

# A Multi-Step Learning Approach to Assist Code Review

---

Oussama Ben Sghaier  
*University of Montreal*  
Montreal, Canada  
oussama.ben.sghaier@umontreal.ca

Houari Sahraoui  
*University of Montreal*  
Montreal, Canada  
houari.sahraoui@umontreal.ca

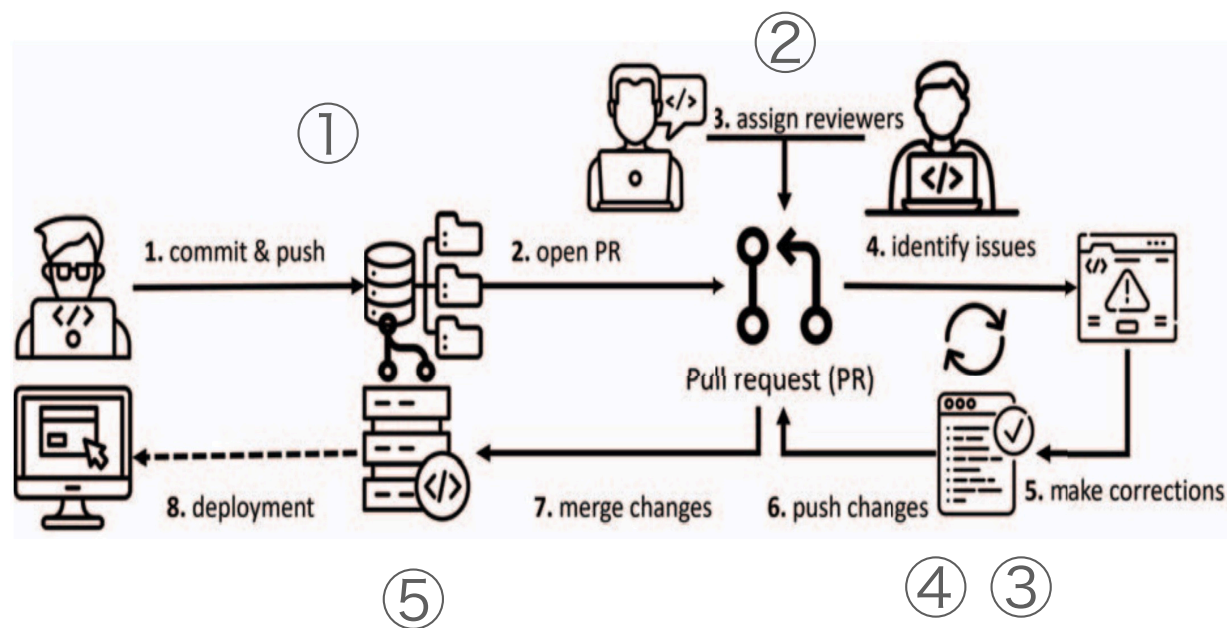
発表：山村慎太郎

# 背景

## コードレビュー

- ・コードの変更を，他の開発者が検査するプロセス
  - 目的：品質の維持，バグの検出，改善点の提案

## コードレビューの流れ



- ① **コード変更**内容を含むpull requestを作成
- ② 別の開発者が，変更内容を検討
- ③ 問題点の指摘や改善案など，フィードバックを提供
- ④ レビューを踏まえて，修正・変更
- ⑤ 問題が解決したら承認・マージ

※問題：変更により発生しうる不具合など

# 背景

---

## コードレビューの課題

- ・ **手作業**で行うため，時間・労力・コストがかかる

- コードレビュー自動化の研究

例) 静的解析による問題の特定，レビュー例やコード変更の提案

## コードレビュー自動化の課題

- ・ プロセスの複雑性や自動化の精度の低さ・使い勝手の悪さ

- レビューの完全な自動化は，現状ではかなり困難

→完全自動化ではなく，レビュアーの作業を支援

# 目的

---

## 本論文の目的

- ・ コードレビュープロセスにおけるレビュアーの支援

## 具体的には？

- ・ **コードの変更**によって引き起こされる**問題**の,  
①**種類** ②**原因となる場所**  
の**予測**(機械学習)によって, コードレビューを支援
- ・ 提案手法のための**データセットの作成**

