lab
sesa
SOFTWARE ENGINEERING
SALERNO

When Code Smells Meet ML: On the Lifecycle of ML-specific Code Smells in ML-enabled Systems

Gilberto Recupito, **Giammaria Giordano**, Filomena Ferrucci, Dario Di Nucci, Fabio Palomba

University of Salerno (Italy)
Department of Computer Science
Software Engineering (SeSa) Lab

giagiordano@unisa.it
giammariagiordano.github.io/giammaria-giordano
@giammariagiord1

```
from sklearn.cluster import KMeans
kmeans = KMeans()
```

```python
from sklearn.cluster import KMeans
kmeans = KMeans()
```

A simple invocation of KMeans function

```python
from sklearn.cluster import KMeans
kmeans = KMeans()
```

A simple invocation of KMeans function

...Are we sure?

```python
from sklearn.cluster import KMeans
kmeans = KMeans()
```

A simple invocation of KMeans function

...Are we sure?

What if the default hyperparameters change due to some library updates?

```python
from sklearn.cluster import KMeans
kmeans = KMeans()
```

The model performance could change...

```
from sklearn.cluster import KMeans
```

ML-specific code smell!

The model performance could cha

Background and Context

Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particular, code smells have rarely been studied in this domain. Although machine learning code is usually integrated as a small part of an overarching system, it usually plays an important role in its core functionality. Hence ensuring code quality is quintessential to avoid issues in the long run. This paper proposes and identifies a list of 22 machine learning-specific code smells collected from various sources, including papers, grey literature, GitHub commits, and Stack Overflow posts. We pinpoint each smell with a description of its context, potential issues in the long run, and proposed solutions. In addition, we link them to their respective pipeline stage and the evidence from both academic and grey literature. The code smell catalog helps data scientists and developers produce and maintain high-quality machine learning application code.

KEYWORDS

Code Smell, Anti-pattern, Machine Learning, Code Quality, Technical Debt

ACM Reference Format:
Haiyin Zhang, Luís Cruz, and Arie van Deursen. 2022. Code Smells for Machine Learning Applications. In *1st Conference on AI Engineering - Software Engineering for AI (CAIN'22), May 16–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3522664.3528620

1 INTRODUCTION

Despite the large increase in the popularity of machine learning applications [3], there are several concerns regarding the quality control and the inevitable technical debt growing in these systems [16]. Moreover, machine learning teams tend to be very heterogeneous, having experts from different disciplines that are not necessarily aware of Software Engineering (SE) practices backgrounds and there is a limited number of training and guidelines on machine learning-related software development issues. Hence, software engineering best practices are often overlooked when developing machine learning applications [12, 17]. Yet, previous research shows that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].

There has been a lot of interest in various machine learning system artifacts, including models and data. Researchers make efforts to improve machine learning model quality [10] and data quality [7]. However, the quality assurance of machine learning code has not been highlighted [12]. Recent work studied the code quality for machine learning applications in a general way, finding some code quality issues such as duplicated code [20] and violations of traditional naming convention [17]. These works highlighted the fact that the existing code conventions do not necessarily fit the context of machine learning applications. For example, the typical math notation in data science tasks clashes with the naming conventions of Python [20]. Thus, we argue that more research is needed to accommodate the particularities of data-oriented codebases.

As an important artifact in the machine learning application, the quality of the code is essential. Low-quality code can lead to catastrophic consequences. In the meantime, different from traditional software, machine learning code quality is more challenging to evaluate and control. Low-quality code can lead to silent pitfalls that exist somewhere that affect the software quality, which takes a lot of time and effort to discover [22]. Therefore, it is non-trivial to improve the code quality during the development process and consider code quality assurance in the deployment process.

A common strategy to improve code quality is eliminating code smells and anti-patterns. When we talk about code smells in this paper, we refer them to the pitfalls that we can inspect at the code level but not at the data or model level. We use the term "pitfall" to represent issues that degrade the software quality. Listing 1 shows an example of such pitfalls using Python and the Pandas library. In the red (-) part of the listing, an inefficient loop is created. A better alternative is highlighted in green (+), using Pandas built-in API to replace the loop, which operates faster. While some alternative solutions might lead to improvements in runtime efficiency, other solutions might be essential to prevent problems in the long run. For example, previous work shows that code smells affect the maintainability, understandability, and complexity of software [11].

Listing 1: Coding Pitfall Example from [4]

```
import pandas as pd
df = pd.DataFrame([1, 2, 3])

- result = []
- for index, row in df.iterrows():
-     result.append(row[0] + 1)
- result = pd.DataFrame(result)
+ result = df.add(1)
```

With the concern of improving machine learning application code quality and easing the machine learning development process,

CAIN'22, May 21-22 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9275-4/22/05…$15.00
https://doi.org/10.1145/3522664.3528620

Background and Context

Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particular, code smells have rarely been studied in this domain. Although machine learning code is usually integrated as a small part of an overarching system, it usually plays an important role in its core functionality. Hence ensuring code quality is quintessential to avoid issues in the long run. This paper proposes and identifies a list of 22 machine learning-specific code smells collected from various sources, including papers, grey literature, GitHub commits, and Stack Overflow posts. We pinpoint each smell with a description of its context, potential issues in the long run, and proposed solutions. In addition, we link them to their respective pipeline stage and the evidence from both academic and grey literature. The code smell catalog helps data scientists and developers produce and maintain high-quality machine learning application code.

KEYWORDS

Code Smell, Anti-pattern, Machine Learning, Code Quality, Technical Debt

ACM Reference Format:
Haiyin Zhang, Luís Cruz, and Arie van Deursen. 2022. Code Smells for Machine Learning Applications. In *1st Conference on AI Engineering - Software Engineering for AI (CAIN'22), May 16–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3522664.3528620

1 INTRODUCTION

Despite the large increase in the popularity of machine learning applications [3], there are several concerns regarding the quality control and the inevitable technical debt growing in these systems [16]. Moreover, machine learning teams tend to be very heterogeneous, having experts from different disciplines that are not necessarily aware of Software Engineering (SE) practices backgrounds and there is a limited number of training and guidelines on machine learning-related software development issues. Hence, software engineering best practices are often overlooked when developing machine learning applications [12, 17]. Yet, previous research shows that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].

There has been a lot of interest in various machine learning system artifacts, including models and data. Researchers make efforts to improve machine learning model quality [10] and data quality [7]. However, the quality assurance of machine learning code has not been highlighted [12]. Recent work studied the code quality for machine learning applications in a general way, finding some code quality issues such as duplicated code [20] and violations of traditional naming convention [17]. These works highlighted the fact that the existing code conventions do not necessarily fit the context of machine learning applications. For example, the typical math notation in data science tasks clashes with the naming conventions of Python [20]. Thus, we argue that more research is needed to accommodate the particularities of data-oriented codebases.

As an important artifact in the machine learning application, the quality of the code is essential. Low-quality code can lead to catastrophic consequences. In the meantime, different from traditional software, machine learning code quality is more challenging to evaluate and control. Low-quality code can lead to silent pitfalls that exist somewhere that affect the software quality, which takes a lot of time and effort to discover [22]. Therefore, it is non-trivial to improve the code quality during the development process and consider code quality assurance in the deployment process.

A common strategy to improve code quality is eliminating code smells and anti-patterns. When we talk about code smells in this paper, we refer them to the pitfalls that we can inspect at the code level but not at the data or model level. We use the term "pitfall" to represent issues that degrade the software quality. Listing 1 shows an example of such pitfalls using Python and the Pandas library. In the red (-) part of the listing, an inefficient loop is created. A better alternative is highlighted in green (+), using Pandas built-in API to replace the loop, which operates faster. While some alternative solutions might lead to improvements in runtime efficiency, other solutions might be essential to prevent problems in the long run. For example, previous work shows that code smells affect the maintainability, understandability, and complexity of software [11].

Listing 1: Coding Pitfall Example from [4]

```
import pandas as pd
df = pd.DataFrame([1, 2, 3])

- result = []
- for index, row in df.iterrows():
-   result.append(row[0] + 1)
- result = pd.DataFrame(result)
+ result = df.add(1)
```

With the concern of improving machine learning application code quality and easing the machine learning development process,

CAIN'22, May 21-22 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9275-4/22/05…$15.00
https://doi.org/10.1145/3522664.3528620

Background and Context

Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particular, code smells have rarely been studied in this domain. Although machine learning code is usually integrated as a small part of an overarching system, it usually plays an important role in its core functionality. Hence ensuring code quality is quintessential to avoid issues in the long run. This paper proposes and identifies a list of 22 machine learning-specific code smells collected from various sources, including papers, grey literature, GitHub commits, and Stack Overflow posts. We pinpoint each smell with a description of its context, potential issues in the long run, and proposed solutions. In addition, we link them to their respective pipeline stage and the evidence from both academic and grey literature. The code smell catalog helps data scientists and developers produce and maintain high-quality machine learning application code.

