4-2024

Improving automated code reviews: Learning from experience

Hong Yi LIN
University of Melbourne

Patanamon THONGTANUNAM
University of Melbourne

Christoph TREUDE
Singapore Management University, ctreude@smu.edu.sg

Wachiraphan CHAROENWET
University of Melbourne

Citation

LIN, Hong Yi; THONGTANUNAM, Patanamon; TREUDE, Christoph; and CHAROENWET, Wachiraphan. Improving automated code reviews: Learning from experience. (2024). *MSR '24: Proceedings of the 21st International Conference on Mining Software Repositories, Lisbon, Portugal, April 15-16*. 278-283.
Available at: https://ink.library.smu.edu.sg/sis_research/8882

Improving Automated Code Reviews: Learning from Experience

Hong Yi Lin
The University of Melbourne
Melbourne, Australia
holin2@student.unimelb.edu.au

Patanamon Thongtanunam
The University of Melbourne
Melbourne, Australia
patanamon.t@unimelb.edu.au

Christoph Treude
Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Wachiraphan Charoenwet
The University of Melbourne
Melbourne, Australia
wcharoenwet@student.unimelb.edu.au

ABSTRACT

Modern code review is a critical quality assurance process that is widely adopted in both industry and open source software environments. This process can help newcomers learn from the feedback of experienced reviewers; however, it often brings a large workload and stress to reviewers. To alleviate this burden, the field of automated code reviews aims to automate the process, teaching large language models to provide reviews on submitted code, just as a human would. A recent approach pre-trained and fine-tuned the code intelligent language model on a large-scale code review corpus. However, such techniques did not fully utilise quality reviews amongst the training data. Indeed, reviewers with a higher level of experience or familiarity with the code will likely provide deeper insights than the others. In this study, we set out to investigate whether higher-quality reviews can be generated from automated code review models that are trained based on an experience-aware oversampling technique. Through our quantitative and qualitative evaluation, we find that experience-aware oversampling can increase the correctness, level of information, and meaningfulness of reviews generated by the current state-of-the-art model without introducing new data. The results suggest that a vast amount of high-quality reviews are underutilised with current training strategies. This work sheds light on resource-efficient ways to boost automated code review models.

CCS CONCEPTS

• **Software creation and management**; • **Machine translation**;

KEYWORDS

Code Review, Review Comments, Neural Machine Translation

ACM Reference Format:
Hong Yi Lin, Patanamon Thongtanunam, Christoph Treude, and Wachiraphan Charoenwet. 2024. Improving Automated Code Reviews: Learning from Experience. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643991.3644910

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04.
https://doi.org/10.1145/3643991.3644910

1 INTRODUCTION

Modern code review is a widely adopted quality assurance process that leverages the expertise of experienced reviewers to help maintain the source code. This activity provides an avenue in which novice and new team members can learn from the feedback of experienced reviewers and improve the overall quality of code changes [4, 12, 14, 36]. In the absence of code reviews, code changes become more defect-prone [15, 20, 31].

In practice, code reviews are demanding workloads for reviewers as they require large amounts of time and attention. To help alleviate this stressful and time-consuming process, several works attempt to automate the practice by leveraging large language models to imitate reviewers [16, 17, 19, 23, 32–35]. Most recently, Li et al. [18] proposed CodeReviewer, a pre-trained code model on the largest code review dataset that achieved state-of-the-art performance. However, such methods still treat all review examples (i.e., training data) as equal in quality, irrespective of the experience of the reviewer behind the comment.

In this work, we hypothesise that spending more training on experienced reviewers' examples can help the model pay more attention to critical issues within code changes and communicate better insights into underlying problems, resulting in better quality reviews. Rather than acquiring more data to train the model, we treat the experienced reviewers' examples as low-resource data [7, 24, 30]. Thus, we use oversampling to over-represent target examples during training, such that these examples yield more influence over the model's behaviour, enabling higher-quality review generation.

