

An Empirical Study on Just-in-time Conformal Defect Prediction

Khulja Shahini

paluno, University of Duisburg-Essen
Essen, Germany

xhulja.shahini@paluno.uni-due.de

Andreas Metzger

paluno, University of Duisburg-Essen
Essen, Germany

andreas.metzger@paluno.uni-due.de

Klaus Pohl

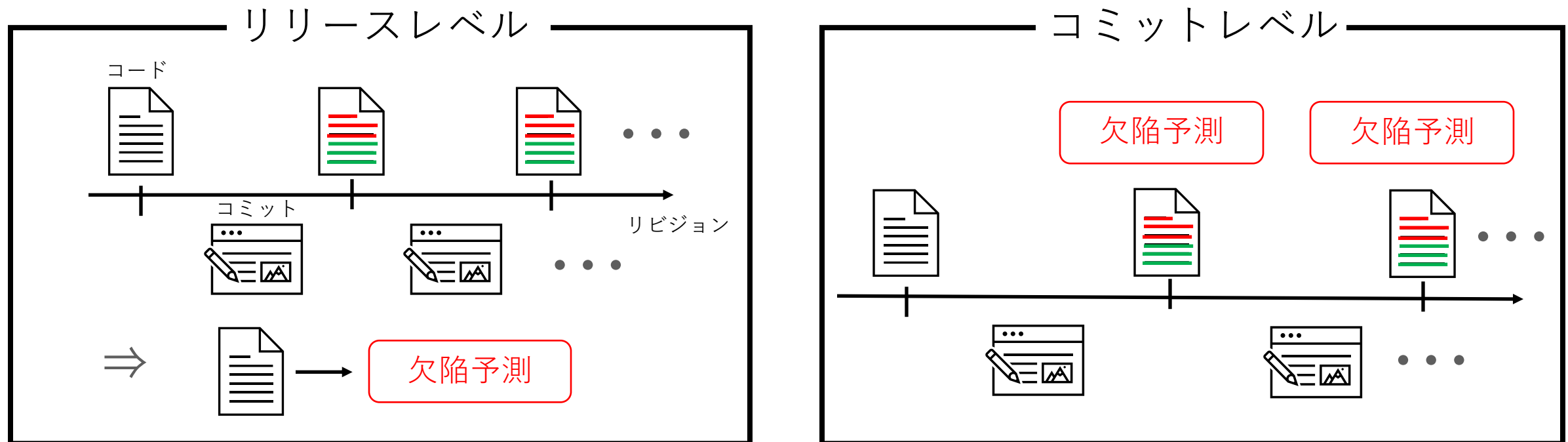
paluno, University of Duisburg-Essen
Essen, Germany

klaus.pohl@paluno.uni-due.de

Proceedings of the 21st International Conference on Mining Software Repositories, Pages 88 - 99