KEYWORDS

Code Smell, Anti-pattern, Machine Learning, Code Quality, Technical Debt

ACM Reference Format:
Haiyin Zhang, Luís Cruz, and Arie van Deursen. 2022. Code Smells for Machine Learning Applications. In *1st Conference on AI Engineering - Software Engineering for AI (CAIN'22), May 16–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3522664.3528620

1 INTRODUCTION

Despite the large increase in the popularity of machine learning applications [3], there are several concerns regarding the quality control and the inevitable technical debt growing in these systems [16]. Moreover, machine learning teams tend to be very heterogeneous, having experts from different disciplines that are not necessarily aware of Software Engineering (SE) practices backgrounds and there is a limited number of training and guidelines on machine learning-related software development issues. Hence, software engineering best practices are often overlooked when developing machine learning applications [12, 17]. Yet, previous research shows that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].

There has been a lot of interest in various machine learning system artifacts, including models and data. Researchers make efforts to improve machine learning model quality [10] and data quality [7]. However, the quality assurance of machine learning code has not been highlighted [12]. Recent work studied the code quality for machine learning applications in a general way, finding some code quality issues such as duplicated code [20] and violations of traditional naming convention [17]. These works highlighted the fact that the existing code conventions do not necessarily fit the context of machine learning applications. For example, the typical math notation in data science tasks clashes with the naming conventions of Python [20]. Thus, we argue that more research is needed to accommodate the particularities of data-oriented codebases.

As an important artifact in the machine learning application, the quality of the code is essential. Low-quality code can lead to catastrophic consequences. In the meantime, different from traditional software, machine learning code quality is more challenging to evaluate and control. Low-quality code can lead to silent pitfalls that exist somewhere that affect the software quality, which takes a lot of time and effort to discover [22]. Therefore, it is non-trivial to improve the code quality during the development process and consider code quality assurance in the deployment process.

A common strategy to improve code quality is eliminating code smells and anti-patterns. When we talk about code smells in this paper, we refer them to the pitfalls that we can inspect at the code level but not at the data or model level. We use the term "pitfall" to represent issues that degrade the software quality. Listing 1 shows an example of such pitfalls using Python and the Pandas library. In the red (-) part of the listing, an inefficient loop is created. A better alternative is highlighted in green (+), using Pandas built-in API to replace the loop, which operates faster. While some alternative solutions might lead to improvements in runtime efficiency, other solutions might be essential to prevent problems in the long run. For example, previous work shows that code smells affect the maintainability, understandability, and complexity of software [11].

Listing 1: Coding Pitfall Example from [4]

```
import pandas as pd
df = pd.DataFrame([1, 2, 3])

- result = []
- for index, row in df.iterrows():
-   result.append(row[0] + 1)
- result = pd.DataFrame(result)
+ result = df.add(1)
```

With the concern of improving machine learning application code quality and easing the machine learning development process,

ACM ISBN 978-1-4503-9275-4/22/05…$15.00
https://doi.org/10.1145/3522664.3528620

Multivocal Literature Review

Background and Context

Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particular, code smells have rarely been studied in this domain. Although machine learning code is usually integrated as a small part of an overarching system, it usually plays an important role in its core functionality. Hence ensuring code quality is quintessential to avoid issues in the long run. This paper proposes and identifies a list of 22 machine learning-specific code smells collected from various sources, including papers, grey literature, GitHub commits, and Stack Overflow posts. We pinpoint each smell with a description of its context, potential issues in the long run, and proposed solutions. In addition, we link them to their respective pipeline stage and the evidence from both academic and grey literature. The code smell catalog helps data scientists and developers produce and maintain high-quality machine learning application code.

KEYWORDS

Code Smell, Anti-pattern, Machine Learning, Code Quality, Technical Debt

ACM Reference Format:
Haiyin Zhang, Luís Cruz, and Arie van Deursen. 2022. Code Smells for Machine Learning Applications. In *1st Conference on AI Engineering - Software Engineering for AI (CAIN'22), May 16–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3522664.3528620

1 INTRODUCTION

Despite the large increase in the popularity of machine learning applications [3], there are several concerns regarding the quality control and the inevitable technical debt growing in these systems [16]. Moreover, machine learning teams tend to be very heterogeneous, having experts from different disciplines that are not necessarily aware of Software Engineering (SE) practices backgrounds and there is a limited number of training and guidelines on machine learning-related software development issues. Hence, software engineering best practices are often overlooked when developing machine learning applications [12, 17]. Yet, previous research shows that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].

There has been a lot of interest in various machine learning system artifacts, including models and data. Researchers make efforts to improve machine learning model quality [10] and data quality [7]. However, the quality assurance of machine learning code has not been highlighted [12]. Recent work studied the code quality for machine learning applications in a general way, finding some code quality issues such as duplicated code [20] and violations of traditional naming convention [17]. These works highlighted the fact that the existing code conventions do not necessarily fit the context of machine learning applications. For example, the typical math notation in data science tasks clashes with the naming conventions of Python [20]. Thus, we argue that more research is needed to accommodate the particularities of data-oriented codebases.

As an important artifact in the machine learning application, the quality of the code is essential. Low-quality code can lead to catastrophic consequences. In the meantime, different from traditional software, machine learning code quality is more challenging to evaluate and control. Low-quality code can lead to silent pitfalls that exist somewhere that affect the software quality, which takes a lot of time and effort to discover [22]. Therefore, it is non-trivial to improve the code quality during the development process and consider code quality assurance in the deployment process.

A common strategy to improve code quality is eliminating code smells and anti-patterns. When we talk about code smells in this paper, we refer them to the pitfalls that we can inspect at the code level but not at the data or model level. We use the term "pitfall" to represent issues that degrade the software quality. Listing 1 shows an example of such pitfalls using Python and the Pandas library. In the red (-) part of the listing, an inefficient loop is created. A better alternative is highlighted in green (+), using Pandas built-in API to replace the loop, which operates faster. While some alternative solutions might lead to improvements in runtime efficiency, other solutions might be essential to prevent problems in the long run. For example, previous work shows that code smells affect the maintainability, understandability, and complexity of software [11].

Listing 1: Coding Pitfall Example from [4]

```
import pandas as pd
df = pd.DataFrame([1, 2, 3])

- result = []
- for index, row in df.iterrows();
-     result.append(row[0] + 1)
- result = pd.DataFrame(result)
+ result = df.add(1)
```

With the concern of improving machine learning application code quality and easing the machine learning development process,

CAIN'22, May 21-22 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9275-4/22/05…$15.00
https://doi.org/10.1145/3522664.3528620

Multivocal Literature Review

22 ML-specific code smells identified

State-Of-The-Art

Prevalence of Code Smells in Reinforcement Learning Projects

Nicolás Cardozo
Universidad de los Andes
Bogotá, Colombia
n.cardozo@uniandes.edu.co

Ivana Dusparic
Trinity College Dublin
Dublin, Ireland
ivana.dusparic@tcd.ie

Christian Cabrera
University of Cambridge
Cambridge, UK
chc79@cam.ac.uk

Abstract—**Reinforcement Learning (RL) is being increasingly used to learn and adapt application behavior in many domains, including large-scale and safety critical systems, as for example**

RL to improve on the processing power or accuracy of an existing solution as an immediate goal, but set aside middle

The Prevalence of Code Smells in Machine Learning projects

Bart van Oort[1,2], Luís Cruz[2], Maurício Aniche[2], Arie van Deursen[2]
Delft University of Technology
[1] *AI for Fintech Research, ING*
[2] Delft, Netherlands
bart.van.oort@ing.com, {l.cruz, m.f.aniche, arie.vandeursen}@tudelft.nl

Abstract—**Artificial Intelligence (AI) and Machine Learning (ML) are pervasive in the current computer science landscape. Yet, there still exists a lack of software engineering experience and best practices in this field. One such best practice, static code analysis, can be used to find code smells, i.e., (potential) defects in the source code, refactoring opportunities, and violations of common coding s**

which we amalgamate into 'code smells' for the rest of this paper. Research has shown that the attributes of quality most affected by code smells are maintainability, understandability and complexity, and that early detection of code smells reduces the cost of maintenance [7].

