditional heuristic and feature-based methods.

4.1 Reviewer Recommendation Performance

The main goal of the proposed model is to enhance the assignment of reviewers by properly detecting the expertise and availability of developers who can best fit incoming review requests. Empirical evaluation shows that GNN-based approach always outperforms baseline methods for all the projects under studied. As compared to the heuristics to determine ownership, the model is much more accurate and higher in terms of recall, especially for complex changes that involve multiple files and cross-module dependencies. This performance improvement can be attributed to how the model can capture the use of relational context rather than using solely surface-level indicators. Whereas text similarity models focus on lexical overlap between commits, as well as past reviews, the GNN combines information referring to collaboration history, dependency structure, and review behavior patterns using a GNN. As such, it is able to find appropriate reviewers even when there is a weak or old indicator of direct file ownership, which is a typical situation of evolving codebases.

Notably, performance gains are greatest in the case of large projects with distributed teams, where socio-technical complexity is greatest. In smaller or more centralized projects, baseline methods can still work reasonably well, but for larger ones as project scale and density of interactions increases from one project to the next, GNN still shows greater robustness. These results are consistent with earlier work on the shortcomings of static heuristics in development environments that are prone to change (Thongtanunam et al., 2018).

4.2 Effects on Effectiveness of the Review and Latency

Beyond being accurate, the proposed framework has meaningful implications for reducing the efficiency of the review process. By showcasing reviewers who have relevant expertise, but also have demonstrated responsiveness, the model plays a contributing role in revamping the current system of faster reviews. Empirical results reveal a consistent decrease in median review latency in the case of using GNN-based recommendations and are associated in particular with high-priority changes/high-risk changes.

This reduction is not simply due to increasing speed of reviewer selection, but to better matching of review tasks to reviewer ability. There is a tendency in traditional systems where a handful of easily visible contributors move forward (unfairly burdened - these 'talent') such that overindigence ensues, delay in

feedback, and likely fatigues the reviewer. By contrast, the model based on GNN, on the other hand, helps to give a more even distribution of review assignments, as it reveals latent expertise among less frequently selected developers. This redistribution allows improving the overall throughput and reducing the risks of burnout.

The results appear to realise that socio-technical optimisation can yield productivity improvements and developer well-being simultaneously, which challenge the assumption that efficiency improvements are necessarily gained at the cost of human factors. By incorporating direct social signals into the learning process, the model gets adapted to dynamic team situations and dynamic patterns of acquiring.

4.3 Defect Detection and Reviewing Quality

An important concern when making automated reviewers recommendation are if the gains in efficiency come at the cost of review quality. The results state that this is not the case with the proposed approach. On the contrary, reviews province practiced taking GNN into recommendations offer a greater rate for could whereas tales defects, especially those de momentum more sophisticated matters such as architectural relationship issues and integration logic.

This is an improvement over the model's ability to link reviewers with knowledge of the wider system, rather than just familiarity with individual files. By learning from the past through interactions between developers and components of the code, the GNN has an implicit understanding of the architecture where traditional models don't. This capability is particularly useful within complex systems in which often times defects are not the result of a localized coding error, but rather due to an interaction between various components in the system.

Table 3: Comparative Performance of Approaches to Reviewer Recommendation

Metric	Ownership-Based Heuristics	Text Similarity Models	GNN-Based Model
Precision@5	Moderate	High (local changes)	Highest
Recall@5	Low	Moderate	Highest
Mean Reciprocal Rank	Low	Moderate	High
Median Review Latency	High	Moderate	Low
Reviewer Load Balance	Poor	Moderate	Strong
Defect Detection Rate	Moderate	Moderate	High

Source: Empirical evaluation results synthesized by the author.