To investigate this, we fine-tune CodeReviewer [18] with oversampled experienced reviewers' examples to automatically generate code review comments. Then we evaluate our experience-aware oversampling models in terms of correctness [18, 34], level of information [18], and meaningfulness [4, 8, 12, 14]. Through our quantitative and qualitative evaluation, we found that our experience-aware oversampling models can generate more comments that are semantically correct (16%-21%), and more applicable suggestions to the proposed changes (32%-34%) with explanation (9%-16%) than the original model, which achieves 15% for semantic correctness, 22% for applicable suggestions, and 4% for explanation. Lastly, we discovered that our models could provide comments for various issues, especially critical ones related to logic, validation, and resources. These results suggest that a higher quality of reviews can be generated by oversampling examples from experienced reviewers, boosting the automated code review performance without introducing new data.

2 RELATED WORK AND MOTIVATION

Automated code reviews. In its earliest form, the proposed task was to directly refine the submitted pre-review code changes to an improved post-review version, i.e., ($M_{pre} \rightarrow M_{post}$) method pairs mined from the Gerrit code review tool [33]. Later, Tufano et al. [35] tested the vanilla transformer [37] on a multimodal input scenario ($M_{pre}, R_{nl} \rightarrow M_{post}$), where R_{nl} represents a natural language comment that helps guide the code refinement. Other studies also investigated different techniques to improve performance. Thongtanunam et al. [32] used subword tokenisation to unlock the ability to handle previously unseen tokens appearing in M_{post}, whilst others found benefits in using code diff [23] and structure information [19].

Review comment generation. As the ($M_{pre}, R_{nl} \rightarrow M_{post}$) form of code refinement still relies on the comment R_{nl} of a human reviewer, recent techniques [17, 18] have focused on incorporating the ability to perform ($M_{pre} \rightarrow R_{nl}$) review comment generation [34]. Review comment generation embodies the core task of automated code reviews, where the model needs to identify the exact issue within the submitted code M_{pre} from an expansive potential problem space and output a useful and detailed comment R_{nl} that will assist a human developer in improving the quality of code changes for M_{post}. Although efforts have been made to filter out noisy and unrelated comments [17, 18, 34, 35], previous works have not focused on the variation in review quality within the datasets.

Reviewer experience and comment quality. At the core of code review comment quality, having a wealth of experience enables reviewers to provide better insights. Mozilla core developers [14] argue that a meaningful code review needs to provide more valuable feedback than mere suggestions on code formatting and style. Moreover, they also argue that feedback from experienced reviewers is preferred as these experienced reviewers can leverage their understanding of the codebase to share insights on what could break and what can be re-used.

Microsoft developers [4] stated that useful comments identify functional and validation issues. Developers at Samsung Research Bangladesh [12] suggested that comments related to optimisation, redundant code, corner cases, code integration, deprecated features, and coding standards were also useful. In a study with OpenDev developers [36], defects, code improvement opportunities, and alternative solutions were considered the primary usefulness criteria. Furthermore, [4] report that reviewers who had authoring experience with the file under review had a greater number of useful comments. As studies in both industry [4, 8, 12, 25] and open-source environments [14, 15, 36] converge on this finding, it becomes evident that the experienced reviewer demographic requires additional attention.

3 STUDY DESIGN

3.1 Research Questions

Following the notion that review quality is associated with reviewer experience [4, 8, 12, 14, 15, 25, 36], we argue that spending more training on experienced reviewers' examples can help the model improve the quality of reviews. To do so, we treat the experienced reviewers' examples as a low-resource corpus [7, 24, 30]. As such, we propose an experience-aware oversampling approach, where we train a model by targeting reviewers with high authoring and/or reviewing experience and overrepresent their examples during training. With this approach, the oversampled examples would yield more influence over the model's behaviour, enabling higher-quality review generation. Since review comment generation embodies the core task of automated code reviews, in this work, we focus on the effectiveness on *code review comment generation.* To evaluate our approach, we formulate the following research questions.

RQ1: *Can experience-aware oversampling models correctly generate comments?*
RQ2: *Can experience-aware oversampling models generate more informative comments?*
RQ3: *What kind of comments do experience-aware oversampling models generate?*

RQ1 evaluates the correctness of the generated comment against the ground truth [18, 34]. RQ2 evaluates the models' ability to generate actionable and understandable reviews [18]. RQ3 evaluates the models' ability to generate review comments that target more critical issues as expected by developers [4, 8, 12, 14].