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)

An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems

Yiming Tang*, Raffi Khatchadourian†*, Mehdi Bagherzadeh‡, Rhia Singh§, Ajani Stewart†, Anita Raja†*
*CUNY Graduate Center, †CUNY Hunter College, ‡Oakland University, §CUNY Macaulay Honors College
Email: ytang3@gradcenter.cuny.edu, raffi.khatchadourian@hunter.cuny.edu, mbagherzadeh@oakland.edu, rhia.singh@macaulay.cuny.edu, ajani.stewart42@myhunter.cuny.edu, anita.raja@hunter.cuny.edu

Abstract—**Machine Learning (ML), including Deep Learning (DL), systems, i.e., those with ML capabilities, are pervasive in today's data-driven society. Such systems are complex; they are comprised of ML models and many subsystems that support learning processes. As with other complex systems, ML systems are prone to classic technical debt issues, especially when such systems are long-lived, but they also exhibit debt specific to these systems. Unfortunately, there is a gap of knowledge in how ML systems actually evolve and are maintained. In this paper, we fill this gap by studying refactorings, i.e., source-to-source semantics-**

open-source ML systems. We set out to discover (i) the kinds of *refactorings*—both specific and tangential to ML—performed, (ii) whether particular refactorings occurred *more often* in model code vs. other supporting subsystems, (iii) the types of *technical debt* being addressed and whether they correspond to established ML-specific technical debt [1], and (iv) whether any *new*—potentially generalizable—ML-specific refactorings and technical debt categories could be derived.

Knowing the kinds of refactorings and technical debt

Understanding Developer Practices and Code Smells Diffusion in AI-Enabled Software: A Preliminary Study

Giammaria Giordano[1], Giusy Annunziata[1], Andrea De Lucia[1] and Fabio Palomba[1]

[1]*University of Salerno (Italy) - SeSa Lab*

Abstract

To deal with continuous change requests and the strict time-to-market, practitioners and big companies constantly update their software systems to meet users' requirements. This practice force developers to release immature products, neglecting best practices to reduce delivery times. As a possible result, *technical debt* can arise, *i.e.,* potential design issues that can negatively impact software maintenance and evolution and, in turn, increase both the time-to-market and costs. *Code smells*—sub-optimal design decisions identifiable by computing software metrics and providing a general overview of code quality —are common symptoms of technical debt. While previous research focused on code smells primarily considering them in the context of Java, the growing popularity of Python, particularly for developing artificial intelligence (AI)-Enabled systems, calls for additional investigations. This preliminary analysis addresses this gap by exploring the diffusion of Python-specific code smells, and the activities performed by developers that induce the introduction of code smells in their systems. To perform our preliminary investigation, we selected 200 AI-Enabled systems available in the NICHE dataset; We extracted 10,611 information on the releases using PYDRILLER, and PYSMELL to extract information about code smells. The results reveal several insights: 1) Code smells related to object-oriented principles are rarely detected in Python; 2) Complex List Comprehension is the most prevalent and the most long-alive

State-Of-The-Art

Prevalence of Code Smells in Reinforcement Learning Projects

Nicolás Cardozo
Universidad de los Andes
Bogotá, Colombia
n.cardozo@uniandes.edu.co

Ivana Dusparic
Trinity College Dublin
Dublin, Ireland
ivana.dusparic@tcd.ie

Christian Cabrera
University of Cambridge
Cambridge, UK
chc79@cam.ac.uk

The Prevalence of Code Smells in Machine Learning projects

Bart van Oort[1,2], Luís Cruz[2], Maurício Aniche[2], Arie van Deursen[2]
Delft University of Technology
[1] AI for Fintech Research, ING
[2] Delft, Netherlands
bart.van.oort@ing.com, {l.cruz, m.f.aniche, arie.vandeursen}@tudelft.nl

No empirical studies on ML-specific code smells!

An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems

Yiming Tang*, Raffi Khatchadourian†*, Mehdi Bagherzadeh‡, Rhia Singh§, Ajani Stewart†, Anita Raja†*
*CUNY Graduate Center, †CUNY Hunter College, ‡Oakland University, §CUNY Macaulay Honors College
Email: ytang3@gradcenter.cuny.edu, raffi.khatchadourian@hunter.cuny.edu, mbagherzadeh@oakland.edu,
rhia.singh@macaulay.cuny.edu, ajani.stewart42@myhunter.cuny.edu, anita.raja@hunter.cuny.edu

Abstract—Machine Learning (ML), including Deep Learning (DL), systems, i.e., those with ML capabilities, are pervasive in today's data-driven society. Such systems are complex; they are comprised of ML models and many subsystems that support learning processes. As with other complex systems, ML systems are prone to classic technical debt issues, especially when such systems are long-lived, but they also exhibit debt specific to these systems. Unfortunately, there is a gap of knowledge in how ML systems actually evolve and are maintained. In this paper, we fill this gap by studying refactorings, i.e., source-to-source semantics-

open-source ML systems. We set out to discover (i) the kinds of *refactorings*—both specific and tangential to ML—performed, (ii) whether particular refactorings occurred *more often* in model code vs. other supporting subsystems, (iii) the types of *technical debt* being addressed and whether they correspond to established ML-specific technical debt [1], and (iv) whether any *new*—potentially generalizable—ML-specific refactorings and technical debt categories could be derived.

[1]University of Salerno (Italy) - SeSa Lab

IDEA

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 43, NO. 11, NOVEMBER 2017 1063

When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)

Michele Tufano, *Student Member, IEEE*, Fabio Palomba, *Member, IEEE*, Gabriele Bavota, *Member, IEEE*, Rocco Oliveto, *Member, IEEE*, Massimiliano Di Penta, *Member, IEEE*, Andrea De Lucia, *Senior Member, IEEE*, and Denys Poshyvanyk, *Member, IEEE*

Abstract—Technical debt is a metaphor introduced by Cunningham to indicate "not quite right code which we postpone making it right". One noticeable symptom of technical debt is represented by code smells, defined as symptoms of poor design and implementation choices. Previous studies showed the negative impact of code smells on the comprehensibility and maintainability of code. While the repercussions of smells on code quality have been empirically assessed, there is still only anecdotal evidence on *when* and *why* bad smells are introduced, what is their *survivability*, and *how* they are *removed* by developers. To empirically corroborate such anecdotal evidence, we conducted a large empirical study over the change history of 200 open source projects. This study required the development of a strategy to identify smell-introducing commits, the mining of over half a million of commits, and the manual analysis and classification of over 10K of them. Our findings mostly contradict common wisdom, showing that most of the smell instances are introduced when an artifact is created and not as a result of its evolution. At the same time, 80 percent of smells survive in the system. Also, among the 20 percent of removed instances, only 9 percent are removed as a direct consequence of refactoring operations.

Index Terms—Code smells, empirical study, mining software repositories

1 Introduction

The technical debt metaphor introduced by Cunningham [22] explains well the trade-offs between delivering the most appropriate but still immature product, in the shortest time possible [14], [22], [42], [47], [70]. Bad code smells (shortly "code smells" or "smells"), i.e., symptoms of poor design and implementation choices [27], represent one important factor contributing to technical debt, and possibly affecting the

empirically proven, there is still noticeable lack of empirical evidence related to how, when, and why they occur in software projects, as well as whether, after how long, and how they are removed [14]. This represents an obstacle for an effective and efficient management of technical debt. Also, understanding the typical life-cycle of code smells and the actions undertaken by developers to remove them is of paramount

IDEA

Wh
Ba

Gabriele Bavota
Ar

Abstract—Te
right". One not
implementation
code. While th
when and *why*
corroborate su
This study req
and the manua
of the smell ins
smells survive
consequence

Index Terms—

1 INTRODUCTI

THE technical d
[22] explains
most appropriate
time possible [14],
"code smells" or "
implementation c
contributing to te

When Code Smells Meet ML: On the Lifecycle of ML-specific Code Smells in ML-enabled Systems

Gilberto Recupito
Sesa Lab - University of Salerno
Salerno, Italy
grecupito@unisa.it

Giammaria Giordano
Sesa Lab - University of Salerno
Salerno, Italy
giagiordano@unisa.it

Filomena Ferrucci
Sesa Lab - University of Salerno
Salerno, Italy
fferrucci@unisa.it

Dario Di Nucci
Sesa Lab - University of Salerno
Salerno, Italy
ddinucci@unisa.it

Fabio Palomba
Sesa Lab - University of Salerno
Salerno, Italy
fpalomba@unisa.it

ABSTRACT

Context. The adoption of Machine Learning (ML)–enabled systems is steadily increasing. Nevertheless, there is a shortage of ML-specific quality assurance approaches, possibly because of the limited knowledge of how quality-related concerns emerge and evolve in ML-enabled systems. **Objective.** We aim to investigate the emergence and evolution of specific types of quality-related

1 INTRODUCTION

Machine Learning (ML) evolved through the emergence of complex software integrating ML modules, defined as ML-enabled systems [13]. Self-driving cars, voice assistance instruments, or conversational agents like ChatGPT[1] are just some examples of the successful integration of ML within software engineering projects.

