

Automating the Removal of Obsolete TODO Comments

Zhipeng Gao
Monash University
Australia
zhipeng.gao@monash.edu

Xin Xia*
Monash University
Australia
xin.xia@acm.org

David Lo
Singapore Management University
Singapore
davidlo@smu.edu.sg

John Grundy
Monash University
Australia
john.grundy@monash.edu

Thomas Zimmermann
Microsoft Research
United States
tzimmer@microsoft.com

In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021), pp.218–229

Obsolete TODO Comment

- タスクを終えているのにも関わらず，削除されずに残り続けている“TODO”のコメント
- 忙しさなどでソースコードの変更時に削除（もしくは更新）することを忘れる ^{[33][39]}

開発に与える影響

- 開発者を混乱させ，品質の低下を招く ^{[14][25][33]}
- バグの混入を引き起こす可能性がある ^{[33][35]}

- [14] Walid Mlbrahim, Nicolas Bettenburg, Bram Adams, and Ahmed E Hassan. 2012. On the relationship between comment update practices and software bugs. *Journal of Systems and Software* 85, 10 (2012), 2293–2304.
- [25] David Lorge Parnas. 2011. Precise documentation: The key to better software. In *The Future of Software Engineering*. Springer, 125–148.
- [33] Lin Tan, Ding Yuan, Gopal Krishna, and Yuanyuan Zhou. 2007. /* iComment: Bugs or bad comments?*. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 145–158.
- [35] Lin Tan, Yuanyuan Zhou, and Yoann Padioleau. 2011. aComment: mining annotations from comments and code to detect interrupt related concurrency bugs. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 11–20.
- [39] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 53–64.

ツール実現化の障壁

- コメントとソースコードの関係
 - コメントにフォーマットが存在しないため、ソースコードのようにテストできない
 - コメントとソースコードの対応関係の特定が容易でない
- コメントの有効性
 - コメントのタスクが終了しているのかどうか（陳腐化）の検証することは困難である

“TODO”コメントが解決されたときの
変更コード，コミットメッセージも学習する

TDCLEANER

- TODO comment CLEANER

- コードの変更が行われたときに“TODO”コメントが陳腐化していないか2値分類する
- ベースモデルはBERT (Transformer)

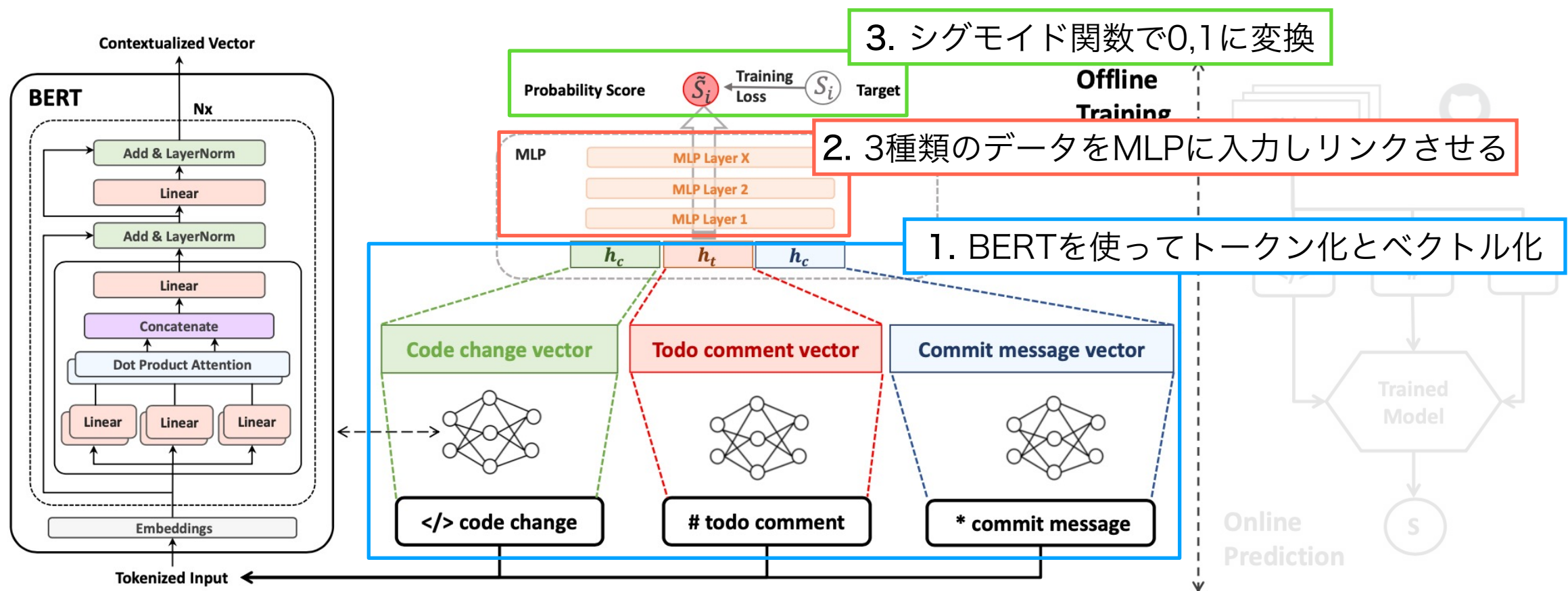


Figure 3: Overview of Our Approach

データセット (1)