3.2 Reviewer Experience Heuristics

To identify experienced reviewers' examples, we calculate ownership metrics that measure the experience (i.e., familiarity with the codebase) of individual reviewers. We leverage traditional ownership metrics from both the authoring [3] and reviewing [31] perspective to represent the reviewer's experience. We calculate the code ownership of a reviewer at the repository level to fit the review environment of the studied data (i.e., GitHub). This level of granularity allows us to capture overall reviewer experience in the project, as target file experience [4, 12, 25] can often be inaccurate due to deletion. The authoring based **Authoring Code Ownership (ACO)** metric is calculated as $ACO(D, R) = \frac{\alpha(D,R)}{C(R)}$ where $\alpha(D, R)$ is the number of commits the reviewer D has contributed to the repository R and $C(R)$ is the total number of commits to the repository. The **Review-Specific Ownership (RSO)** metric is calculated as $RSO(D, R) = \frac{r(D,R)}{\rho(R)}$ where $r(D, R)$ is the number of closed pull request reviews in a repository R for which the reviewer D has provided comments and $\rho(R)$ is the total number of closed pull request reviews that have been conducted in the repository R.

Following the traditional approaches, reviewers with ACO ≥ 5% are considered ***major authors***, whilst those with ACO < 5% are considered ***minor authors*** [3]. Similarly, those with RSO ≥ 5% are considered ***major reviewers***, whilst those with RSO < 5% are considered ***minor reviewers*** [31].

3.3 Data Preparation

Dataset selection. We used the datasets of Li et al. [18] who introduced the largest multilingual code review dataset to date. This dataset was mined from GitHub pull requests and contains code reviews from the top 10k most starred projects. The training set was built from repositories with more than 2,500 PRs, while the validation and test sets were built from those with [1,500,2,500) PRs. Li et

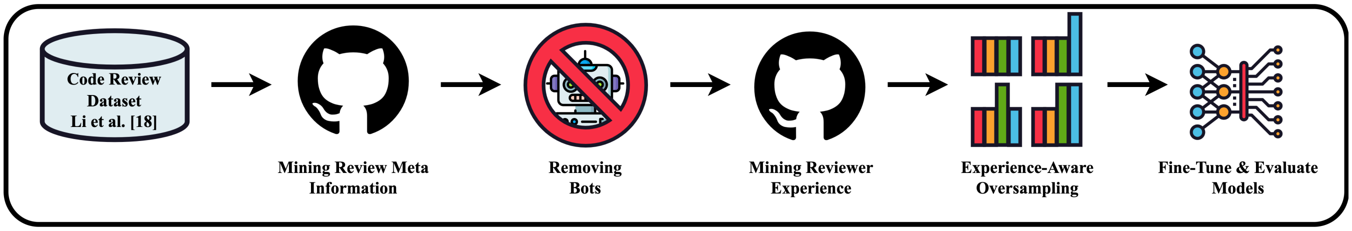
Code Review Dataset Li et al. [18]
Mining Review Meta Information
Removing Bots
Mining Reviewer Experience
Experience-Aware Oversampling
Fine-Tune & Evaluate Models

Figure 1: The Process of Creating Experience-Aware Automated Code Review Models

al. [18] provide three separate datasets for three standalone automation tasks, i.e., code change quality estimation ($M_{pre} \rightarrow revise?$), review comment generation ($M_{pre} \rightarrow R_{nl}$), and code refinement ($M_{pre}, R_{nl} \rightarrow M_{post}$). Since we need to further mine the ownership of reviewers from GitHub repositories, we used the code refinement dataset which still retained PR IDs along with repository names, and we repurposed this dataset for the task of review comment generation, i.e., ($M_{pre} \rightarrow R_{nl}$). The original sizes for the training, validation, and test sets were 150,406 comments, 13,103 comments, and 13,104 comments, respectively.

Mining review meta information. To gather the meta information of the reviews for ownership calculation, we used the GitHub REST API through PyGithub[1] to retrieve the original pull requests. Since there can be many review comments on a single pull request, we obtained only the code review comment that exactly matched the comments in the dataset. For each of the code review comments obtained, we extracted the username of the reviewer and the time of the comment. In total, we identified 10,583 reviewer accounts in the training set and 2,763 reviewer accounts in the validation and test set, covering the period between 2011 and 2022.