JIT (Just In Time) 欠陥予測

コード変更による欠陥をコミットレベルで予測



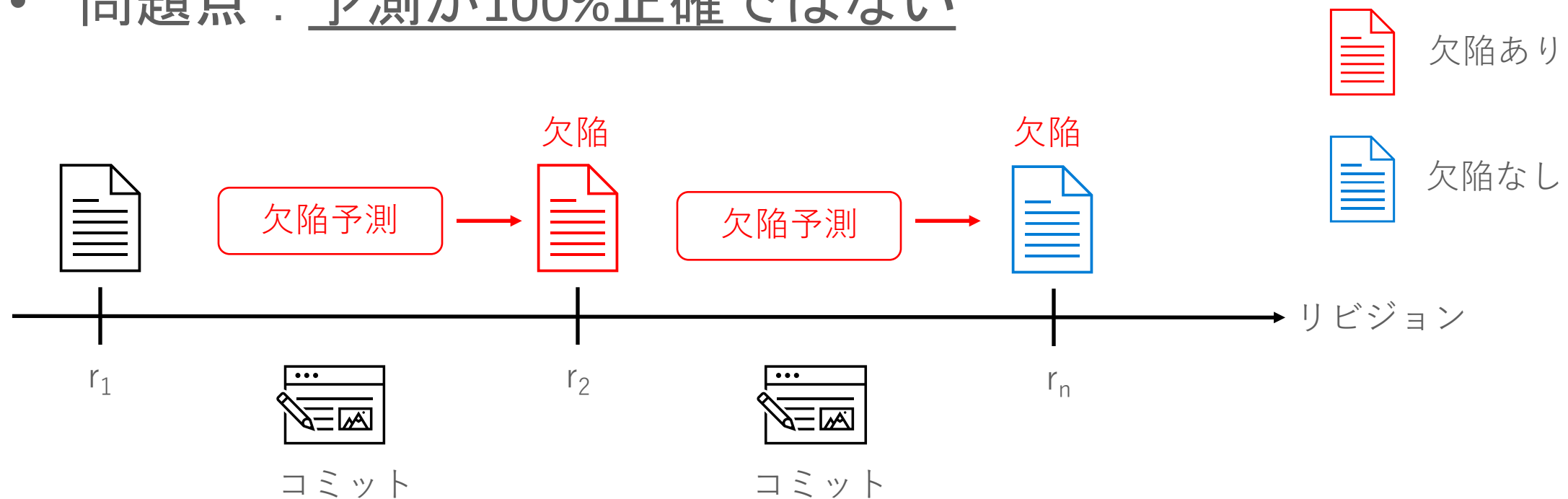
リリースレベルの欠陥予測と比較

1. 欠陥を調査するコード行が少ない
2. リアルタイムでフィードバックを得ることができる

JIT欠陥予測での課題点

不確実性での意思決定

- 最先端のJIT欠陥予測
→アンサンブル学習や深層学習で予測精度を向上
- 問題点：予測が100%正確ではない



- 欠陥を含まないコミットを調査
 - 開発者の無駄な時間と労力を消費してしまう

JIT欠陥予測の不確実性：定量化

ヒューリスティックな不確実性

- 予測の不確実性を定量化

- クラスラベルに事後確率を提供
- 決定木ルールの信頼度スコアを抽出

} 予測の正確性に対する
モデルの信頼度を保証しない

厳密な不確実性

- 不確実性の予測モデルをコンフォーマル予測と組み合わせる

- コンフォーマル予測（CP）

- 経験的な不確実性の概念を厳密なものに変換