# 提案手法

## 提案手法の流れ

1. データセットの作成
2. 機械学習モデルによる, 問題の種類の予測
3. 機械学習モデルによる, 問題の場所の予測

## 1.データセット作成の理由

- 2.の目的: **コードの変更**を入力, **問題の種類**を出力
  - **コードの変更** + **種類**のペアをモデルで学習
- プロジェクトごとに, 設定したい**問題の種類**は異なる
  - 幅広いプロジェクトで利用できるようにするため,  
**コードの変更** + **種類**のデータセットを自動で作成できるようにする

# 1. データセットの作成

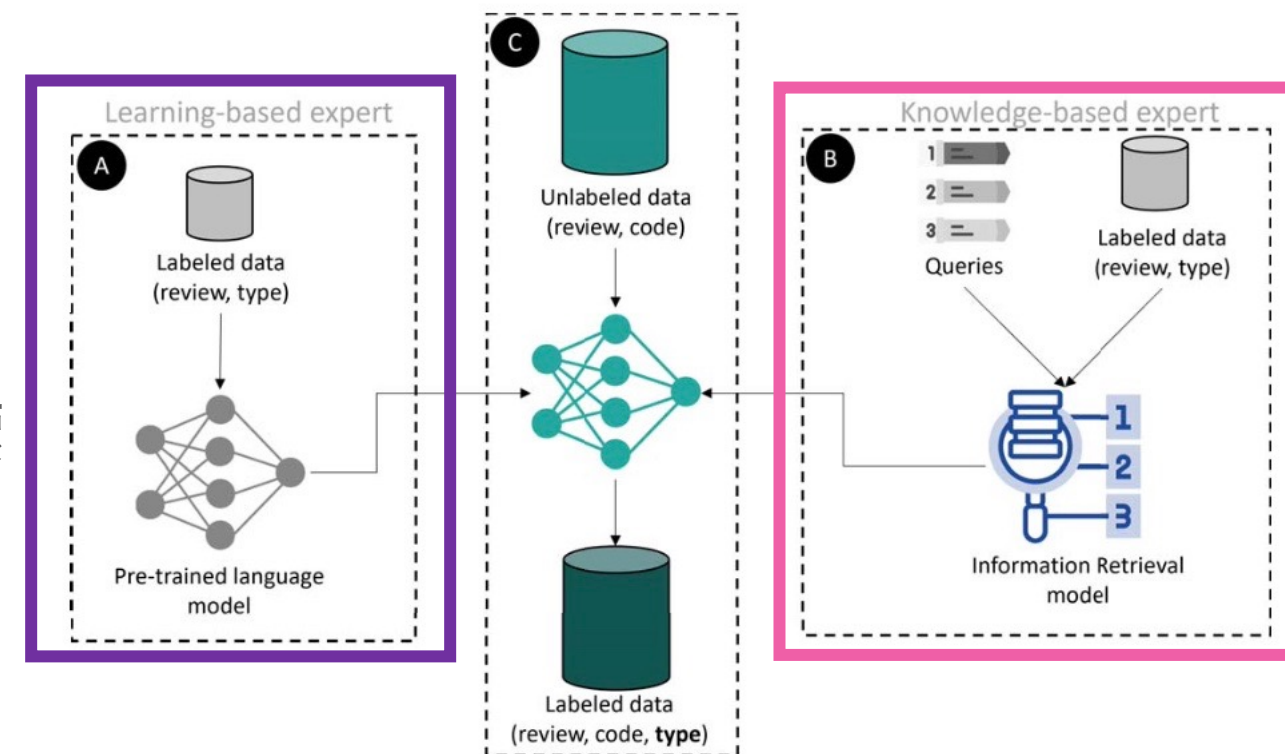
## アンサンブル学習

- ・ タイプのラベル付けには、2手法のアンサンブル学習を採用

### A. 事前学習済みモデル

大量のテキストデータを事前に学習させたモデル RoBERTa を使用

**B. IR (Information Retrieval) … 情報検索**  
大量の情報から特定の情報を効率的に取得する手法  
例) 検索エンジンなど



## 問題タイプの分類（ラベル付け）

- ・ レビュー+コード変更のペアを6種類のタイプに分類
  - 入力：レビュー+コード変更 出力：レビュー+コード変更+タイプ

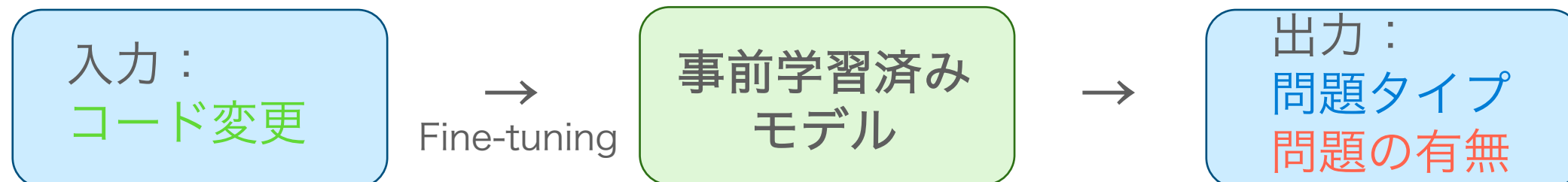
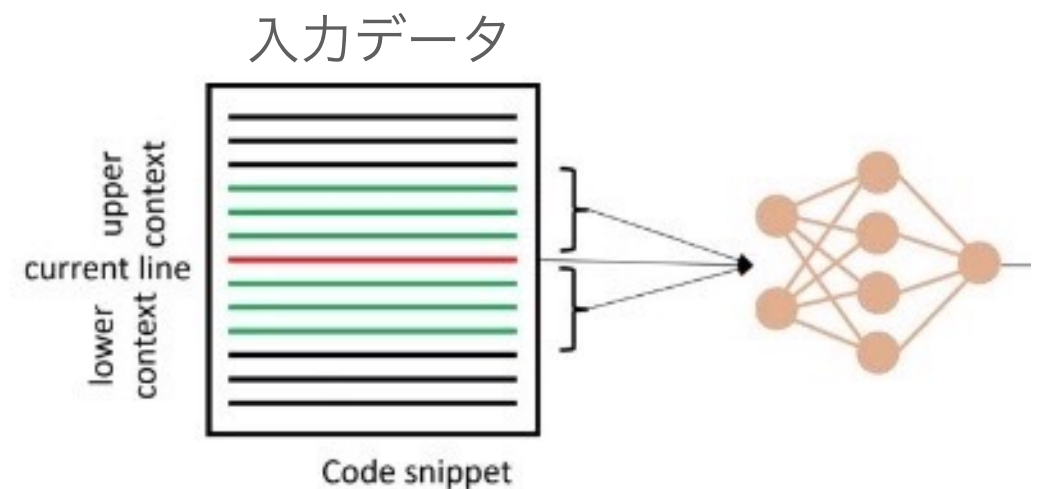
## 2.問題タイプ・3.場所の予測

### 2.問題タイプの予測

- 事前学習済みモデルに1.のデータを与え、問題タイプを予測  
—使用モデル：CodeBERT、GraphCodeBERT、RoBERTa、BERT

### 3.問題の場所の予測

- モデルにコード変更+問題の有無のデータを与え、場所を予測
- 入力の変更行とその上下数行  
→文脈を含めるため  
行数は0~5行に変更して実験
- 予測の流れ



# RQ

## RQ1：データセットの分類は正しく行えていたか？

- 適切な分類かどうかを人間(31人)が評価

| Type (問題の種類)      | Precision | Recall | F1   |