Removing bots. While reviews from bots (e.g., CI bots, style checkers) do not harm the model, the main goal of automated code reviews is to replicate human reviews as a means to complement traditional tools. We utilise robust heuristics [10] to 1) remove accounts with the *"bot"* suffix in their username [39] and 2) remove accounts within an established list of bots [11]. In total, we removed 1,207 comments from nine bots in the training set and 96 comments from five bots in the validation and test set. Similar to previous work [18], we removed comments that only suggest code without providing any natural language comment. The final training, validation, and test sets have sizes of 141,259 comments, 12,406 comments, and 12,369 comments, respectively.

Mining reviewer experience. Since reviewers have different ACO and RSO profiles at each point in time, we retrieved their authoring and reviewing histories with respect to each PR in the dataset. For ACO of a reviewer D for a PR p, we used PyDriller [28] to retrieve the number of previous commits that D authored and the total number of commits in the corresponding repository. For RSO of a reviewer D for a PR p, we used the GitHub search API via their GraphQL implementation to retrieve the number of previous PRs in which the reviewer D participated and the total number of PRs in the corresponding repository. We capture all previous commits and PRs that were submitted before the submission of a PR p since the inception of the repository.

Table 1: Transformation of Training(Tr), Validation(Val) and Test(Te) sets.

	Tr	Val	Te
Original Size	150,406	13,103	13,104
Deleted Reviews	618	24	41
Bot Reviews	1,207	41	55
No Natural Language Comment	7,322	632	639
Final Size	141,259	12,406	12,369

Table 2: Accounts in Training(Tr), Validation(Val) and Test(Te) sets.

	Tr	Val	Te
Reviewer Accounts	10,583	2,148	2,125
Identified Bot Accounts	9	3	4

3.4 Experimental Setup

Model Selection. We experiment with the state-of-the-art automated code review model, CodeReviewer [18]. This model is a 225M parameter transformer which was pre-trained from CodeT5's weights [38]. The code review oriented pre-trained model was then fine-tuned into three standalone models for different review tasks. In this work, we focus on the review comment generation model. Since we repurpose the code refinement dataset to the comment generation task, we replicate the original comment generation model using our newly prepared dataset for a fair comparison in our experiment. Note that although the model is re-finetuned, we achieve a similar performance (BLEU-4 of 7.27) as in the original paper (BLEU-4 of 5.32) [18].

Experience-aware oversampling. We explore different types of experienced reviewers: a) Major Reviewer Major Authors only (***MRMA***), b) all Major Reviewers (***MR***) and c) all Major Authors (***MA***). Table 3 shows a proportion of examples in each experience type. We fine-tune the pre-trained CodeReviewer model by targeting one of the three types. To do so, we upsample the subset of reviews associated with the target experience type by 400% in the training data to achieve a 2:3 ratio for the smallest partition [24].

Settings. We fine-tune the original and the three oversampling models (i.e., MRMA, MR, MA) using the same hyperparameters as Li et al. [18], i.e., a batch size of 72, learning rate of 0.0003, beam search

[1] https://github.com/PyGithub/PyGithub

Table 3: Distribution of Examples in the Training (Tr), Validation (Val), and Test (Te) sets.

	Major Reviewer			**Minor Reviewer**		
Major Author	14%	40%	42%	7%	18%	19%
Minor Author	21%	18%	17%	58%	24%	22%
	Tr	***Val***	***Te***	***Tr***	***Val***	***Te***

width of 10, and trained the model for 30 epochs. For hardware, we used a 32-core server with four NVIDIA A100-80G GPUs.