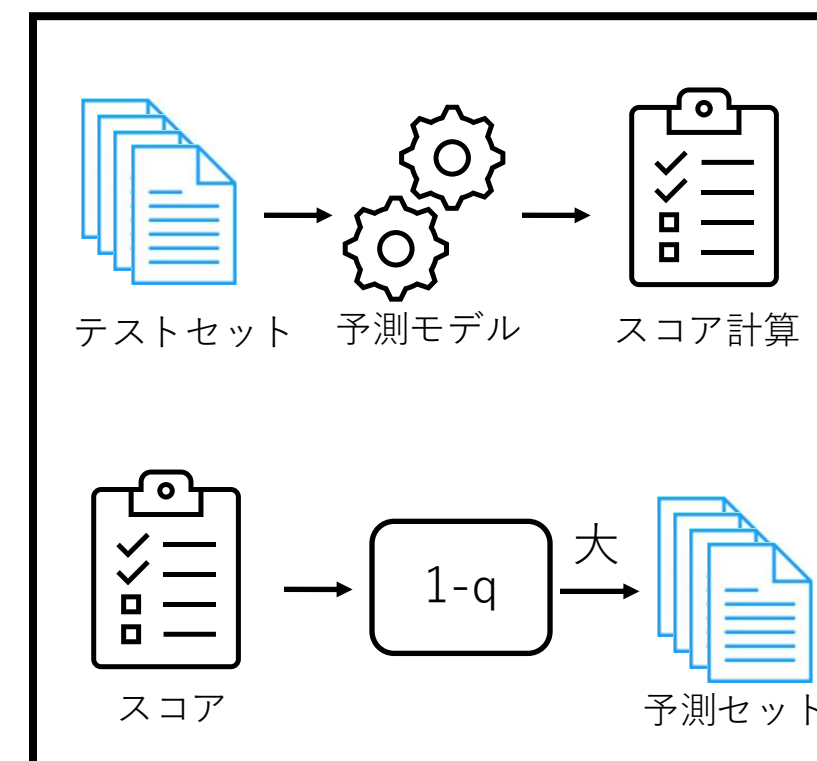
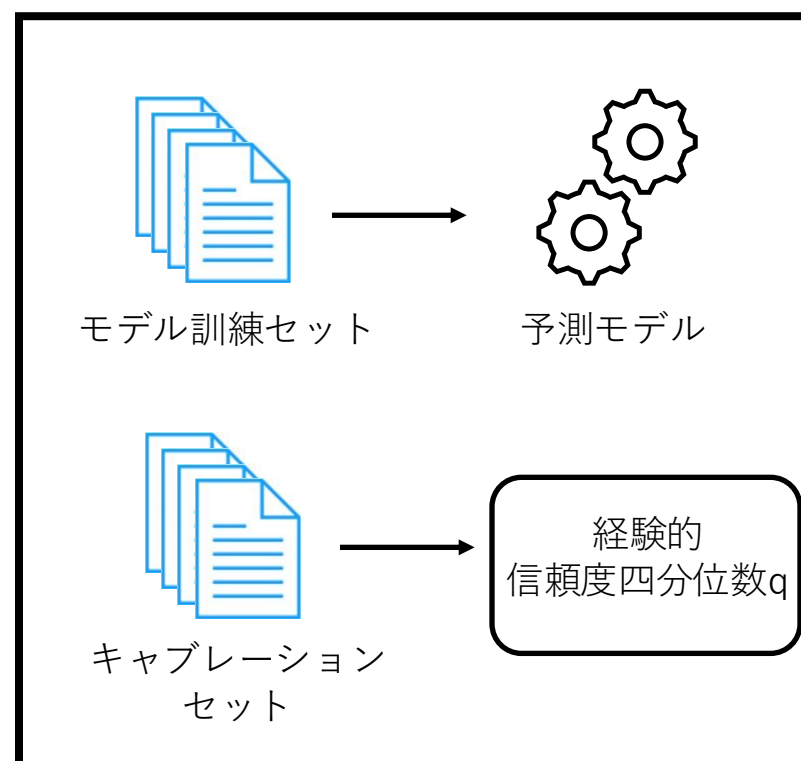
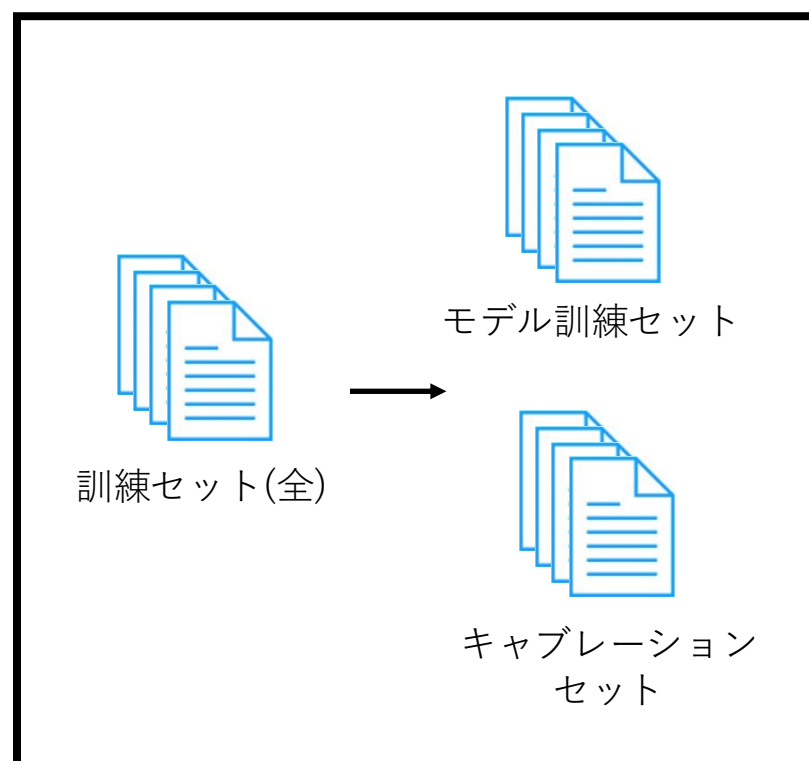
⇒確率 $1-\alpha$ の真のラベルを含む予測セットを生成（ α ：エラー閾値）

最適な場合、一つのラベルで予測セットが構築

コンフォーマル予測 (CP)

予測モデルの予測がどれだけ正しいかを評価

CPアルゴリズム



$$\hat{q} = \frac{\lceil (n+1)(1-\alpha) \rceil}{n}$$

n : インスタンス数

α : エラー閾値

目的とリサーチクエスチョン

目的

- JIT欠陥予測によるCPの適応可能性と有効性を評価

リサーチクエスチョン（RQ）

- RQ1：CPはどの程度の割合で有益な結果を提供するか
- RQ2：CPが有益な結果を提供するクラスラベルの分布は？
- RQ3：CPが誤った予測をどの程度フィルタリングできるか

