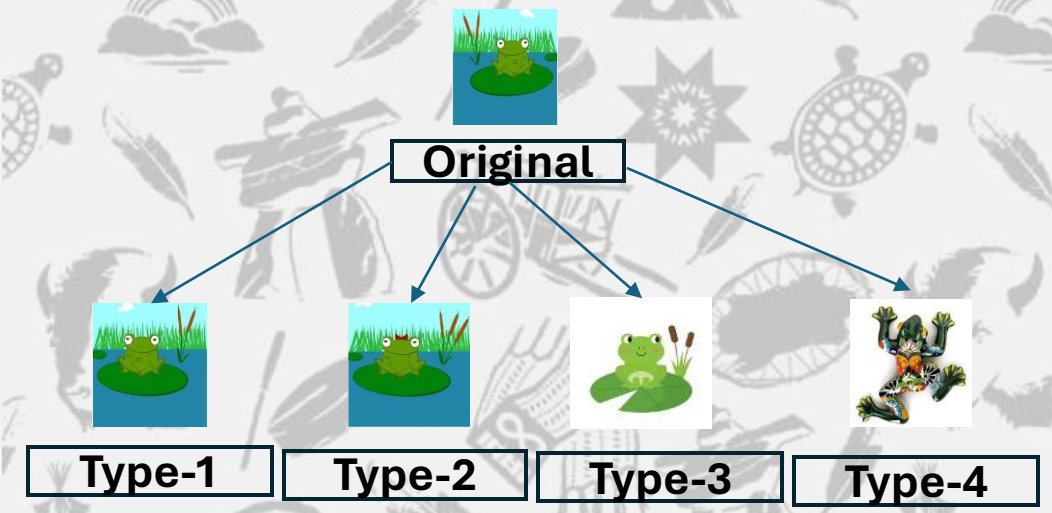
Are Classical Clone Detectors Good Enough For the AI Era?

Ajmain I. Alam, **Palash R. Roy,** Farouq Al-omari, Chanchal k. Roy, Banani Roy, Kevin A. Schneider
Dept. Of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada

Clones



Semantic Clones

```python
def factorial(n):
    x = n
    for i in range(1,n,1):
        x = x*(n-i)
    return x
```

```python
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)
```

Cross-Language Clones

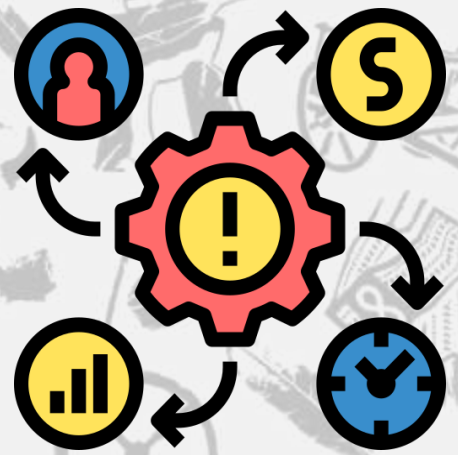
```java
// Java
public static boolean longerTF (boolean [] guess) {
    int variableTrue = 0;
    int variableFalse = 0;
    for (int x = 0;
    x < guess.length; x ++) {
        if (guess [x]) {
            variableTrue ++;
        } else {
            variableFalse ++;
        }
    }
    return variableTrue >= variableFalse;
}
```

```python
// Python
def longerTF(guess):
    variableTrue = 0
    variableFalse = 0
    for x in range(len(guess)):
        if (guess[x]):
            variableTrue += 1
        else:
            variableFalse += 1
    return variableTrue >= variableFalse
```

Impacts of Clone

Positive Impacts *[Krinke WCRE 2007, Krinke SCAM 2008, Krinke, IWSC 2011, Hotta et al. IWPSE 2010, Gode and Harder CSMR 2011, Kapser and Godfrey EMSE 2008, Alam et al. ICSME 2023, Roy et al. IWSC 2023, Mondal et al EMSE 2017]*

- Increased productivity by code reuse.
- Easing Program Comprehension.
- Avoiding risks by reuse of mature and tested code.



Negative Impacts *[Lozano and Wermelinger, ICSM 2008 , Lozano and Wermelinger, IWSC 2010, Juergens et al. ICSE 2009, Barbour et al. ICSM 2011, Mondal et al. ACR 2012, SAC 2012]*

- Hidden bug propagation.
- Inconsistent change.
- Higher change proneness.
- Larger codebase (code bloating).
- Increased maintenance efforts.

Trends of AI Code

Adoption and frequency of AI coding assistant use

Among developers who say they use an AI coding assistant:

49% use them daily — and another 39% use them weekly

58% started using these tools within the last 6 months

71% use these tools more often today than when they first started



Key survey findings:

- **AI is here and it's being used at scale.** 92% of U.S.-based developers are already using AI coding tools both in and outside of work.
- **Waiting on builds and tests is still a problem.** Despite industry-wide investments in DevOps, developers still say the most time-consuming thing they're doing at work besides writing code is waiting on builds and tests.
- **Developers want more collaboration**. Developers in enterprise settings work with an average of 21 other engineers on projects–and want collaboration to be a top metric in performance reviews.
- **And they think AI will help.** More than 4 out of 5 developers expect AI coding tools will make their team more collaborative.
- **Developers also see big benefits to AI.** 70% say AI coding tools will offer them an advantage at work and cite better code quality, completion time, and resolving incidents as some of the top anticipated benefits.

AI Copilot Code Quality: 2025 Look Back at 12 Months of Data

Emerging trends: 4x more code cloning, "copy/paste" exceeds "moved" code for first time in history. Includes 2025 projections.

Stack Overflow's 2024 Developer Survey showed that 63% of professional developers are currently using AI in their development process, with another 14% saying they plan to begin using AI soon.

We examine 5 years worth of data (2020-2024), using the largest dataset that has been brought to bear -- 211 million changed lines from repos owned by Google, Microsoft, Meta, and enterprise C-Corps. The focus was to understand how AI Assistants influence the type of code being written. Are there measurable differences in code quality metrics between **AI-assisted blocks** and **non-AI-assisted** code?

We observe a spike in the prevalence of duplicate code blocks, along with increases in short-term churn code, and the continued decline of moved lines (code reuse).