3.5 Evaluation

We used one quantitative measure and five human evaluation tasks:

- **BLEU-4 (RQ1)**: Similar to Li et al. [18], we use BLEU-4 [22] to assess the deviations in performance as a canonical benchmark.
- **Sematic Equivalence (RQ1)**: We manually examined whether the generated comments are semantically equivalent, i.e., same intention as the ground truth, regardless of the degree of textual overlap [18, 34].
- **Applicability (RQ2)**: We manually determine that the generated comment is considered applicable if it raises a valid suggestion or concern in the context of the submitted PR, regardless of its semantic equivalence with the ground truth.
- **Feedback type (RQ2)**: We categorise the comments into three distinct types: *Suggestion* (i.e., proposing a solution), *Concern* (i.e., raising issues), and *Confused Question* (i.e., showing a lack of understanding or a need for clarification).
- **Presence of explanation (RQ2)**: We examine whether a comment includes its rationale or explanation.
- **Comment Category (RQ3)**: We categorise a comment based on 18 categories developed by past work [4, 12, 21]: *Larger Defect*, *Validation*, *Logical*, *Interface*, *Solution Approach*, *Question*, *Design Discussion*, *Resource*, *Documentation*, *Organization of Code*, *Alternate Output*, *Support*, *Timing*, *Naming Convention*, *Praise*, *Visual Representation*, *False Positives*, and *others*.

We measure BLEU-4 on the entire test set. The manual evaluation tasks were conducted on 100 samples in the test set, which should allow us to generalise conclusions with a confidence level of 95% and a confidence interval of 10%. The first and fourth authors evaluated 25 samples separately, achieving Cohen's kappa (κ) between [0.28, 0.45] for semantic equivalence, [0.12, 0.35] for applicability, [0.52, 1] for feedback type, [0.46, 0.63] for explanation, and [0.17, 0.33] for comment category. After resolving the discrepancies, the first author continued to classify the remaining 75 samples. The classification of these 75 samples are then reviewed by the fourth author to ensure the consistency of manual evalution.

4 RESULTS

RQ1 - Can experience-aware oversampling models correctly generate comments? Table 4 shows the average BLEU-4 and the number of comments that are semantically equivalent to the actual comments by any reviewers (All), by major reviewer major authors (◊♦), by major reviewers (◊), and by major authors (♦) in the test set. The results show that although the BLEU-4 scores of our oversampling models are slightly lower than those of the original model, all of our oversampling models achieve a higher number of comments that are semantically equivalent to the ground truth compared to the original model. The lower BLEU is due to harsher penalties to any differences in short sentences [9, 27]. Prior work also report that BLEU is known to poorly correlate with human judgement [2, 5, 13, 26, 29] and ignore the semantic quality [1]. Specifically, the *MA* model achieves the highest number of semantically equivalent comments. The findings highlight a better alignment with experienced reviewers' perspectives in our models.

Table 4: BLEU-4 (B4) on the Test Set & Semantic Equivalence (SE) on the 100 Samples

			Oversampling models					
	Original		MRMA		MR		MA	
	B4	*SE*	*B4*	*SE*	*B4*	*SE*	*B4*	*SE*
All	7.27	15/100	7.12	18/100	7.1	16/100	7.11	21/100
◊♦	6.99	6/32	6.78	6/32	6.94	7/32	6.85	11/32
◊	7.09	7/52	6.93	10/52	6.98	9/52	6.89	14/52
♦	7.12	9/48	6.9	10/48	6.99	11/48	7	15/48

◊♦ comments by Major Reviewer Major Authors in test set

◊ comments by Major Reviewers & ♦ comments by Major Authors in test set

RQ2 - Can experience-aware oversampling models generate more informative comments? Table 5 shows that all of our oversampling models have produced more applicable comments than the original model. Based on the applicable comments, we further evaluate the level of information. We find that our oversampling models provide more comments with suggestions and less concerns than the original model. On the contrary, only 55% of the original approach's applicable comments were suggestions. The original model has a higher tendency to generate confused questions (6/40) than our oversampling models, which generate 1–4 confused questions. We also find that the original model seldom explains (4/40), whilst our oversampling models justified themselves more. Our results suggest that our oversampling models exhibit a notable increase in solution-oriented suggestions with explanation, whilst showing less confusion, which is a common review anti-pattern [6].