データセット

JIT欠陥予測技術

- DeepJITとCC2Vecを使用
 - トップ学会で発表され、多くの引用がある

使用するデータ

- QTとOpenStackを使用^[33]
 - コードやプロセス、著者などのメトリクスが含まれている

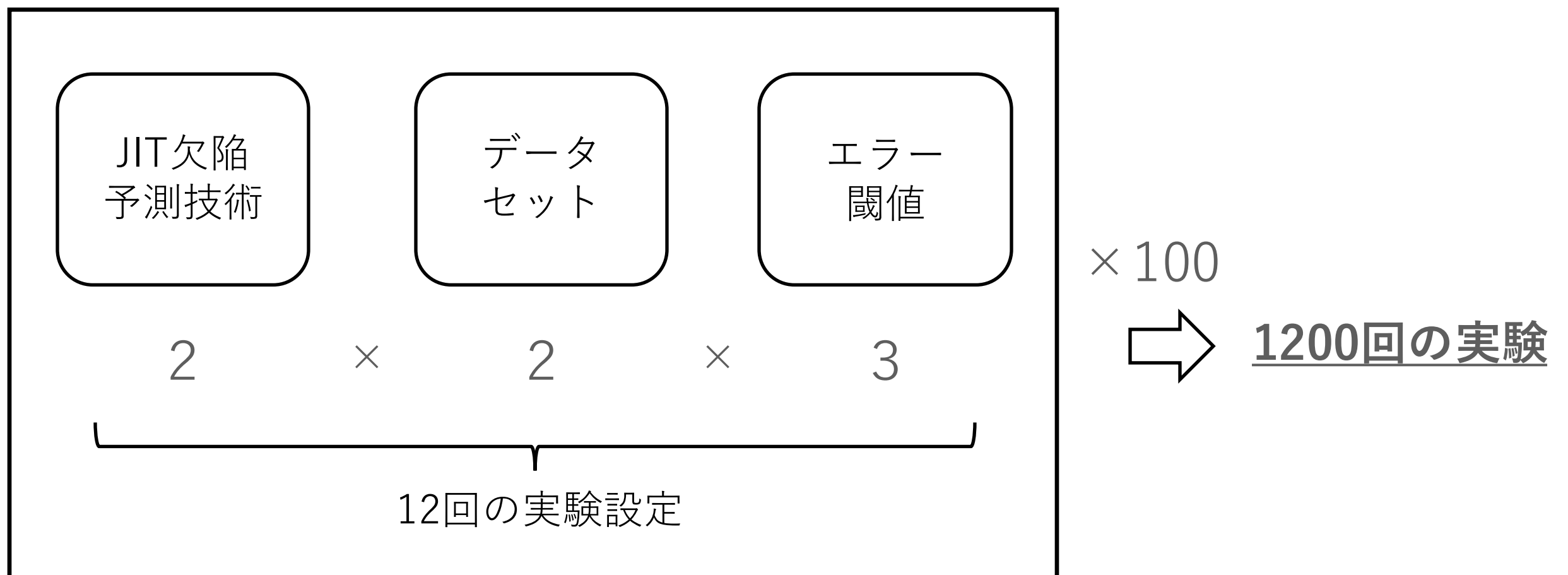
データセット	コミット数	欠陥あり（予測）
QT	25,150	2012(8%)
OpenStack	12,374	1608(13%)

[33] Shane McIntosh and Yasutaka Kamei. 2017. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. IEEE Transactions on Software Engineering PP (04 2017), 1–1. <https://doi.org/10.1109/TSE.2017.2693980>

実験設定

実験

- JIT欠陥予測技術 : DeepJIT, CC2VeC
- データセット : QT, OpenStack
- エラー閾値 : 0.05, 0.10, 0.15



RQ1：どの割合で有益な結果を提供するか

CP予測の割合

$$Inf = \frac{m_{s=1}}{n}$$

$m_{s=1}$ ：サイズ1の予測セットを生成したインスタンス数
 n ：テストセット内のインスタンス数

本研究では $n=1000$ と設定

結果

DeepJIT

OPENSTACK				QT		
α	% _{ss=1}	Min-max	STDev	% _{ss=1}	Min-max	STDev
(0.05)	27%	175-550	87	28%	187-563	83
(0.10)	46%	398-682	41	52%	480-910	92
(0.15)	55%	601-807	29	70%	748-1134	84

⇒AUC：0.771

⇒AUC：0.765

CC2Vec

OPENSTACK				QT		
α	% _{ss=1}	Min-max	STDev	% _{ss=1}	Min-max	STDev
(0.05)	9%	92-170	14	8%	79-160	20
(0.10)	16%	158-299	23	17%	173-283	24
(0.15)	24%	262-433	26	24%	263-456	41