However, the strict time-to-market and change requests pres-

IDEA

ML-specific code smells

Sesa Lab - University of Salerno
Salerno, Italy
grecupito@unisa.it

Sesa Lab - University of Salerno
Salerno, Italy
giagiordano@unisa.it

Sesa Lab - University of Salerno
Salerno, Italy
fferrucci@unisa.it

Dario Di Nucci
Sesa Lab - University of Salerno
Salerno, Italy
ddinucci@unisa.it

Fabio Palomba
Sesa Lab - University of Salerno
Salerno, Italy
fpalomba@unisa.it

ABSTRACT

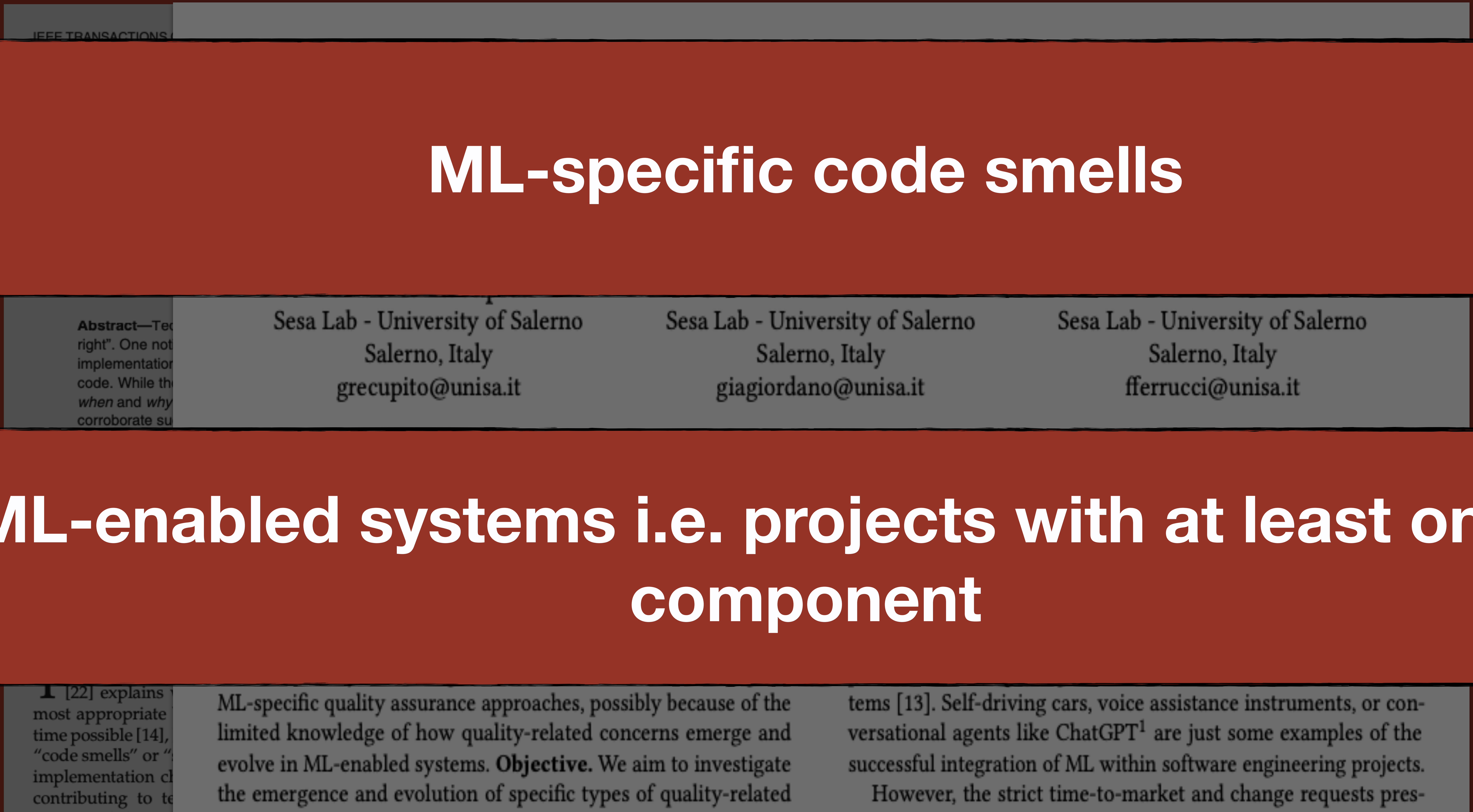
Context. The adoption of Machine Learning (ML)–enabled systems is steadily increasing. Nevertheless, there is a shortage of ML-specific quality assurance approaches, possibly because of the limited knowledge of how quality-related concerns emerge and evolve in ML-enabled systems. **Objective.** We aim to investigate the emergence and evolution of specific types of quality-related

1 INTRODUCTION

Machine Learning (ML) evolved through the emergence of complex software integrating ML modules, defined as ML-enabled systems [13]. Self-driving cars, voice assistance instruments, or conversational agents like ChatGPT[1] are just some examples of the successful integration of ML within software engineering projects.

However, the strict time-to-market and change requests pres-

IDEA


Sesa Lab - University of Salerno
Salerno, Italy
grecupito@unisa.it

Sesa Lab - University of Salerno
Salerno, Italy
giagiordano@unisa.it

Sesa Lab - University of Salerno
Salerno, Italy
fferrucci@unisa.it

ML-specific quality assurance approaches, possibly because of the limited knowledge of how quality-related concerns emerge and evolve in ML-enabled systems. **Objective.** We aim to investigate the emergence and evolution of specific types of quality-related

tems [13]. Self-driving cars, voice assistance instruments, or conversational agents like ChatGPT[1] are just some examples of the successful integration of ML within software engineering projects.
However, the strict time-to-market and change requests pres-

ML-specific code smells

ML-enabled systems i.e. projects with at least one ML component

Goal

Analyze the **prevalence**, **introduction**, **removal**, and **survival** of ML-specific code smells in ML-enabled systems

Research Questions

RQ0 | **How** are **ML-specific code smells prevalent** in ML-enabled systems?

RQ1 | **When** are **ML-specific code smells introduced** in ML-enabled systems?

RQ2 | **What tasks** were **performed** when the **ML-CSs** were **introduced**?

RQ3 | **When** and **how ML-specific code smells** are **removed** in ML-enabled systems?

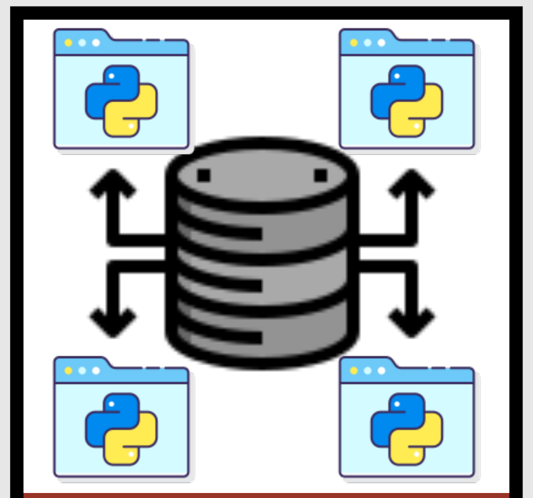
RQ4 | **How** long do **ML-specific code smells survive** in ML-enabled systems?