Human vs AI Code

```python
from collections import deque
def is_palindrome(s):
    d = deque(s)
    while len(d) > 1:
        if d.popleft() != d.pop():
            return False
    return True
```

```python
def ispalindrome(s):
    for c1, c2 in zip(s, reversed(s)):
        if c1 != c2:
            return False
    return True
```

Experts Opinions

> AI-generated code introduces a "detection blind spot" for traditional CCD tools

> we're seeing false negatives in our clone detection pipeline that weren't there before our team started using AI assistants

Tools Tested

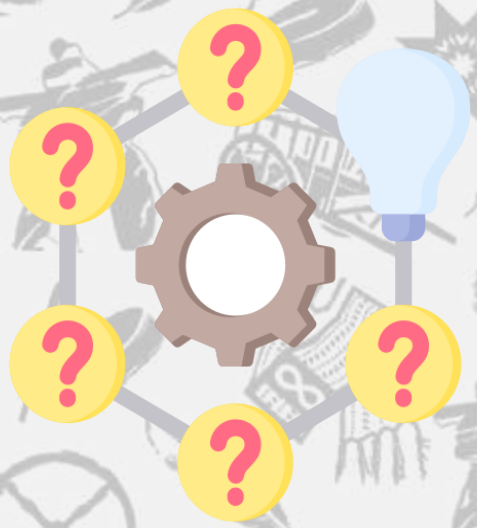
Tools	Type	Configuration
PMD/CPD		Min token= 50
Simian		Threshold=6
Deckard		sim=0.7, stride=2
Clone Digger		default
CCCD		default
CloneWorks		T3 conservative
Stone Detector		default
ASTNN		finetuned
CodeBERT		pretrained

Research Questions

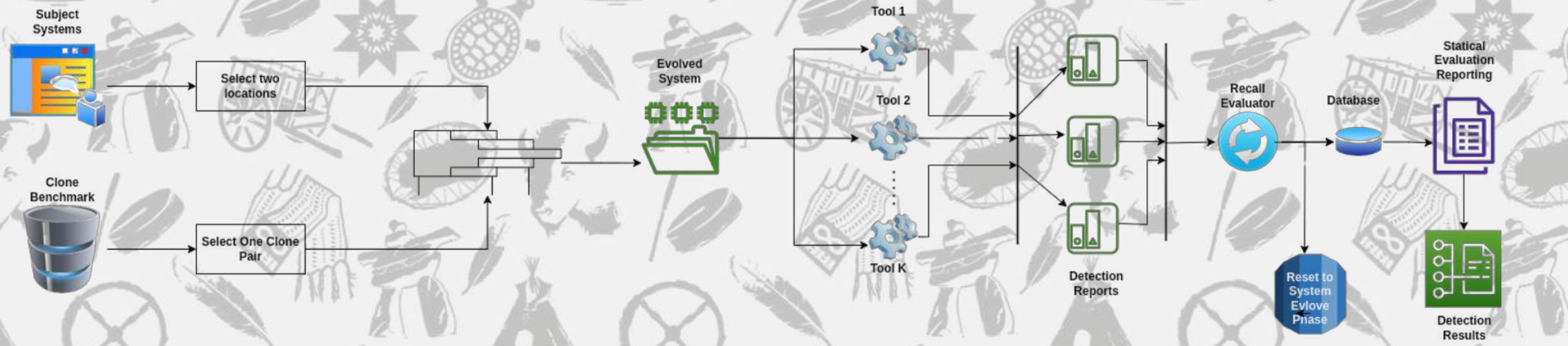
RQ1: How do classical clone detection tools perform on AI-generated clones?

RQ2: Do tools perform differently in AI versus non-AI clone benchmarks?

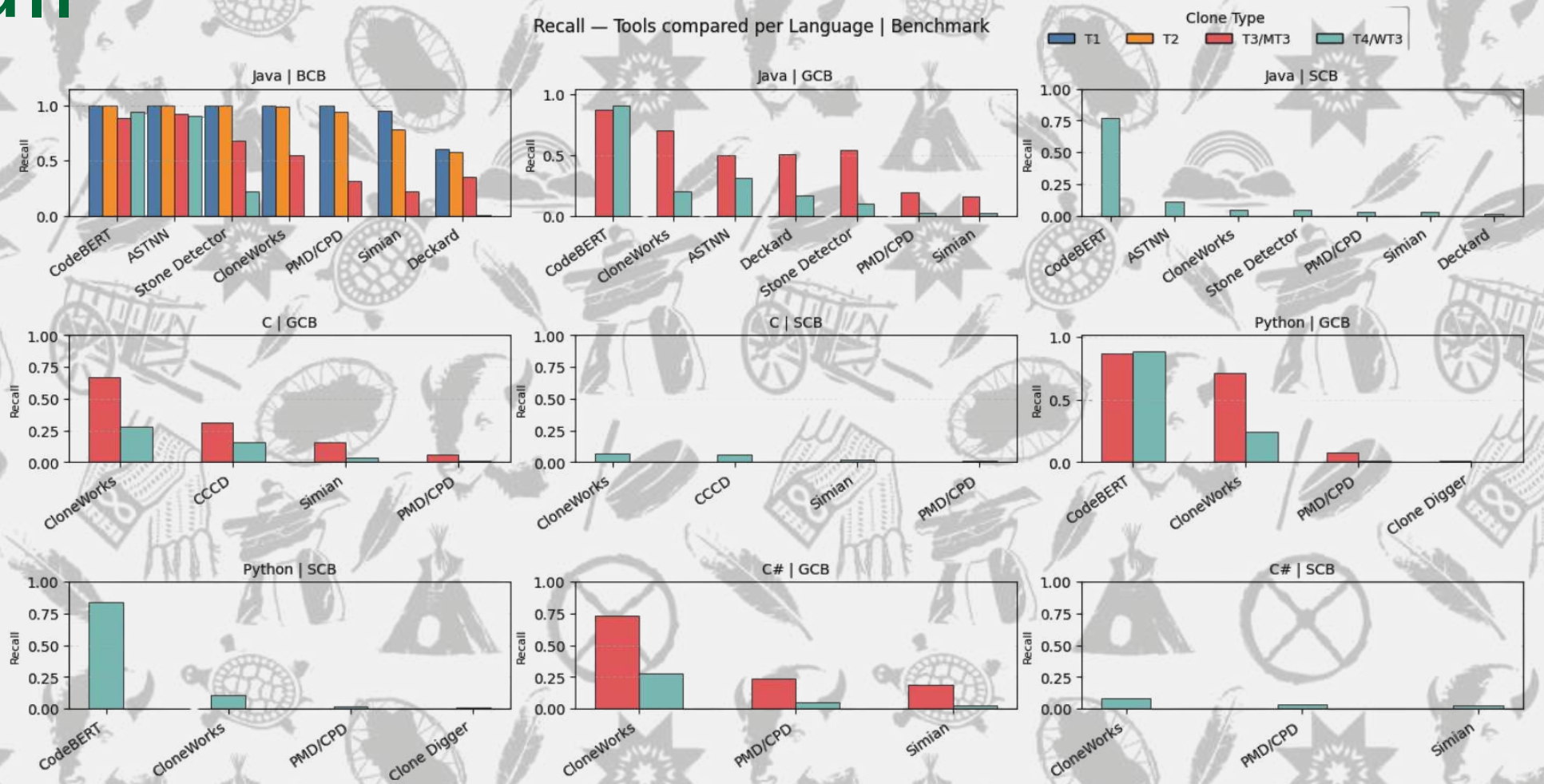
RQ3: What is the execution time and scalability for each tool?



