





# Automating Code Review Activities by Large-Scale Pre- training

Authors: Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks,  
Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, Neel Sundaresan



# Problem Statement

- The paper addresses the critical challenge of automating code review activities, which play a pivotal role in ensuring software quality and reliability.
- Code review involves the process of systematically examining source code changes to identify errors, improve code quality, and ensure adherence to coding standards.
- The importance of automated code review has grown significantly in software development, driven by the need for faster development cycles, increased code complexity, and the growing demand for high-quality software products.
- Automating code review tasks can lead to significant efficiency gains, allowing developers to focus more on innovation and less on manual code inspection and validation.

# Motivation

- **Challenges in Manual Code Review:** Manual code review is time-consuming, labor-intensive, and prone to human error. Reviewers must meticulously inspect code changes, which can lead to bottlenecks in the development process.
- **Importance of Automation:** Automating code review activities can significantly improve software development efficiency by reducing the time and effort required for code inspection. It allows developers to focus on higher-level tasks, such as innovation and problem-solving, rather than routine code examination.
- **Need for Efficiency:** With the rapid pace of software development and the increasing complexity of codebases, there is a growing need for efficient code review processes that can keep up with the demands of modern software development practices.

# Proposed Solution

**CodeReviewer Model:** The proposed solution is the CodeReviewer model, an encoder-decoder architecture based on Transformer. It is designed to automate code review activities by understanding code changes and generating review comments.

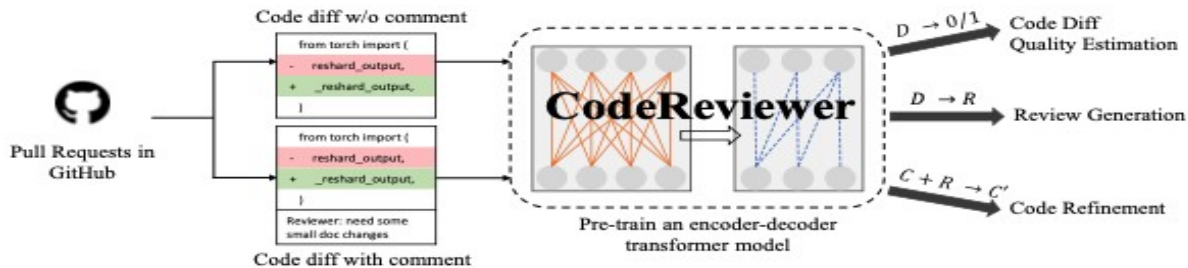


Figure 1: Overview of the workflow of CodeReviewer.

## **Pre-training Tasks:**

**Diff Tag Prediction:** Teaches the model to understand the special line tags in code diff, distinguishing between unchanged and updated lines.

**Denoising Objective:** Trains the model to predict masked spans in code diff and review comments, improving its understanding of code changes.

**Review Comment Generation:** Enables the model to generate natural language review comments based on code changes, enhancing its ability to capture the relationship between code changes and review comments.

# Research Questions Addressed

RQ1: How does CodeReviewer perform on the code change quality estimation task?

RQ2: How does CodeReviewer perform on the review generation task?

RQ3: How does CodeReviewer perform on the code refinement task?

RQ4: What role does each pre-training task play in CodeReviewer?

RQ5: Can multilingual dataset benefit model performance on understanding single programming language?

# Evaluation Metrics:

- Code Change Quality Estimation: Accuracy, Precision, Recall, F1 Score.
- Review Generation: BLEU Score, Human Evaluation (Information, Relevance).
- Code Refinement: BLEU Score, Exact Match Rate.

## **Results:**

Evaluation results on code change quality estimation, review generation, and code refinement tasks.

# Novelty of Proposed Solution

- Introduction of CodeReviewer: A transformer-based encoder-decoder model tailored for automating code review activities.
- Utilization of a large-scale multilingual dataset: Collected from GitHub pull requests, covering nine popular programming languages.
- Integration of four pre-training tasks: Enhancing CodeReviewer's understanding of code changes and review comments.
- Comprehensive evaluation across multiple code review tasks: Demonstrating the effectiveness of the proposed solution.



# Assumptions Made

- Availability of open-source code repositories: Assumes access to GitHub data for collecting a large-scale dataset.
- Single-comment review structure: Each code change is associated with a single review comment, potentially limiting the diversity of perspectives.
- Homogeneity of code review tasks: Assumes similar underlying principles across different code review tasks.

# Limitations of the Proposed Solution

- Dependency on dataset quality: Performance heavily relies on the quality and representativeness of the collected GitHub dataset.
- Interpretability challenges: Transformer-based models like CodeReviewer may lack interpretability, hindering understanding of predictions.
- Generalizability across programming languages: Effectiveness may vary across different programming paradigms and styles.

# Practical significance of the proposed solution

- Automating code review activities streamlines the software development process by reducing manual effort and time spent on reviewing code changes.
- CodeReviewer offers several benefits in software development, including:
  - Faster turnaround time for code reviews.
  - Consistency in review feedback.
  - Increased productivity by enabling developers to focus on high-level tasks.
- The adoption of CodeReviewer can lead to significant improvements in code quality, as it ensures adherence to coding standards, best practices, and bug identification.
- Ultimately, CodeReviewer has the potential to enhance efficiency and effectiveness in software development teams, contributing to faster delivery of high-quality software products.

# Discussion Points

- Which methods can be explored to extend CodeReviewer's language support beyond the current dataset?
- How might changes in software development methodologies affect CodeReviewer's generalizability?
- Can CodeReviewer be applied to tasks beyond code review, such as bug detection?



Q/A