- GitHubからPython・Javaのスター数上位10,000プロジェクトを選定
- 以下のようにデータを成形
 - 正例：削除行に“TODO”コメントが含まれている
 - 負例：“TODO”コメントがそのままにされている

Commit Message	issue #3326 - adds log for request failing	Issue #372 - Forward all params from client to GitHub
Diff	<pre>except requests.exceptions.HTTPError as err: - # TODO: log the message - pass + logging.waring('Fetching comments failed')</pre>	<pre># TODO: handle sort and other parapeters - params = {} + params = request.args.copy()</pre>
code_change	<pre>except requests . exception . httperror as err : <nl> - pass <nl> + logging . warning (' fetching comments failed ')</pre>	<pre>- params = { } <nl> + params = request . args . copy ()</pre>
commit_msg	issue <issue_id> — adds log for request failing	issue <issue_id> — forward all params from client to github
todo_comment	todo : log the message	todo : handle sort and order parameters
Label	POSITIVE	NEGATIVE

Figure 4: Data Preparation Examples

データセット (2)

- データの内訳

種別	正例 [ペア]	負例 [ペア]
Python	38,175	35,995
Java	33,797	32,490

- データの信頼性

- ランダムサンプリングした200のデータ (正:100, 負:100) を手動で確認
 - 正例 : 93 / 100 (正答率)
 - 負例 : 100 / 100 (正答率)

評価実験：ベースライン

- TCO : TODO-CodeChange-Overlap
 - “TODO”コメントと変更コードに共通の単語が含まれているか判定
- TMO : TODO-CommitMsg-Overlap
 - “TODO”コメントとコミットメッセージに共通の単語が含まれているか判定
- TCMO
 - TCOとTMOの合わせたやつ
- IRSC
 - メソッドのシグネチャと内容を入力して, “TODO”コメントが陳腐化しているかを判定^[29]

[29] GiriprasadSridhara.2016.Automaticallydetectingtheup-to-datestatusofToDo comments in Java programs. In Proceedings of the 9th India Software Engineering Conference. 16–25.

評価実験：結果

- 提案手法が全ての指標において上回っている

Table 2: Effectiveness Evaluation (Python)

Measure	Accuracy	Precision	Recall	F1
TCO	55.3%	45.3%	58.1%	50.9%
TMO	56.9%	23.1%	75.5%	35.4%
TCMO	58.4%	54.8%	60.2%	57.4%
IRSC	60.4%	61.0%	62.1%	61.6%
TDCLEANER	84.7%	82.6%	86.8%	84.7%

Table 3: Effectiveness Evaluation (Java)

Measure	Accuracy	Precision	Recall	F1
TCO	56.5%	47.5%	58.7%	52.5%
TMO	56.9%	23.4%	73.3%	35.5%
TCMO	59.7%	57.6%	60.8%	59.2%
IRSC	60.1%	59.0%	70.1%	64.0%
TDCLEANER	85.0%	86.2%	84.4%	85.3%

アブレーションスタディ

- コードの情報を取り入れた方が精度がいい

Table 4: Component-Wise Evaluation (Python)

Measure	Accuracy	Precision	Recall	F1
TD_CC_Encoder	78.4%	78.5%	79.0%	78.8%
TD_MSG_Encoder	61.9%	62.0%	63.0%	62.5%
CC_MSG_Encoder	76.1%	67.7%	82.4%	74.3%
TDCLEANER	84.7%	82.6%	86.8%	84.7%

Table 5: Component-Wise Evaluation (Java)

Measure	Accuracy	Precision	Recall	F1
TD_CC_Encoder	78.6%	82.6%	76.9%	79.7%
TD_MSG_Encoder	60.9%	59.6%	61.8%	60.7%
CC_MSG_Encoder	77.0%	81.3%	75.3%	78.2%
TDCLEANER	85.0%	86.2%	84.4%	85.3%

まとめ

- 陳腐化した“TODO”コメントを判定し自動除去するツールの提案
 - F1値は84.7～85.3%
 - 現存のプロジェクトに適用したところ、いくつか正しく判定できているものが確認できた

所感

- “TODO”コメントのみ除去されているデータの扱いはどうしているのか？
- “TODO”コメントがコミットされたと同時に追加されたコードが関係あるのではないか？
 - “TODO”コメントとコードのペアができ、ペアとなっているコード箇所に変更があれば何かしらのアクションが必要
 - 労力がかかる？難しい？