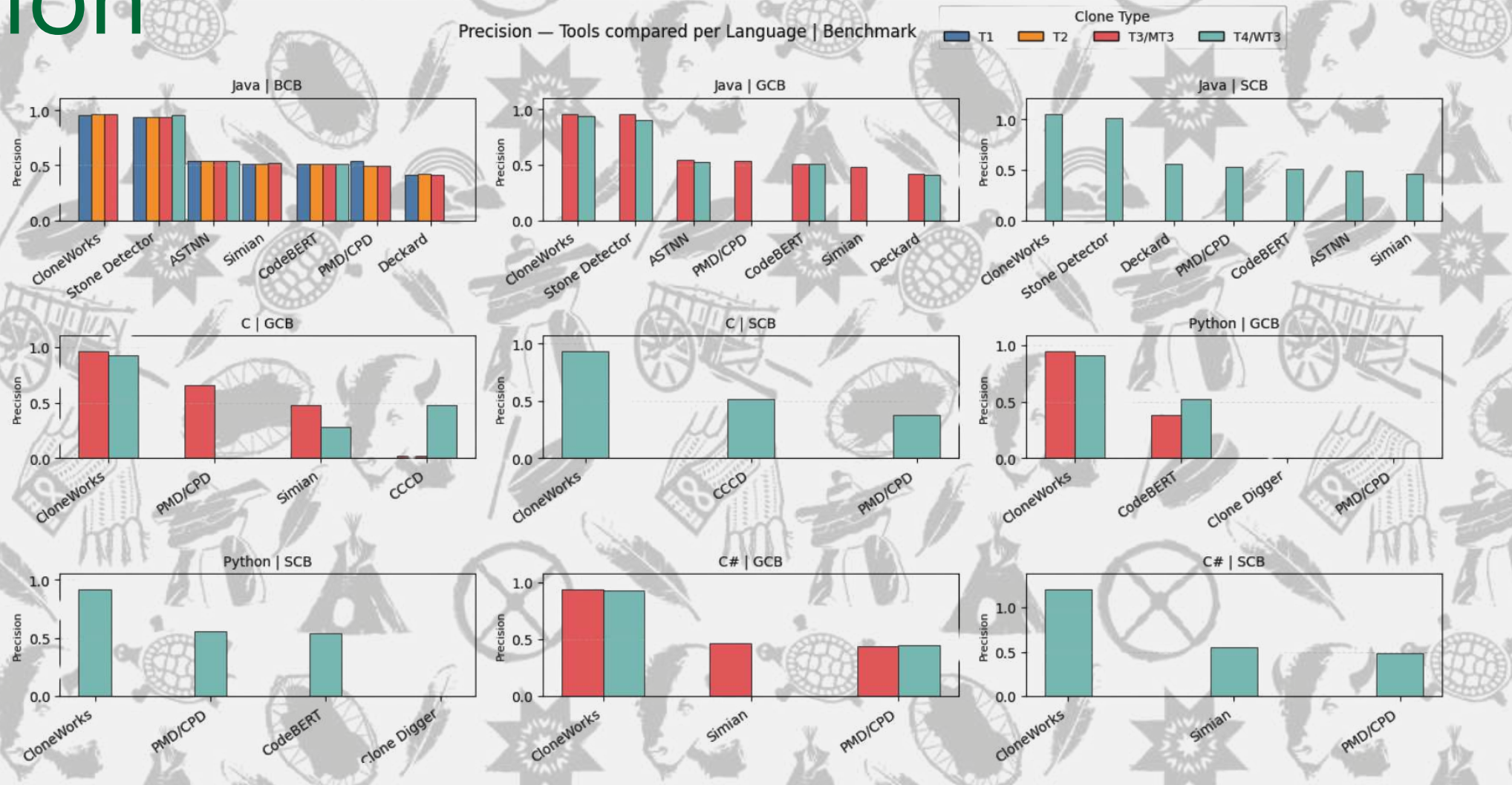
Methodology



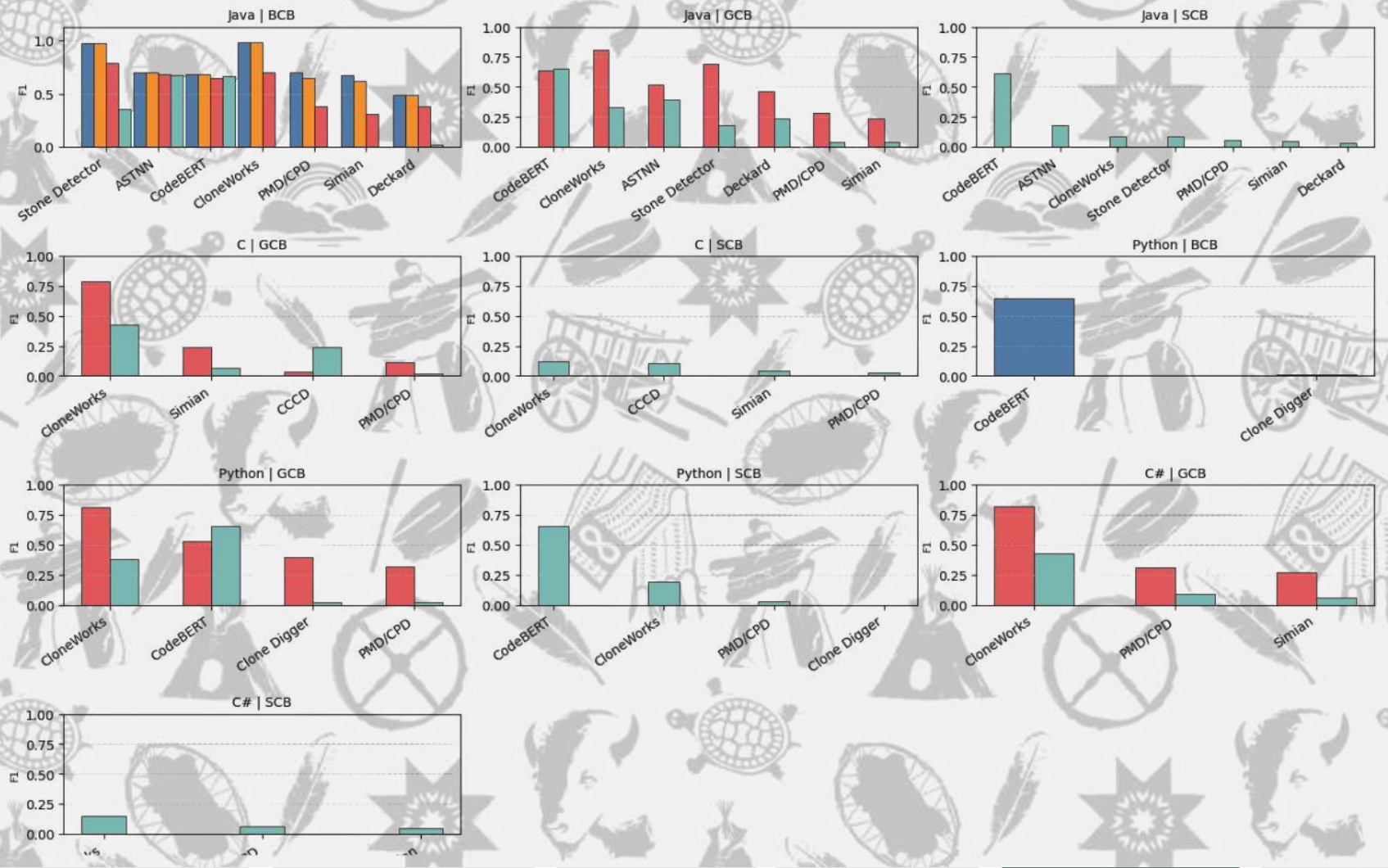
Recall


Recall — Tools compared per Language | Benchmark
Clone Type
T1
T2
T3/MT3
T4/WT3
Java | BCB
Java | GCB
Java | SCB
C | GCB
C | SCB
Python | GCB
Python | SCB
C# | GCB
C# | SCB
Recall
CodeBERT
ASTNN
Stone Detector
CloneWorks
PMD/CPD
Simian
Deckard
CCCD
Clone Digger