Research Process

Projects Selection

Data Extraction

Data Analysis



NICHE

ML Projects

NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python

Ratnadira Widyasari, Zhou Yang, Ferdian Thung, Sheng Qin Sim, Fiona Wee, Camellia Lok, Jack Phan, Haodi Qi, Constance Tan, Qijin Tay, and David Lo
School of Computing and Information System, Singapore Management University
{ratnadiraw.2020,zyang,ferdianthung,sqsim.2018,fiona.wee.2018,camellialok.2017,jack.phan.2018}@smu.edu.sg
{haodi.qi.2017,hytan.2018,qijin.tay.2018,davidlo}@smu.edu.sg

***Abstract*—Machine learning (ML) has gained much attention and been incorporated into our daily lives. While there are numerous publicly available ML projects on open source platforms such as GitHub, there have been limited attempts in filtering those projects to curate ML projects of high quality. The limited availability of such high-quality dataset poses an obstacle in understanding ML projects. To help clear this obstacle, we present *NICHE*, a manually labelled dataset consisting of 572 ML projects. Based on evidences of good software engineering practices, we label 441 of these projects as engineered and 131 as non-engineered. This dataset can help researchers understand the practices that are followed in high-quality ML projects. It can also be used as a benchmark for classifiers designed to identify engineered ML projects.**

***Index Terms*—Engineered Software Project, Machine Learning, Python, Open Source Projects**

I. Introduction

There are many valuable pieces of information stored in a version control system of a project; they include: source code, documentation, issue reports, test cases, list of contributors, etc. Researchers mine these software repositories to get useful insights related to how bugs are fixed [1], how developers collaborate [2] and so on. With the abundance of the open source repositories in GitHub, researchers can mine for insights and validate hypotheses on a large corpus of data. However, Kalliamvakou et al. showed that most repositories in Github are of low-quality [3], [4], which can lead to wrong and biased conclusions. To avoid skewed findings, researchers usually take some measures to filter out low-quality projects, e.g., by choosing projects with a high number of stars (which is considered to reflect the projects' popularity). Unfortunately, popularity may not be correlated with project quality [5]. Therefore, Munaiah et al. propose an approach to find high-quality software projects, more specifically; by identifying engineered software projects [6]. Such projects are essential for mining software repository (MSR) research, as they allow for high-quality findings to be uncovered (from high-quality data).

Machine learning (ML) projects are becoming increasingly popular and play essential roles in various domain, e.g., code processing [7], [8], self-driving cars, speech recognition [9], etc. Despite widespread usage and popularity, only a few research works try to examine AI and ML projects to identify unique properties, development patterns, and trends. Gonzalez et al. [10] find that the AI & ML community has unique characteristics that should be considered in future software engineering and MSR research. For example, more support is needed for Python as the main programming language, and there are significant differences between internal and external contributors in AI & ML projects. We coin a term for such research: Mining Machine Learning Repository (MLR). Similar to conventional MSR research, MLR also requires high-quality projects. In GitHub, there are many tutorials, resource pages, courseworks and toy projects that are related to ML; some of which are very popular but unsuitable for MLR research. To facilitate MLR research, we present a curated dataset of E**N**g**I**neered Ma**CH**ine **LE**arning Projects in Python (*NICHE*). We first automatically identify projects in GitHub that: (1) use one of the popular ML libraries, and (2) satisfy some basic quantitative quality metrics. This process returns 572 ML projects from GitHub. Next, we manually analyze the 572 ML projects and label them as engineered or not engineered. This dataset can be used as the raw material for MLR research, or as the benchmark for evaluating classifiers designed to identify engineered ML projects.

We label the dataset manually to ensure high quality and accurate labels. Our criteria for assessing an ML project are rooted in Munaiah et al. work [6]. We check 8 distinct dimensions of a project (architecture, community, CI, documentation, history, issues, license and unit testing) to evaluate whether the project is engineered or not. Out of the 572 projects we collected, 441 projects are labelled as engineered ML projects, and 131 projects are labelled as non-engineered ML projects. There are several related datasets in the literature. Datasets from [6] and [11] have labels indicating whether a project is engineered or not, but they do not contain ML projects. Gonzalez et al. [10] collected a dataset of ML & AI projects, but these projects are not comprehensively assessed based on their adoption of good software engineering practices. They only eliminate tutorials, homework assignments and so on. We make our dataset publicly available[1].

The rest of this paper is organized as follow. Section 2 describes the methodology used to collect and filter the dataset, as well as how the dataset is stored. Section 3 gives an overview of the dataset. In Section 4, we propose some

[1] https://doi.org/10.6084/m9.figshare.21967265

ML Projects

NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python

Ratnadira Widyasari, Zhou Yang, Ferdian Thung, Sheng Qin Sim, Fiona Wee, Camellia Lok, Jack Phan, Haodi Qi, Constance Tan, Qijin Tay, and David Lo
School of Computing and Information System, Singapore Management University
{ratnadiraw.2020,zyang,ferdianthung,sqsim.2018,fiona.wee.2018,camellialok.2017,jack.phan.2018}@smu.edu.sg
{haodi.qi.2017,hytan.2018,qijin.tay.2018,davidlo}@smu.edu.sg

***Abstract*—Machine learning (ML) has gained much attention and been incorporated into our daily lives. While there are numerous publicly available ML projects on open source platforms such as GitHub, there have been limited attempts in filtering those projects to curate ML projects of high quality. The limited availability of such high-quality dataset poses an obstacle in understanding ML projects. To help clear this obstacle, we present *NICHE*, a manually labelled dataset consisting of 572 ML projects. Based on evidences of good software engineering practices, we label 441 of these projects as engineered and 131 as non-engineered. This dataset can help researchers understand the practices that are followed in high-quality ML projects. It can also be used as a benchmark for classifiers designed to identify engineered ML projects.**

***Index Terms*—Engineered Software Project, Machine Learning, Python, Open Source Projects**

I. Introduction

There are many valuable pieces of information stored in a version control system of a project; they include: source code, documentation, issue reports, test cases, list of contributors, etc. Researchers mine these software repositories to get useful insights related to how bugs are fixed [1], how developers collaborate [2] and so on. With the abundance of the open source repositories in GitHub, researchers can mine for insights and validate hypotheses on a large corpus of data. However, Kalliamvakou et al. showed that most repositories in Github are of low-quality [3], [4], which can lead to wrong and biased conclusions. To avoid skewed findings, researchers usually take some measures to filter out low-quality projects, e.g., by choosing projects with a high number of stars (which is considered to reflect the projects' popularity). Unfortunately, popularity may not be correlated with project quality [5]. Therefore, Munaiah et al. propose an approach to find high-quality software projects, more specifically; by identifying engineered software projects [6]. Such projects are essential for mining software repository (MSR) research, as they allow for high-quality findings to be uncovered (from high-quality data).

Machine learning (ML) projects are becoming increasingly popular and play essential roles in various domain, e.g., code processing [7], [8], self-driving cars, speech recognition [9], etc. Despite widespread usage and popularity, only a few research works try to examine AI and ML projects to identify unique properties, development patterns, and trends. Gonzalez et al. [10] find that the AI & ML community has unique characteristics that should be considered in future software engineering and MSR research. For example, more support is needed for Python as the main programming language, and there are significant differences between internal and external contributors in AI & ML projects. We coin a term for such research: Mining Machine Learning Repository (MLR). Similar to conventional MSR research, MLR also requires high-quality projects. In GitHub, there are many tutorials, resource pages, courseworks and toy projects that are related to ML; some of which are very popular but unsuitable for MLR research. To facilitate MLR research, we present a curated dataset of E**N**g**I**neered Ma**CH**ine L**E**arning Projects in Python (*NICHE*). We first automatically identify projects in GitHub that: (1) use one of the popular ML libraries, and (2) satisfy some basic quantitative quality metrics. This process returns 572 ML projects from GitHub. Next, we manually analyze the 572 ML projects and label them as engineered or not engineered. This dataset can be used as the raw material for MLR research, or as the benchmark for evaluating classifiers designed to identify engineered ML projects.

We label the dataset manually to ensure high quality and accurate labels. Our criteria for assessing an ML project are rooted in Munaiah et al. work [6]. We check 8 distinct dimensions of a project (architecture, community, CI, documentation, history, issues, license and unit testing) to evaluate whether the project is engineered or not. Out of the 572 projects we collected, 441 projects are labelled as engineered ML projects, and 131 projects are labelled as non-engineered ML projects. There are several related datasets in the literature. Datasets from [6] and [11] have labels indicating whether a project is engineered or not, but they do not contain ML projects. Gonzalez et al. [10] collected a dataset of ML & AI projects, but these projects are not comprehensively assessed based on their adoption of good software engineering practices. They only eliminate tutorials, homework assignments and so on. We make our dataset publicly available[1].