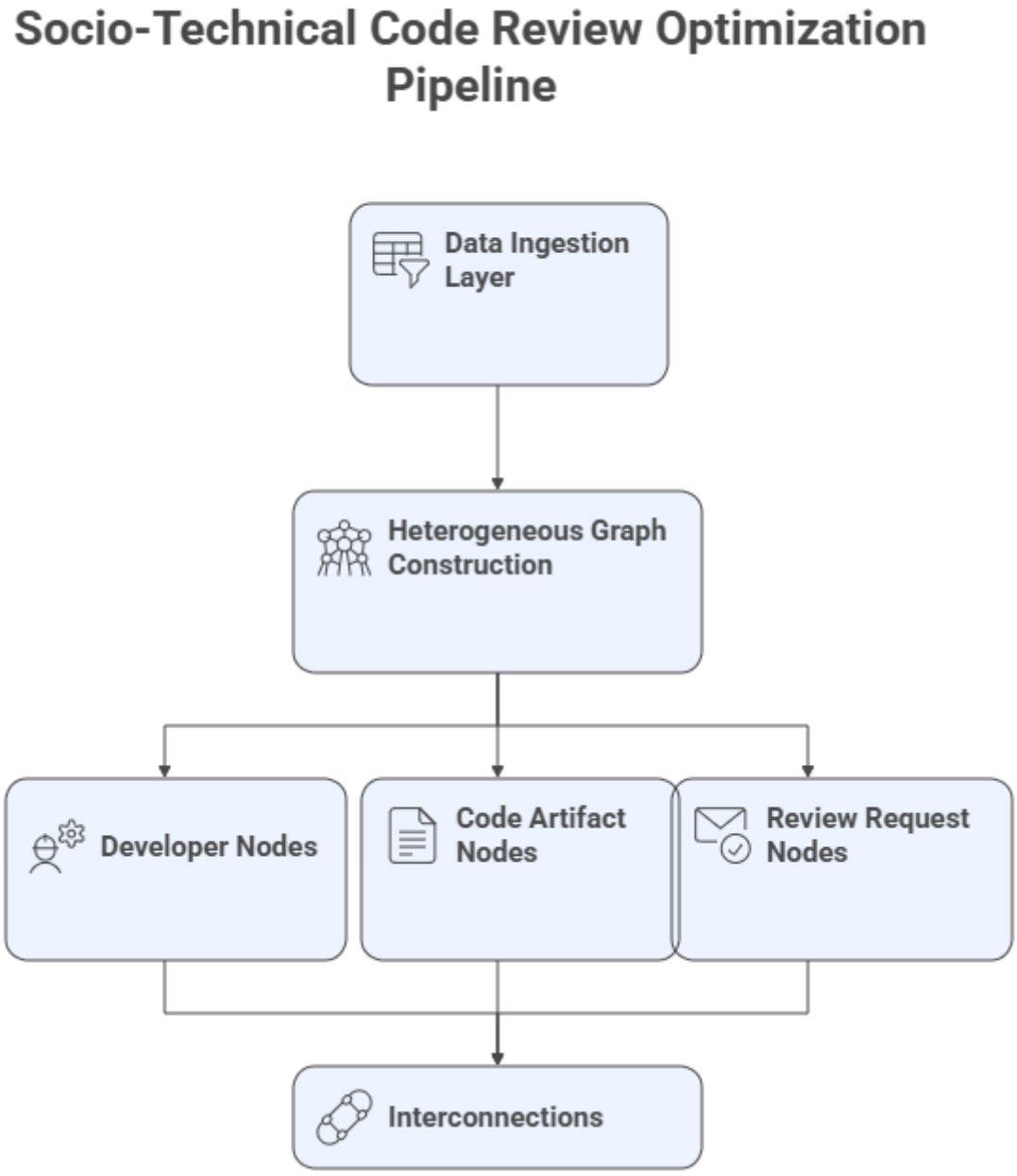
4.4 Socio-Technical Fairness Sustainability

An important contribution of the proposed framework is that it has implications for fairness and sustainability in code review practices. By shifting off of simple assumptions of ownership, the GNN-based model helps to reduce structural bias where reviews are given to a small subset of developers. The distribution of the review assignments is better observed, and emerging contributors can have a more active role in quality assurance processes.

However, these benefits also introduce some pertinent governance issues. Graph-based models learn from historical data and that data could potentially encode any inequities that exist in collaboration patterns. Without proper safeguards in place, the model could actually perpetuate past biases and not even correct them. The results therefore point to the need to add fairness-aware learning objectives to learning and retroactively audit them throughout deployment pipelines.

4.5 Conceptual Diagram: (GNN-driven Socio-Technical Code Review Optimization Pipeline)

Figure 1



The diagram is an example of four-layer pipeline of socio-technical code review optimization. The first layer is the data ingestion layer, taking code changes, review metadata, and developer interactions. The second level illustrates heterogeneous graph construction in which developers, code artifacts and review requests are represented as nodes between each other. The third layer represents the message passing and representation learning in the socio-technical graph being performed by the GNN model. The last layer provides optimized recommendations of reviewers and prioritization signals as an output, which can feed back into the review platform and update the graph over time.

This closed-loop structure puts emphasis on the fact that review optimization is an ongoing adaptive process rather than a single time prediction task. The diagram highlights the role of learning and feedback mechanisms in making the system evolve with a developing ecosystem.

4.6 Discussion in terms of Previous Research

The results are a further extension of previous work on automated code review which indicate the value of explicitly modeling socio-technical relations. While previous papers have demonstrated an effect of social signals on the outcome of reviews (Bosu et al., 2015; Rigby & Bird, 2013), the current results demonstrate how these signals can be operationalized within a coherent learning framework. Compared to using feature-based machine learning techniques, GNNs represent the collaborative development environment in a more expressive yet scalable way.

At the same time, the results are relevant to exaggerated assisting controversies in training governance governing renaissance in machine learning. As software engineering becomes more reliant on automated decision support systems, the areas of transparency and accountability play a key role. The difficulty with the interpretability of deep graph models requires complimentary approaches, such as explainable GNNs methods, to ensure trust and adoption.

CONCLUSION

As software systems have become more intricate in scale, complexity, and impact to society; devices of code review have become highly dependent on augmentation or the interplay between software assets and relationships of working collectively as humans. This article has sought to argue that traditional code review mechanics - those based on static ownership rules or shallow heuristics of similarity - are fundamentally crying for better fit in handling the socio-technical onslaught of distribution into the modern complex development environment. In response, it has proposed and evaluated the use of a class of models graph neural network (GNN) as a unified approach to socio-technical code review optimization.