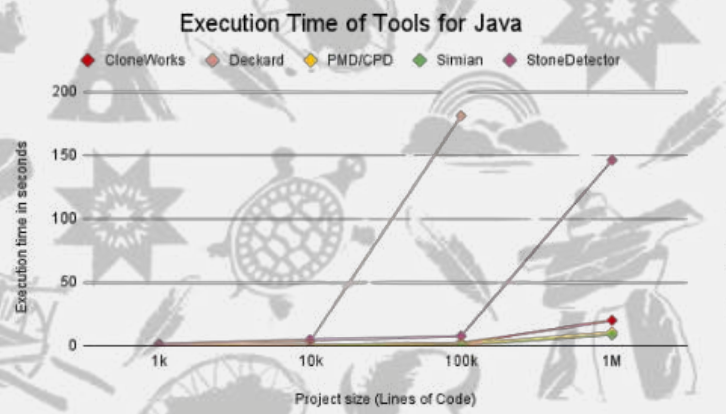
Precision

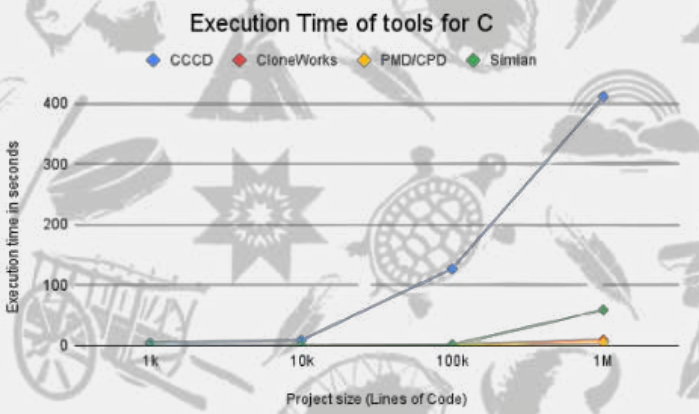

Precision — Tools compared per Language | Benchmark
Clone Type
T1
T2
T3/MT3
T4/WT3
Java | BCB
Java | GCB
Java | SCB
C | GCB
C | SCB
Python | GCB
Python | SCB
C# | GCB
C# | SCB
Precision
CloneWorks
Stone Detector
ASTNN
Simian
CodeBERT
PMD/CPD
Deckard
CCCD
Clone Digger