The rest of this paper is organized as follow. Section 2 describes the methodology used to collect and filter the dataset, as well as how the dataset is stored. Section 3 gives an overview of the dataset. In Section 4, we propose some

[1]https://doi.org/10.6084/m9.figshare.21967265

572 ML projects

ML Projects

NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python

Ratnadira Widyasari, Zhou Yang, Ferdian Thung, Sheng Qin Sim, Fiona Wee, Camellia Lok, Jack Phan, Haodi Qi, Constance Tan, Qijin Tay, and David Lo
School of Computing and Information System, Singapore Management University
{ratnadiraw.2020,zyang,ferdianthung,sqsim.2018,fiona.wee.2018,camellialok.2017,jack.phan.2018}@smu.edu.sg
{haodi.qi.2017,hytan.2018,qijin.tay.2018,davidlo}@smu.edu.sg

***Abstract*—Machine learning (ML) has gained much attention and been incorporated into our daily lives. While there are numerous publicly available ML projects on open source platforms such as GitHub, there have been limited attempts in filtering those projects to curate ML projects of high quality. The limited availability of such high-quality dataset poses an obstacle in understanding ML projects. To help clear this obstacle, we present *NICHE*, a manually labelled dataset consisting of 572 ML projects. Based on evidences of good software engineering practices, we label 441 of these projects as engineered and 131 as non-engineered. This dataset can help researchers understand the practices that are followed in high-quality ML projects. It can also be used as a benchmark for classifiers designed to identify engineered ML projects.**

***Index Terms*—Engineered Software Project, Machine Learning, Python, Open Source Projects**

I. Introduction

There are many valuable pieces of information stored in a version control system of a project; they include: source code, documentation, issue reports, test cases, list of contributors, etc. Researchers mine these software repositories to get useful insights related to how bugs are fixed [1], how developers collaborate [2] and so on. With the abundance of the open source repositories in GitHub, researchers can mine for insights and validate hypotheses on a large corpus of data. However, Kalliamvakou et al. showed that most repositories in Github are of low-quality [3], [4], which can lead to wrong and biased conclusions. To avoid skewed findings, researchers usually take some measures to filter out low-quality projects, e.g., by choosing projects with a high number of stars (which is considered to reflect the projects' popularity). Unfortunately, popularity may not be correlated with project quality [5]. Therefore, Munaiah et al. propose an approach to find high-quality software projects, more specifically; by identifying engineered software projects [6]. Such projects are essential for mining software repository (MSR) research, as they allow for high-quality findings to be uncovered (from high-quality data).

Machine learning (ML) projects are becoming increasingly popular and play essential roles in various domain, e.g., code processing [7], [8], self-driving cars, speech recognition [9], etc. Despite widespread usage and popularity, only a few research works try to examine AI and ML projects to identify unique properties, development patterns, and trends. Gonzalez et al. [10] find that the AI & ML community has unique characteristics that should be considered in future software engineering and MSR research. For example, more support is needed for Python as the main programming language, and there are significant differences between internal and external contributors in AI & ML projects. We coin a term for such research: Mining Machine Learning Repository (MLR). Similar to conventional MSR research, MLR also requires high-quality projects. In GitHub, there are many tutorials, resource pages, courseworks and toy projects that are related to ML; some of which are very popular but unsuitable for MLR research. To facilitate MLR research, we present a curated dataset of ENgIneered MaCHine LEarning Projects in Python (*NICHE*). We first automatically identify projects in GitHub that: (1) use one of the popular ML libraries, and (2) satisfy some basic quantitative quality metrics. This process returns 572 ML projects from GitHub. Next, we manually analyze the 572 ML projects and label them as engineered or not engineered. This dataset can be used as the raw material for MLR research, or as the benchmark for evaluating classifiers designed to identify engineered ML projects.

We label the dataset manually to ensure high quality and accurate labels. Our criteria for assessing an ML project are rooted in Munaiah et al. work [6]. We check 8 distinct dimensions of a project (architecture, community, CI, documentation, history, issues, license and unit testing) to evaluate whether the project is engineered or not. Out of the 572 projects we collected, 441 projects are labelled as engineered ML projects, and 131 projects are labelled as non-engineered ML projects. There are several related datasets in the literature. Datasets from [6] and [11] have labels indicating whether a project is engineered or not, but they do not contain ML projects. Gonzalez et al. [10] collected a dataset of ML & AI projects, but these projects are not comprehensively assessed based on their adoption of good software engineering practices. They only eliminate tutorials, homework assignments and so on. We make our dataset publicly available[1].

The rest of this paper is organized as follow. Section 2 describes the methodology used to collect and filter the dataset, as well as how the dataset is stored. Section 3 gives an overview of the dataset. In Section 4, we propose some

[1] https://doi.org/10.6084/m9.figshare.21967265

572 ML projects

"engineered" and **"not engineered"** according to **8 dimensions** including Continuous Integration

ML Projects

NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python

Ratnadira Widyasari, Zhou Yang, Ferdian Thung, Sheng Qin Sim, Fiona Wee, Camellia Lok, Jack Phan, Haodi Qi, Constance Tan, Qijin Tay, and David Lo
School of Computing and Information System, Singapore Management University
{ratnadiraw.2020,zyang,ferdianthung,sqsim.2018,fiona.wee.2018,camellialok.2017,jack.phan.2018}@smu.edu.sg
{haodi.qi.2017,hytan.2018,qijin.tay.2018,davidlo}@smu.edu.sg

***Abstract*—Machine learning (ML) has gained much attention and been incorporated into our daily lives. While there are numerous publicly available ML projects on open source platforms such as GitHub, there have been limited attempts in filtering those projects to curate ML projects of high quality. The limited availability of such high-quality dataset poses an obstacle in understanding ML projects. To help clear this obstacle, we present *NICHE*, a manually labelled dataset consisting of 572 ML projects. Based on evidences of good software engineering practices, we label 441 of these projects as engineered and 131 as non-engineered. This dataset can help researchers understand the practices that are followed in high-quality ML projects. It can also be used as a benchmark for classifiers designed to identify engineered ML projects.**

***Index Terms*—Engineered Software Project, Machine Learning, Python, Open Source Projects**

I. Introduction

There are many valuable pieces of information stored in a version control system of a project; they include: source code, documentation, issue reports, test cases, list of contributors, etc. Researchers mine these software repositories to get useful insights related to how bugs are fixed [1], how developers collaborate [2] and so on. With the abundance of the open source repositories in GitHub, researchers can mine for insights and validate hypotheses on a large corpus of data. However, Kalliamvakou et al. showed that most repositories in Github are of low-quality [3], [4], which can lead to wrong and biased conclusions. To avoid skewed findings, researchers usually take some measures to filter out low-quality projects, e.g., by choosing projects with a high number of stars (which is considered to reflect the projects' popularity). Unfortunately, popularity may not be correlated with project quality [5]. Therefore, Munaiah et al. propose an approach to find high-quality software projects, more specifically; by identifying engineered software projects [6]. Such projects are essential for mining software repository (MSR) research, as they allow for high-quality findings to be uncovered (from high-quality data).

Machine learning (ML) projects are becoming increasingly popular and play essential roles in various domain, e.g., code processing [7], [8], self-driving cars, speech recognition [9], etc. Despite widespread usage and popularity, only a few research works try to examine AI and ML projects to identify unique properties, development patterns, and trends. Gonzalez et al. [10] find that the AI & ML community has unique characteristics that should be considered in future software engineering and MSR research. For example, more support is needed for Python as the main programming language, and there are significant differences between internal and external contributors in AI & ML projects. We coin a term for such research: Mining Machine Learning Repository (MLR). Similar to conventional MSR research, MLR also requires high-quality projects. In GitHub, there are many tutorials, resource pages, courseworks and toy projects that are related to ML; some of which are very popular but unsuitable for MLR research. To facilitate MLR research, we present a curated dataset of ENgIneered MaCHine LEarning Projects in Python (*NICHE*). We first automatically identify projects in GitHub that: (1) use one of the popular ML libraries, and (2) satisfy some basic quantitative quality metrics. This process returns 572 ML projects from GitHub. Next, we manually analyze the 572 ML projects and label them as engineered or not engineered. This dataset can be used as the raw material for MLR research, or as the benchmark for evaluating classifiers designed to identify engineered ML projects.

We label the dataset manually to ensure high quality and accurate labels. Our criteria for assessing an ML project are rooted in Munaiah et al. work [6]. We check 8 distinct dimensions of a project (architecture, community, CI, documentation, history, issues, license and unit testing) to evaluate whether the project is engineered or not. Out of the 572 projects we collected, 441 projects are labelled as engineered ML projects, and 131 projects are labelled as non-engineered ML projects. There are several related datasets in the literature. Datasets from [6] and [11] have labels indicating whether a project is engineered or not, but they do not contain ML projects. Gonzalez et al. [10] collected a dataset of ML & AI projects, but these projects are not comprehensively assessed based on their adoption of good software engineering practices. They only eliminate tutorials, homework assignments and so on. We make our dataset publicly available[1].