|-------------------|-----------|--------|------|
| Other             | 0.90      | 0.91   | 0.90 |
| Code organization | 0.86      | 0.82   | 0.84 |
| Documentation     | 0.90      | 0.78   | 0.83 |
| Error handling    | 0.79      | 0.96   | 0.87 |
| Testing           | 0.95      | 0.86   | 0.90 |
| Naming            | 0.90      | 0.93   | 0.91 |
| Average           | 0.88      | 0.88   | 0.88 |

- 分類は**かなり正確**に行われていると言える

→データセットとして十分活用可能

## RQ2：アンサンブル学習の効果は？

- 機械学習・IR単体と、2手法のアンサンブルを比較

| Type           | 機械学習 IR アンサンブル |      |      |
|----------------|----------------|------|------|
|                | F1             | F1   | F1   |
| Other          | 0.86           | 0.81 | 0.90 |
| Code organiza  | <b>0.84</b>    | 0.67 | 0.84 |
| Documentation  | 0.78           | 0.75 | 0.83 |
| Error handling | <b>0.85</b>    | 0.78 | 0.87 |
| Testing        | 0.88           | 0.88 | 0.90 |
| Naming         | 0.81           | 0.83 | 0.91 |
| Average        | 0.83           | 0.79 | 0.88 |