RQ3 - What kind of comments do experience-aware oversampling models generate? Figure 2 shows that our models can generate new comments on functional issues e.g.,*Logical*, *Validation*, *Resource*, despite that they are rarely raised [8]. In terms of maintenance-related issues, the *MR* model excelled in this domain, while the *MRMA* model displayed a higher propensity to suggest *Naming Convention* related fixes. On the other hand, the original model could not provide any *Organization of Code* related feedback. Moreover, the original model tends to ask questions more often. The results also highlight that our oversampling techniques can elicit more critical types of comments.

5 THREATS TO VALIDITY

We use the same pre-trained model, hyper-parameters, and training setup as CodeReviewer [18] so that the only varying factor is the fine-tuning dataset and our oversampling strategies. To ensure the validity of the results, we replicate the original model and confirm

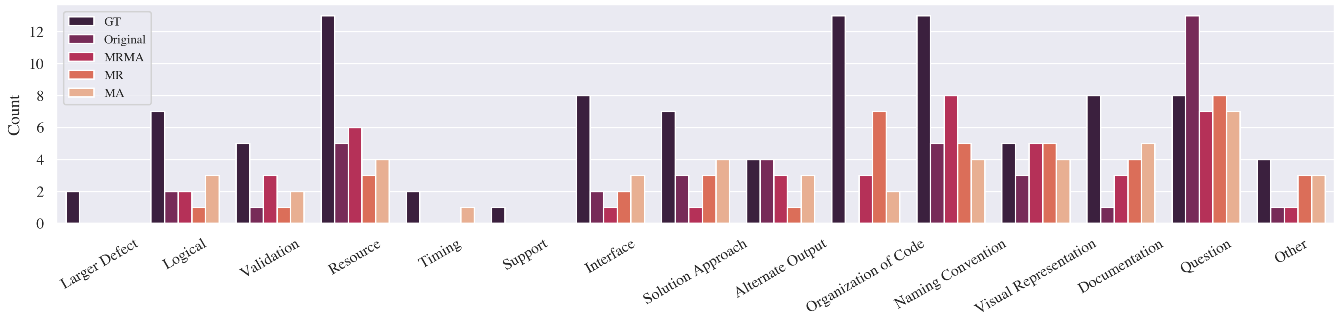

Figure 2: Comment Categories of Applicable Code Review Comments

Table 5: Human Evaluation of Applicability, Feedback Type, and Presence of Explanation on the 100 Samples

			Oversampling models		
	GT	Original	MRMA	MR	MA
Applicability	100	40	43	43	45
Suggestion	77	22	32	34	32
Concern	20	12	7	8	11
Confused Question	3	6	4	1	2
Explanation	68	4	15	16	9

that the model performance is similar to the original paper's results (5.35 vs 5.32) based on BLEU-4 on their test set.

Experience metrics may be underestimated when reviews or commits are deleted, users are unsearchable or use multiple accounts. We treat these false negatives as noise, which should only under represent the potential impact of our technique.

Selected upsampling ratios are arbitrary and do not reflect the best upsampling performance that can be achieved in this dataset. We leave tuning for the optimal ratio to future work.

The outcomes of the manual evaluations are prone to subjective perspectives of human evaluators. To mitigate this, two annotators independently evaluate the sample and discuss to (1) resolve all disagreements and (2) apply the shared understanding to the rest of the annotation. We include annotation results in the materials for transparency. The data is subject to the confines of GitHub. As such, the targeted reviewer demographic does not reflect their counterparts in other software development environments.

6 CONCLUSION

This study explores the ability of the automated code review model to generate higher-quality reviews by oversampling experienced reviewers' examples within the training set. Our results show that experience-aware oversampling allowed the model to generate more semantically correct comments and convey better information by providing more suggestions and explanations. Additionally, the model was able to generate more comments related to functional issues. As the underlying dataset is fundamentally unchanged, we demonstrate the existence of untapped knowledge within the experienced reviewer partitions of the training data.

In future work, we intend to better understand the behavioural differences caused by experience-aware oversampling by investigating changes in attention weights. We plan to experiment with different training methods to learn more effectively from experienced reviewers. Additionally, we will also compare with oversampling of novice reviewers to investigate if contrasting effects arise.

Data Availability. All the materials produced from this study are available on Zenodo[2].

ACKNOWLEDGMENT

This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative. Patanamon Thongtanunam was supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE210101091).