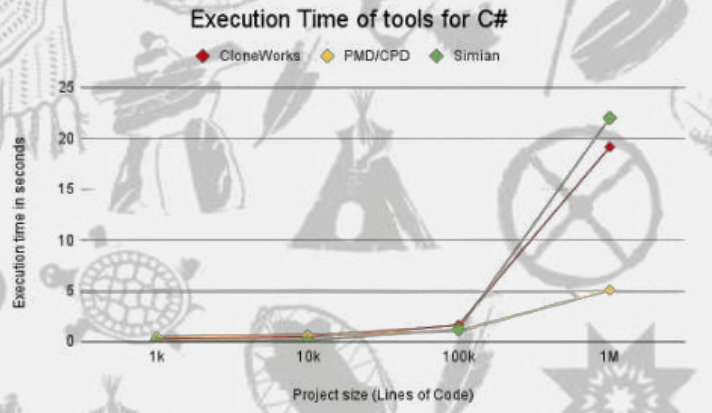
F1 Score

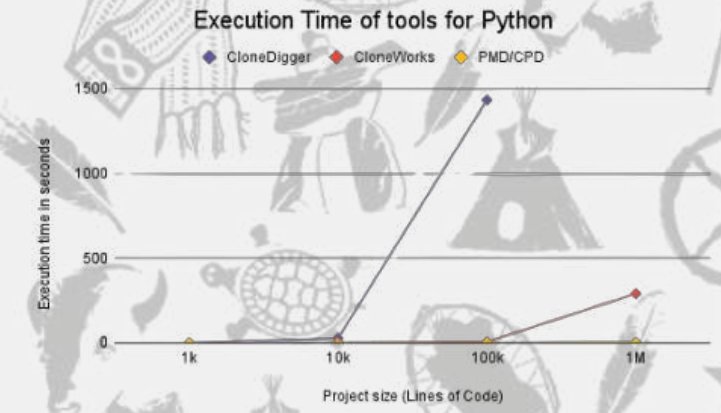

Java | BCB
Stone Detector
ASTNN
CodeBERT
CloneWorks
PMD/CPD
Simian
Deckard
Java | GCB
CodeBERT
CloneWorks
ASTNN
Stone Detector
Deckard
PMD/CPD
Simian
Java | SCB
CodeBERT
ASTNN
CloneWorks
Stone Detector
PMD/CPD
Simian
Deckard
C | GCB
CloneWorks
Simian
CCCD
PMD/CPD
C | SCB
CloneWorks
CCCD
Simian
PMD/CPD
Python | BCB
CodeBERT
Clone Digger
Python | GCB
CloneWorks
CodeBERT
Clone Digger
PMD/CPD
Python | SCB
CodeBERT
CloneWorks
PMD/CPD
Clone Digger
C# | GCB
CloneWorks
PMD/CPD
Simian
C# | SCB
F1

Execution Time









But!

2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)

Why Don't XAI Techniques Agree? Characterizing the Disagreements Between Post-hoc Explanations of Defect Predictions

Saumendu Roy, *University of Saskatchewan, Canada*, plz937@usask.ca
Gabriel Laberge, *Polytechnique Montréal, Canada*, gabriel.laberge@polymtl.ca
Banani Roy, *University of Saskatchewan, Canada*, banani.roy@usask.ca
Foutse Khomh, *Polytechnique Montréal, Canada*, foutse.khomh@polymtl.ca
Amin Nikanjam, *Polytechnique Montréal, Canada*, amin.nikanjam@polymtl.ca
Saikat Mondal, *University of Saskatchewan, Canada*, saikat.mondal@usask.ca

***Abstract*—Machine Learning (ML) based defect prediction models can be used to improve the reliability and overall quality of software systems. However, such defect predictors might not be deployed in real applications due to the lack of transparency. Thus, recently, application of several post-hoc explanation methods (e.g., LIME and SHAP) have gained popularity. These explanation methods can offer insight by ranking features based on their importance in black box decisions. The explainability of ML techniques is reasonably novel in the Software Engineering community. However, it is still unclear whether such explainability methods genuinely help practitioners make better decisions regarding software maintenance. Recent user studies show that data scientists usually utilize multiple post-hoc explainers to understand a single model decision because of the lack of ground truth. Such a scenario causes disagreement between explainability methods and impedes drawing a conclusion. Therefore, our study first investigates three disagreement metrics between LIME and SHAP explanations of 10 defect-predictors, and exposes that disagreements regarding the rankings of feature importance are most frequent. Our findings lead us to propose a method of aggregating LIME and SHAP explanations that puts less emphasis on those disagreements while highlighting the aspect on which explanations agree.**

***Index Terms*—Empirical, Defect Prediction, eXplainable AI, LIME, SHAP, Software Maintenance**

I. Introduction

With the growing complexity of modern software [1], data-intensive application [2], there has come an increased difficulty in assuring high quality, and maintainability [1]. Thus, detecting defect-prone modules early in the development cycle is becoming primordial to better allocate limited resources for testing and maintenance. Different techniques have been proposed to manage the task of defect prediction. The most commonly known are Machine Learning (ML) based techniques [3]–[5]. However, ML models remain black boxes despite their apparent success, meaning it is hard for humans to understand the reasoning behind their decisions. This is a significant reason why eXplainable AI (XAI) methods such as LIME [6] and SHAP [7] have gained in popularity recently. These techniques provide explanations for any decisions made by a model (model-agnostic). Hence, they are a versatile tool to add to the traditional ML-based methodology for defect predictions. The usage of these te
in the Software Engineering con
crucial questions are still unansv
whether or not post-hoc explan
making regarding software main
noted, different post-hoc explain
even contradicting) explanations
[10]. Practitioners being shown
why a given module is predicted
to disregard the explainability te
is critical to study the disagree
explanations in a Software Engin

In this study, we investigate *th*
tween LIME and SHAP (*Feature*
and *Sign Agreement*) in the cont
show that disagreements regardi
features importance occur more
the sign (positive/negative) of the
insight, we propose a novel met
SHAP explanations by only pres
most important features accordin
and whose importance shares the
across both explainers. This agg
ranking among top-k features as
be inconsistent between explaine

II. Backgr

When a user is employing a
burden of proof is always on th
soning behind its decision [11].
completely disregard predictions
using the detection tools. To reso
a growing interest in applying tec
of defect prediction [8], [12]. In
among developers and managers

2576-3148/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSME55016.2022.00056

Authorized licensed use limited to: University of Saskatchewan. Downloaded on September 11,2025 at 19:07:04 UTC from

Tackling the XAI Disagreement Problem with Regional Explanations

Gabriel Laberge[1] Yann Pequignot[2] Mario Marchand[2] Foutse Khomh[1]
[1]Polytechnique Montréal [2]Université Laval à Québec

Abstract

The XAI Disagreement Problem concerns the fact that various explainability methods yield different local/global insights on model behavior. Thus, given the lack of ground truth in explainability, practitioners are left wondering "Which explanation should I believe?". In this work, we approach the Disagreement Problem from the point of view of Functional Decomposition (FD). First, we demonstrate that many XAI techniques disagree because they handle feature interactions differently. Secondly, we reduce interactions locally by fitting a so-called FD-Tree, which partitions the input space into regions where the model is approximately additive. Thus instead of providing global explanations aggregated over the whole dataset, we advocate reporting the FD-Tree structure as well as the regional explanations extracted from its leaves. The beneficial effects of FD-Trees on the Disagreement Problem are demonstrated on toy and real datasets.