The rest of this paper is organized as follow. Section 2 describes the methodology used to collect and filter the dataset, as well as how the dataset is stored. Section 3 gives an overview of the dataset. In Section 4, we propose some

[1] https://doi.org/10.6084/m9.figshare.21967265

572 ML projects

"**engineered**" and "**not engineered**" according to **8 dimensions** including Continuous Integration

Due to possible computational issues, we want to select a **statistically significant sampling** of **337 projects**

Research Process

Code Smell Detector

We **want** to **build** a **static analyzer** able to **detect** **ML-specific code smells** starting from the **catalog** proposed **by Zhang et al.**

CodeSmile

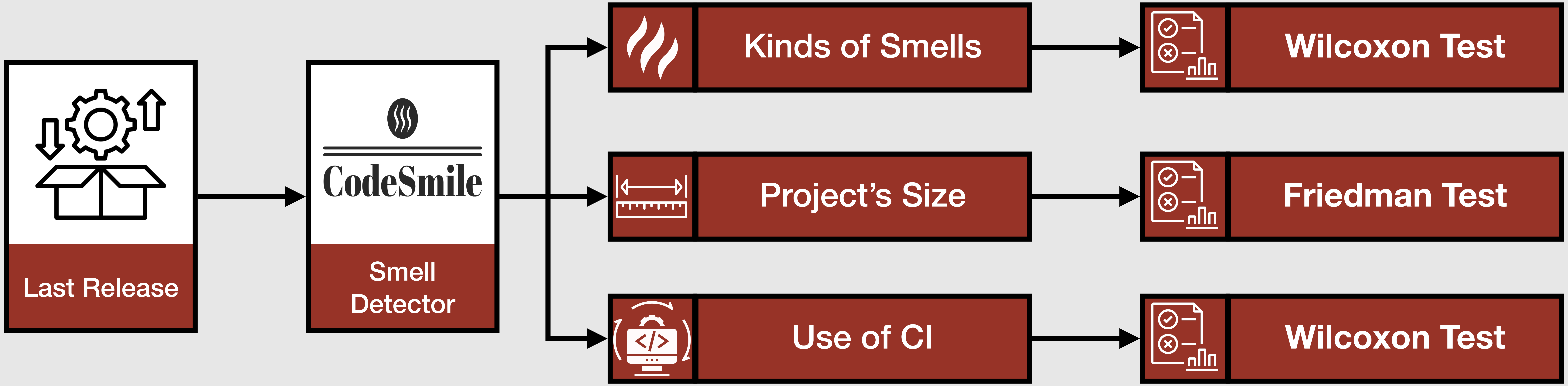
We plan to analyze historical information on over 400k commits

Research Process

How will we analyze the results?

RQ0: Prevalence

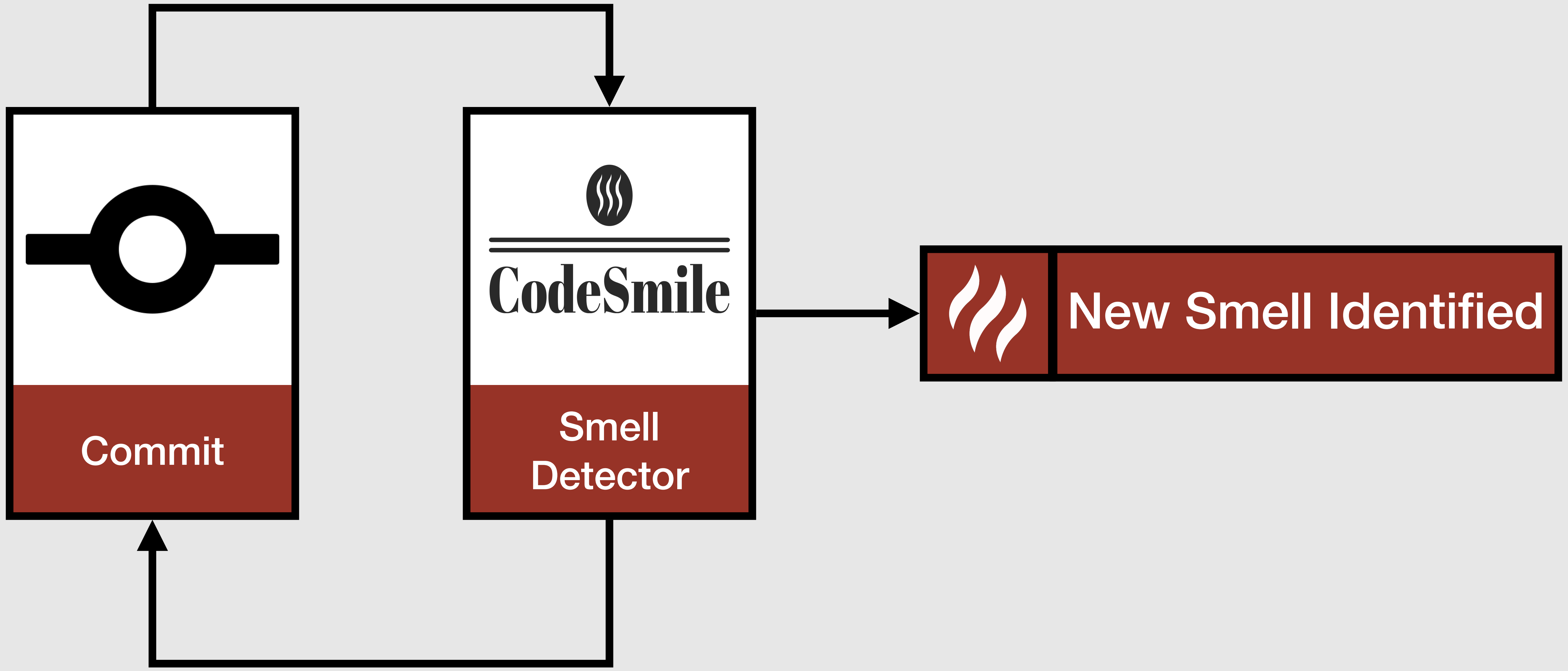
We will **statistically** assess the **differences between smells** by considering: the **kinds of smells**, the **project's size**, and the use of **CI** considering only the **last commit**



How will we analyze the results?

RQ1: When are introduced

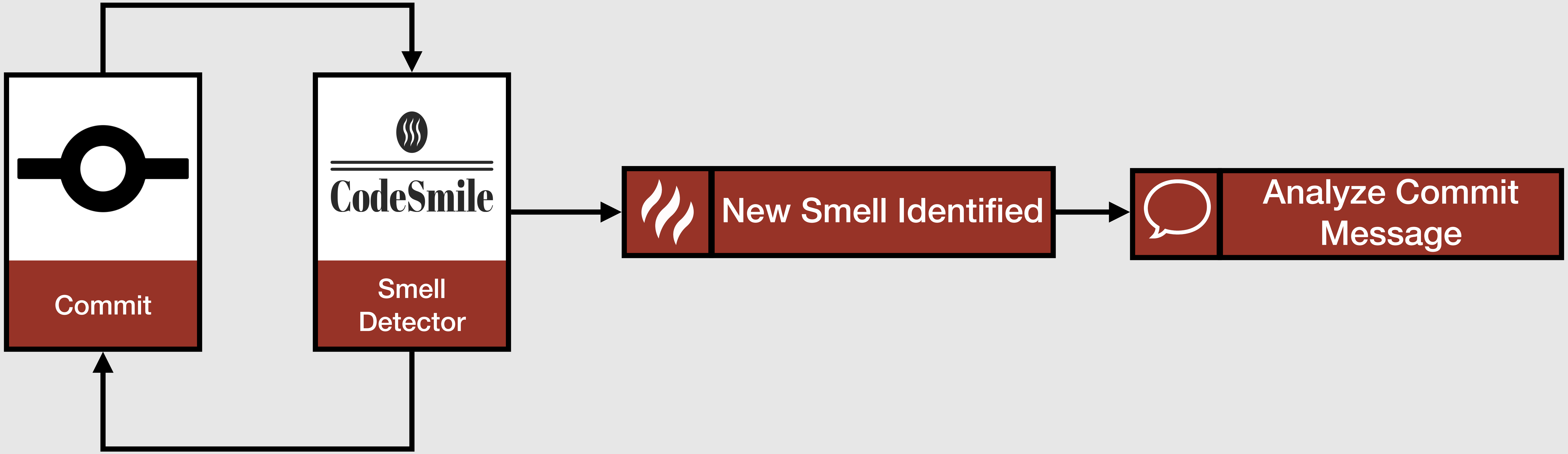
We will run **CodeSmile commit by commit** to discover **when a code smell** is **introduced in** ML-enabled systems



How will we analyze the results?