REFERENCES

[1] Bogdan Babych and Anthony Hartley. 2008. Sensitivity of Automated MT Evaluation Metrics on Higher Quality MT Output: BLEU vs Task-Based Evaluation Methods.. In *LREC*, Vol. 2008. ELRA, Marrakech, Morocco, 6.

[2] Anja Belz and Ehud Reiter. 2006. Comparing automatic and human evaluation of NLG systems. In *Proc. of EACL*. ACL, Trento, Italy, 313–320.

[3] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *Proc. of ESEC/FSE*. ACM, New York, NY, USA, 4–14.

[4] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proc. of MSR*. ACM, New York, NY, USA, 146–156. https://doi.org/10.1109/MSR.2015.21

[5] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. 2006. Re-evaluating the role of BLEU in machine translation research. In *Proc. of EACL*. ACL, Trento, Italy, 249–256.

[6] Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamon Thongtanunam, Mohamed Wiem Mkaouer, and Kenichi Matsumoto. 2021. Anti-patterns in Modern Code Review: Symptoms and Prevalence. In *Proc. of SANER*. IEEE, NJ, USA, 531–535. https://doi.org/10.1109/SANER50967.2021.00060

[7] Anna Currey, Prashant Mathur, and Georgiana Dinu. 2020. Distilling Multiple Domains for Neural Machine Translation. In *Proc. of EMNLP*. ACL, Online, 4500–4511. https://doi.org/10.18653/v1/2020.emnlp-main.364

[8] Jacek Czerwonka, Michaela Greiler, and Jack Tilford. 2015. Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down. In *Proc. of ICSE*, Vol. 2. IEEE, NJ, USA, 27–28. https://doi.org/10.1109/ICSE.2015.131

[9] Marina Fomicheva and Lucia Specia. 2019. Taking MT evaluation metrics to extremes: Beyond correlation with human judgments. *Comput Linguist Assoc Comput Linguist* 45, 3 (2019), 515–558.

[2] https://zenodo.org/records/10572047

[10] Mehdi Golzadeh, Alexandre Decan, and Natarajan Chidambaram. 2022. On the Accuracy of Bot Detection Techniques. In *Proc. of BotSE*. ACM, New York, NY, USA, 1–5. https://doi.org/10.1145/3528228.3528406

[11] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. 2021. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *J Syst Softw* 175 (2021), 110911.

[12] Masum Hasan, Anindya Iqbal, Mohammad Rafid Ul Islam, AJM Imtiajur Rahman, and Amiangshu Bosu. 2021. Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report. *Empir. Softw. Eng.* 26 (2021), 1–34.

[13] Katharina Kann, Sascha Rothe, and Katja Filippova. 2018. Sentence-Level Fluency Evaluation: References Help, But Can Be Spared!. In *Proc. of CoNLL*. ACL, Brussels, Belgium, 313–323. https://doi.org/10.18653/v1/K18-1031

[14] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code review quality: how developers see it. In *Proc. of ICSE*. ACM, New York, NY, USA, 1028–1038. https://doi.org/10.1145/2884781.2884840

[15] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *Proc. of ICSME*. IEEE, NJ, USA, 111–120. https://doi.org/10.1109/ICSM.2015.7332457

[16] Jia Li, Ge Li, Zhuo Li, Zhi Jin, Xing Hu, Kechi Zhang, and Zhiyi Fu. 2022. CodeEditor: Learning to Edit Source Code with Pre-trained Models. *ACM Trans. Softw. Eng. Methodol.* 32, 6 (2022), 1–22.

[17] Lingwei Li, Li Yang, Huaxi Jiang, Jun Yan, Tiejian Luo, Zihan Hua, Geng Liang, and Chun Zuo. 2022. AUGER: automatically generating review comments with pre-training models. In *Proc. of ESEC/FSE*. ACM, New York, NY, USA, 1009–1021.

[18] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In *Proc. of ESEC/FSE*. ACM, New York, NY, USA, 1035–1047.