⇒AUC：0.822

⇒AUC：0.809

95%正しいと保証

DeepJIT：27%， CC2Vec：9%の予測

RQ2：有益な結果のクラスラベル分布は？

CP予測の分布

$$\left. Inf_{c=0} = \frac{m_{s=1,c=0}}{m_{s=1}} \right\} \text{clean (クラス=0)} \quad \left. Inf_{c=1} = \frac{m_{s=1,c=1}}{m_{s=1}} \right\} \text{detective (クラス=1)}$$

結果

DeepJIT

α	OPENSTACK			
	#defective	#clean	%defective	%clean
(0.05)	27	260	9	91
(0.10)	51	417	11	89
(0.15)	74	437	14	86

α	QT			
	#defective	#clean	%defective	%clean
(0.05)	27	273	9	91
(0.10)	45	505	9	91
(0.15)	57	657	8	92

CC2Vec

α	OPENSTACK			
	#defective	#clean	%defective	%clean
(0.05)	36	1	97	3
(0.10)	64	2	97	3
(0.15)	82	6	93	7

α	QT			
	#defective	#clean	%defective	%clean
(0.05)	32	1	97	3
(0.10)	56	9	86	14
(0.15)	70	26	73	27

DeepJITではcleanインスタンスが多く占め、
CC2Vecではインスタンスをdetectiveと分類する傾向がある

RQ3：誤った予測のフィルタリング精度は

CPがフィルタリングする偽陽性(FP)と偽陰性(FN)予測の割合を測定

$$filteredFP = \frac{FP_{filtered}}{FP_{tot}}, \quad filteredFN = \frac{FN_{filtered}}{FN_{tot}}$$

結果

	DeepJIT				CC2Vec			
	QT		OPENSTACK		QT		OPENSTACK	
	FP	FN	FP	FN	FP	FN	FP	FN
(Avg)	354	34	529	25	808	3	812	7
(Min)	304	29	437	12	601	0	625	2
(Max)	414	38	658	42	1006	8	981	12
(STDev)	21.5	1.8	46.2	6.3	83.9	1.6	68.9	2.1

FPとFNの予測結果



α	OPENSTACK				QT			
	% FP	STDev	%FN	STDev	%FP	STDev	%FN	STDev
(0.05)	86	0.02	82	0.06	81	0.03	83	0.07
(0.10)	74	0.03	65	0.07	63	0.05	57	0.1
(0.15)	60	0.04	62	0.08	42	0.06	34	0.09

DeepJIT

α	OPENSTACK				QT			
	% FP	STDev	%FN	STDev	%FP	STDev	%FN	STDev
(0.05)	90	0.02	100	0	90	0.02	95	0.2
(0.10)	81	0.03	99	0.02	80	0.03	95	0.2
(0.15)	72	0.03	99	0.02	71	0.04	95	0.2

CC2Vec

CPがJIT欠陥予測技術によって生成されるエラーをフィルタリングする能力があることが明らかになった

まとめ

目的

- JIT欠陥予測によるCPの適応可能性と有効性を評価

結果

- CPはJIT欠陥予測で予測されたもので、95%正しいと言えるものがDeepJITで27%, CC2Vecで9%であった
- CPのフィルタリング能力
 - DeepJITで偽陽性で最大83%, 偽陰性で最大86%をフィルタリング
 - CC2Vecで偽陽性で最大100%, 偽陰性で最大90%をフィルタリング

補足：DeepJIT^[14]

コード，プロセス，開発者のメトリクスを入力データとして，CNNを訓練する手法

メトリクス例

- ソースコードから抽出されたサイズ
- 変更履歴から抽出されたコード行数
- 著者，レビュー担当者の経験メトリクス

モデルの訓練

1. コミットメッセージや変更内容などの自然言語を前処理し訓練
2. コミットに欠陥が含むかどうかを予測

[14] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. 2019. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada, Margaret Anne D. Storey, Bram Adams, and Sonia Haiduc (Eds.). IEEE / ACM, 34–45.

補足：CC2Vec^[15]

ソフトウェアコード変更のための特徴表現技術

- DeepJITと組み合わせることで、予測性能を向上
- 階層的注意ネットワークを活用し、コード変更アーティファクト（ログメッセージ、コード行）の効率的な表現を生成
- 表現の生成
 1. ログメッセージの最初の文からコード変更のセマンティクスを抽出
 2. 追加，削除されたコード行から変更のベクトル表現を生成
 3. アテンションメカニズムを使用し、コード変更の構造的依存関係を捉え、変更のより良い表現を生成

[15] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. 2020. Cc2vec: Distributed representations of code changes. Proceedings - International Conference on Software Engineering (2020), 518 – 529. <https://doi.org/10.1145/3377811.3380361> All Open Access, Green Open Access.