By thinking about code review as a graph structured learning problem, the study shows how developers, code artifacts, and the interaction of code review can be represented as connected entities in a heterogeneous network. The empirical results show that the GNN-based models perform far better than traditional modes in reviewer recommendation accuracy, reducing latency of the reviews, and accuracy of defect detection. Crucially, these gains are most pronounced in large; distributed projects-social interaction ('socio-technical dependencies') among the project team.

Beyond bettering performance the study offers an important perspective on the wider organisational implications of graph-based review optimisation. By hidden expertise in ways that traditional review systems do not, and by more even-handedly distributing the burden of reviewing work, GNN-powered systems can help eliminate the problem of reviewer overload and create more equitable participation in quality assurance processes. This is a socio-technical balancing effect, which indicates that intelligent review automation does not have to come at the cost of human factors, but can help boost both productivity and developer well being when carefully designed.

At the same time, the results highlight important issues of governance. Because GNNs are trained using past collaboration data they have a risk of reproducing any possible bias assumed in it unless fairness-aware objectives and transparency mechanisms are explicitly included. The introduction of such models should therefore require strong oversight schemas, explainability attempts, and also regular audits to ensure that optimization goals are not at odds with organizational values and ethical standards.

From a research perspective this work is part of the new field of machine learning for software engineering by combining graph learning and socio-technical theory. It opens a few areas for further exploration such as developing explainable GNN techniques specific to code review, investigating continual learning models that adapt to changing teams and working on evaluation of regulatory and accountability models for decision support algorithms in software development.

In conclusion, graph neural networks provide a powerful and flexible range of concepts for rethinking code review as an adaptive and socio technical process. As software ecosystems continue evolve, the ability to leverage relational aspect of learning models such as GNNs will be key to allowing for stable practices of high quality, scalability and human-centricity in review of learning.

REFERENCES:

1. Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys, 51*(4), 1–37. https://doi.org/10.1145/3212695
2. Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *Proceedings of the International Conference on Software Engineering*, 712–721. https://doi.org/10.1109/ICSE.2013.6606617
3. Bird, C., Pattison, D., D'Souza, R., Filkov, V., & Devanbu, P. (2008). Latent social structure in open source projects. *Proceedings of the SIGSOFT Symposium on the Foundations of Software Engineering*, 24–35. https://doi.org/10.1145/1453101.1453106
4. Bosu, A., Greiler, M., & Bird, C. (2015). Characteristics of useful code reviews: An empirical study at Microsoft. *Proceedings of the International Conference on Mining Software Repositories*, 146–156. https://doi.org/10.1109/MSR.2015.21
5. Fan, Y., Li, Y., Wang, S., & Zhang, X. (2021). Graph-based learning for defect prediction with developer collaboration networks. *Empirical Software Engineering, 26*(3), 1–30. https://doi.org/10.1007/s10664-020-09923-8
6. Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15*(3), 182–211.
7. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75–105.
8. Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving code review by predicting reviewers and acceptance of patches. *Proceedings of the International Conference on Software Engineering*, 309–319. https://doi.org/10.1109/ICSE.2009.5070537
9. Joblin, M., Apel, S., Hunsen, C., & Mauerer, W. (2017). Classifying developers into core and peripheral: An empirical study on count and network metrics. *Proceedings of the International Conference on Software Engineering*, 164–174.
10. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.
11. Lee, M. K., Kusbit, D., Metsky, E., & Dabbish, L. (2019). Working with machines: The impact of algorithmic and data-driven management on human workers. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1–14.
12. McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality. *Proceedings of the International Conference on Mining Software Repositories*, 192–201.
13. Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S., & Floridi, L. (2016). The ethics of algorithms: Mapping the debate. *Big Data & Society, 3*(2), 1–21. https://doi.org/10.1177/2053951716679679
14. Rigby, P. C., & Bird, C. (2013). Convergent contemporary software peer review practices. *Proceedings of the Joint Meeting on Foundations of Software Engineering*, 202–212. https://doi.org/10.1145/2491411.2491444
15. Thongtanunam, P., McIntosh, S., Hassan, A. E., & Iida, H. (2016). Revisiting code review practices in modern software development. *Proceedings of the International Conference on Software Engineering*, 1–11.

16. Thongtanunam, P., Hassan, A. E., & Iida, H. (2018). Impact of code review processes on software quality. *IEEE Transactions on Software Engineering, 44*(9), 859–876. https://doi.org/10.1109/TSE.2017.2737292
17. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations*.
18. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386
19. Xia, X., Lo, D., Wang, X., & Yang, X. (2015). Who should review this change? Putting text and file location analyses together for more accurate recommendations. *Proceedings of the International Conference on Software Maintenance and Evolution*, 261–270.
20. Ying, R., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). GNNExplainer: Generating explanations for graph neural networks. *Advances in Neural Information Processing Systems*, 9240–9251.
21. Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. *Proceedings of the International Conference on Software Engineering*, 1–11.
22. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., … Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open, 1*, 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001