[19] Hong Yi Lin and Patanamon Thongtanunam. 2023. Towards Automated Code Reviews: Does Learning Code Structure Help?. In *Proc. of SANER*. IEEE, NJ, USA, 703–707. https://doi.org/10.1109/SANER56733.2023.00075

[20] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2015. An empirical study of the impact of modern code review practices on software quality. *Empir. Softw. Eng.* 21 (2015), 2146–2189. https://doi.org/10.1007/s10664-015-9381-9

[21] Mika V. Mäntylä and Casper Lassenius. 2009. What Types of Defects Are Really Discovered in Code Reviews? *IEEE Trans. Softw. Eng.* 35, 3 (2009), 430–448. https://doi.org/10.1109/TSE.2008.71

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proc. of ACL*. ACL, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[23] Chanathip Pornprasit, Chakkrit Tantithamthavorn, Patanamon Thongtanunam, and Chunyang Chen. 2023. D-ACT: Towards Diff-Aware Code Transformation for Code Review Under a Time-Wise Evaluation. In *Proc. of SANER*. IEEE, NJ, USA, 296–307. https://doi.org/10.1109/SANER56733.2023.00036

[24] Michael Przystupa and Muhammad Abdul-Mageed. 2019. Neural Machine Translation of Low-Resource and Similar Languages with Backtranslation. In *Proc. of WMT*. ACL, Florence, Italy, 224–235. https://doi.org/10.18653/v1/W19-5431

[25] Mohammad Masudur Rahman, Chanchal K. Roy, and Raula G. Kula. 2017. Predicting Usefulness of Code Review Comments Using Textual Features and Developer Experience. In *Proc. of MSR*. ACM, New York, NY, USA, 215–226. https://doi.org/10.1109/MSR.2017.17

[26] Aaron Smith, Christian Hardmeier, and Jörg Tiedemann. 2016. Climbing Mont BLEU: the strange world of reachable high-BLEU translations. In *Proc. of EAMT*. ACL, Riga, Latvia, 269–281.

[27] Xingyi Song, Trevor Cohn, and Lucia Specia. 2013. BLEU Deconstructed: Designing a Better MT Evaluation Metric. *Int. J. Comput. Linguistics Appl.* 4, 2 (2013), 29–44.

[28] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python Framework for Mining Software Repositories. In *Proc. of ESEC/FSE*. ACM, New York, NY, USA, 908–911. https://doi.org/10.1145/3236024.3264598

[29] Liling Tan, Jon Dehdari, and Josef van Genabith. 2015. An awkward disparity between bleu/ribes scores and human judgements in machine translation. In *Proc. of WAT*. ACL, Kyoto, Japan, 74–81.

[30] Xu Tan, Jiale Chen, Di He, Yingce Xia, Tao Qin, and Tie-Yan Liu. 2019. Multilingual Neural Machine Translation with Language Clustering. In *Proc. of EMNLP-IJCNLP*. ACL, Hong Kong, China, 963–973. https://doi.org/10.18653/v1/D19-1089

[31] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In *Proc. of ICSE*. IEEE, NJ, USA, 1039–1050. https://doi.org/10.1145/2884781.2884852

[32] Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. 2022. AutoTransform: Automated Code Transformation to Support Modern Code Review Process. In *Proc. of ICSE*. IEEE, NJ, USA, 237–248. https://doi.org/10.1145/3510003.3510067

[33] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *Proc. of ICSE*. IEEE, NJ, USA, 25–36.

[34] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using Pre-Trained Models to Boost Code Review Automation. In *Proc. of ICSE*. IEEE, NJ, USA, 2291–2302.

[35] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In *Proc. of ICSE*. IEEE, NJ, USA, 163–174.

[36] Asif Kamal Turzo and Amiangshu Bosu. 2024. What makes a code review useful to opendev developers? an empirical investigation. *Empir. Softw. Eng.* 29 (2024), 6.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017), 6000–6010.

[38] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proc. of EMNLP*. ACL, Online and Punta Cana, Dominican Republic, 8696–8708.

[39] Mairieli Wessel, Bruno Souza, Igor Steinmacher, Igor Wiese, Ivanilton Polato, Ana Paula Chaves Steinmacher, and Marco Aurelio Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. of ACM on HCI* 2 (11 2018), 1–19. https://doi.org/10.1145/3274451