1 Introduction

The Machine Learning paradigm is growing in popularity in data-centric domains. However, there are rising concerns regarding the black-box nature of the performant models *e.g.* Random Forests (Breiman, 2001) and Gradient Boosted Trees (Friedman, 2001). To address these concerns, the field of eXplainable AI (XAI) was introduced, and various post-hoc explanation methods were proposed to provide insight into complex model behaviors (Arrieta et al., 2020). Notable examples of post-hoc explanations include Partial Dependence Plots (PDP) (Friedman, 200
ley Additive exPlanations (SHAP) (Lundberg
2017), and Permutation Feature Importan
(Breiman, 2001).

Despite the explosion in XAI research, there
fundamental challenges to address. For inst
so-called *Disagreement Problem* (DP) (Krish
2022) refers to the observation that different
explanation methods disagree on the local/g
havior of models. This is not necessarily
nad it is actually expected since different
niques characterize models differently. Yet,
tioners are expected to make decisions based
explanations or said explanations are used
decisions impacting human beings, then th
comes critical. Which explanations should
tioner consider? Which explanation should
to a client being impacted by the model? (
lack of ground truth in XAI, these questions
rently left unanswered.

Our partial mitigation of the disagreement p
inspired by Hybrid Interpretable Models (Wa
Pan et al., 2020; Ferry et al., 2023). In the
it is shown that regions exist where complex
can be replaced by simple rules without de
in overall performance. These analyses sug
models may be globally complex yet simple
regions. Note that it may be possible to ex
model on those regions without any disagreem
post-hoc explainers. As a toy example, con
features $x_i \sim U(-1, 1)$ for $i = 0, 1, \ldots 4$ and t

$$h(\boldsymbol{x}) = \begin{cases} x_0 & \text{if } x_1 \geq 0 \\ x_2 & \text{otherwise,} \end{cases}$$

which is globally complex but locally sin
ear). To compute global feature importa
PDP/SHAP/PFI, one must provide refere
samples. The typical approach is to use t
dataset which leads to Figure 1 (a). We obser
disagreements regarding the importance of
can we do to reduce this disagreement? We p
learn two disjoint regions $\Omega_- = \{\boldsymbol{x} \in \mathbb{R}^5 : x_1$

Proceedings of the 27th International Conferences on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Copyright 2024 by the author(s).

Compressing Pre-trained Models of Code into 3 MB

Jieke Shi, Zhou Yang, Bowen Xu*, Hong Jin Kang and David Lo
School of Computing and Information Systems
Singapore Management University
{jiekeshi,zyang,bowenxu.2017,hjkang.2018,davidlo}@smu.edu.sg

ABSTRACT

Although large pre-trained models of code have delivered significant advancements in various code processing tasks, there is an impediment to the wide and fluent adoption of these powerful models in software developers' daily workflow: these large models consume hundreds of megabytes of memory and run slowly on personal devices, which causes problems in model deployment and greatly degrades the user experience.

It motivates us to propose *Compressor*, a novel approach that can compress the pre-trained models of code into extremely small models with negligible performance sacrifice. Our proposed method formulates the design of tiny models as simplifying the pre-trained model architecture: searching for a significantly smaller model that follows an architectural design similar to the original pre-trained model. *Compressor* proposes a genetic algorithm (GA)-based strategy to guide the simplification process. Prior studies found that a model with higher computational cost tends to be more powerful. Inspired by this insight, the GA algorithm is designed to maximize a model's Giga floating-point operations (GFLOPs), an indicator of the model computational cost, to satisfy the constraint of the target model size. Then, we use the knowledge distillation technique to train the small model: unlabelled data is fed into the large model and the outputs are used as labels to train the small model. We evaluate *Compressor* with two state-of-the-art pre-trained models, i.e., CodeBERT and GraphCodeBERT, on two important tasks, i.e., vulnerability prediction and clone detection. We use our method to compress pre-trained models to a size (3 MB), which is 160× smaller than the original size. The results show that compressed CodeBERT and GraphCodeBERT are 4.31× and 4.15× faster than the original model at inference, respectively. More importantly, they maintain 96.15% and 97.74% of the original performance on the vulnerability prediction task. They even maintain higher ratios (99.20% and 97.52%) of the original performance on the clone detection task.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; **Designing software**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Model Compression, Genetic Algorithm, Pre-Trained Models

ACM Reference Format:
Jieke Shi, Zhou Yang, Bowen Xu*, Hong Jin Kang and David Lo. 2022. *Compressing Pre-trained Models of Code into 3 MB.* In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3551349.3556964

1 Introduction

Pre-trained models of code have achieved great success in various program understanding and generation tasks [1, 10, 15, 28, 43]. However, these models are large-sized with billions of parameters and have low inference speed; thus they do not suit consumer-grade devices or applications that require strictly low latency. To put things in perspective, CodeBERT [10], a state-of-the-art pre-trained model for processing multiple programming languages, has a total of 125 million parameters, which results in a large model size of 476 MB. When such a large model performs inference on a consumer-grade laptop, the response latency is as high as 3.5 seconds.[1] According to the experience of the Visual Studio teams who have deployed integrated development environments (IDEs) to numerous users, Svyatkovskiy et al. [40] suggest that a reasonable upper bound size for an IDE component or an editor plug-in is 50 MB, while a 3 MB model is preferred as it can be deployed even in severely restricted environments (e.g., low-end hardware for teaching students to code). Meanwhile, Ayre et al. [2] argue that the inference latency should be reduced to 0.1 seconds as much as possible. Recently, researchers [18, 48] also try to minimize the response time of models for code. The pre-trained models of code are designed to serve coding tasks, promising to be a component in modern IDEs to assist developers in software development and maintenance. However, the huge storage and runtime memory consumption, coupled with high inference latency, make these large models prohibitive for integration in modern IDEs. Therefore, it is imperative to reduce the size of these pre-trained models before deploying them.