- 単体モデルよりも、**アンサンブルモデル**の分類の方が優れている

→アンサンブル学習の効果あり

# RQ

## RQ3.コードの変更から問題タイプを正しく分類できるか

| 問題の種類                         |      |      |      | モデルごとの精度 |      |      |             |             |             |               |             |             |
|-------------------------------|------|------|------|----------|------|------|-------------|-------------|-------------|---------------|-------------|-------------|
| Label                         | BERT |      |      | RoBERTa  |      |      | CodeBERT    |             |             | GraphCodeBERT |             |             |
|                               | P    | R    | F    | P        | R    | F    | P           | R           | F           | P             | R           | F           |
| Other                         | 0.36 | 0.37 | 0.36 | 0.38     | 0.40 | 0.39 | 0.53        | 0.54        | 0.53        | <b>0.57</b>   | <b>0.60</b> | <b>0.59</b> |
| Code organization/Refactoring | 0.29 | 0.35 | 0.32 | 0.27     | 0.31 | 0.29 | 0.71        | 0.69        | 0.70        | <b>0.79</b>   | <b>0.76</b> | <b>0.77</b> |
| Documentation                 | 0.64 | 0.67 | 0.65 | 0.63     | 0.67 | 0.65 | 0.83        | <b>0.85</b> | <b>0.84</b> | <b>0.84</b>   | 0.84        | <b>0.84</b> |
| Error/Resource handling       | 0.79 | 0.80 | 0.79 | 0.78     | 0.79 | 0.78 | 0.87        | 0.88        | 0.87        | <b>0.92</b>   | <b>0.92</b> | <b>0.92</b> |
| Testing                       | 0.82 | 0.81 | 0.81 | 0.83     | 0.84 | 0.83 | 0.84        | <b>0.87</b> | 0.85        | <b>0.86</b>   | 0.86        | <b>0.86</b> |
| Naming                        | 0.49 | 0.42 | 0.45 | 0.51     | 0.47 | 0.49 | <b>0.81</b> | <b>0.82</b> | <b>0.81</b> | <b>0.81</b>   | 0.80        | 0.80        |
| Accuracy                      |      | 0.56 |      |          | 0.57 |      |             | 0.77        |             |               | 0.80        |             |

- GraphCodeBERTモデルが最も正確に分類できた
  - CodeBERT, GraphCodeBERTは, プログラム言語用の事前学習済み
- BERT, RoBERTaは正確性が低い
  - 自然言語に特化したモデルであるため
- 問題の種類ごとにも, 精度に差がみられた
  - コードの内容に関わってくる種類は, 自然言語モデルでは低く出やすい

# RQ

## RQ4.コードの変更から問題の場所を正しく特定できるか

| Label     | BERT | RoBERTa | CodeBERT | GraphCodeBERT |
|-----------|------|---------|----------|---------------|
| Precision | 0.55 | 0.59    | 0.67     | <b>0.71</b>   |
| Recall    | 0.51 | 0.56    | 0.62     | <b>0.69</b>   |
| F1-score  | 0.53 | 0.57    | 0.64     | <b>0.70</b>   |

場所予測の精度(行数3)

- ・ CodeBERT, GraphCodeBERTの精度がやはり高い

→コードを入力とするため

行数

| Window size | Precision   | Recall      | F1-score    |
|-------------|-------------|-------------|-------------|
| $w = 0$     | 0.29        | 0.31        | 0.30        |
| $w = 1$     | 0.52        | 0.57        | 0.54        |
| $w = 3$     | 0.71        | 0.69        | 0.70        |
| $w = 5$     | <b>0.73</b> | <b>0.72</b> | <b>0.72</b> |

行数ごとの精度(GraphCodeBERT)

- ・ 含める行数は、多いほど精度が上がった

→前後の関係・文脈情報を含むため

# まとめ

---

コードレビューの自動化ではなく、支援を目的とした論文

事前学習済み機械学習モデルを用いて、コード変更によって起こりうる問題の種類・場所を予測するモデルを作成

問題の種類分類のため、データセットを作成するためのモデルを、アンサンブル学習を用いて作成

—事前学習済みモデル・IR(情報検索)のアンサンブル