RQ2: What are the tasks during the introduction

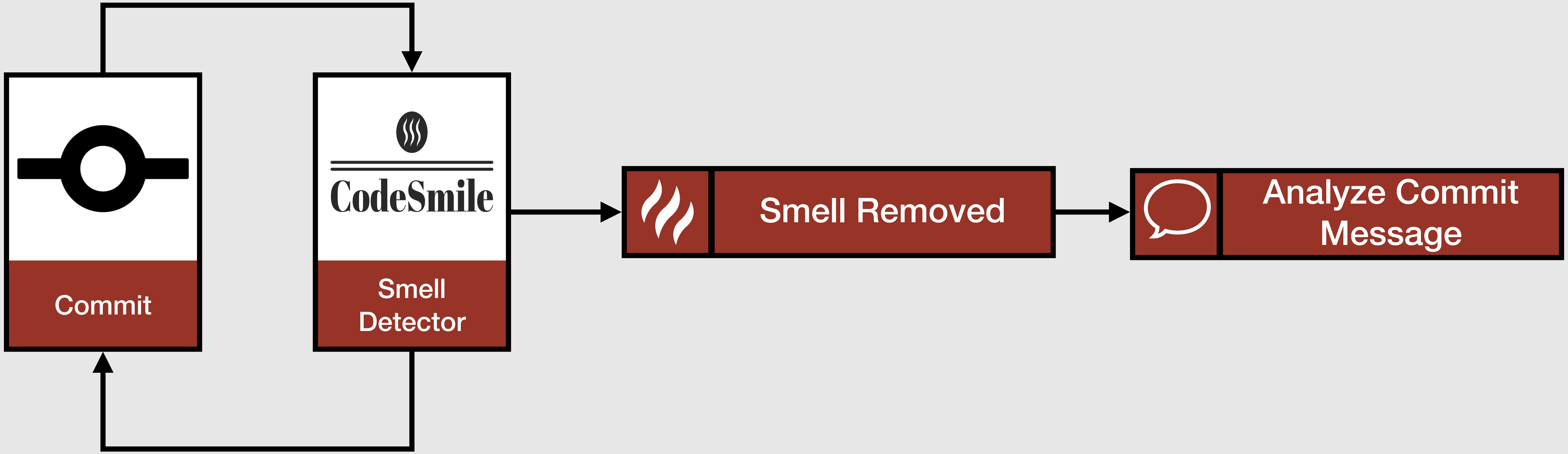
We will analyze **commit messages** for each commit and label them by applying a **keyword pattern-matching**



How will we analyze the results?

RQ3: When and How are removed

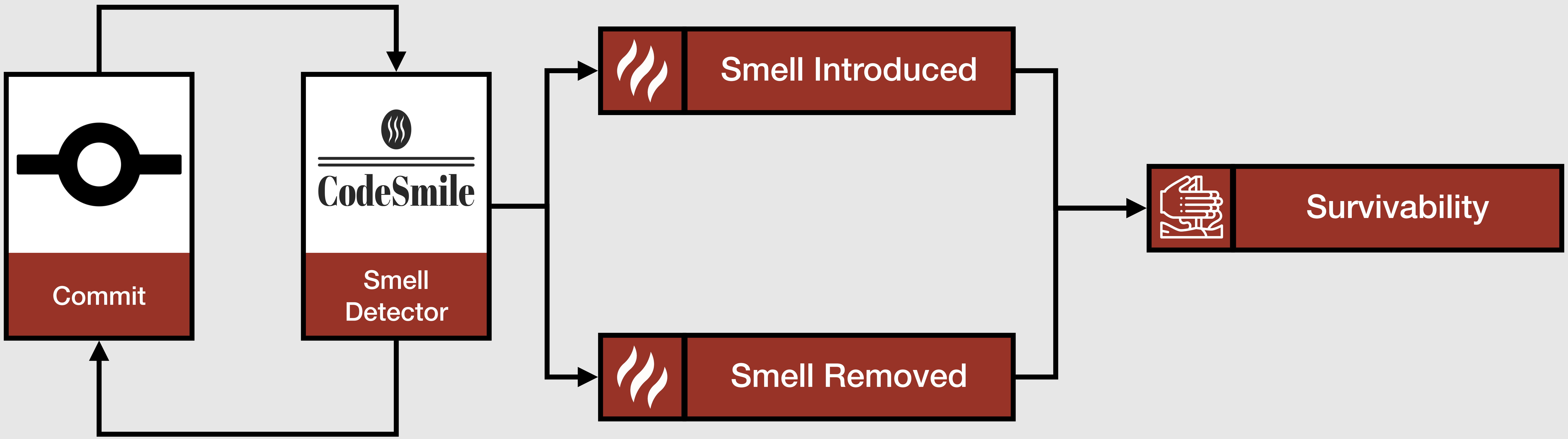
We want to run CodeSmile to discover **when** a code smell is removed and what **task** performed during its removal



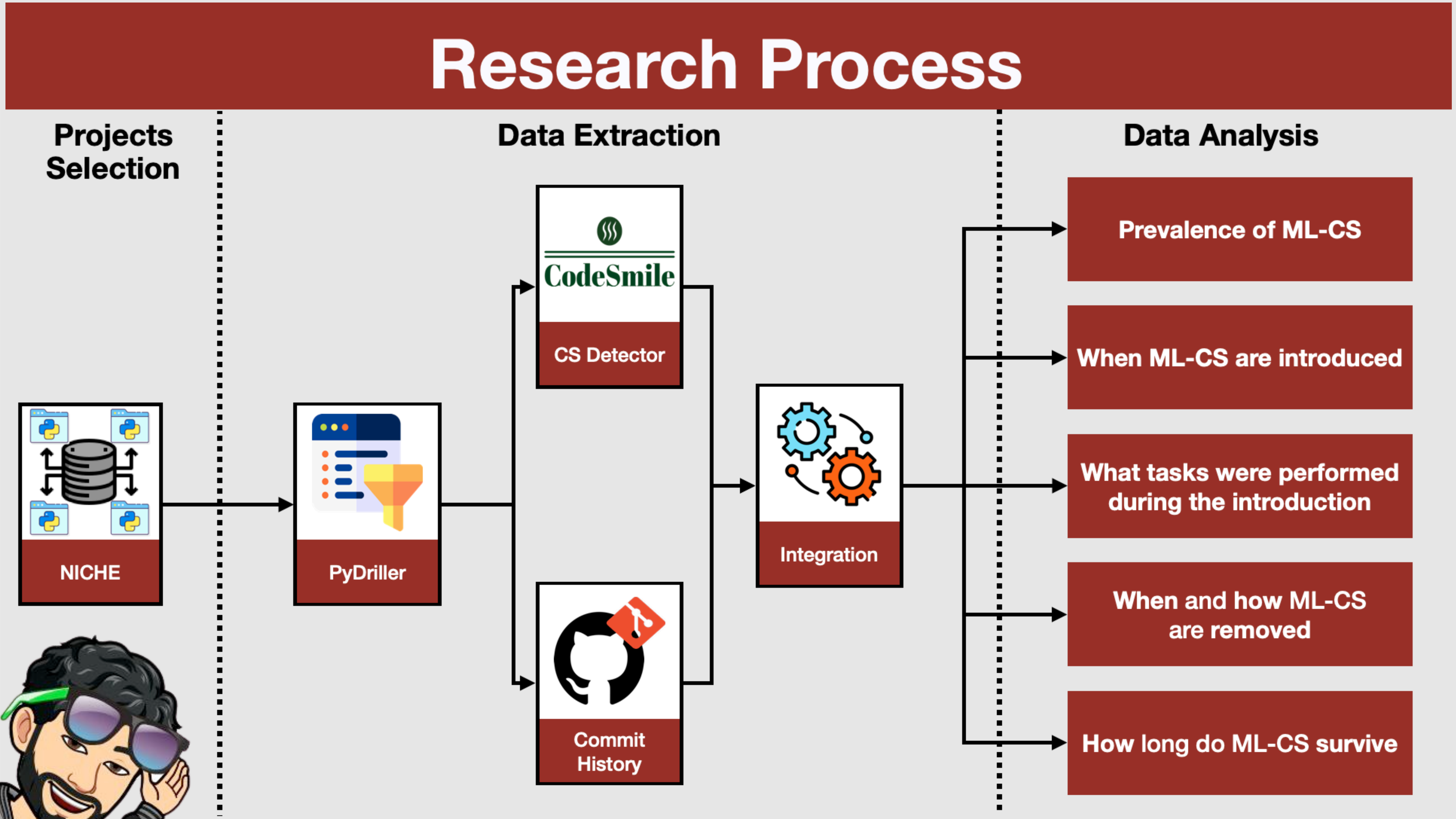
How will we analyze the results?

RQ4: Survivability

We will **combine** the information of the previous RQs to identify the **survivability** time of ML-CS



Thanks!



sesa lab
SOFTWARE ENGINEERING SALERNO

SCAN ME!
I'm the paper!

giagiordano@unisa.it

https://giammariagiordano.github.io/giammaria-giordano/

@GiammariaGiord1