To date, some approaches have been proposed to compress pre-trained models in natural language processing or other tasks [20, 37, 44, 52]. These existing studies fall under three types: model pruning, model quantization, and knowledge distillation. Model pruning works by replacing partial model parameters with zero [13] or removing architectural components (e.g., network layers or attention heads) [29]. Model quantization techniques convert the model's parameters from 32-bit floating-point numbers into low-bit fixed-point numbers [36]. Unfortunately, model pruning cannot compress CodeBERT and GraphCodeBERT to be less than 50 MB –

[1]The results are elicited by running a handful of examples with a video/night of 460 on the vulnerability prediction task. The number of used CPU cores is limited to 8.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASE '22, October 10–14, 2022, Rochester, MI, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3556964

2018 ACM/IEEE 40th International Conference on Software Engineering: New Ideas and Emerging Results

Explainable Software Analytics

Hoa Khanh Dam, University of Wollongong, Australia, hoa@uow.edu.au
Truyen Tran, Deakin University, Australia, truyen.tran@deakin.edu.au
Aditya Ghose, University of Wollongong, Australia, aditya@uow.edu.au

ABSTRACT

Software analytics has been the subject of considerable recent attention but is yet to receive significant industry traction. One of the key reasons is that software practitioners are reluctant to trust predictions produced by the analytics machinery without understanding the rationale for those predictions. While complex models such as deep learning and ensemble methods improve predictive performance, they have limited explainability. In this paper, we argue that making software analytics models explainable to software practitioners is as *important* as achieving accurate predictions. Explainability should therefore be a key measure for evaluating software analytics models. We envision that explainability will be a key driver for developing software analytics models that are useful in practice. We outline a research roadmap for this space, building on social science, explainable artificial intelligence and software engineering.

KEYWORDS

Software engineering, software analytics, Mining software repositories

ACM Reference Format:
Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2018. Explainable Software Analytics. In *ICSE-NIER'18: 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183399.3183424

1 INTRODUCTION

Software analytics has in recent times become the focus of considerable research attention. The vast majority of software analytics methods leverage machine learning algorithms to build prediction models for various software engineering tasks such as defect prediction, effort estimation, API recommendation, and risk prediction. Prediction accuracy (assessed using measures such as precision, recall, F-measure, Mean Absolute Error and similar) is currently the dominant criterion for evaluating predictive models in the software analytics literature. The pressure to maximize predictive accuracy has led to the use of powerful and complex models such as SVMs, ensemble methods and deep neural networks. Those models are however considered as "black box" models, thus it is difficult for software practitioners to understand them and interpret their predictions.

The lack of explainability results in the lack of *trust*, l
to industry reluctance to adopt or deploy software analy
software practitioners do not understand a model's predi
they would not blindly trust those predictions, nor commit
resources or time to act on those predictions. High predicti
formance during testing may establish some degree of trust
model. However, those test results may not hold when the
is deployed and run "in the wild". After all, the test data is a
known to the model designer, while the "future" data is unl
and can be completely different from the test data, potentiall
ing the predictive machinery far less effective. Trust is esp
important when the model produces predictions that are di
from the software practitioner's expectations (e.g., generating
for parts of the code-base that developers expect to be "clean
plainability is not only a pre-requisite for practitioner trust, b
potentially a trigger for new insights in the minds of practit
For example, a developer would want to understand why a
prediction model suggests that a particular source file is de
so that they can fix the defect. In other words, merely flaggin
as defective is often not good enough. Developers need a tra
a justification of such predictions in order to generate appro
fixes.

We believe that making predictive models explainable t
ware practitioners is as *important* as achieving accurate pred
Explainability should be an important measure for evaluatin
ware analytics models. We envision that explainability will b
driver for developing software analytics models that actua
liver value in practice. In this paper, we raise a number of re
questions that will be critical to the success of software an
in practice:

(1) What forms a (good) explanation in software engin
tasks?
(2) How do we build explainable software analytics mod
how do we generate explanations from them?
(3) How might we evaluate explainability of software an
models?

Decades of research in social sciences have established a understanding of how humans explain decisions and behav each other [20]. This body of knowledge will be useful for d ing software analytics models that are truly capable of pro explanations to human engineers. In addition, *explainable ar* intelligence (XAI) has recently become an important field AI researchers have started looking into developing new m learning systems which will be capable of explaining their de and describing their strengths and weaknesses in a way that users can understand (which in turn engenders *trust* that systems will generate reliable results in future usage scenar

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE-NIER'18, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183424

From Questions to Insights: Exploring XAI Challenges Reported on Stack Overflow Questions

Saumendu Roy Saikat Mondal Banani Roy Chanchal Roy
University of Saskatchewan, Canada
{saumendu.roy,saikat.mondal,banani.roy,chanchal.roy}@usask.ca

ABSTRACT

The lack of interpretability is a major barrier that limits the practical usage of AI models. Several eXplainable AI (XAI) techniques (e.g., SHAP, LIME) have been employed to interpret these models' performance. However, users often face challenges when leveraging these techniques in real-world scenarios and thus submit questions in technical Q&A forums like Stack Overflow (SO) to resolve these challenges. We conducted an exploratory study to expose these challenges, their severity, and features that can make XAI techniques more accessible and easier to use. Our contributions to this study are fourfold. First, we manually analyzed 663 SO questions that discussed challenges related to XAI techniques. Our careful investigation produced a catalog of seven challenges (e.g., disagreement issues). We then analyzed their prevalence and found that model integration and disagreement issues emerged
llenges. Second, we attempt to estimate
hallenge by determining the correlation
nd answer metadata (e.g., the presence
analysis suggests that model integration
hallenge. Third, we attempt to perceive
ges based on practitioners' ability to use
in their work. Practitioners' responses
t issues most severely affect the use of
e seek agreement from practitioners on
that could make XAI techniques more
ly. The majority of them suggest consis-
implified integration. Our study findings
he accessibility and usability of XAI and
mark that can inspire future research.

ACM Reference Format:
Saumendu Roy Saikat Mondal Banani Roy Chanchal Roy. 2025. From Questions to Insights: Exploring XAI Challenges Reported on Stack Overflow Questions. In *The 29th International Conference on Evaluation and Assessment in Software Engineering (EASE 2025), 17–20 June, 2025, Istanbul, Türkiye.* ACM, New York, NY, USA, 11 pages. https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

1 INTRODUCTION

The growing importance of XAI lies in its ability to make AI (e.g., Machine Learning) models more transparent and understandable. This transparency builds trust and allows users to make informed decisions based on the insights provided by the model. Popular XAI techniques such as LIME (Local Interpretable Model-agnostic Explanations) [30] and SHAP (SHapley Additive Explanations) [16] have been widely adopted to explain model decisions. However, users often face challenges when applying these methods in real-world scenarios, and thus seek solutions by submitting questions to Technical Q&A forums like SO. Consider the example in Figure 1, where a user encounters difficulty applying SHAP's *TreeExplainer* to a scikit-learn *StackingClassifier*. In particular, the user receives an error that states the model type is not supported, which reflects a *compatibility issue* between XAI techniques (e.g., SHAP) and specific model types. However, this is a representative example of the many challenges users encounter when leveraging XAI techniques. Thus, it is crucial to identify all such challenges and address them to (a) improve the functionality of XAI tools, (b) enhance user accessibility, and (c) make AI systems more interpretable, trustworthy, and actionable.

Several studies have investigated the inconsistencies in post-hoc explanations [11, 28, 43]. For example, Roy et al. [35] and Parashar et al. [25] studied disagreements among post-hoc explanations in
r studies suggested that different
daining a model, which challenged
and Longo et al.[15] discussed the
erformance and interpretability in
ocused on how users understand
, aiming to make them practical for
ongo et al. explored design strate-
racy and ease of understanding in
es primarily focused on addressing
ements among explainers), overar-
usability in explainability methods
ploring theoretical trade-offs. How-
er all the major practical challenges
g XAI techniques in their tasks (e.g.,
litionally, to the best of our knowl-
on user-driven recommendations
techniques.

This research project was supported by the National Research Foundation, under its Investigatorship Grant (NRF-NRFI08-2022-0002). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.
Authors' Contact Information: Jieke Shi, School of Computing and Information Systems, Singapore Management University, Singapore, Singapore; e-mail: jiekeshi@smu.edu.sg; Zhou Yang (corresponding author), School of Computing and Information Systems, Singapore Management University, Singapore, Singapore; e-mail: zyang@smu.edu.sg; David Lo, School of Computing and Information Systems, Singapore Management University, Singapore, Singapore; e-mail: davidlo@smu.edu.sg.
This work is licensed under Creative Commons Attribution-NonCommercial-ShareAlike International 4.0.
© 2025 Copyright held by the owner/author(s).
ACM 1557-7392/2025/5-ART137
https://doi.org/10.1145/3709025

ACM Transactions on Software Engineering and Methodology, Vol. 34, No. 5, Article 137. Publication date: May 2025.

Definitions → **AI Era** → **Developers** → **Used Tools** → **RQs** → **Method** → **Findings** → **Concluding..**

UNIVERSITY OF SASKATCHEWAN

BE WHAT THE WORLD NEEDS

ICSME 2025 Auckland, New Zealand

Are classical Clone Detectors Good Enough for the AI Era?

Threats to Validity

- Tool Configuration.
- Precision measurement.
- Clone Injection to Software system.



Conclusion

- Classical tools are not that irrelevant yet.
- With some more attention they might be able to do better.
- Performance wise and resource wise they are faster and quicker.
- Most importantly Explainable!!!!



Thank You



Questions!

Key Take Away

Let's not disregard classical clone detectors. They are reliable, explainable, and, with the right support, can perform even better at detecting clones in the AI era

palashroy.me



References

- Krinke, J. (2007, October). A study of consistent and inconsistent changes to code clones. In *14th working conference on reverse engineering (WCRE 2007)* (pp. 170-178). IEEE.
- Krinke, J. (2008, September). Is cloned code more stable than non-cloned code?. In *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation* (pp. 57-66). IEEE.
- Krinke, J. (2011, May). Is cloned code older than non-cloned code?. In *Proceedings of the 5th International Workshop on Software Clones* (pp. 28-33).
- Alam, A. I., Roy, P. R., Al-Omari, F., Roy, C. K., Roy, B., & Schneider, K. A. (2023, October). GPTCloneBench: A comprehensive benchmark of semantic clones and cross-language clones using GPT-3 model and SemanticCloneBench. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 1-13). IEEE.
- Roy, P. R., Alam, A. I., Al-Omari, F., Roy, B., Roy, C. K., & Schneider, K. A. (2023, October). Unveiling the potential of large language models in generating semantic and cross-language clones. In *2023 IEEE 17th International Workshop on Software Clones (IWSC)* (pp. 22-28). IEEE.
- Mondal, M., Roy, B., Roy, C. K., & Schneider, K. A. (2020, February). HistoRank: History-based ranking of co-change candidates. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 240-250). IEEE.
- Lozano, A., & Wermelinger, M. (2008, September). Assessing the effect of clones on changeability. In *2008 IEEE International Conference on Software Maintenance* (pp. 227-236). IEEE.
- Lozano, A., & Wermelinger, M. (2010, May). Tracking clones' imprint. In *Proceedings of the 4th International Workshop on Software Clones* (pp. 65-72).
- Juergens, E., Deissenboeck, F., Hummel, B., & Wagner, S. (2009, May). Do code clones matter?. In *2009 IEEE 31st International Conference on Software Engineering* (pp. 485-495). IEEE.
- Barbour, L., Khomh, F., & Zou, Y. (2011, September). Late propagation in software clones. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)* (pp. 273-282). IEEE.
- Mondal, M., Roy, C. K., & Schneider, K. A. (2018). Bug-proneness and late propagation tendency of code clones: A comparative study on different clone types. *Journal of Systems and Software*, *144*, 41-59.
- Roy, C. K., Zibran, M. F., & Koschke, R. (2014, February). The vision of software clone management: Past, present, and future (keynote paper). In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)* (pp. 18-33). IEEE.